Semantics of linear logic and higher-order model-checking

Charles Grellois

Ecole Doctorale 386 — Sciences Mathématiques de Paris Centre

Semantics of linear logic and higher-order model-checking

Thèse pour l'obtention du titre de Docteur de l'Université Paris Diderot (Paris 7) Sorbonne Paris Cité, spécialité Informatique.

par

Charles Grellois

Thèse soutenue publiquement le 8 avril 2016 devant le jury constitué de

Ugo Dal Lago	Examinateur
Thomas Ehrhard	Président du jury
Jean Goubault-Larrecq	Examinateur
Paul-André Melliès	Directeur de thèse
Luke Ong	Examinateur
Olivier Serre	Co-directeur de thèse
Kazushige Terui	Rapporteur
Igor Walukiewicz	Rapporteur

Résumé

Dans cette thèse, nous envisageons des problèmes de model-checking d'ordre supérieur à l'aide d'approches issues de la sémantique et de la logique. Le model-checking d'ordre supérieur étudie la vérification de propriétés, exprimées en *logique monadique du second ordre*, sur des arbres infinis générés par une classe de systèmes de réécriture appelés *schémas de récursion d'ordre supérieur*. Ces systèmes sont équivalents au λ-calcul simplement typé avec récursion, et peuvent donc être étudiés à l'aide d'outils sémantiques.

Plus précisément, l'objet de cette thèse est de relier le model-checking d'ordre supérieur à une série de concepts de premier plan en sémantique contemporaine, tels que la *logique linéaire* et sa *sémantique relationnelle*, la *logique linéaire indexée*, les *lois distributives* entre comonades, les *comonades paramétrées* et la *logique tensorielle*. Nous verrons que ces concepts contribuent de façon particulièrement naturelle à l'étude du model-checking d'ordre supérieur.

Notre approche débute par une étude du système de types intersection de Kobayashi et Ong, qui permet de typer un schéma de récursion d'ordre supérieur avec les états d'un automate donné encodant une formule de la logique monadique du second ordre. Le schéma admet pour type l'état initial de l'automate si et seulement si l'arbre infini qu'il représente satisfait la propriété encodée par l'automate. En dépit de cette adéquation, le système de types de Kobayashi et Ong a été pensé indépendamment de la connexion existant entre les types intersections et les modèles de la logique linéaire, relation observée par Bucciarelli, Ehrhard, de Carvalho et Terui. Nous avons donc cherché à relier ces deux domaines.

Notre analyse nous a permis de définir un système de types intersection dérivé de celui de Kobayashi et Ong, capturant lui aussi le model-checking d'ordre supérieur de façon adéquate. Contrairement au système original, notre système est formulé de façon *modale*, et correspond à une *sémantique finitaire* de la logique linéaire obtenue en composant la modalité exponentielle usuelle avec une comonade colorant les formules. Nous équipons cette sémantique de la logique linéaire avec un opérateur de point fixe inductif-coinductif, et obtenons ainsi un modèle du λ-calcul avec récursion dans lequel l'interprétation d'un schéma de récursion d'ordre supérieur est l'ensemble des états depuis lesquels l'arbre infini qu'il représente est accepté. La finitude de la sémantique nous permet de donner de nouvelles preuves de plusieurs résultats de décidabilité pour des problèmes de model-checking d'ordre supérieur, dont le *problème de la sélection* formulé récemment par Carayol et Serre.

La sémantique finitaire que nous définissons est inspirée du théorème d'écrasement extensionnel d'Ehrhard, qui montre que l'écrasement extensionnel du modèle relationnel de la logique linéaire correspond à sa sémantique finitaire donnée par le modèle de Scott. Ce résultat nous permet de définir dans un premier temps la comonade de coloration et l'opérateur de point fixe inductif-coinductif dans une sémantique quantitative correspondant à une variante infinie (et non-continue) du modèle relationnel de la logique linéaire.

Abstract

This thesis studies problems of higher-order model-checking from a semantic and logical perspective. Higher-order model-checking is concerned with the verification of properties expressed in *monadic second-order logic*, specified over infinite trees generated by a class of rewriting systems called *higher-order recursion schemes*. These systems are equivalent to simply-typed λ-terms with recursion, and can therefore be studied using semantic methods.

The more specific purpose of this thesis is to connect higher-order model-checking to a series of advanced ideas in contemporary semantics, such as *linear logic* and its *relational semantics*, *indexed linear logic*, *distributive laws* between comonads, *parametric comonads* and *tensorial logic*. As we will see, all these ingredients meet and combine surprisingly well with higher-order model-checking.

The starting point of our approach is the study of the intersection type system of Kobayashi and Ong. This intersection type system enables one to type a higher-order recursion scheme with states of a given automaton, associated with a formula of monadic second-order logic. The recursion scheme is typable with the initial state of the automaton if and only if the infinite tree it represents satisfies the formula of interest. In spite of this soundness-and-completeness result, the original type system by Kobayashi and Ong was not designed with the connection between intersection types and models of linear logic observed by Bucciarelli, Ehrhard, de Carvalho and Terui in mind. Our work has thus been to connect these two fields.

Our analysis leads us to the definition of an alternative intersection type system, which enjoys a similar soundness-and-completeness theorem with respect to higher-order model-checking. In contrast to the original type system by Kobayashi and Ong, our *modal* formulation is the proof-theoretic counterpart of a *finitary semantics* of linear logic, obtained by composing the traditional exponential modality with a coloring comonad. We equip the semantics of linear logic with an inductive-coinductive fixpoint operator. We obtain in this way a model of the λ-calculus with recursion in which the interpretation of a higher-order recursion scheme is the set of states from which the infinite tree it represents is accepted. The finiteness of the semantics enables us to reestablish several results of decidability for higher-order model-checking problems, among which the *selection problem* recently formulated and proved by Carayol and Serre.

This finitary semantics are inspired from the extensional collapse theorem of Ehrhard, who shows that the relational semantics of linear logic collapses extensionally to the finitary semantics provided by Scott lattices. For that reason, we start in a preliminary approach to define the coloring comonad and the inductive-coinductive fixpoint operator in the quantitative semantics provided by an infinitary (and non-continuous) version of the relational model of linear logic.

Remerciements

Si ce manuscrit n'est signé que d'un nom, il résulte de nombreux échanges et soutiens ; de critiques nécessaires, et d'inspirations bienvenues. J'ai souvent vu des thèses commencer par ces mots :

« Cette thèse n'aurait pu voir le jour sans... »,

mots dont je ne saisis véritablement le sens qu'à l'issue de ce travail éprouvant et passionnant. Ce manuscrit terminé, voici venu mon tour de remercier ceux qui ont contribué à ce cheminement intellectuel et humain ; qui m'ont éclairé scientifiquement, et m'ont soutenu dans la difficulté.

C'est bien naturellement que mes remerciements vont en premier lieu à mon directeur de thèse, Paul-André Melliès, qui m'a apporté sa confiance et m'a permis de travailler avec une grande liberté sur un sujet de synthèse, réunissant deux facettes élégantes de l'informatique théorique. La profondeur de sa vision, guidée par le cadre unificateur et révélateur qu'est la logique tensorielle, a fortement développé et influencé ma compréhension du domaine. Je le remercie également pour son aide précieuse dans les moments les plus délicats de la rédaction de ce manuscrit.

Je dois également beaucoup à mon co-directeur de thèse, Olivier Serre, qui a su être présent à chaque fois que cela était nécessaire, et m'a apporté bien des conseils avisés. Ma compréhension de la théorie des automates et des jeux de parité a beaucoup bénéficié de ses explications. Je tiens aussi à le remercier pour son aide précieuse face aux diverses démarches administratives, qu'elles concernent la thèse ou la recherche de postdocs.

Les deux rapporteurs de cette thèse, Kazushige Terui et Igor Walukiewicz, ont eu la gentillesse de relire avec minutie ce document pourtant un peu trop long. Leurs remarques exigeantes et pertinentes ont très certainement contribué à l'amélioration du manuscrit. Je les en remercie profondément.

Je suis reconnaissant aux quatre autres membres du jury — pour leur participation lors de cette soutenance, mais également à d'autres titres. Je dois à Thomas Ehrhard, directeur du défunt laboratoire PPS, l'autorisation de participer à bien des événements scientifiques, qui m'ont beaucoup apporté, mais aussi d'organiser au laboratoire un groupe de travail connexe à ma thématique de recherche. Cette confiance m'a profondément touché, tout comme j'ai véritablement apprécié la liberté laissée aux doctorants à PPS. Je remercie aussi

Ugo dal Lago pour mes débuts en postdoc à Bologne, où je trouve beaucoup de satisfaction intellectuelle à découvrir les aspects probabilistes du calcul. Ugo, merci aussi pour ta patience face à l'allongement conséquent de la durée de rédaction de ma thèse... Merci à Jean Goubault-Larrecq pour ses cours passionnants à Cachan, il y a bien des années déjà. Et merci à Luke Ong, qui m'a accueilli en stage mais aussi en visite durant ma thèse, et m'a permis de découvrir l'atmosphère unique d'Oxford. Je le remercie aussi pour les récentes invitations à NII Shonan et à Singapour.

Cette thèse a été en partie rédigée à Dundee, où j'ai fait la connaissance de Marco Gaboardi, que je remercie pour sa gentillesse et son humanité profonde, et avec qui je suis heureux de collaborer. J'y ai également rencontré Gian Pietro, à qui je dois de très bons souvenirs, même si la période était à la rédaction. Je remercie aussi le reste du groupe de Dundee pour son accueil au cours de ces quelques mois en Ecosse.

Lors de mon monitorat à Paris 7, j'ai eu la chance d'enseigner avec des collègues compétents et agréables, notamment Daniele Varacca, que je remercie tout particulièrement pour la confiance qu'il m'a accordée, Antonio Bucciarelli, Yann Régis-Gianas, Jean Krivine, et Fabrice Ben Hamouda. Je suis très reconnaissant à Mai Gehrke de m'avoir proposé d'assurer les TDs de son cours au LMFI. J'ai beaucoup appris lors de cette expérience, et ai été très honoré de cette marque de confiance.

Je remercie également l'ensemble des membres de PPS, dont Alexis et Christine pour leur encadrement des thésards, Odile pour son rôle précieux dans le laboratoire, et Michele pour sa belle réponse sur la réduction linéaire de tête l'été dernier. Merci à tous les doctorants côtoyés au cours de ces années, dont la liste exacte serait trop longue à établir, et particulièrement à la « secte du 3026 » : Amina, Clément, Yann, Marie, à qui je souhaite beaucoup de bonheur avec sa petite famille, Pierre Cagne, Pierre Vial et Gabriel par assimilation récente, et Kenji, visiteur régulier. Merci pour les discussions divertissantes et constructives, et pour la belle expérience de la colocation à ETAPS l'an dernier. Je remercie aussi, parmi les doctorants et post-doctorants, Axel, Nathanaël, Kuba, Matthieu, Pierre-Marie, Jovana, Hadrien, Thomas, Shahin, Ioana, Alexis, Guillaume, Giulio, Jonas, Noam, Thibaut, Maxime, Ludovic, Cyrille, Gabriel et Cyprien – mais aussi Flavien et Raphaëlle, que j'ai le plaisir de retrouver à Bologne.

Merci aussi aux chercheurs rencontrés au cours d'événements scientifiques tels que le séminaire Chocola, à ceux qui sont venus régulièrement au groupe de travail sur les schémas, et à ceux avec qui j'ai interagi lors de la recherche de postdocs, pour leur accueil bienveillant. Je pense notamment à Arnaud Carayol, à David Baelde, à Sylvain Salvati, à Pierre Clairambault, à Luigi Santocanale, à Lionel Vaux, à Daniel Hirschkoff et à Jean-Marc Talbot.

Avant de commencer ma thèse, au cours de mes années de normalien, j'ai eu la chance et le plaisir de participer au groupe de travail de logique des étudiants qui se tenait à Ulm, groupe fondé par Marc Bagnol et qui réunissait entre autres, outre Marc, Aloïs Brunel, Guillaume Brunerie, Adrien Guatto, Bap-

tiste Mélès, Guillaume Munch-Maccagnoni, Pierre-Marie Pédrot, Maël Pégny, Pablo Rauzy, Silvain Rideau, Gabriel Scherer, Anne-Sophie de Suzzoni... Merci à Kenji d'avoir repris le flambeau. A la même période, j'ai eu le plaisir de fréquenter régulièrement la « bande de la Butte aux Cailles » : Adrien, Olivier et Alain, que je remercie pour tous ces bons moments. Je rencontrais aussi Silvain et Catherine, que j'ai toujours beaucoup de plaisir à voir, et Pierre, à qui j'espère une fin de thèse proche. En parallèle, j'ai eu le véritable plaisir d'enseigner à Nasser, Nour et Naïla, trois jeunes élèves brillants et stimulants auxquels je souhaite réussite et bonheur dans les voies qu'ils choisiront. Je remercie aussi Edouard et Richard, camarades de prépa restés fidèles amis depuis toutes ces années.

Je tiens à remercier la « bande de la F@O » – puisse la signification de cet acronyme rester méconnue –, c'est-à-dire Lucas, Thomas et Antoine, pour leur amitié depuis quinze ans déjà. Mon penchant pour l'informatique doit beaucoup à Antoine, aux « coding parties » et aux réparations de vieilles machines, en des temps déjà lointains. Merci à Xavier Lafaysse, qui nous aida à souder notre bande. A peu près à la même époque, je rencontrais Augustin, que je remercie de son amitié. Merci aussi à la vieille bande des scouts, pour notre amitié qui dure depuis quinze ans et qui nous emmena presque au bout du monde : Eloïse, Arnaud, Mathieu, Benjamin, et le « vingt-quatrième », Pierre-Antoine.

Il est temps d'utiliser ces mots : cette thèse n'aurait pu voir le jour sans l'enseignement de Jacques Jeanjean, sans lequel je serais vraisemblablement devenu biologiste. Mon parcours mathématique a également bénéficié du regard d'esthète de Denis Favennec. Je dois aussi à Gilbert Munos, et à Guillaume Brevet.

Cette thèse a bénéficié du soutien précieux de Dominique Laurent, qui n'a jamais compté son temps. Je l'en remercie du fond du cœur.

Je remercie profondément ma famille pour son soutien, tout particulièrement lors de la phase de rédaction, et notamment ma mère, mon père, mes frères et sœurs, mais aussi ma marraine Béatrice, et Mamette. Grand-Père, je pense bien à toi en ce moment. Merci aussi à tous les autres membres de notre grande famille.

Merci à ceux qui contribué à la relecture de cette thèse : Clément Aubert, Clément Jacq, Olivier, et surtout Paul-André.

Je termine (presque) cette longue liste par un remerciement plus généralement adressé à tous ceux que j'aurais pu oublier, mais aussi par un encouragement pour Pierre et Clément, mes « petits frères » de thèse, auxquels je souhaite un beau parcours.

Pour terminer, je remercie Astrid de m'avoir accompagné au cours de cette thèse. Et je remercie également Athéna, mon quadrupède préféré, pour ses tentatives de contribution au manuscrit. Certaines coquilles pourraient provenir de sa patte...

Contents

Chapter 1

Introduction

1.1 A semantic approach to higher-order model-checking

In a world in which automated processes play an increasingly important rôle, notably in critical systems, software reliability is an essential issue. Among the various techniques used to improve the reliability of the high-level as well as low-level code are

- *semantics*, which studies the mathematical structure and modeling of software programming,
- *program verification*, whose aim is to check whether a given program satisfies a given logical specification.

A key method in verification is *model-checking*, introduced in the eighties by Clarke, Emerson and Sifakis [CE81, QS82]. The main idea is to reason not on the original program itself, but to build an *abstract model* of this program, keeping only some of its original features. For instance, the abstract model may consider the conditional `if` statements no longer as computable expressions, but as uninterpreted constants. In this case, the model will typically be a tree, in which branching indicates the potential execution traces resulting of the evaluation of the `if` statement in the original program. This abstraction of the original program enables one to consider models which are not Turing-complete, and over which the satisfiability of properties expressed in adequate logics may be decidable. The ultimate goal of model-checking is to determine automatically whether the abstracted program has the property of interest. At first, *finite* models were thus considered, as finite graphs for instance; but in the second part of the nineties the focus of the model-checking community moved to *finitely presentable* infinite models – as infinite trees generated by pushdown systems for instance [Wal96, BEM97].

Higher-order model-checking. The successful emergence of higher-order programming languages – such as C++, Haskell, OCaml, Javascript, Python, or Scala – is a real challenge for program verification, notably due to the presence of higher-order recursion. In these languages, a function may receive other functions as inputs, a typical example being the function `map` using a function

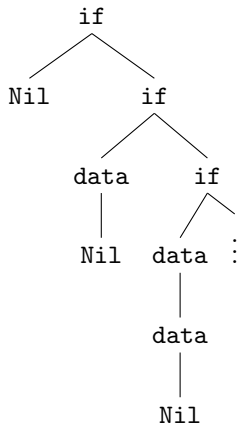
$A \to A$ and a list of elements of A to output the list obtained by applying the function to every element of the input list. An abstract model for such programs is *higher-order recursion schemes* (HORS). They originate from Nivat's recursive program schemes [Niv72] which intended at giving a semantics to LISP programs. These recursive schemes were extended to higher-order by Damm [Dam77a,Dam77b] in order to model ALGOL 68 programs, a step which led to the definition of higher-order recursion schemes.

Let us explain informally how these higher-order recursion schemes act as models of programs, and at the same time as finite representations of the possibly infinite trees of their execution traces. Consider a very simple program which receives a connection of a client and builds a list of all the data it receives until the connection is closed for some reason. This program can be abstracted as the following HORS:

$$\mathcal{G}_1 \quad = \quad \begin{cases} \texttt{S} & = & \texttt{L Nil} \\ \texttt{L}\ x & = & \texttt{if}\ x\ (\texttt{L}\ (\texttt{data}\ x)) \end{cases}$$

`S` is the start symbol, and can be understood as the representation of the `main` function of the program. It calls `L`, which represents a function `listen`, on a representation of the empty list. In this abstraction of the program, the conditional statement `if` is treated as a constant: it is just a symbol which is used to compute a tree of execution traces of the original program. We also abstract the precise data that was received as a constant symbol `data`. The HORS $\mathcal{G}_1$ defines a rewriting system, which starts from the start symbol `S` and applies infinitely the rules of the recursion scheme, see Figure 3.6 on p.69 for details. This process computes the following infinite tree, whose maximal branches describe the set of traces of executions of the original program:

$\mathcal{T}_1 \quad =$



In spite of the simplicity of $\mathcal{G}_1$, this tree is *not* regular: it can not be represented as the unfolding of a finite graph. In fact, it was remarked in the early 2000s that the infinite trees with finite representation typically studied in model-checking could be represented by classes of higher-order recursion schemes. These recursion schemes are in particular deeply related to pushdown systems. It was thus compelling to study the model-checking of the infinite trees they represent, but for which logic?

Given a model, the choice of a logic to describe its properties requires to consider a trade-off between *expressivity* and *complexity*. In the case of

infinite trees, *monadic second-order logic* (MSO) is a logic of choice, as its model-checking is decidable over the complete binary tree [Rab69] but also on infinite graphs generated by pushdown structures [MS85]. It is therefore appealing to consider the following *local higher-order model-checking problem*: given an MSO formula φ and a higher-order recursion scheme $\mathcal{G}$, does φ hold at the root of the infinite tree $\langle\, \mathcal{G} \,\rangle$ represented by $\mathcal{G}$? The first decidability proof was given in 2001 for order-2 safe recursion schemes by Knapik, Niwinski and Urzyczyn [KNU01], and then extended to all safe schemes in 2002 by the same authors [KNU02]. At the same time, Caucal proved this result using a different representation of the trees generated by higher-order recursion schemes [Cau02]. A first step towards the relaxation of the safety hypothesis was made in 2005 by Knapik, Niwinski, Urzyczyn and Walukiewicz [KNUW05] and independently by Aehlig, de Miranda and Ong [AdMO05]. Both approaches prove the decidability of MSO over the trees produced by order-2 recursion schemes, would they be safe or not. In 2006, Aehlig restricted the expressive power of the logic instead of the one of the recursion schemes, and proved that the satisfiability of MSO properties expressible by means of trivial alternating parity automata is decidable at the root of trees generated by all higher-order recursion schemes [Aeh06]. The same year, Ong gave the first general decidability proof of the local higher-order model-checking problem [Ong06], using a game semantics-based study of the recursion scheme. This result has been established a number of times since then, using different techniques. They are listed in our related works section (§1.2).

From the early times of model-checking, two more elaborate problems have been considered, namely *global* model-checking, which consists in computing a finite representation of the set of nodes satisfying the formula of interest, and *witness generation*. For higher-order recursion schemes, Carayol and Serre formulated such a witness generation property under the name of *MSO selection* [CS12]. It was reformulated automata-theoretically by Haddad in his PhD thesis [Had13b]: given a HORS $\mathcal{G}$ and an MSO formula φ, represented as an equivalent *alternating parity tree automaton* (APT) $\mathcal{A}$, the selection problem is to compute effectively a finite representation of an accepting execution of the automaton $\mathcal{A}$ over the infinite tree $\langle\, \mathcal{G} \,\rangle$.

In this thesis, we explain how tree automata theory and the verification of infinite systems is somewhat surprisingly related to the stream of ideas which emerged from the discovery in the late eighties of linear logic by Jean-Yves Girard [Gir87]: the relational semantics of linear logic and its connection to Bucciarelli and Ehrhard's indexed linear logic [BE00, BE01], the formal definition of the exponential modality in these models, Ehrhard's extensional collapse theorems [Ehr12a, Ehr12b], and finally Melliès' tensorial logic [MT10, Mel12, Mel16b]. Our approach notably leads us to a semantic resolution of the selection problem of Carayol and Serre.

Automata, algebraic recognition, and type systems. The notion of alternating parity tree automaton plays a central rôle in this thesis, as an automata-theoretic counterpart to monadic second-order logic formulas. Such an automaton is a non-deterministic tree automaton enriched with *alternation* and a discrimination of *accepting* and *rejecting* executions with respect to the *parity* condition. Before giving more details on these two features, let us con-

sider the simplest model-checking problem: we model an execution trace of a program as a word over an alphabet of actions, and the property to check is encoded as a finite automaton. For instance, we may consider the word of actions

$$\texttt{open} \cdot \texttt{read} \cdot \texttt{write} \cdot \texttt{close} \tag{1.1}$$

and check whether every `read` is immediately followed by a `write` using an automaton of states q_0 and q_r, the first being the initial and final state. The transition function, as expected, notably contains

$$\delta(q_0, \texttt{read}) \;=\; q_r \qquad\qquad \delta(q_r, \texttt{write}) \;=\; q_0$$

As a preliminary to the treatment of the case of *alternating* automata, let us recall that the Church encoding defines a correspondence between trees over a signature Σ and simply-typed λ-terms of ground type over the same signature, considered modulo $\beta\eta$-conversion. Let us explain this correspondence in the case of words, which correspond to trees with unary and nullary nodes. We consider a signature Σ of first-order symbols of arity 0 or 1 – the symbols of arity 0 being precisely the final letters of words. For instance, the word (1.1) is defined over the signature

$$\Sigma \;=\; \{\, \texttt{open} : 1, \texttt{read} : 1, \texttt{write} : 1, \texttt{close} : 0 \,\}$$

meaning that `close` is a final letter. The Church encoding of the letter `close` is the λ-term `close` of simple type o, and the Church encoding of the letter `read` is the λ-term `read` of simple type $o \to o$. More generally, every word w over the signature Σ is translated as a λ-term $\{\!|\, w \,|\!\}$ of simple type o, and the concatenation of letters at the beginning of a word corresponds to the application of λ-terms:

$$\{\!|\, \texttt{read} \cdot w \,|\!\} \;=\; \texttt{read}\ (\ \{\!|\, w \,|\!\}\ ) \quad \text{and} \quad \{\!|\, \texttt{write} \cdot w \,|\!\} \;=\; \texttt{write}\ (\ \{\!|\, w \,|\!\}\ )$$

The transition function may be understood as a type refinement: the transition

$$\delta(q_0, \texttt{read}) \;=\; q_r$$

amounts to attributing to the λ-term `read` the refined type

$$\texttt{read} : q_r \to q_0$$

expressing the fact that if the λ-term $\{\!|\, w \,|\!\}$ is accepted from q_r, then the λ-term `read` ($\{\!|\, w \,|\!\}$) is accepted from q_0. In the type-theoretic reformulation allowed by this Church encoding, executions of the automaton over the word w correspond to typing derivations of the word $\{\!|\, w \,|\!\}$, as for instance:

$$\dfrac{\vdash \texttt{open} : q_0 \to q_0 \qquad \dfrac{\vdash \texttt{read} : q_r \to q_0 \qquad \dfrac{\vdash \texttt{write} : q_0 \to q_r \qquad \vdash \texttt{close} : q_0}{\vdash \texttt{write}\ (\ \texttt{close}\ ) : q_r}}{\vdash \texttt{read}\ (\ \texttt{write}\ (\ \texttt{close}\ )\ ) : q_0}}{\vdash \texttt{open}\ (\ \texttt{read}\ (\ \texttt{write}\ (\ \texttt{close}\ )\ )\ ) : q_0}$$

A great advantage of this type-theoretic view is that it naturally lifts to higher-order: it is straightforward to design a type system in which the transition function of the automaton provides the typings of constants, all other

rules being standard. Subject reduction and expansion hold, and allow us to prove that the refined type a term admits is an invariant of the computation, so that a term computing a word has type q_0 if and only if the word it represents is accepted by the finite automaton of interest: typing enables us to reason directly on programs computing words, *without executing them.* This idea will be crucial for higher-order recursion schemes: instead of considering infinite executions of automata over trees produced by infinite computations, we will lift, using type systems, the properties of automata to higher-order types, and perform model-checking directly on the finite representation the recursion scheme is. The idea of relating tree automata with type systems appears in the work of Hosoya, Vouillon and Pierce on languages for XML processing [HVP05], and was adapted for the first time to higher-order recursion schemes by Kobayashi in [Kob09b].

Another interesting point of view is the one of *recognition by homomorphism*: recall that, given a language $L \subseteq A^*$, there exists a finite automaton $\mathcal{A}$ recognizing L if and only if there exists a finite *monoid* M, a subset $K \subseteq M$ and a *homomorphism*

$$\varphi : A^* \to M$$

such that

$$L = \varphi^{-1}(K).$$

In other words, there is an *algebraic structure* in which we interpret words, and their interpretations determine whether they belong to the regular language L. A nice and deep idea, appearing independently in Aehlig's [Aeh06] and in Salvati's work [Sal09], is then to look for an extension to terms — and then to higher-order recursion schemes, which are equivalent to a certain class of simply-typed λ-terms with recursion — of this idea of recognition by homomorphism. The interpretation is no longer computed in monoids, but in finite *domains*, which are algebraic structures adapted to the interpretation of terms and of higher-order recursion; these structures are naturally connected to some refined *intersection* type systems. In the simple case of finite words, the interpretation of a word in the model can be understood as the set of states from which it is accepted by the automaton; and this set coincides with the set of refined types the word admits. Aehlig's approach [Aeh06] extends to higher-order types the idea that a tree automaton divides the set of infinite trees into finitely many classes, and obtains in this way a finite semantics for λ-terms with recursion. Salvati uses connections between finite domain theory and intersection type theory to define a notion of tree automata executing on λ-terms without recursion [Sal09]. He also studies the closure properties of these automata.

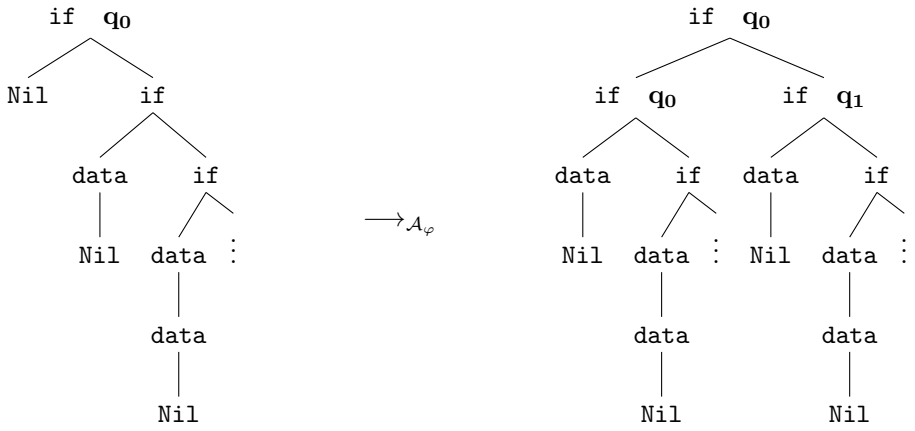
From these considerations, it appears that there is a real and difficult challenge to establish decidability of higher-order model-checking by constructing a denotational interpretation of λ-terms with recursion in a way meaningful to model-checking and alternating tree automata endowed with parity condition. This is precisely what will be achieved in this thesis: guided by intuitions coming from the relational semantics of linear logic, we will construct a denotational semantics where the interpretation of a lambda-term is precisely the

set of its refined types; and where this set of refined types reflects the infinitary and inductive-coinductive behavior of the lambda-term.

Intersection types and models of linear logic. In a first step, we focus on alternating tree automata – also called *trivial APT*, for that they are essentially APT without the parity condition. A typical transition of such a top-down automaton is

$$\delta(q_0, \texttt{if}) \;=\; (2, q_0) \wedge (2, q_1) \tag{1.2}$$

meaning that the automaton executes with state q_0 on a copy of the second child of the `if` node, and with state q_1 on another copy of this second child. As it does not require to visit the first child, it simply drops the subtree rooted at it. The effect of an execution of this transition on the infinite tree $\mathcal{T}_1$ is:



Using the correspondence between λ-terms and trees again, we can see `if` as a constant of simple type $o \to o \to o$. Aehlig's semantic investigation [Aeh06] of the model-checking of properties described by alternating tree automata over trees finitely represented by HORS was reformulated type-theoretically by Kobayashi [Kob09b], the main idea being that the transition (1.2) can be seen as a refined *intersection* type for `if`:

$$\texttt{if} : \emptyset \to \bigwedge_{i \in \{0,1\}} q_i \to q_0 \;::\; o \to o \to o \tag{1.3}$$

where the intersection type

$$\emptyset \to \bigwedge_{i \in \{0,1\}} q_i \to q_0$$

expresses the fact that given any tree T_1 and a tree T_2 accepted from states q_0 and q_1, the tree $\texttt{if}\ T_1\ T_2$ is accepted from q_0. In the corresponding intersection type system, this is proved by the derivation:

$$\dfrac{\dfrac{\dfrac{}{\emptyset \vdash \texttt{if} : \emptyset \to (q_0 \wedge q_1) \to q_0}\,\delta \qquad \emptyset}{\emptyset \vdash \texttt{if}\ T_1 : (q_0 \wedge q_1) \to q_0}\,\text{App} \qquad \dfrac{\vdots}{\Gamma_{21} \vdash T_2 : q_0} \qquad \dfrac{\vdots}{\Gamma_{22} \vdash T_2 : q_1}}{\Gamma_{21}, \Gamma_{22} \vdash \texttt{if}\ T_1\ T_2 : q_0}\,\text{App}$$

whose effect is exactly the one of the alternating automaton of interest: it *duplicates* T_2, to prove it accepted both from q_0 and q_1, and does *not* explore T_1 at all, as this subtree T_1 does not need to be accepted from any state of the automaton. Just as in the naive case of finite words and automata, the typing of constants with intersection types lifts to higher-order types, leading to a full type system for λ-terms. As a matter of fact, using this idea together with a coinductive definition of types, Kobayashi established that the local higher-order model-checking problem is decidable, for coinductive properties expressed without the need for parity conditions. But let us stick to the simple case of terms without recursion for now. It is particularly instructive to have a closer look at the Application rule:

$$\frac{\Gamma \vdash t : (\bigwedge_{i=1}^{n} \theta_i) \to \theta' :: \kappa \to \kappa' \qquad \Gamma_i \vdash u : \theta_i :: \kappa \quad \text{for every } i \in \{1, \dots n\}}{\Gamma, \Gamma_1, \dots, \Gamma_n \vdash t\ u : \theta' :: \kappa'} \tag{1.4}$$

which, in a traditional sequent calculus, would decompose in two steps:

$$\text{App}\ \frac{\Gamma \vdash t : (\bigwedge_{i=1}^{n} \theta_i) \to \theta' :: \kappa \to \kappa' \qquad \dfrac{\Gamma_i \vdash u : \theta_i :: \kappa \quad \text{for every } i \in \{1, \dots n\}}{\Gamma_1, \dots, \Gamma_n \vdash u : \bigwedge_{i=1}^{n} \theta_i :: \kappa}\ \text{Right}\ \bigwedge}{\Gamma, \Gamma_1, \dots, \Gamma_n \vdash t\ u : \theta' :: \kappa'} \tag{1.5}$$

One main novelty of our work with respect to higher-order model-checking is that we will see this introduction rule with the spectacles of linear logic — a logic based on the linear decomposition of the intuitionistic arrow:

$$A \Rightarrow B \quad = \quad !\,A \multimap B$$

This fundamental property of linear logic means that every intuitionistic arrow building a formula $A \Rightarrow B$ — or, under the Curry-Howard correspondence, the simple type $A \to B$ — can be understood as a two-step operation:

- The first step performs a number of *duplications* or *erasures* of its argument A: this is the effect of the modality ! on the formula A.
- The second step consists in a *linear* use of each element of $!\,A$: each copy made in the previous step is used exactly once in the program of type $!\,A \multimap B$ we consider.

This two-step decomposition of the intuitionistic application $A \Rightarrow B$ is strikingly similar to the decomposition of the Application rule (1.4) as a pair of introduction rules (1.5), and it turns out that intersection types relate very precisely to models of linear logic. We need, however, to clarify the meaning of the comma appearing between contexts in the Application rule (1.4): what does

$$\Gamma, \Gamma_1, \dots, \Gamma_n$$

mean? In particular, does

$$x : q_0 :: o, x : q_0 \wedge q_1 :: o$$

evaluate to

$$x \ : q_0 \wedge q_1 \ :: \ o$$

or to

$$x \ : q_0 \wedge q_0 \wedge q_1 \ :: \ o \qquad ?$$

Both solutions are admissible. In the first case, we say that the intersection type system is *idempotent*, and that it is *non-idempotent* in the second. These two intersection type disciplines relate to semantics of linear logic:

- *Idempotency* amounts to considering the intersection operator as a representation of a *set* of refined types: idempotent types relate to the Scott semantics of linear logic, in which

  $$[\![A]\!] \ = \ \mathcal{P}_{fin}([\![A]\!]).$$

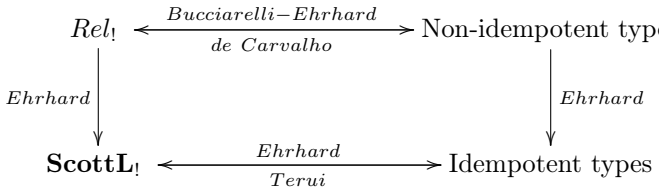
  It should be emphasized that these Scott semantics feature order-theoretic conditions which correspond, on the type-theoretic side, to the presence of *subtyping*.

- *Non-idempotency* connects to the relational semantics of linear logic, in which the exponential is interpreted using *finite multisets*, that is, finite sets enriched with finite multiplicities of elements:

  $$[\![A]\!] \ = \ \mathcal{M}_{fin}([\![A]\!]).$$

In the non-idempotent case, the rule Right $\bigwedge$ of the decomposition (1.5) corresponds in a precise sense to the Promotion rule of Bucciarelli and Ehrhard's *indexed linear logic* [BE00, BE01]. This variant of linear logic acts as a bridge between non-idempotent intersection types and denotations of the relational model of linear logic. Consider indeed the composition of a proof of the formula $!\,A$ with a proof of the formula $!\,A \multimap B$. Indexation ensures that every proof of A used to obtain $!\,A$ by a Promotion rule has the same underlying derivation tree — this corresponds, using the Curry-Howard correspondence, to the fact that the intersection type associated with $!\,A$ was built by taking the intersection of typing derivations of a same term. Indexation therefore allows to reflect in the logic the construction of intersection types, in a non-idempotent way. Further investigations on the relation between non-idempotent type systems and the relational semantics were conducted by de Carvalho [dC09] and appear in Ehrhard's work [Ehr12a]. Relations between indexed linear logic and higher-order model-checking are discussed in our paper with Melliès [GM15b]. A similar connection between idempotent intersection types and the Scott semantics of linear logic appears both in Terui's [Ter12] and Ehrhard's [Ehr12a] work. Note, however, that this connection is not associated to some indexed variant of linear logic.

An important conceptual guideline of this thesis is Ehrhard's extensional collapse result, relating Scott semantics with the relational semantics. Informally, it means that forgetting the multiplicities appearing in the relational denotations gives the denotations in the Scott semantics. To summarize, we have the following picture, where the categories $\mathit{Rel}_!$ and $\mathbf{ScottL}_!$ are the models of the λ-calculus induced by the relational and Scott models of linear logic:



(1.6)

- *North*: the connection appears in Bucciarelli and Ehrhard's work on indexed linear logic [BE00, BE01], in which indexation relates the finite multiset construction of the relational model of linear logic with the non-idempotent intersection operator of intersection type theory. The characterization of the relational semantics using a non-idempotent type system also appears in de Carvalho's work on the measure of the complexity of head normalization using Krivine machines [dC09].
- *South*: this connection appears independently in Terui's work on the complexity of normalization of λ-terms of Boolean type [Ter12], and in Ehrhard's type-theoretic characterization of his extensional collapse theorem [Ehr12a],
- *West*: this arrow is the extensional collapse theorem of Ehrhard [Ehr12b],
- *East*: this type-theoretic counterpart of the extensional collapse theorem represented by *West* is due to Ehrhard as well [Ehr12a].

In all four corners of this diagram, we can obtain a model-checking result: given a term t computing a tree and a trivial APT $\mathcal{A}$, there is an execution of $\mathcal{A}$ from q over the normal form of t if and only if $q \in [\![t]\!]$ — or, in the corresponding type system, if $\emptyset \vdash t : q :: o$. It follows that both $Rel_!$ and $\mathbf{ScottL}_!$ are suitable categorical models for recognition by alternating tree automata of λ-terms without recursion. Moreover, the existence of the finite model (South-West corner of the diagram) implies that recognition is decidable. Note that the decidability of recognition could also have been established using the intersection type system with idempotent types (South-East corner of the diagram).

Coloring annotations and the parity condition. To capture the whole higher-order model-checking problem, we need to extend the picture (1.6) with

- the *recursion* of higher-order recursion schemes,
- and the *parity condition* of alternating parity automata.

The parity condition allows to encode in an automaton $\mathcal{A}$ properties mixing inductive and coinductive behaviors. It is defined on execution trees of APT by considering a *coloring function* $\Omega : Q \to \mathbb{N}$. On every infinite branch, since there are finitely many states and thus finitely many colors, at least one of them appears infinitely often. We consider the greatest such color. If it is even, we declare the branch winning; otherwise, it is loosing. An execution tree of $\mathcal{A}$ is winning precisely when all its infinite branches are. An APT has a winning

execution over an infinite tree if and only if its root satisfies the associated monadic second-order logic formula.

To capture this parity condition, Kobayashi and Ong [KO09] extend Kobayashi's intersection type system with coloring annotations, and use the resulting system to type the rules of higher-order recursion schemes. They define a parity game $Adamic(\mathcal{G}, \mathcal{A})$ which models the recursion of the HORS $\mathcal{G}$ consistently with the parity condition of the APT $\mathcal{A}$; the idea is that Adam unfolds rules of the recursion scheme, and Eve proves that she can always type the resulting term. Kobayashi and Ong obtain that Eve has a winning strategy in this parity game if and only if there is a winning execution of $\mathcal{A}$ over the tree $\langle\, \mathcal{G} \,\rangle$ represented by $\mathcal{G}$. The decidability of finite parity games implies in turn the decidability of the local model-checking problem.

In this thesis, following our approach with Melliès [GM15d], we propose a variant of their type system in which the coloring annotation is turned into a coloring *modality* of linear logic, leading to several important consequences, as we will discuss. Before that, to understand the coloring modality, let us consider a run-tree t with a hole, and set m the maximal color seen on the finite path leading from the root of the tree – excluded – to the hole – included. We introduce in [GM15d] a new, neutral color ε for the case where $t = [\,]$, corresponding to the absence of colors on the path. Then the type of t will be $\boxdot_m\ q' \to q$: the coloring modality reflects in the refined types the color seen on the finite paths leading to holes. Informally, the fixpoint operator Y of the λ-calculus with recursion will produce infinite branches of a run-tree by composing infinitely many such trees, and will check that the parity condition is respected by reading from the typing annotations the colors occurring along the branches. The constants are typed according to the transition function of the automaton: the transition (1.2) now corresponds to the annotated typing

$$\texttt{if} \,:\, \emptyset \to \bigwedge_{i\in\{0,1\}} \boxdot_{\Omega(q_i)}\ q_i \to q_0 \,::\, o \to o \to o \tag{1.7}$$

In the colored intersection type system we formulate in §6.3, the Application rule is

$$\frac{\Gamma \vdash t : (\bigwedge_{i=1}^{n}\ \boxdot_{m_i}\ \theta_i) \to \theta \,::\, \kappa \to \kappa' \qquad \Gamma_i \vdash u \,:\, \theta_i \,::\, \kappa \quad \text{for every } i \in \{1, \dots n\}}{\Gamma,\, \boxdot_{m_1}\Gamma_1,\, \dots,\, \boxdot_{m_n}\Gamma_n \ \vdash\ t\,u \,:\, \theta :: \kappa'}$$

where $\boxdot_{m_i}\Gamma_i$ means that every color m appearing in Γ_i is updated to $\max(m, m_i)$. If we think of t and u as trees, the idea is that the maximal color seen from the root of $t[u[\,]]$ to a distinguished hole $[\,]$ of u is the maximum of the color seen from the root of t to the hole u is plugged in, and of the color seen from the root of u to the hole of interest. The point is that this basic intuition on trees lifts to higher-order types. This Application rule would decompose, in the traditional sequent calculus of a modal logic (typically S4 or linear logic), as

$$\dfrac{\Gamma \vdash t : (\bigwedge_{i=1}^{n}\ \boxdot_{m_i}\ \theta_i) \to \theta \qquad \dfrac{\dfrac{\Gamma_1 \vdash u \,:\, \theta_1}{\boxdot_{m_1}\ \Gamma_1 \ \vdash\ u \,:\, \boxdot_{m_1}\ \theta_1} \quad \dots \quad \dfrac{\Gamma_n \vdash u \,:\, \theta_n}{\boxdot_{m_n}\ \Gamma_n \ \vdash\ u \,:\, \boxdot_{m_n}\ \theta_1}\ \text{Right } \boxdot}{\boxdot_{m_1}\Gamma_1, \dots, \boxdot_{m_n}\Gamma_n \ \vdash\ u \,:\, \bigwedge_{i=1}^{n}\ \boxdot_{m_i}\ \theta_i}\ \text{Right } \bigwedge}{\Gamma,\, \boxdot_{m_1}\Gamma_1,\, \dots,\, \boxdot_{m_n}\Gamma_n \ \vdash\ t\,u \,:\, \theta} \tag{1.8}$$

where the rule Right $\boxdot$ is reminiscent of the Promotion rule for the exponential in linear logic. In the variant we give of the Kobayashi-Ong type system, this connection is tight: we observe that $\boxdot$ satisfies the same comonadic principles as the exponential modality [GM15d]. Moreover, the fact that we use the rules Right $\boxdot$ and Right $\bigwedge$ in a row in the decomposition (1.8) suggests to *compose* the exponential modality ! with the coloring modality $\boxdot$.

In the parity game $\mathit{Adamic}(\mathcal{G}, \mathcal{A})$ simulating the fixpoint in the approach of Kobayashi and Ong, an assumption of uniformity is made: suppose that several leaves of a typing proof introduce a same non-terminal F with a same intersection type θ. If Adam plays $F : \theta$, the game $\mathit{Adamic}(\mathcal{G}, \mathcal{A})$ will simulate the unfolding of *all* the occurrences of F typed with θ, and Eve will provide the *same* typing proof for each unfolding. Another interesting aspect of our reformulation is that we relax this uniformity condition, as a first step towards proof theory. Additionally, we internalize the parity fixpoint dynamics of the game $\mathit{Adamic}(\mathcal{G}, \mathcal{A})$ directly in the type system by adding a fixpoint rule which unfolds the rules of the recursion scheme, and by defining a parity condition on typing derivations, lifting to higher-order types this parity condition traditionally formulated over trees. This proof-theoretic reformulation is an important step towards the formulation of an appropriate inductive-coinductive fixpoint operator, based on the parity condition, in the denotational models of linear logic we consider in this thesis.

Colored models of linear logic and higher-order model-checking. This modal reformulation of the Kobayashi-Ong type system provides us with an essential ingredient for extending the models of linear logic previously considered to models capturing the whole higher-order model-checking problem. Semantically, the fact that $\boxdot$ is a modality implies that we can define a corresponding *comonad* $\square$ in the semantics, which is parametrized in the sense of Melliès [Mel06b, Mel14b]. In order to define a fixpoint reflecting the parity condition of alternating parity automata in the semantics, and thus able to iterate coinductively, one conceptual difficulty needs to be addressed in the relational semantics: the traditional interpretation of $!A$ in these semantics is biased towards an inductive (rather than coinductive) interpretation of the fixpoint operator $\mathbf{Y}$. Technically speaking, this comes from the fact that the multisets in $!A$ are *finite*. For that reason, we develop in Chapter 9 an alternative relational semantics of linear logic where the exponential modality noted $A \mapsto \lightning A$ is interpreted as the set $\mathcal{M}_{\leq\omega}(A)$ of *finite-or-countable* multisets of elements of A. In the resulting infinitary relational semantics, a distributive law allows us to compose the exponential modality $\lightning$ with the coloring comonad $\square$. We obtain in this way a colored exponential modality $\bolt$ of linear logic, defined as $\bolt = \lightning \circ \square$. The Kleisli construction applied to this colored exponential provides us with a denotational model of the λ-calculus, which can be extended with an *inductive-coinductive* fixpoint operator, iterating inductively or coinductively depending on the color of the element it considers. We conjecture that this infinitary model allows to recognize MSO properties over infinite trees represented by higher-order recursion schemes.

Taking Ehrhard's extensional collapse theorem as a guideline, we extend the Scott model of linear logic with a coloring comonad and a distributive law

between ! and □. This distributive law enables us to define a colored exponential $\lightning = ! \circ \Box$ and an inductive-coinductive fixpoint lifting the corresponding constructions of our infinitary relational semantics. We then shift to the idempotent intersection type systems of Terui [Ter12] and Ehrhard [Ehr12a] and equip them with coloring and an inductive-coinductive fixpoint operator Y. We also introduce a Scott colored semantics and we prove that it is equivalent in a precise sense with the colored and idempotent intersection type system just obtained. By revising Kobayashi and Ong's original type system (our system differs in some key aspects from their original system, and is not equivalent to it) we are able to reflect the recognition of a higher-order recursion scheme by an alternating parity tree automaton just by looking at the interpretation of the higher-order recursion scheme (seen as an equivalent λ-term with fixpoint Y) in the finitary and colored Scott semantics. The decidability of the local higher-order model-checking problem follows, as the interpretation of a HORS is always computed in finite time. In fact, the finiteness of the semantics has stronger implications: it allows us to prove the decidability of the *selection* problem, which follows from a form of *regularity* of the semantics: any denotation of a λY-term can be obtained from a finitely representable computation; and this finite representation can be formulated in the language of higher-order recursion schemes. The finitary model we obtain is close to the finitary model independently defined by Salvati and Walukiewicz [SW15a], and that they used to prove the decidability of the local higher-order model-checking problem, as well as a *transfer theorem* we detail on the related works (§1.2).

As it appears in this introduction, this thesis uses the unifying power of linear logic and of contemporary semantics to connect

- the theory of intersection types for MSO recognition formulated by Kobayashi and Ong in [KO09],
- the proof-theoretic approach of Bucciarelli and Ehrhard, relating non-idempotent intersection type theory with the relational semantics of linear logic using indexed linear logic [BE00, BE01],
- and the semantic approach of Salvati and Walukiewicz using finite domains for MSO recognition [SW15a].

1.2 Related works

In addition to the articles we cited in this introduction, we give an overview of the different proofs of decidability of the higher-order model-checking problems, and discuss infinitary extensions of linear logic related to the infinitary relational semantics we introduce in Chapter 9.

1.2.1 Decidability of higher-order model-checking problems

Early results. The first result on higher-order model-checking dates back from 2001; it is due to Knapik, Niwinski and Urzyczyn [KNU01], and proves the

decidability of monadic second-order logic over the trees produced by order-2 *safe* recursion schemes. Safety is a restriction on higher-order recursion schemes which appeared first in Damm's work [Dam82], and was rephrased in [KNU01]. It was expressed on λ-terms by Blum and Ong [BO09], who remarked that in this *safe* fragment of the λ-calculus, the usual capture-avoiding substitution can be harmlessly replaced with syntactic substitution.

Knapik, Niwinski and Urzyczyn extended their result in 2002 to the decidability of monadic second-order logic over the trees produced by all *safe* higher-order recursion schemes [KNU02]. Their result proceeds by a translation of safe HORS to higher-order pushdown automata (abbreviated as PDA). Relaxing the safety condition on HORS requires to extend PDA with a collapse operation [Par12].

An alternative approach was developed by Caucal who introduced in [Cau02] two infinite hierarchies, one made of infinite trees and the other made of infinite graphs, defined by means of two simple transformations: unfolding, which goes from graphs to trees, and inverse rational mapping (or MSO-interpretation [CW03]), which goes from trees to graphs. He showed that the tree hierarchy coincides with the trees generated by safe schemes.

A first step towards CPDA was made in 2005, in two independent papers proving the decidability of MSO over trees generated by all order-2 HORS, would they be safe or not. The first one is by Knapik, Niwinski, Urzyczyn and Walukiewicz [KNUW05], the second one by Aehlig, de Miranda and Ong [AdMO05]. These approaches rely on translations of HORS respectively to *panic automata* and to *automata with links*.

In 2006, Aehlig proved that the MSO properties expressible by means of tree automata with trivial acceptance conditions were decidable on the trees produced by all HORS [Aeh06], see also the long version [Aeh07]. His approach relies on the construction of a finite model for evaluating recursion schemes, and was expressed as an intersection type system for verification by Kobayashi [Kob09b], who used this type-theoretic translation to design a practical model-checker.

Local decidability (first proof). The first decidability proof of the local model-checking problem was given in 2006 by Ong [Ong06], see also the long version [Ong], and relies on game semantics. This article uses a result relating *traversals*, which are infinite explorations corresponding to the head normalization of a λ-term $t(\mathcal{G})$ associated with the recursion scheme $\mathcal{G}$ of interest,[1] with the branches of the value tree $\langle\, \mathcal{G} \,\rangle$. This correspondence is proved in [Ong15]. Given an APT $\mathcal{A}$ running over the same signature as $\mathcal{G}$, Ong defines another APT $\mathcal{B}$ running on terms with constants in Σ, and he relates the existence of a winning run-tree of $\mathcal{A}$ over $\langle\, \mathcal{G} \,\rangle$ with the one of a winning run-tree of $\mathcal{B}$ over $t(\mathcal{G})$. Since $t(\mathcal{G})$ is a regular tree, it admits a representation as a finite graph and the existence of a winning run-tree of $\mathcal{B}$ over it can thus be decided, as it amounts to solving a finite parity game. The complexity of the problem,

[1] Actually, it would be more precise to say that $t(\mathcal{G})$ is the infinite λ-term corresponding to a *frozen* form of the β-normal η-long form of $\mathcal{G}$. By *frozen*, we mean that every redex $(\lambda x\,.t)\ u$ is replaced with @ $(\lambda x\,.t)\ u$, where @ is a constant representing applications. As a result, $t(\mathcal{G})$ is an infinite term in β-normal η-long form, which can be analyzed in game semantics using innocent strategies.

namely n-EXPTIME, follows from the size of $\mathcal{B}$ and thus of the parity game.

Local decidability (second proof). The second proof, given in 2008 by Hague, Murawski, Ong and Serre [HMOS08], translates higher-order recursion schemes to *collapsible pushdown automata* (CPDA), a class of automata with higher-order stacks – stacks of stacks of stacks... of symbols – in which the symbols contain pointers to stacks below them and featuring a *collapse* operation, which removes all the stacks above the target of the pointer of the topmost symbol. This collapse operation strictly extends the expressive power of the automaton, as proved by Parys [Par12]. The authors show that CPDA can be used to generate the same trees as HORS; the way the translation uses CPDA is reminiscent of the evaluation mechanism of the Krivine machine. They then translate the local model-checking problem to a parity game over the infinite configuration graph over CPDA. A procedure of order reduction, inspired from the treatment of order 1 by Walukiewicz [Wal01], allows to translate the game over the configuration graph of a CPDA of order n to such a game for a CPDA of order $n-1$ at the price of an exponential blowup of its size. An iterated application of this procedure allows to reduce to order 0, and to obtain in this way a finite graph over which the parity game is decidable. A presentation of [Ong06] and of [HMOS08] can be found in the author's Master thesis [Gre10].

Local decidability (third proof). The third proof, given in 2009 by Kobayashi and Ong [KO09], extends the approach of Kobayashi [Kob09b] in which the model-checking of properties modelled by tree automata with trivial acceptance conditions over infinite trees generated by HORS is solved by means of intersection types. The analysis of these two articles is central in this thesis; the use of intersection types, introduced in [Kob09b], is motivated in the introduction of Chapter 5, and the approach of [KO09] is presented in §6.1 and refined in Chapter 6.

Global decidability (first proof). In 2010, the global higher-order model-checking problem was introduced by Broadbent, Carayol, Ong and Serre in [BCOS10], under the name of *logical reflection*. The authors prove the decidability of the problem by translating HORS to CPDA and by analyzing the structure of the set of winning positions on the infinite parity game generated from the configurations of the automaton. From this analysis follows the decidability of the global HOMC problem. It should be noted that the annotation with $\bullet$ of the symbols of Σ appearing in $\langle\, \mathcal{G} \,\rangle$ at positions where the formula of interest is satisfied is performed on the CPDA, and then translated back to a HORS.

Global decidability (second proof). In [SW11] (see also the long version [SW14]), Salvati and Walukiewicz use Krivine machines to evaluate λY-terms. They design an infinitary parity game over the infinite set of configurations of the machine, which synchronizes the evaluation of the parity automaton

of interest with the head computation of a branch of the normal form of the λY-term of interest. They then collapse this infinite parity game to an equivalent *finite* parity game, leading to a decidability proof of the local and global higher-order model-checking problems.

Selection problem (first proof). The selection problem was introduced, and proved decidable, by Carayol and Serre in 2012 [CS12]. It relies again on a translation between HORS and CPDA, but this translation differs from the one of [HMOS08] and seems more effective, even if the original translation could have been used as well to prove the decidability of the selection problem. As in [BCOS10], the transformation is not performed on the recursion scheme, but on the equivalent automaton.

Selection problem (second proof). In the approach of Carayol and Serre, the annotated HORS obtained by the effective selection procedure can differ a lot from the original one. This led Haddad to study *shape-preserving transformations* of higher-order recursion schemes [Had13a, Had13b]: by an analysis of the type-theoretic result of Kobayashi and Ong, he extracts a transformation *directly on recursion schemes* solving the selection problem.

Transfer theorem (first proof). In [SW13a], Salvati and Walukiewicz use again a parity game on the configurations of a Krivine machine to prove their *transfer theorem*, which states that:

> For every formula φ of monadic second-order logic, there exists an MSO formula $\widehat{\varphi}$ such that, for every infinite term t (thus including λY-terms), φ holds at the root of the normal form of t if and only if $\widehat{\varphi}$ holds at the root of t.

In other terms, an MSO formula φ over Σ-labelled ranked infinite trees can be lifted to an MSO formula $\widehat{\varphi}$ over simply-typed infinite terms (not necessarily representable as finite terms with recursion), whose constants are in Σ. This lifting is such that an infinite term t satisfies $\widehat{\varphi}$ if and only if its infinitary normal form satisfies φ. Salvati and Walukiewicz notably show that this theorem implies global – and thus local – higher-order model-checking.

Selection problem (third proof). Tsukada and Ong gave another proof of the decidability of the selection problem in 2014 in [TO14], see also the long version [TO]. Their approach follows from their quest for a cartesian closed category of games, initiated in [OT12] where they introduce a *two-level* game semantics for interpreting Kobayashi's type system [Kob09b]. The idea is to interpret terms at two levels at the same time, in a fibered way: in a base category playing on the simple type, and in a refined category modelling intersection types. In [TO14], this approach is extended to two-level arenas with effects and coeffects, for interpreting the parity condition or more generally an ω-regular winning condition. This interpretation is connected to a formulation of the Kobayashi-Ong type system which, interestingly, uses effects and subtyping. The main differences with our approach are that they do not use linear logic as a framework for model construction; that their model is based on games, while ours is based on domains; and that they need to consider not only

a comonad, but also an adjunct monad which is not required in our model. On the other hand, their formulation allows to deal with all ω-regular conditions. However, changing the winning condition of the fixpoint in our model would probably allow to consider smoothly such alternative winning conditions.

Selection problem (fourth proof). Grellois and Melliès proved the decidability of the selection problem in 2015 using an interpretation of higher-order recursion schemes in a finitary model of linear logic, extended with a coloring modality and a colored fixpoint operator [GM15a]. This result is the culmination of a series of articles developped independently and in parallel of the approaches of Tsukada and Ong and of Salvati and Walukiewicz. The first connection between linear logic and higher-order model-checking appears in [GM15b], in which the authors show that Kobayashi's intersection type system [Kob09b] can be translated into an equivalent non-idempotent type system, which is in turn connected to the relational semantics of linear logic using indexed linear logic. Motivated by their discovery of the modal nature of the coloring operation of the Kobayashi-Ong type system, the authors introduce an infinitary and colored variant of the usual relational semantics of linear logic, enriched with an inductive-coinductive fixpoint operator [GM15c]. The design of this fixpoint operator is inspired by the proof-theoretic reformulation of the parity game introduced by Kobayashi and Ong in [KO09]. This reformulation and the modal nature of the coloring annotation of the type system are explained in [GM15d]. Finally, Grellois and Melliès adapt the constructions of the infinitary case to the finitary Scott semantics of linear logic, and obtain a finitary model for the recognition of MSO properties over higher-order recursion schemes in [GM15a]. This notably enables them to solve the selection problem. Note that the modal nature of the coloring operation and the colored Scott semantics, enriched with an inductive-coinductive fixpoint operator, were presented by Melliès in 2014 [Mel14a].

Transfer theorem (second proof). Salvati and Walukiewicz gave another proof of their transfer theorem in 2015, restricted this time to the case of λY-terms, in [SW15a]. This article is the result of a series of articles by the authors on the use of finitary models for recognizing λ-terms. The idea of interpreting a term in a model to determine whether it is accepted by an automaton generalizes the traditional recognition by monoid for finite automata over words, and appears already in Salvati's 2009 work [Sal09]. Note that this idea appears independently in Aehlig's work [Aeh06] on finitary models for the recognition of properties expressed by trivial APT over the value trees of higher-order recursion schemes. In [SW13b], Salvati and Walukiewicz use Scott models to interpret λY-terms, recursion being interpreted using the least fixpoint operator, and they show that these models allow to capture tree automata with trivial acceptance condition. This result improves Aehlig's [Aeh06] by considering *insightful* automata, which are able to detect whether a term has a head normal form. Decidability follows from the finiteness of the interpretation of each simple type. They extend their result to weak alternating automata, and thus to weak MSO, in [SW15b]. In this goal, they need to add *ranks* – which we called *colors* in this document – to the model, and to stratify the fixpoint rule in an appropriate way: the rule is either the typical one for inductive, or

coinductive fixpoint operators, depending on the current rank. Their analysis, as ours, is based on a companion intersection type system. The use of inductive and coinductive fixpoint rules differs from our approach, and from Tsukada and Ong's, both relying on a parity condition formulated on infinite plays. Such an alternation of fixpoint rules however appears in Melliès' 2014 model construction [Mel14a], and we conjecture that our model could be equivalently defined using a stratification of fixpoints similar to the one of Melliès and of Salvati and Walukiewicz.

They treat the full case of recognition of properties formulated in monadic second-order logic in [SW15a]. The construction is more elaborated, as it requires to fully model the coloring discipline. An aspect is reminiscent of linear logic: to interpret a simple type, say $o \to o$, they consider the finite domain

$$\mathcal{D}_{o\to o} \quad = \quad \mathcal{R}_o \to \mathcal{D}_o$$

where $\mathcal{D}_o$ is simply $\mathcal{P}(Q)$, and $\mathcal{R}_o$ consists of sets of pairs (q, r) of a state q together with a rank r such that $r \geq \Omega(q)$. Such families of domains $\mathcal{D}$ and $\mathcal{R}$ are defined for all simple types; we could think of the former as a linear interpretation of the simple type, and of the latter as obtained using an "exponential" modality. However, this domain-theoretic construction of Salvati and Walukiewicz is based on the color management of the original type system of Kobayashi and Ong [KO09], and notably does not introduce a neutral color ε for interpreting the Axiom rule, but uses the color of the return state of the type introduced by this rule. As such, the coloring operation is not a parametric comonad, so that the construction lifting from $\mathcal{D}$ to $\mathcal{R}$ is not computed by an exponential modality. In fact, their model is precisely built on the coloring information given by the automaton of interest, while in our model this coloring information could be decorrelated from the states of the automaton.

Another difference between their approach and ours lies in the fact that they need to restrict the morphisms they consider to the monotone functions satisfying a *stratification* property, which is a requirement on the color management during the composition of morphisms. It would be very interesting to understand why this condition is necessary in their work but not in ours.

Our constructions also differ on their behavior on unproductive recursion schemes, whose value trees contain the divergence symbol $\bot$. In our approach, this leads in the semantic run-tree built by $\mathbf{Y}$ to an infinite branch containing only the neutral color ε after a finite prefix of other colors, so that the branch is rejected and the semantic witness tree is as well. By contrast, their model reflects Ω-*even* automata, which accept the divergence symbol if and only if it is labelled with a state of even color.

We should also point out that in their model for MSO recognition, just as in their approach for weak MSO [SW15b], Salvati and Walukiewicz use an alternation of inductive and coinductive fixpoint operators to define their semantic fixpoint $\mathbf{Y}$, and not a parity condition on infinite semantic composition trees as we do. We believe however that our definition would be equivalent to one based on these two inductive and coinductive fixpoint operators.

1.2.2 Infinitary variants of linear logic and of its semantics

The idea developed in Chapter 9 of shifting from the traditional finitary relational semantics of linear logic to infinitary variants is far from new. The closest to our work in this respect is probably the work by Miquel [Miq01] where stable but non-continuous functions between coherence spaces are considered. However, our motivations are different, since we focus here on the case of a modality $\frac{1}{\ell} A$ defined by *finite-or-countable* multisets in A, which is indeed crucial for higher-order model-checking, but is not considered by Miquel.

In another closely related line of work, Carraro, Ehrhard and Salibra [CES10] formulate a general and possibly infinitary construction of the exponential modality $A \mapsto !\, A$ in the relational model of linear logic. However, the authors make the extra finiteness assumption in [CES10] that the support of a possibly infinite multiset in $!\, A$ is necessarily finite. Seen from that prospect, one purpose of our work is precisely to relax this finiteness condition which appears to be too restrictive for our semantic account of higher-order model-checking based on linear logic.

Coming back to linear logic, we would like to mention the works by Baelde [Bae12] and Montelatici [Mon03] who developed infinitary variants (either inductive-coinductive or recursive) of linear logic, with an emphasis on the syntactic rather than semantic side.

1.3 Outline and summary of contributions

This thesis is divided into three parts and nine main chapters.

Part I introduces the core ingredients of higher-order model-checking. Chapter 2 defines trees, two equivalent logics over them, namely monadic second-order logic and the modal μ-calculus; and a companion automata model, alternating parity tree automata, before ending with a quick reminder on parity games. Chapter 3 recalls rudiments of simply-typed λ-calculus, and uses them to define higher-order recursion schemes as well as the simply-typed extension of the λ-calculus with fixpoints. It states the three main problems of higher-order model-checking, and explains why we can restrict our study to productive higher-order recursion schemes. These two first chapters are essentially a pedagogical contribution. Chapter 4 is also pedagogical, but includes as well a small scientific contribution. We introduce informally corecursive structures and coinduction, and we adapt existing frameworks of infinitary rewriting to the case of higher-order recursion schemes. We obtain a coinductive and non-deterministic relation computing the executions of an alternating tree automaton over the value tree of a recursion scheme.

Part II investigates and reformulates the intersection type systems for higher-order model-checking of [Kob09b] and [KO09], revealing in this way a fruitful connection with linear logic and its semantics. In Chapter 5, we study the preliminary problem of the verification of properties expressed by alternating tree automata (without parity) over finite trees generated by simply-typed λ-terms (without recursion). We consider in a first time a non-idempotent variant of the intersection type system of [Kob09b] which solves this problem, and happens to connect naturally to the relational semantics of linear logic. Following

Ehrhard's extensional collapse result [Ehr12b], we explain how this approach can be recasted in a system of idempotent intersection types and in a qualitative semantics of linear logic. In Chapter 6, we investigate the type system of Kobayashi and Ong [KO09] in order to understand how the connection between intersection types and models of linear logic exhibited in the previous chapter could be accommodated with recursion and parity conditions. This leads us to a redefinition of the original type system, disclosing the modal nature of the coloring operation. We briefly explain the main ideas of our reformulation of Kobayashi and Ong's soundness-and-completeness proof, and formulate remarks on potential extensions. We then add some more structural rules to the system as a preliminary to the connections developed in Chapter 10. Our adaptation of the soundness-and-completeness proof of Kobayashi and Ong to our modal reformulation of their type system is split in two steps. Chapter 7 details the completeness proof, after emphasizing on its main concern: defining an optimal strategy for Eve, indexed on the head reduction of the recursion scheme. Chapter 8 contains the proof of soundness of the type system, which notably requires a lemma from [KO]. Using our type-theoretic reformulation of Kobayashi and Ong's parity game, this lemma states that every infinite branch of a run-tree of an alternating parity tree automaton over the value tree of a recursion scheme comes from an infinite branch of the associated typing derivation in our modal type system extended with a parity fixpoint rule.

Part III exploits the results of Chapter 6 to define extensions of the two models considered for finite terms in Chapter 5. In Chapter 9, we introduce an infinitary and non-continuous extension of the traditional relational semantics, in which we define a fixpoint mixing inductive and coinductive behavior. We prove, after adapting the notion to models of linear logic, that it is a Conway operator, meaning that it has the expected « good properties » of a fixpoint operator. We conjecture that the infinitary model we obtain allows the recognition of MSO properties over the infinite trees represented by higher-order recursion schemes. In relation with this model, we introduce a colored and infinitary extension of tensorial logic. In Chapter 10, we take advantage of the conceptual guideline that Ehrhard's extensional collapse result is to reformulate in the finitary Scott semantics of linear logic the constructions of coloring comonad and of inductive-coinductive Conway operator performed in the infinitary relational semantics. We introduce a type system reflecting the computation of denotations in this model, and connect it to the type system for higher-order model-checking we defined in Chapter 6. We then obtain a semantic recognition theorem: the interpretation of a higher-order recursion scheme $\mathcal{G}$ with respect to an alternating parity tree automaton $\mathcal{A}$ is the set of states from which $\mathcal{A}$ accepts the infinite tree $\langle\, \mathcal{G} \,\rangle$ represented by $\mathcal{G}$. The finiteness of the semantics leads to the decidability of the local higher-order model-checking problem, but also to the decidability of the selection problem.

We conclude the thesis by a list of contributions and perspectives.

Part I

The higher-order model-checking problem

Chapter 2

Logic and automata

As explained in the Introduction, the purpose of this thesis is to decide the validity of logical formulas over infinite trees with a finite representation. In this chapter, after defining labeled trees in §2.1, we briefly introduce the two equivalent logics over them which will be under focus in this thesis, namely *monadic second-order logic* (MSO), in §2.2, and *modal μ-calculus*, in §2.4. Prior to the introduction of the latter, which involves fixpoint operators, we recall the necessary elements of order theory in §2.3. We then introduce *alternating parity automata* (APT), which are the automata-theoretic counterpart to both these logics, in §2.5. Finally, we connect these parity automata to *parity games* in §2.6, whose decidability on finite graphs is a key result in model-checking.

Additional references. In addition to the references appearing in this chapter, we find useful to refer to [CDG$^+$07] for additional notions on trees and tree automata theory, to [DP90] for a more elaborate introduction to order theory, to [AN01, Wil01, BW15] for more material on the connection between μ-calculus, automata and parity games, and to [Tho96, GTW02] for a more general connection between MSO, automata and games.

2.1 Trees, graphs

Before we start defining the logics used for the specification of properties in higher-order model-checking, we find useful to recall the definition of an elementary, yet central notion in computer science: trees. They will appear under several different forms in this thesis: as trees labeled with actions, approximating the set of behaviors of a program, as trees generated by the interaction of an automaton with another tree, and as derivation trees, proving either judgments in an appropriate logic, or type inferences. The trees we consider will typically be labeled by a ranked alphabet.

Definition 1 (Ranked alphabet, signature)**.** A *ranked alphabet*, or *signature*, is a finite set Σ together with an *arity* function $\mathrm{ar} : \Sigma \to \mathbb{N}$.
If $\Sigma = \{ a_i \mid i \in I \}$, the signature defined by the arity function ar is often denoted as $\{ a_i : \mathrm{ar}(a_i) \mid i \in I \}$

Definition 2 (Trees)**.**
- A *tree* $t \subseteq \mathbb{N}^*$ is a prefix-closed set of finite words on natural numbers.

- Given a ranked alphabet Σ, a Σ-*labeled tree* is a function $t : Dom(t) \to \Sigma$ whose domain is a tree $Dom(t)$.
- Given a ranked alphabet Σ, a Σ-labeled *ranked* tree is a Σ-labeled tree t such that

$$\forall \alpha \in Dom(t),\ \{i \mid \alpha \cdot i \in Dom(t)\} = \{1, \ldots, ar(t(\alpha))\}$$

  For a node α of a ranked tree t, the node $\alpha \cdot i$ is often referred to as its *child*, or *successor*, in *direction* i. It is denoted $succ_i(\alpha)$.

Σ-labeled trees not satisfying the last condition will often be explicitly called *unranked*. In the definition we give, trees are of *finite degree*: each node has finitely many successors. In the last chapters of this document, this constraint will be relaxed, but until then all trees we consider are implicitly of finite degree. Paths starting from the root are called *branches*:

Definition 3. A maximal *branch* $b = i_0 \cdots i_n \cdots$ of a tree t is a finite or countable sequence of integers whose prefixes $i_0 \cdots i_n$ are all in $Dom(t)$, and which, if finite, ends on a node without successor – if t is ranked, this implies that $\mathrm{ar}(t(i_0 \cdots i_n)) = 0$.

When not explicitly stated otherwise, we implicitly consider branches that are maximal. In this thesis, we will consider trees with finite or countable branches. Since our goal is to *decide* properties over such trees, it will be crucial to have suitable *finite* representations of them. The most canonical representation is perhaps given by *regular trees*, which are obtained by unfolding Σ-labeled graphs:

Definition 4. Given a ranked alphabet Σ, set $n = \max_{a \in \Sigma}(\mathrm{ar}(a))$. A Σ-*labeled graph* is then a tuple $G = \big(V, (E_i)_{i \in \{0,\ldots,n-1\}}, \lambda\big)$, where:

- V is a set of *vertices*
- $\lambda : V \to \Sigma$ is a *labeling* of vertices,
- for every $i \in \{0, \ldots, n-1\}$, $E_i \subseteq V \times V$ is a set such that for every $v \in V$:
  1. for every $i \in \{\, 0, \ldots, \mathrm{ar}(\lambda(v)) - 1 \,\}$, there is a unique $v' \in V$ such that $(v, v') \in E_i$,
  2. for every $i \geq \mathrm{ar}(\lambda(v))$, there is no vertice v' such that $(v, v') \in E_i$.

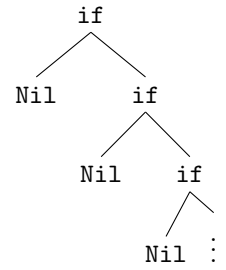
We say that the graph G is *finite* precisely when V is.

In other terms, a Σ-labeled graph is a generalization of Σ-labeled ranked trees, where edges may "backtrack". We will refer to *successors* and *directions* in these graphs in the same way as for trees. We define the *unfolding* of a graph G from a vertice v as the potentially infinite Σ-labeled ranked tree of paths of G starting from v.

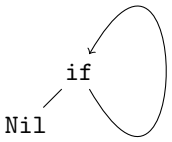
Definition 5 (Regular tree)**.** A Σ-labeled ranked tree is *regular* if it has finitely many subtrees. Equivalently, a Σ-labeled ranked tree is regular if and only if it can be obtained as the unfolding of a finite Σ-labeled graph.

Example 1. Consider the signature $\Sigma_1 = \{\texttt{if} : 2, \texttt{data} : 1, \texttt{Nil} : 0\}$, which can be understood as a set of *actions*. The Σ_1-labeled tree $\mathcal{T}_0$ is ranked and regular, as it can be obtained as the unfolding of the graph $\mathcal{G}_0$ starting from the node labeled with `if`:

$\mathcal{T}_0 =$

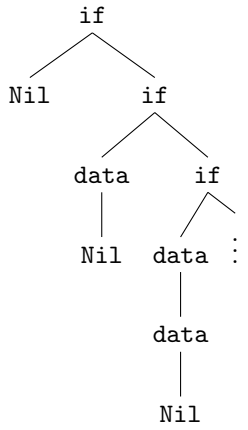


$\mathcal{G}_0 =$



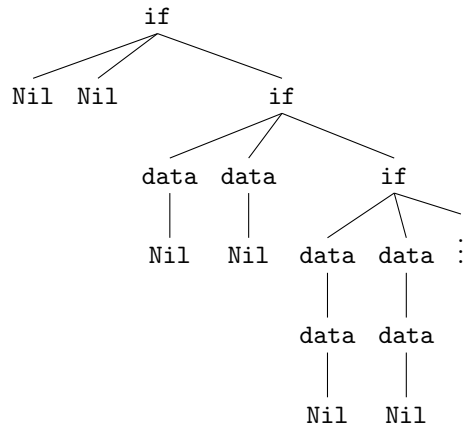
where in $\mathcal{G}_0$ the edge to `Nil` is in direction 0, and the backtracking one is in direction 1. Contrary to $\mathcal{T}_0$, the tree $\mathcal{T}_1$ of the Introduction

$\mathcal{T}_1 =$



is not regular: since its branches, except the rightmost, encode the natural numbers, it has countably many distinct subtrees. A finite representation of this tree can be obtained using *higher-order recursion schemes*, and will be given in §3.2. This Σ_1-labeled tree is ranked, unlike

$\mathcal{T}_{1,dup}$ =



which is obtained by duplicating the leftmost branch on each `if` node — a behavior typical of *alternating tree automata*, see §2.5.

2.2 Monadic second-order logic

As explained in the Introduction, the purpose of a tree of actions such as $\mathcal{T}_1$ is to model the set of potential executions of a program. Following the principles of model-checking, the verification of a property over the program will be performed on this abstract model. Over trees, a convenient and widely used logic is *monadic second order logic* (MSO), since it is a well-balanced choice between expressivity – it contains most other usual logics over trees – and decidability: the satisfiability of a formula is decidable for infinite structures of interest, such as the complete infinite binary tree [Rab69]. However, in higher-order program verification, our goal will not be to determine whether a given formula has a model, that is, whether there exists *some* tree satisfying it, but to determine whether a *given* tree satisfies a given formula. In this section, we provide a short and mostly informal introduction to MSO; more emphasis will be put on modal μ-calculus in §2.4, for reasons which will appear shortly.

MSO is an extension of first-order logic with second-order unary quantification, that is, quantification over sets. Given a ranked alphabet Σ, a unary predicate $\underline{a}$ is introduced for every symbol $a \in \Sigma$; this predicate is true on nodes labeled with a. The choice of a Σ-labeled ranked tree t then induces a family of relations $succ_i$, relating a node to its successor in direction i, if any, as well as a relation $succ$ defined as the union of the former, and a relation $<$ obtained by iterating $succ$ at least once. The MSO formulas over t are then defined by the grammar

$$\begin{array}{rcl} \varphi & ::= & x \mid X \mid x = y \mid x < y \mid succ(x,y) \mid succ_i(x,y) \mid x \in X \mid X \subseteq Y \\ & & \mid \underline{a}(x) \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \neg\varphi \mid \exists x.\, \varphi \mid \forall x.\, \varphi \mid \exists X.\, \varphi \mid \forall X.\, \varphi \end{array}$$

In this setting, first-order variables x correspond to *nodes* of the tree we consider, while second-order variables X represent *sets* of nodes. For instance, $x < y$ is true if and only if the node y belongs to the subtree rooted at x, while $succ_i(x,y)$ holds if and only if y is the successor in direction i of x.

The presentation of MSO we give is redundant, for convenience. The formula $X \subseteq Y$ could be obtained from $\forall x \in X \;\; x \in Y$, while $<$ and $succ$ may be defined from the family of relations $succ_i$.

Example 2. Consider the tree $\mathcal{T}_1$ of Example 1. We define $root \in X$ as $\forall x \;\; (\forall y \;\; x \neq y \Rightarrow \neg\,(\,y < x\,)) \;\Rightarrow x \in X$. We may characterize the existence of an infinite branch labeled only with the symbol `if` using the MSO formula

$$\varphi^{M}_{1,coind} \;=\; \exists X.\,(\,root \in X \;\wedge\; \forall x \in X.\,(\,\underline{\texttt{if}}(x) \;\wedge\; (\,\exists y \in X.\,succ(x,y)\,)\,)\,)$$

which is true if and only if the Σ_1-labeled ranked tree considered contains a set X of nodes all labeled with `if`, and all admitting a successor with the same property.

Monadic second-order logic can be more generally defined on transition systems, as graphs for instance. It notably allows to distinguish $\mathcal{T}_0$ from $\mathcal{G}_0$, using the formula

$$\exists x.\, succ(x,x)$$

which is true on the node labeled with `if` of $\mathcal{G}_0$ (of Example 1), but nowhere in $\mathcal{T}_0$, since by definition a tree does not contain loops. However, these two structures are bisimilar:

Definition 6 (Bisimulation, bisimilarity)**.** Given a signature Σ and two Σ-labeled graphs $G \;=\; \big(V,\,(E_i)_{i\in\{1,\ldots,n\}},\,\lambda\big)$ and $G' \;=\; \big(V',\,(E'_i)_{i\in\{1,\ldots,n\}},\,\lambda'\big)$ a bisimulation is a relation $\equiv$ such that for every $(v,\,v') \in V \times V'$, we have that $v \;\equiv\; v'$ if and only if

- $\lambda(v) \;=\; \lambda'(v')$,
- and for every $i \in \{\,1,\ldots,\,\mathrm{ar}(\lambda(v_1))\,\}$, $succ_i(v) \;\equiv succ_i(v')$.

G and G' are *bisimilar* if and only if there exist a bisimulation $\equiv$ between them, and $(v_0,\,v'_0) \in V \times V'$ such that $v_0 \;\equiv\; v'_0$.

As a consequence, MSO is not closed under bisimulation: a graph and its unfolding, yet bisimilar, will not always satisfy the same formulas. However, since in this thesis we focus on logical formulas over trees, we can restrict ourselves to the bisimulation-invariant fragment of MSO, that is to formulas which do not have different truth values over bisimilar structures. We will consider in the sequel formulas specified in modal μ-calculus, defined in §2.4, which are equivalent in this framework:

Theorem 1 ([Niw88])**.** *Monadic second-order logic and the modal μ-calculus are equi-expressive over trees: for every formula φ of monadic second-order logic, there is an equivalent formula φ_μ of the modal μ-calculus such that φ holds at the root of a tree if and only if φ_μ does.*

Conversely, for every formula φ of the modal μ-calculus, there is an equivalent formula φ_{MSO} of monadic second-order logic such that φ holds at the root of a tree if and only if φ_{MSO} does.

Note that this result, which will be sufficient for our work, can be extended: any MSO formula which does not distinguish bisimilar structures can be translated to an equivalent formula of the modal μ-calculus, and conversely [JW96].

As we will see soon, modal μ-calculus allows to describe finite sets over binary trees of finite degree and of at most countable depth; in this setting, a finiteness predicate is definable in MSO as well [Cou97, Proposition 3.7].

The next two sections are devoted to the introduction of the modal μ-calculus, which we will prefer to MSO in this thesis. Before ending this section, let us mention an interesting fragment of monadic second-order logic, *weak MSO*, obtained from MSO by restricting quantification to finite sets.

2.3 Preliminaries on order theory

Both syntax and semantics of the modal μ-calculus make use of fixpoint operators. We recall in this section elementary facts about these operators, defined over lattices, starting from elementary concepts:

- A *partial order* is a pair $(P, \leq)$, where P is a set and $\leq \ \subseteq \ P \times P$ an antisymmetric, reflexive and transitive relation. We often omit the distinction between P and $(P, \leq)$ when it is clear from context.
- The *dual* P^{op} of $(P, \leq)$ is the pair $(P, \geq)$, where $x \geq y$ iff $y \leq x$.
- The product $\prod_{i \in I} \ (P_i, \leq_i)$ of partial orders is $\left(\prod_{i \in I} \ P_i, \prod_{i \in I} \ \leq_i\right)$, where the product of relations is defined componentwise on tuples of elements.
- In a partial order $(P, \leq)$, the *least element* (resp. *greatest*) of a set $S \subseteq P$ is, when it exists, the element $y \in S$ such that $\forall x \in S \ \ y \leq x$ (resp. $x \leq y$). The *bottom* element $\bot$ of P is its least element; its *top* element $\top$ is dually its greatest element.
- In a partial order $(P, \leq)$, given a set $S \subseteq P$, its *supremum* $\bigvee S \ \in P$ is defined, when it exists, as the element such that $\forall s \in S \ \ s \leq \bigvee S$, and such that $\forall p \in P \ \ (\forall s \in S \ \ s \leq p) \Rightarrow \bigvee S \leq p$. In other words, $\bigvee S$ is the least element of the set of elements of P greater than all elements of S. It does not belong to S in general. The *infimum* $\bigwedge S$ of S is defined dually as the greatest minorant of S.
- In a partial order $(P, \leq)$, a set $S \subseteq P$ is *directed* if and only if, for every $(x, y) \in S$, there exists $z \in S$ such that $x \leq z$ and $y \leq z$. When the partial order represents the partial outputs of a computation, directed sets typically represent different partial computations of a same output. For instance, in a parallel program consisting of two processes π_1 and π_2, the execution of the action a_1 on π_1 could be interpreted in a model based on partial orders as a value x, while the execution of the action a_2 on the processus π_2 would lead, from the same initial configuration, to a value y. The value z can be obtained by executing a_2 on π_2 in the first case, or a_1 on π_1 in the second.
- A *complete partial order*, also called *domain* in the sequel[1], is a partial order in which every directed set has a supremum. Note that it contains

[1] The reader should be warned that there is no unified definition of the notion of domain. Some authors may assume that they obey additional axioms.

a least element, which is the supremum of the empty set. Following the previous intuition, in a domain, a set of "locally unifiable" computations must be "globally unifiable".

- A *complete lattice* is a partial order with a bottom and a top element, and such that every set has an infimum and a supremum[2]. We will typically denote these lattices L, to distinguish them from partial orders. Note that the powerset of a set is always a complete lattice.
- A function between partial orders is *monotone* if it preserves the ordering.
- A fixpoint of an endofunction $f : (L, \leq) \to (L, \leq)$ is an element $x \in L$ such that $f(x) = x$.

The following theorem ensures the existence of fixpoints for monotone endofunctions over complete lattices:

Theorem 2 (Tarski [Tar55])**.** *Let L be a complete lattice and $f : L \to L$ be a monotone function. Then f has fixpoints, and they form a complete lattice $Fix(f)$.*

Being a complete lattice, $Fix(f)$ has a top and a bottom element, which we respectively denote, as later in modal μ-calculus, $\nu X.\, f(X)$ and $\mu X.\, f(X)$. They respectively correspond to the greatest postfixpoint and to the least prefixpoint of the endofunction f:

$$\nu X.\, f(X) \;=\; \bigvee \{\, X \in L \mid X \leq f(X) \,\}$$

$$\mu X.\, f(X) \;=\; \bigwedge \{\, X \in L \mid f(X) \leq X \,\}$$

To understand these two particular fixpoints, it is enlightening to study how they can be effectively computed.

Definition 7 (Initial and final sequences)**.** Given an endofunction $f : L \to L$ on a complete lattice L, the *initial* (resp. *final*) *sequence* of f is an ordinal-indexed sequence $(f^\alpha)_\alpha$ of elements of L, such that

- $f^0 = \bot$ (resp. $f^0 = \top$),
- $f^{\alpha+1} = f(f^\alpha)$,
- For a limit ordinal λ, $f^\lambda = \bigvee_{\alpha<\lambda} f^\alpha$ (resp. $f^\lambda = \bigwedge_{\alpha<\lambda} f^\alpha$).

A *limit* of such a sequence is an element $x \in L$ such that there exists an ordinal ζ satisfying $f^\alpha(x) = x$ for every $\alpha \geq \zeta$. The least such element ζ is called the *closure ordinal* of the sequence; its cardinality can not exceed the one of L.

Before explaining and illustrating this abstract definition with an example, we relate it to the computation of least and greatest fixpoints:

Proposition 1. *Let $\mathcal{D}$ be a* domain *and $f : \mathcal{D} \to \mathcal{D}$ be a monotone function. Then the initial sequence of f exists, and $\mu X.\, f(X)$ is its limit.*

[2]This is in fact enough to imply the existence of $\bot$, which is the supremum of the empty set, and dually of its infimum $\top$.

Proposition 2. *Let L be a* complete lattice *and $f : L \to L$ be a monotone function. Then the initial and final sequences of f exist, and $\mu X.\, f(X)$ is the limit of the initial sequence, while $\nu X.\, f(X)$ is the one of the final sequence.*

Example 3. The most canonical complete lattices are arguably given by powersets[3]. Let us consider the tree $\mathcal{T}_1$ of Example 1, and in particular its set of nodes N. Denote by $L = \mathcal{P}(N)$ the complete lattice of sets of nodes of $\mathcal{T}_1$, and define the endofunction

$$f_1 \ : \ X \mapsto \{\, x \in N \mid x \text{ is labeled with } \texttt{if} \text{ and } x \text{ has a successor in } X \,\}$$

In a complete lattice obtained as a powerset of some set N, we always have $\bot = \emptyset$ and $\top = N$. So, the initial sequence for f_1 starts with $f_1^0 = \emptyset$. The first iteration already stabilizes the sequence: indeed, $f_1^1 = \emptyset$ since no node of the tree has a successor in $\emptyset$, and we therefore have

$$\mu X.\, f_1(X) \quad = \quad \emptyset$$

On the other hand, the computation of the final sequence for f_1 starts with $f_1^0 = N$. Then f_1^1 is the set of all nodes labeled with `if`, since they all have a successor in N. The sequence stabilizes at the next step: indeed, $f_1^2 = f_1^1$ since every element of f_1^1 is labeled with `if`, and has a successor in f_1^1. We therefore have:

$$\nu X.\, f_1(X) \quad = \quad \{\, n \in N \mid n \text{ is labeled with } \texttt{if} \,\}$$

In other terms, this fixpoint computation allows us to detect the infinite branch labeled by `if`, just as the MSO formula $\varphi^M_{1,coind}$ of Example 2 did.

Notice the difference between the two interpretations of f_1: in this setting of trees with branches of at most countable length, we can understand the least fixpoint μ as a *finitary* iteration of φ on the initial element, a process often referred to as *inductive*, while ν can explore infinite branches: by duality with μ, it starts from the set of all elements, which can be infinite, and its iterations do not construct, but *destruct* this set by removing elements which are not stable under the action of the function of interest. This behavior, dual to the inductive one, is called *coinductive*, and is a central concept when infinite objects are considered, see Chapter 4 for a quick introduction. The associated theory of *coinduction* permits indeed a convenient form of reasoning on non-well-founded branches – which, in our framework, will always be countable branches. Recall that a relation is *well-founded* when there is no infinite decreasing chain. For an order relation, this means that there is no infinite sequence

$$\cdots \ < \ x_2 \ < \ x_1 \ < \ x_0$$

or, alternatively, that every such sequence has a minimal element. In the setting of trees, the order relation $x < y$ typically means that x is an iterated successor of y (or equivalently, that $x \neq y$ and x is in the subtree rooted at y), so that

[3] Birkhoff's representation theorem states indeed that finite distributive lattices are equivalent to sublattices of a powerset lattice, see for instance [DP90]. Stone duality theory is devoted to the study of more general – and typically infinite – situations, where it is not enough to consider a powerset construction.

well-foundedness amount to having only finite branches. Over well-founded relations, one can use an induction principle:

$$\forall x \in X \;\; [(\forall y \in X \;\; (y < x \Rightarrow P(y))) \Rightarrow P(x)] \Rightarrow \forall x \in X \;\; P(x)$$

which is a form of strong induction on finite branches. We will use this principle in §8.8.

2.4 Modal μ-calculus

The function f_1 of Example 3 corresponds to a logical formula over $\mathcal{T}_1$, whose informal meaning is:

"the current node is labeled with `if`,
and it has a successor with the same property"

The two different interpretations we computed correspond to an *inductive*, or a *coinductive* interpretation of the property. Modal μ-calculus provides a logical language to express such specifications, using:

- unary *predicates* such as $\underline{\texttt{if}}$, stating that the current node is labeled with `if`,
- the usual *Boolean operations* on formulas,
- *modalities* $\Diamond$ and $\Box$, to express the fact that a formula holds on at least one ($\Diamond$) — or all ($\Box$) — successors of the node of interest,
- and *variables*, together with two *fixpoint operators* μ and ν, the former being inductive and the latter coinductive, in the spirit of §2.3.

For instance, f_1 gives the semantics of the modal μ-calculus formula

$$\texttt{if} \;\wedge\; \Diamond X$$

over the tree $\mathcal{T}_1$, which has one free variable X to which we may apply a fixpoint. Its inductive interpretation $\mu X.f_1(X)$ corresponds to

$$\varphi_{1,ind} \;=\; \mu X.\; (\,\underline{\texttt{if}} \wedge \Diamond X\,)$$

whose semantics is empty – and which is therefore false on every node, while its coinductive interpretation $\nu X.f_1(X)$

$$\varphi_{1,coind} \;=\; \nu X.\; (\,\underline{\texttt{if}} \wedge \Diamond X\,)$$

holds on every node of the rightmost branch of $\mathcal{T}_1$, as computed in Example 3.

Formulas and their semantics. Formally, given a ranked alphabet Σ, modal μ-calculus formulas are defined by the grammar

$$\varphi \;::=\; X \mid \underline{a} \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \Box\varphi \mid \Diamond_i \varphi \mid \Diamond\varphi \mid \mu X.\,\varphi \mid \nu X.\,\varphi$$

for $a \in \Sigma$. As for MSO, we introduce some redundancy in the grammar: $\Diamond\varphi$, which expresses the existence of some successor - in any direction - of the current node satisfying the property φ, can be expressed as $\bigvee_i \Diamond_i$.

This presentation of modal μ-calculus does not introduce a negation operator. In fact, negation is admissible on the fragment of guarded formulas – to be defined in the next paragraph – and a form of de Morgan duality can be given, sending a modality or a fixpoint to its dual:

- $\neg\, \underline{a} \;=\; \bigvee_{b\in\Sigma, b\neq a} \underline{b}$
- $\neg\, (\,\varphi \vee \psi\,) \;=\; \neg\varphi \wedge \neg\psi$
- $\neg \Diamond \varphi \;=\; \Box \neg \varphi$
- $\neg \Diamond_i \varphi \;=\; \Diamond_i \neg \varphi$
- $\neg\, \mu X.\, \varphi(X) \;=\; \nu X.\, \neg\, \varphi(\neg\, X)$

The latter case introduces negated variables in the scope of a fixpoint. The guarded fragment of modal μ-calculus restricts to formulas giving monotone operators on the complete lattices they are interpreted into; in such guarded a formula, every variable is under the scope of an *even* amount of negations. Note that the negation-free grammar we defined forces formulas to be guarded, so that their semantics will be monotone functions, and admit fixpoints thanks to Theorem 2.

Given a Σ-labeled ranked tree $\mathcal{T}$, whose nodes form the set N, the semantics of a formula is the set of nodes where it holds. It is defined using a valuation $\mathcal{V} : Var \to \mathcal{P}(S)$ for free variables:

- $||a||_{\mathcal{V}} \;=\; \{n \in N \mid n \text{ is labeled with } a\}$
- $||X||_{\mathcal{V}} \;=\; \mathcal{V}(X)$
- $||\neg\varphi||_{\mathcal{V}} \;=\; N \setminus ||\varphi||_{\mathcal{V}}$
- $||\varphi \vee \psi||_{\mathcal{V}} \;=\; ||\varphi||_{\mathcal{V}} \cup ||\psi||_{\mathcal{V}}$
- $||\Diamond_i \varphi||_{\mathcal{V}} \;=\; \{n \in N \mid ar(n) \geq i \text{ and } succ_i(n) \in ||\varphi||_{\mathcal{V}}\}$
- $||\mu X.\, \varphi(X)||_{\mathcal{V}} \;=\; \bigcap \{M \subseteq N \mid ||\varphi(X)||_{\mathcal{V}[X\leftarrow M]} \subseteq M\}$

The semantics of dual connectors is defined using de Morgan duality. The least fixpoint operator μ is expressed as a least prefixpoint, as defined in §2.3, of the endofunction f on the complete lattice $L = \mathcal{P}(N)$ obtained from $||\varphi||$ by fixing the valuation $\mathcal{V}$ on all variables but X, which is its parameter. Dually, the semantics of the greatest fixpoint ν is the coinductive interpretation of this function f.

In our setting, where modal μ-calculus formulas are defined over trees with branches of at most countable length, the intuitive meaning of inductive and of coinductive interpretation given in §2.3 can be helpful to understand the meaning of a formula. We can therefore understand the operator μ as a *finite* iteration of the formula using a fixpoint rule

$$\mu X.\, \varphi[X] \;\to_\mu\; \varphi[\mu X.\, \varphi[X]] \tag{2.1}$$

while the coinductive operator ν allows such an unfolding

$$\nu X.\, \varphi[X] \;\to_\nu\; \varphi[\nu X.\, \varphi[X]] \tag{2.2}$$

finitely or infinitely. The inductive operator μ is thus useful to state *safety* properties, and the coinductive operator ν to express *liveness* properties. Of course, several fixpoints may be nested in a formula:

$$\nu X.\ (\ \Box\, X \wedge \mu Y.\ (\ \underline{\texttt{archive}} \vee \Diamond\, Y\ )\ ) \tag{2.3}$$

would express for instance that, from every node of a tree of actions modeling some program – that is, from any instant of its (approximated) execution – there is some execution which runs the instruction `archive` after a finite time.

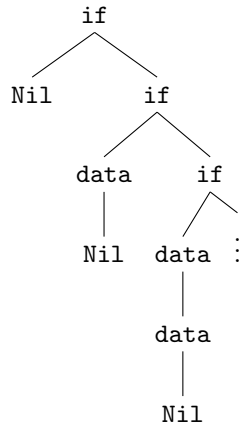
This alternance of inductive and coinductive fixpoints is an appropriate measure of the complexity of a modal μ-calculus formula, see [BW15, Section 3.4] for more details.

Definition 8 (Alternation depth)**.** The *alternation depth* of a modal μ-calculus formula φ is the maximal number of alternations between nested inductive and coinductive fixpoint operators.

For instance, the formula (2.3) has alternation depth 1. When we consider a more general definition of the modal μ-calculus over labeled transition systems [BW15], an interesting restriction of Theorem 1 states that the bisimilar fragment of weak MSO is equivalent to the formulas of modal μ-calculus of alternation depth 0 [Niw88]. The inability of modal μ-calculus to distinguish between bisimilar structures leads to the *tree model property*: if a formula is satisfiable – that is, if there exists a transition system over which it holds – then it is also satisfied by a tree.

Example 4. Recall the tree $\mathcal{T}_1$ of Example 1:

$\mathcal{T}_1 \quad =$



The modal μ-calculus formula

$$\varphi_{1,t} \ =\ \nu X.\ (\ \underline{\texttt{if}} \wedge \Diamond_1\ \ (\ \mu Y.\ (\ \underline{\texttt{Nil}} \vee \Box\, Y\ )\ ) \wedge \Diamond_2\ X\ )$$

holds on every node of the rightmost branch of $\mathcal{T}_1$. It is indeed true on nodes labeled with `if`, from which the symbol `Nil` can be reached on all branches starting in the first direction after finitely many steps, and whose successor in the second direction has the same property – this second part being interpreted

coinductively. An inductive interpretation would have resulted in a formula valid nowhere on the tree, as an infinite set of nodes needs to be considered.

The formula

$$\varphi_{1,alt} \;=\; \nu X.\,(\,\underline{\texttt{if}} \wedge \mu Y.\,(\,\Diamond\, Y \vee \underline{\texttt{data}}\,) \wedge \Diamond\, X\,)$$

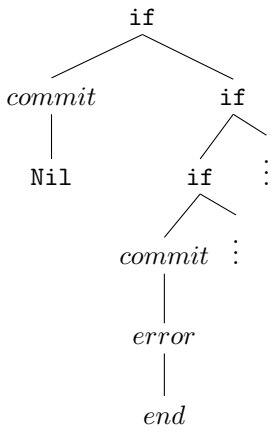
shares a same spirit. It holds on all nodes of the rightmost branch, for the same reasons as before, except for the root. It is true on the root only because the use of the modality $\Diamond$ instead of $\Diamond_1$ allows to look for `data` by exploring the second direction first, and then to explore the left branch labeled by `data`$\cdot$`Nil`. As we will see in §2.5, this necessity to explore *twice* the rightmost subtree of the root will lead an equivalent *alternating* automaton to *duplicate* it, to check in two separate executions that $\varphi_{1,alt}$ and $\mu Y.\,(\,\Diamond\, Y \vee \underline{\texttt{data}}\,)$ hold on the right successor of the root.

We now consider the ranked alphabet

$$\Sigma_2 \;=\; \{\,\texttt{if} : 2,\, \mathit{commit} : 1,\, \mathit{error} : 1,\, \mathit{end} : 0,\, \texttt{Nil} : 0\,\}$$

and the Σ_2-labeled ranked tree

$\mathcal{T}_2 \quad =$



whose finite representation will be given in §3.2. A typical formula over this tree of actions would allow to detect that an execution leads to an `error`:

$$\varphi_{2,t} \;=\; \mu X.\,(\,\Diamond\, X \vee \underline{\texttt{error}}\,)$$

This formula is true at the root of any Σ_2-labeled ranked tree containing an occurrence of the symbol `error`.

Before we translate modal μ-calculus to an equivalent automata model, we find useful to make a few extra remarks. First of all, it is possible to express the semantics of modal μ-calculus formulas as a proof system, in which $\top \vdash \varphi$ if and only if φ is valid, see [BW15, Section 3.5] and [Wal95].

Note also that a form of renaming is allowed on variables of the μ-calculus, just as in the λ-calculus recalled in §3.1. An extra possibility exists however for modal μ-calculus, and is given by Niwinski's "golden rule" [AN01]:

$$\mu X.\,\mu Y.\,\varphi(X,Y) \;=\; \mu Z.\,\varphi(X,\, Y \leftarrow Z)$$

To conclude this section, let us mention that common dynamic or temporal logics such as PDL, CTL and CTL* can be embedded in the modal μ-calculus [BW15, Section 4.2].

2.5 Alternating parity tree automata

The semantics of the modal μ-calculus can be informally understood as a finite or infinite process of unfolding of a formula, using the rules (2.1) and (2.2) inductively or coinductively, in order to verify that this formula holds. This suggests the definition of an associated tree automata model whose execution over a given tree would check whether it satisfies a formula φ of interest. Such an automaton will propagate subformulas of φ along the tree, using its internal state to memorize the subformula it is currently checking. Such an automaton requires:

- a synchronization of the reading of a symbol $a \in \Sigma$ with the propagation of subformulas of φ along branches, which aims at constructing a proof of φ over the tree,
- *non-deterministic choices*, to deal with disjunctive subformulas,
- *alternation*, to deal with conjunctive subformulas: the automaton will duplicate the subtree over which it has to check $\psi_1 \wedge \psi_2$, and check ψ_i on the i^{th} copy,
- and *parity conditions*, which allow to discriminate inductive from coinductive behaviors *a posteriori*, in order to check that the inductive, "finite" unfolding operator μ was not iterated countably.

This leads to the definition of *alternating tree automata* (ATA), and of *alternating parity tree automata* (APT). In order to define their transition functions, we need to introduce *positive Boolean formulas*:

Definition 9 (Positive Boolean formula)**.** The set $B^+(X)$ of positive Boolean formulas θ over a set X is defined by the grammar

$$\theta \ ::= \ \top \mid \bot \mid x \mid \theta \wedge \theta \mid \theta \vee \theta$$

where $x \in X$. We consider these formulas up to the usual equalities, and notably $\varphi \vee \varphi \ = \ \varphi$ and $\varphi \wedge \varphi \ = \ \varphi$.

We say that a set $S \subseteq X$ *satisfies* $\theta \in B^+(X)$ when replacing each element of S occurring in θ by $\top$, and every element of $X \setminus S$ by $\bot$, gives a formula equivalent to $\top$.

Definition 10 (Alternating tree automaton)**.** An *alternating tree automaton* $\mathcal{A} \ = \ \langle \, \Sigma, \, Q, \, \delta, \, q_0 \, \rangle$, also referred to as *ATA*, is given by

- a ranked alphabet Σ,
- a finite set of control states Q,
- a transition function $\delta \, : \, Q \times \Sigma \to B^+(\mathbb{N} \times Q)$ such that

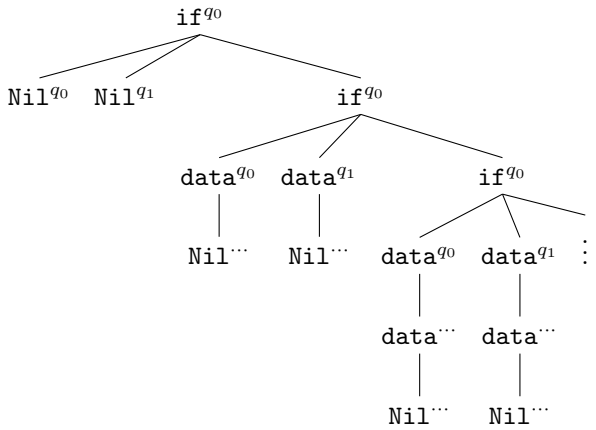
$$\forall (q, \, a) \in Q \times \Sigma \qquad \delta(q, \, a) \in B^+(\{1, \ldots, ar(a)\} \times Q)$$

- and an initial state $q_0 \in Q$.

A typical transition of an ATA over the ranked alphabet Σ_1 introduced in Example 1 is

$$\delta(q_0, \texttt{if}) \;=\; (1, q_0) \wedge (1, q_1) \wedge (2, q_0)$$

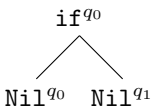
Its effect is to *duplicate* the subtree in direction 1, and to execute on one copy with state q_0, and with q_1 on the other. It executes only on a copy of the second subtree, without changing the state. Over the tree $\mathcal{T}_1$, this duplication procedure produces an infinite unranked tree $\mathcal{R}_1$, called a *run-tree of* $\mathcal{A}$ *over* $\mathcal{T}_1$:

$\mathcal{R}_1 \;=$  (2.4)

Note that transitions may also drop subtrees, as for instance

$$\delta(q_0, \texttt{if}) \;=\; (1, q_0) \wedge (1, q_1)$$

whose execution over the *infinite* tree $\mathcal{T}_1$ results in a *finite* run-tree



More generally, a transition function may stop the execution of the automaton by dropping all subtrees of a node, and trivially accept:

$$\delta(q_0, \texttt{if}) \;=\; \top$$

or reject:

$$\delta(q_0, \texttt{if}) \;=\; \bot$$

Since boolean formulas can be put in disjunctive normal form, a typical transition of an ATA is

$$\delta(q, a) \;=\; \bigvee_{i \in I} \bigwedge_{j \in J_i} (d_{i,j}, q_{i,j}) \;=\; \bigvee_{i \in I} \varphi_i \tag{2.5}$$

For every $i \in I$, we say that φ_i is a *conjunctive clause* of the transition $\delta(q, a)$. A transition can thus be understood as the *non-deterministic* choice of a conjunctive clause φ_i, followed by a *universal choice* (alternation): for every $j \in J_i$, a copy of the subtree in direction $d_{i,j}$ is produced, over which the automaton runs with state $q_{i,j}$.

When, for every $i \in I$ and every direction d, there is a unique j such that $d_{i,j} = d$, we recover the usual notion of *non-deterministic* tree automaton.

We can now define run-trees formally:

Definition 11 (Run-tree)**.** Given a Σ-labeled tree t and an alternating tree automaton $\mathcal{A} = \langle \Sigma, Q, \delta, q_0 \rangle$, a *run-tree*, or *execution tree* of $\mathcal{A}$ over t is a $(\mathbb{N} \times Q)$-labeled unranked tree r such that:

- $\varepsilon \in dom(r)$ and $r(\varepsilon) = (\varepsilon, q_0)$
- For every $\beta \in dom(r)$, denoting $r(\beta) = (\alpha, q)$, there exists $S \subset \mathbb{N} \times Q$ satisfying $\delta(q, t(\alpha))$ and such that

$$\forall (i, q') \in S,\ \exists j \in \mathbb{N},\ (\beta j \in dom(r)) \wedge (r(\beta j) = (\alpha i, q'))$$

Note that the data of the tree t over which the ATA is executed allows to compute the label of each node of a run-tree, so that we will use in the sequel representations of run-trees as in (2.4), with an explicit labeling of nodes.

From modal μ-calculus to tree automata theory. Consider again the modal μ-calculus formula

$$\varphi_{1,t} = \nu X.\ (\ \underline{\texttt{if}} \wedge \Diamond_1\ (\ \mu Y.\ (\ \underline{\texttt{Nil}} \vee \Box Y\ )\ ) \wedge \Diamond_2\ X\ )$$

To check this formula by unfolding, one has to

- check that the current node is labeled by `if`,
- check that its successor in direction 1 satisfies $\mu Y.\ (\ \underline{\texttt{Nil}} \vee \Box Y\ )$,
- check that its successor in direction 2 satisfies $\varphi_{1,t}$.

We can design an automaton which will unfold these two formulas along the branches of the tree. We set $Q = \{q_0, q_1\}$. The state q_0 corresponds to checking that $\varphi_{1,t}$ holds on the current node, while q_1 corresponds to check for $\mu Y.\ (\ \texttt{Nil} \vee \Box Y\ )$. The transition function can then be defined as:

- $\delta(q_0, \texttt{Nil}) = \bot$
- $\delta(q_0, \texttt{data}) = \bot$
- $\delta(q_0, \texttt{if}) = (1, q_1) \wedge (2, q_0)$
- $\delta(q_1, \texttt{Nil}) = \top$
- $\delta(q_1, \texttt{data}) = (1, q_1)$
- $\delta(q_1, \texttt{if}) = (1, q_1) \wedge (2, q_1)$

The first two cases correspond to checking $\varphi_{1,t}$ on the current node, which implies that it is labeled with `if`: if another symbol is found, the automaton stops its execution, and rejects. The third case corresponds to a labeling of the current node with `if` while checking $\varphi_{1,t}$: this formula needs to be checked on the successor in direction 2, while $\mu Y.\ (\ \underline{\texttt{Nil}} \vee \Box Y\ )$ has to be checked in the other direction.

When the current state is q_1, the automaton checks that $\mu Y.\ (\ \underline{\texttt{Nil}} \vee \Box Y\ )$ holds. If the node is labeled with `Nil`, it accepts. If it reads `data` or `if`, it keeps looking for `Nil` in all available directions. Note that this automaton is non-deterministic, but the similar formula $\varphi_{1,alt}$ of Example 4 would require

alternation: indeed, $\varphi_{1,alt}$ and $\mu Y.\ (\ \underline{\texttt{Nil}} \vee \Box Y\ )$ could both hold in the *same* direction.

However, the automaton we designed to check $\varphi_{1,t}$ is incomplete. Suppose that there is an infinite branch whose nodes are all labeled by `data`; the automaton will have an execution tree, since it can always go further in state q_1 along this branch. But this breaks the inductive nature of the formula q_1 is supposed to check. The solution is to add a *color* to each state, that is, an integer which will keep track of an additional information indicating whether the automaton currently unfolds an inductive, or a coinductive fixpoint. We associate the color 0 to q_0, and 1 to q_1. Now, if the automaton reads the infinite branch whose nodes are all labeled by `data`, the maximal color it sees infinitely often is 1 – we call it the color of the branch. By rejecting infinite executions in which an infinite branch has color 1, we exclude "unsound" run-trees, and capture automata-theoretically the inductive-coinductive nature of modal μ-calculus.

In general, this construction leads to the definition of alternating *parity* automata (APT):

Definition 12 (Alternating parity tree automaton)**.** An alternating *parity* tree automaton $\mathcal{A} = \langle\, \Sigma, Q, \delta, q_0, \Omega \,\rangle$, also referred to as *APT*, is given by an ATA $\langle\, \Sigma, Q, \delta, q_0 \,\rangle$, together with a *coloring function* $\Omega : Q \to \mathbb{N}$, mapping each state to its *color*.

We can then generalize the idea we previously sketched:

Definition 13 (Accepting run-tree)**.** Consider an infinite branch $b = i_0 \cdots i_n \cdots$ of a run-tree r of an APT $\mathcal{A} = \langle\, \Sigma, Q, \delta, q_0, \Omega \,\rangle$ over t. Denoting π_2 the projection giving the state labeling a run-tree, we set $m_n = \Omega(\pi_2(r(i_0 \cdots i_n)))$, and call *color of the branch* b the greatest color occurring infinitely often in the sequence $(m_n)_{n\in\mathbb{N}}$. We say that the branch b is *winning*, or *accepting*, if its color is even. A run-tree is *winning*, or *accepting*, when all its infinite branches are.

Given an APT $\mathcal{A} = \langle\, \Sigma, Q, \delta, q_0, \Omega \,\rangle$, we sometimes find convenient in the sequel to say that $\mathcal{A}$ accepts some Σ-labeled ranked tree t *from state* q when the APT $\langle\, \Sigma, Q, \delta, q, \Omega \,\rangle$ obtained from $\mathcal{A}$ by modifying its initial state has an accepting execution tree over t.

The following theorem relates modal μ-calculus (and thus MSO, due to Theorem 1), to alternating parity tree automata:

Theorem 3 ([JW95])**.** *Consider a ranked alphabet* Σ*.*

- *Given a modal* μ*-calculus (resp. MSO) formula* φ*, there exists an alternating parity automaton* $\mathcal{A}_\varphi$ *such that, for every* Σ*-labeled ranked tree* t*,* φ *holds at the root of* t *if and only if* $\mathcal{A}_\varphi$ *has an accepting execution tree over* t*.*

- *Given an alternating parity automaton* $\mathcal{A}$*, there is a modal* μ*-calculus (resp. MSO) formula* $\varphi_\mathcal{A}$ *such that, for every* Σ*-labeled ranked tree* t*,* $\mathcal{A}$ *has an accepting execution tree over* t *if and only if* $\varphi_\mathcal{A}$ *holds at the root of* t*.*

More details on the underlying constructions can be found for instance in [Wil01]. To conclude this section, let us recall that since the tree automata we consider are "top-down", that is, since their execution starts on the root of the tree they run on, they cannot be determinized, see for instance [CDG+07, Proposition 1.6.2]. However, one can *non-determinize* alternating (parity) tree automata:

Theorem 4 (Non-determinization of APT [MS95])**.** *Given an alternating (parity) tree automaton $\mathcal{A}$, one can compute a non-deterministic (parity) tree automaton $\mathcal{A}_{ND}$ recognizing the same language.*

We will nevertheless focus on *alternating* tree automata: as explained in the Introduction and in Chapter 5, the alternating behavior of ATA is strongly similar to the one of the exponential modality of linear logic.

2.6 Parity games

As we explained in the previous section, a transition (2.5) of an alternating tree automaton $\mathcal{A}$ running over a tree t can be understood as a non-deterministic choice, followed by a universal choice. This view leads to an interactive, game-theoretic understanding of ATA, in which a player tries to construct locally a run-tree r by playing non-determinism, while the other one, selecting one of the copies created by alternation, guides this construction of r along a given branch. The goal of the latter player is to find a branch along which no run-tree could be built, leading to the impossibility of the "global" existence of r.

In this section, we formalize this game-theoretic intuition, and incorporate the parity condition of APT, leading to the notion of *parity game*, which is essentially a bipartite graph with a coloring of nodes.

Definition 14 (Parity game)**.** A parity game $P_G = \langle (V_E \uplus V_A, E), v_0, \Omega \rangle$ is given by:

- a directed graph $G = (V = V_E \uplus V_A, E)$,
- an initial vertex $v_0 \in V$,
- and a *coloring function* $\Omega : V \to \mathbb{N}$.

The partition of the graph divides it in nodes controlled by the player Adam, which belong to the set V_A, and nodes controlled by Eve, forming the set V_E. The interaction between players is given by the notion of *play*:

Definition 15 (Play)**.** A *play* of a parity game $P_G = \langle (V_E \uplus V_A, E), v_0, \Omega \rangle$ is a sequence $\pi = v_0 \cdot v_1 \cdots$ such that, for every index i, $(v_i, v_{i+1}) \in E$, and which starts on the initial node v_0.

A play is *maximal* if it is finite and ends on a vertice without outgoing edge, or if it is infinite. Infinite plays receive a color, which is the greatest color occurring infinitely often in $(\Omega(v_i))_{i \in \mathbb{N}}$, similarly to the branches of run-trees of APT.

Following the interactive intuition of APT, we can define a winning condition over plays:

Definition 16 (Winner of a play)**.** A maximal play $\pi = v_0 \cdots v_1 \cdots$ is winning for Eve if it is finite and ends with a node controlled by Adam, or if it is infinite and of even color. Otherwise π is winning for Adam.

Since Adam's rôle is to explore branches, Eve can build a run-tree precisely when she can build one on any branch – that is, when she can win any play where she constructs this run-tree. This construction is modeled by a *strategy* for Eve, that is, by a function indicating her answer to any of Adam's moves; this strategy is winning for Eve when she wins every play following it.

Definition 17 (Strategy)**.**

- A strategy for Eve is a partial map σ from the set of plays ending in V_E to V, and such that for every play π ending in V_E the play $\pi \cdot \sigma(\pi)$ is a play of P_G. When σ is a total map, we say that σ is a *total* strategy.
- A strategy for Eve is *memoryless*, or *history-free*, when it can be represented as a function $\sigma : V_E \to V$, that is, when Eve moves according only to the current node of the graph, independently of the history of the play.
- We say that Eve *follows* σ in the play π if, for every prefix π' of π ending in V_E, the play $\pi' \cdot \sigma(\pi')$ is a prefix of π.
- If every maximal play in which Eve follows σ is winning for her, we say that σ is a *winning strategy*. We define dual notions of (winning (memoryless)) strategy for Adam.
- Given strategies σ_E for Eve and σ_A for Adam, we define their interaction $\langle \sigma_E | \sigma_A \rangle$ as the maximal play starting from v_0 where each player follows his strategy.

The existence of a winning run-tree of an APT $\mathcal{A}$ over a tree t therefore reduces to the existence of a winning strategy for Eve in a parity game. Such games are determined [Mar75], that is, one of the players always have a winning strategy. We can moreover decide which player has a winning strategy from a given node, and algorithms allow to construct memoryless such strategies, see for instance Zielonka [Zie98] or Schewe [Sch07].

Theorem 5. *Given a* finite *parity game, one can effectively compute the winning player, and a memoryless winning strategy for him.*

The problem is known to be in NP $\cap$ coNP. For a finite or regular tree t, we can design a parity game modeling the interactive behavior of an APT $\mathcal{A}_\varphi$ over it, and deduce from this theorem that checking φ over t is decidable.

However, how could we adapt this approach to deal with non-regular trees as $\mathcal{T}_1$? The purpose of this thesis is

- to extend the notion of regularity to *higher-order regularity*, introduced in the next chapter,
- and to use semantic tools to design an appropriate *finite* parity game corresponding to the model-checking of the higher-order regular tree. Part II uses type-theoretic methods in this goal, which we connect to denotational models of linear logic in Part III.

Before we introduce higher-order regularity in the next chapter, let us recall the existence of another duality, whose conceptual interest will arise with this extended notion of regularity: a run of a tree automaton can be understood as a *collaborative* interaction between two players:

- the tree, which is explored during the interaction, and exchanges informations about its labeling,
- and the automaton, which modifies the current state of the flow of interaction, and potentially performs duplications of the tree.

This point of view, deeply related to *game semantics*, will particularly appear in the theorems of §9.2, as well as in their extension as Conjecture 1; and will strongly underly our model-theoretic approaches.

Chapter 3

An abstract model for recursive functional programs

In model-checking, programs are traditionally *abstracted* into more elementary models which, being less expressive, allow to *decide* properties, formulated for instance in monadic second-order logic. To handle functional programs, coded for instance in C++, Haskell, OCaML, Javascript, Python, or Scala, we need models which can take into account *higher-order*, *recursive* function calls. In this chapter, we introduce higher-order recursion schemes, which are a convenient such model.

We start by recalling in §3.1 essential notions about the λ-calculus and its simply-typed fragment. Extending it with a form of recursion leads to higher-order recursion schemes (HORS) in §3.2 or, equivalently, to the λY-calculus, see §3.3. HORS and λY-terms allow to generate trees as $\mathcal{T}_1$, modeling the set of actions of a functional program of interest; we formulate in §3.4 the model-checking problems we will focus on in this thesis. In §3.5, we explain how the abstraction of Turing-complete functional programs into the less expressive formalism that are higher-order recursion schemes allows notably to decide whether they are productive – a notion which may be understood as an infinitary version of the halting problem.

3.1 Simply-typed λ-calculus

Introduced by Church in 1932 [Chu32], the λ-calculus originally intended at replacing set theory with a new foundation for mathematics, based on a calculus of functions and applications. While the resulting system ended up (at least at the time) not being powerful enough to offer an alternative foundation for mathematics, it appeared that it has the same expressive power as a Turing machine, and that it thus provides a simple and convenient universal model of programming languages. In this section, we briefly recall the main ingredients of the λ-calculus, and emphasize on the simply-typed fragment, on which we will focus in the sequel of this thesis – extended with recursion.

λ-terms. Given a signature Σ providing a set of constants, and a set of variables $\mathcal{V}$, λ-terms are defined by the grammar

$$t, u \quad ::= \quad x \mid a \mid \lambda x.\, t \mid t\ u$$

$$\frac{}{(\,\lambda x.\,t\,)\;u \;\;\to_\beta\;\; t[x \leftarrow u]} \qquad \frac{t \;\to_\beta\; u}{\lambda x.\,t \;\to_\beta\; \lambda x.\,u}$$

$$\frac{t \;\to_\beta\; t'}{t\;u \;\to_\beta\; t'\;u} \qquad \frac{u \;\to_\beta\; u'}{t\;u \;\to_\beta\; t\;u'}$$

Figure 3.1: β-reduction of infinite λ-terms.

where $x \in \mathcal{V}$ is a variable and $a \in \Sigma$ a constant. A term $t\ u$ is called the *application* of the term t to the term u, and should be thought of as a function application. The construction $\lambda x.\,t$ *binds* the variable x in the term t, and is to be thought of as a syntactic representation of the function

$$\lambda x.\,t \quad : \quad x \mapsto t[x]$$

A term may have free variables, that is, variables x, $y, \ldots$ which are not bound by a λ. For instance, the term

$$\lambda x.\,\lambda y.\,a\ x\ y$$

where $a \in \Sigma$ is *closed*: all its variables are bound, while the term

$$\lambda x.\,\lambda y.\,a\ x\ z$$

is *open*: it has one free (i.e. not bound) variable, namely z. When a variable is bound, one can uniformly rename every of its occurrences; some attention has nevertheless to be payed, as two different variables may appear with the same name in a term – we will not discuss this problem, and assume that it never occurs in the sequel, as *α-conversion* ensures that there always is a representation of the term where each name corresponds to a unique variable, see [Bar84] for details.

The λ-calculus induces a rewriting system, based on the notion of *redex*. A redex is a term

$$(\,\lambda x.\,t\,)\;u$$

consisting of the application of a term representing the function $x \mapsto t[x]$ to the argument u. Just as in mathematics, where

$$(\,x \mapsto f(x)\,)(5)$$

would rewrite to $f(5)$, the *β-reduction* of this redex is defined as

$$(\,\lambda x.\,t\,)\;u \quad \to_\beta \quad t[x \leftarrow u]$$

where $t[x \leftarrow u]$ is obtained from t by replacing each occurrence[1] of the variable x with the term u. This relation may reduce any redex occurring as a subterm of the term of interest, see Figure 3.1 for its formal definition.

[1] Without the assumption we make in this thesis that the term has been α-converted to a term in which every variable name is the one of a unique variable, more subtlety is required, see [Bar84].

A term is in *normal form* when it does not contain any redex. For instance,

$$(\lambda x.\, \lambda y.\, a\ x\ y)\ b\ c \quad \to_\beta \quad (\lambda y.\, a\ b\ y)\ c \quad \to_\beta \quad a\ b\ c$$

and $a\ b\ c$ is the normal form of the term $(\lambda x.\, \lambda y.\, a\ x\ y)\ b\ c$. A key fact is that the computation may duplicate or drop its inputs, just as an alternating automaton would, a behavior we will analyze precisely in the sequel of this thesis using *linear logic*. Denoting $\to_\beta^3$ the application of the rewriting rule $\to_\beta$ three times, the term

$$(\lambda x.\, \lambda y.\, \lambda z.\, a\ (a\ x\ x)\ (a\ y\ x))\ b\ c\ d \quad \to_\beta^3 \quad a\ (a\ b\ b)\ (a\ c\ b))$$

makes three copies of its first argument b, one of its second argument c, and drops its third argument d. More generally, we denote $\to_\beta^*$ the finitely iterated application of the rule $\to_\beta$ — note that this includes the identity relation, as $\to_\beta$ could be iterated zero times.

Before we discuss the concurrent behavior of the reduction rule $\to_\beta$, let us quickly introduce the relation $\to_\eta$ of *η-reduction*:

$$\lambda x\, .t\ x \quad \to_\eta \quad t \quad \text{where } x \text{ is not free in } t$$

The reverse relation is called *η-expansion*. These relations are independent of β-reduction: in general, we do not have $\lambda x.\, t\ x \quad \to_\beta \quad t$. We will use these relations mainly in Chapter 10.

Reductions and strategies. Terms may contain many redexes, either in parallel:

$$a\ (\,(\,\lambda x_1.\, t_1\,)\ u_1)\ (\,(\,\lambda x_2.\, t_2\,)\ u_2)$$

or in a nested way:

$$(\,\lambda x.\ (\,\lambda y.\, x\ y\,)\ x\,)\ a$$

In both cases, the two possible reductions lead to the same result, as depicted in Figures 3.2 and 3.3. It is not always the case: let us consider

$$\Omega \quad = \quad (\,\lambda x.\, x\ x\,)\ (\,\lambda x.\, x\ x\,)$$

which reduces to itself: $\Omega \to_\beta \Omega \to_\beta \cdots$. Now if we apply it to a term which does not use its argument, obtaining

$$t_w \quad = \quad (\,\lambda x.\, a\,)\ (\,\lambda x.\, x\ x\,)\ (\,\lambda x.\, x\ x\,)$$

we may either

- reduce the *head* redex first, that is, the leftmost one:

$$(\,\lambda x.\, a\,)\ (\,\lambda x.\, x\ x\,)\ (\,\lambda x.\, x\ x\,) \quad \to_\beta \quad a$$

- or reduce the other one, and obtain the same term t_w again.

The behavior of these terms t_w and Ω illustrate several key notions of rewriting theory:

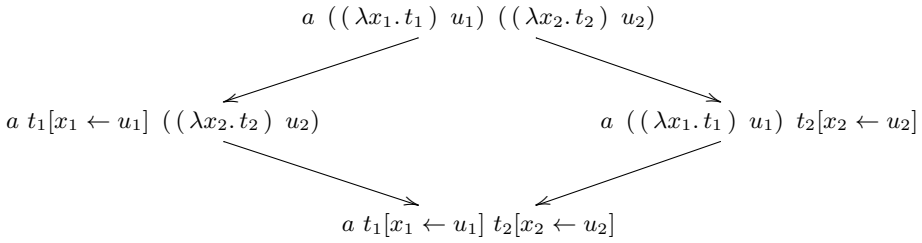


Figure 3.2: Confluence of two parallel reductions.

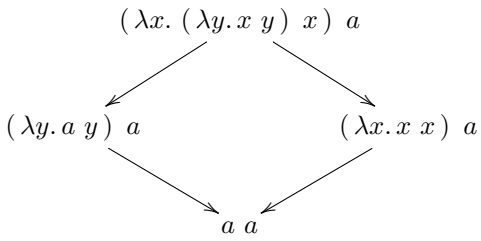
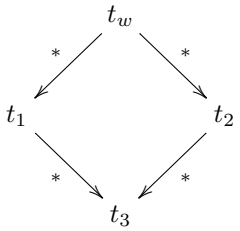


Figure 3.3: Confluence of two nested reductions.

- *confluence*: from any two reductions $t_w \rightarrow_\beta^* t_1$ and $t_w \rightarrow_\beta^* t_2$, there is a term t_3 such that

  In this easy case, t_3 can always be taken to be a, but in some cases it can be t_w as well. In general, the whole λ-calculus is confluent, see [Bar84]: replacing t_w with any term t, the same property holds.

- *Weak normalization*: a term is weakly normalizing when there exists a finite reduction sequence computing its normal form – which, when it exists, is unique by confluence. The term t_w is weakly normalizing, its normal form being a, while Ω is not, and consequently does not have a normal form.

- *Strong normalization*: a term is strongly normalizing when every maximal reduction sequence is finite and computes its normal form. Neither t_w nor Ω have this property. However, the *simply-typed* λ-terms we will soon introduce are all strongly normalizing.

Since not all reduction sequences give the same result, it makes sense to study *reduction strategies*, which indicate in which sequential order redexes

should be evaluated. Call *innermost* a redex which does not contain any other redex, and *outermost* a redex which is not contained in any other. Among all the possibilities[2], we distinguish two main classes of reduction strategies:

- *outermost* strategies, which evaluate first the outermost redexes, and correspond to *call-by-name* evaluation: when a functional application is encountered, the core of the function is evaluated, and its arguments are accessed and reduced every time they appear in a computation. It is a very costly evaluation, since it recomputes its argument every time it accesses it, but it computes the normal form of t_w,
- and *innermost* strategies, which evaluate first the innermost redexes, and can be understood as *call-by-value* evaluations: on a functional evaluation, they compute first the value of the argument, which is somehow memorized by the program, so as to avoid multiple evaluations later on when the body of the function is evaluated. However, the application of an innermost strategy to t_w leads to an infinite, unproductive computation: the argument Ω, although useless for the evaluation of t_w, is computed and leads to a divergent computation.

Notice that the main difference between outermost (resp. innermost) strategies consists in the order in which they compute redexes of the same priority – that is, in how they *sequentialize* a canonical, parallel reduction. In general, when a term has a normal form, it can be computed using the *standard reduction*, also called *head reduction* $\to_h$, and defined using the deduction rules

$$\frac{}{(\lambda x.\, s)\ t \quad \to_w \quad s[x \leftarrow t]} \qquad \frac{s \quad \to_w \quad s'}{s\ t \quad \to_w \quad s'\ t}$$

$$\frac{s \quad \to_w \quad s'}{s \quad \to_h \quad s'} \qquad \frac{s \quad \to_h \quad s'}{\lambda x.\, s \quad \to_h \quad \lambda x.\, s'}$$

where $\to_w$ defines *weak head reduction*, which can not reduce under λ, and $\to_h$ denotes head reduction.

Theorem 6 (Standardization of the λ-calculus [CCF58])**.** *If t is weakly normalizing, then its normal form is computed (in finite time) by the head reduction strategy.*

More generally, a denotational semantics of the λ-calculus can be defined using Böhm trees, which come from Böhm's work on separability [Böh68], and were defined by Barendregt, see [Bar84, Chapter 10]:

Definition 18 (Böhm tree)**.** Given a λ-term t defined over the set $\mathcal{V}$ of variables, and the set Σ of constants, define its *Böhm tree* as the labeled, unranked tree $BT(t)$ defined inductively as

- $BT(t) \quad = \quad \Omega$ if t has no head normal form,
- if $x \in \mathcal{V}$, $BT(x) \quad = \quad x$ is the tree consisting of the single node x,

[2] In practice, more involved strategies can be used, as *call-by-need* [Wad71], *call-by-push-value* [Lev06], probabilistic strategies...

- if $a \in \Sigma$, $BT(a) \quad = \quad a$ is the tree consisting of the single node a,
- $BT(\lambda x.\, t) \quad = \quad \lambda x.\, BT(t)$,
- $BT(t_0\ t_1) \quad = \quad BT(t_0)\ BT(t_1)$.

The Böhm tree of a weakly normalizing term is its normal form itself, due to Theorem 6. More interesting phenomena arise for terms which do not normalize. While we trivially have that the Böhm tree of the λ-term Ω is the divergence symbol Ω, more interesting things arise if we consider a term as Church's fixpoint combinator

$$Y_{Church} \quad = \quad \lambda f.\ (\ \lambda x.\ f\ \ (\ x\ x\ )\ )\ \ (\ \lambda x.\ f\ \ (\ x\ x\ )\ )$$

applied to a symbol $a \in \Sigma$: we get

$$\begin{array}{lcl} BT(Y_{Church}\ a) & = & BT\,((\ \lambda x.\ a\ \ (\ x\ x\ )\ )\ \ (\ \lambda x.\ a\ \ (\ x\ x\ )\ )) \\ & = & BT\,(a\ \ (\ (\ \lambda x.\ a\ \ (\ x\ x\ )\ )\ \ (\ \lambda x.\ a\ \ (\ x\ x\ )\ )\ )) \\ & = & BT\,(a\ a\ \ (\ (\ \lambda x.\ a\ \ (\ x\ x\ )\ )\ \ (\ \lambda x.\ a\ \ (\ x\ x\ )\ )\ )) \\ & = & \cdots \end{array}$$

so that

$$BT(Y_{Church}\ a) \quad = \quad \begin{array}{c} a \\ | \\ a \\ | \\ a \\ | \\ a \\ | \\ \vdots \end{array}$$

For any term t, the term $Y_{Church}\ t$ and the term $t\ (\ Y_{Church}\ t\ )$ have the same normal form, if any; however, to get a recursive behavior, we need a combinator Y such that $Y\ t\ \ \rightarrow_\beta^*\ t\ (\ Y\ t\ )$. Such a combinator exists: one can for instance take the Turing combinator [Tur37]

$$Y_{Turing} \quad = \quad (\ \lambda f.\, \lambda g.\, g\ \ (\ f\ f\ g\ )\ )\ \ (\ \lambda f.\, \lambda g.\, g\ \ (\ f\ f\ g\ )\ )$$

The substitutions of Y_{Turing} simulating the unfolding of a fixpoint may create additional redexes; as an exercise, the brave reader may for instance consider checking that the λ-term over the set of constants Σ_1 of Example 1

$$(\ Y_{Turing}\ \ (\ \lambda F.\, \lambda x.\, \texttt{if}\ \ x\ \ \ (\ F\ \ (\ \texttt{data}\ x\ )\ )\ )\ )\ \ \texttt{Nil} \tag{3.1}$$

has $\mathcal{T}_1$ for Böhm tree – or, in other words, that an *infinitary* head reduction computes it.

The combinator Y_{Turing} allows to incorporate recursion in the λ-calculus[3]; however, there is no guarantee that this recursion will actually produce a normal form, as it may be applied to a diverging term. The notion of typing, and its extension to a recursive framework in §3.2, will provide a more convenient setting for the generation by Böhm evaluation of trees as $\mathcal{T}_1$. The use

[3]There are quite many choices of fixpoint combinators in the untyped λ-calculus – in fact, they form a recursively enumerable set [Gol05].

of recursion schemes, or the introduction of fixpoint combinators directly in the calculus, will tremendously ease the computation of the normal form of a recursive term as (3.1).

Remark 1 (Relating reduction strategies.)**.** While outermost policies are in general strictly more productive than innermost ones, some syntactic transformations allow to simulate the behavior of a strategy of a class into the other. The simulation of call-by-name (CBN) strategies using a call-by-value (CBV) mechanism is not very hard, and relies on the notion of *thunks*, which are functions delaying the computation of arguments until they actually are accessed in the CBV evaluation.

The converse direction is harder, and relies on the *continuation-passing style* (CPS) mechanism, a notion which happens to be deeply related to the double negation translation by Gödel of classical logic into intuitionistic logic, and with the dialogical interpretation of proofs as interactive strategies. In a program in continuation-passing style, every function receives among its arguments a number of *continuations*, from which it picks which function should be called next in the execution flow of the program. The translation of "usual" programs to their CPS counterpart can be performed automatically, and is often used as an intermediate representation of programs in the compilers of functional languages.

Regarding the simulation of innermost policies using outermost ones: given a λ-term t normalizing in a call-by-value policy to a form $\langle\, t \,\rangle$, one can define another term t' whose call-by-name evaluation computes $\langle\, t \,\rangle$, using a CPS transform.

The relation between call-by-name and call-by-value strategies by means of continuations appears in Plotkin's work [Plo75]. For an history of the early discovery of continuations, see [Rey93].

The simply-typed fragment. The λ-term Ω is quite strange: it notably contains

$$(\,\lambda x.\, x\ x\,)$$

which takes as input a term which will be at the same time used as a function, and as an argument of this function. This astonishing behavior allows Ω to loop forever, by always applying this term to itself. It is thus tempting to restrict the λ-calculus to terms which conform to some type – and as Milner summarized [Mil78],

well-typed programs can't go wrong.

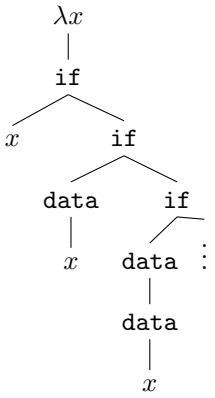
In this section, *"going wrong"* will refer to the *divergence* of terms, that is, to the inability of the standard reduction to compute a normal form in finite time. We define *simple types*, using the grammar[4]

$$\kappa \ ::=\ o \mid \kappa \to \kappa$$

where o is the *base type*, also called *ground type*. In this thesis, o is the type of terms representing trees, that is, the type of terms whose normal form consist

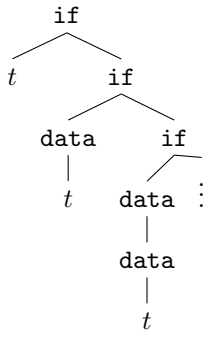
[4]We use the letter κ for simple types, following Kobayashi and Ong [KO09] who called them *kinds* to distinguish them from *refined* types, in which an intersection operator appears, see Chapter 5.

only in constants which are fully-applied, in the sense that a constant of arity i will have i arguments. Relating the type of a term with the one of its normal form follows from a property of stability of typing under reduction called *subject reduction*, see Proposition 3. The set of simple types is denoted $\mathcal{K}$. Under this interpretation of the base type o as the type of trees, a typical term of type $o \to o$ would be



(3.2)

which can be understood as the function mapping a tree t to the tree



By convention, the arrow associates to the right, so that every simple type may be decomposed as

$$\kappa = \kappa_1 \to \cdots \to \kappa_n \to o$$

where n is called the *arity* $\mathrm{ar}(a)$ of the simple type κ. Considering this decomposition, the order $order(\kappa)$ of the simple type κ is defined as

$$\begin{cases} 0 & \text{if } n = 0 \\ 1 + \max(order(\kappa_1), \ldots, order(\kappa_n)) & \text{else.} \end{cases}$$

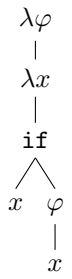
For instance, the type

$$\kappa_2 \quad = \quad (o \to o) \to o \to o$$

is of arity 2, and of order

$$\begin{array}{lcl} order(\kappa_2) & = & 1 + \max(order(o \to o), order(o)) \\ & = & 1 + \max(1 + order(o), 0) \\ & = & 1 + 1 + 0 \\ & = & 2 \end{array}$$

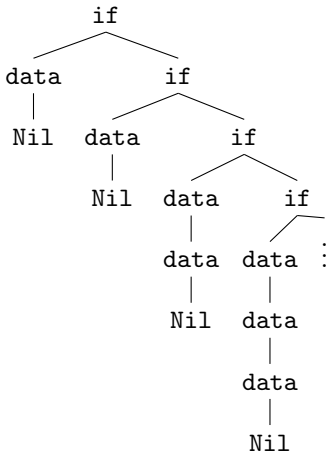
A typical term of type κ_2, denoted as a tree, would be



(3.3)

which, applied to the tree function (3.2) and to the tree

evaluates to the term of type o (that is, to the tree)



Following this idea that *types describe properties of the arguments of terms* – as being a tree, a function mapping a tree to a tree. . . , and of *how they evolve during computations*, a whole inference system can be designed to assign a simple type to terms, see Figure 3.4. In this figure, as well as in the sequel of this document, we take the convention that $o^n \to o$ describes the simple type

$$\underbrace{o \to \cdots \to o}_{n \text{ times}} \to o$$

Not all terms have a simple type: Ω, for instance, does not admit one, as it would require some type σ describing the fact that x is used at the same time as a value of type σ and as a function of type $\sigma \to \sigma$. More general type systems would allow such things, as *intersection type systems*, which give several types to a same term, see Chapter 5 where we introduce *refined* type systems[5],

[5]In fact, the type systems we will consider in this thesis will not allow to type Ω, as general intersection type systems would, notably because we restrict our attention to decidable type systems. See Chapter 5 for more on the subject.

$$\text{Var}\quad \frac{x \in \mathcal{V}}{\Gamma,\, x : \kappa \vdash x : \kappa} \qquad \frac{a \in \Sigma \text{ has arity } n}{\Gamma \vdash a : o^n \to o}\ \Sigma$$

$$\text{App}\quad \frac{\Gamma \vdash t : \kappa \to \kappa' \qquad \Gamma \vdash u : \kappa}{\Gamma \vdash t\ u : \kappa'} \qquad \frac{\Gamma,\, x : \kappa \vdash t : \kappa'}{\Gamma \vdash \lambda x.\, t : \kappa \to \kappa'}\ \lambda$$

Figure 3.4: The simple type system for the λ-calculus (additive presentation).

allowing to track more precise properties of terms during computations than just being trees, tree functions...

An important property of the simply-typed λ-calculus states, in some sense, its stability with respect to reduction:

Proposition 3 (Subject reduction)**.** *If* $\Gamma \vdash t : \kappa$*, and* $t \to_\beta^* t'$*, then* $\Gamma \vdash t' : \kappa$*.*

For a proof, see [GL15, Lemme 1] (in French), or [Ré13, Theorem 9]. We prove subject reduction in the more general framework of infinitary λ-calculus in Proposition 7 – the basic idea being similar, yet extended to an infinitary framework using coinductive techniques.

Note that the converse property, namely *subject expansion*, does not hold for the simply-typed λ-calculus: recall that the term

$$t_w \quad = \quad (\,\lambda x.\, a\,)\ (\,\lambda x.\, x\ x\,)\ (\,\lambda x.\, x\ x\,)$$

we considered earlier can be reduced to a, to which we may attribute some simple type, while t_w itself does not admit one, notably due to Theorem 7, which we are about to state. In Part II, we will consider more expressive type systems, which will be precise enough to allow subject expansion to hold.

A fundamental property of the simply-typed λ-calculus is the following theorem:

Theorem 7 ([Tai67])**.** *The simply-typed* λ*-calculus is strongly normalizing.*

Since the normalization of a term implies that its normal form can be computed in finite time, and that each reduction step manipulates finite data, simply-typed λ-terms only allow to compute *finite* trees. Again, the standard reduction, which computes terms by reducing head redexes first, and choosing the leftmost one in a concurrent situation, computes the normal form we look for. In order to compute infinite trees like $\mathcal{T}_1$ using terms, we need to extend them with some infinitary, yet finitely representable behavior: *recursion*. As discussed previously in the section, we could consider the Turing combinator in the whole (untyped) λ-calculus. We would however like to restrict our attention to an "infinitary" fragment of the λ-calculus in which an appropriate extension of the simple type system allows to *decide* properties on λ-terms and on their infinitary normal forms. Historically, several such extensions were given, in different communities, such as *higher-order recursion schemes*, and the λY-calculus, to which the next two sections are devoted. Before introducing them, we find useful to relate two different presentations of the simple type system of the λ-calculus, which differ in their management of contexts – an issue deeply related to *linear logic*.

$$\text{Var}\quad \frac{x \in \mathcal{V}}{x : \kappa \vdash x : \kappa} \qquad \frac{\Gamma, x : \kappa, y : \kappa \vdash t : \kappa' \qquad z \notin \Gamma}{\Gamma, z : \kappa \vdash t[z \leftarrow x, y] : \kappa'}\ \text{Contraction}$$

$$\Sigma\quad \frac{a \in \Sigma \text{ has arity } n}{\emptyset \vdash a : o^n \to o} \qquad \frac{\Gamma \vdash t : \kappa' \qquad x \notin \Gamma}{\Gamma, x : \kappa \vdash t : \kappa'}\ \text{Weakening}$$

$$\text{App}\quad \frac{\Gamma_1 \vdash t : \kappa \to \kappa' \qquad \Gamma_2 \vdash u : \kappa}{\Gamma_1, \Gamma_2 \vdash t\ u : \kappa'} \qquad \frac{\Gamma, x : \kappa \vdash t : \kappa'}{\Gamma \vdash \lambda x.t : \kappa \to \kappa'}\ \lambda$$

(where $dom(\Gamma_1) \cap dom(\Gamma_2) = \emptyset$)

Figure 3.5: The simple type system for the λ-calculus, in its multiplicative presentation.

Additive vs. multiplicative presentations of the type system. The simple type system of the λ-calculus admits several presentations, which all have different advantages. The one we gave in Figure 3.4 is called its *additive* presentation. If we look closely at it, it does not precisely keep track in contexts of the free variables occurring in the term, if we think of these derivation trees as proof-search devices working bottom-up. For instance, in the Application rule

$$\text{App}\quad \frac{\Gamma \vdash t : \kappa \to \kappa' \qquad \Gamma \vdash u : \kappa}{\Gamma \vdash t\ u : \kappa'}$$

the term u may use a free variable x which does not occur in t, so that we could remove it from the context used to type t. This leads to the multiplicative presentation of this rule:

$$\text{App}\quad \frac{\Gamma_1 \vdash t : \kappa \to \kappa' \qquad \Gamma_2 \vdash u : \kappa}{\Gamma_1, \Gamma_2 \vdash t\ u : \kappa'}$$

where the sum of two contexts Γ_1 and Γ_2, often denoted «Γ_1, Γ_2» is defined here as their union – we assume that a variable can not occur in Γ_1 and Γ_2 with different types. If it does, in this simple-typed framework, it means that the same name was assigned twice, to two different variables, a case we excluded thanks to α-conversion. This idea of *splitting contexts* leads to the *multiplicative presentation* of simple type system, in Figure 3.5. We define the *domain* $dom(\Gamma)$ of a context Γ as the set of variables occurring in it.

This type system features two extra rules, namely *contraction* and *weakening*, which are crucial to make the additive and multiplicative presentations equivalent. Contraction notably allows to apply terms with a common free variable: in $t\ u$, since contexts have to be disjoint to use the Application rule, we give two different names to this same variable in t and u, and contract them to the same variable just after the Application. Note that it is frequent to consider an Application rule embedding the necessary contractions of contexts; most type systems with multiplicative presentations we will consider in the sequel will have this property.

The restriction of structural rules leads to *linear logic*, see for instance [Cur08]. To illustrate the equivalence of the presentations, let us consider the most elaborate step of the translation from an additive typing proof to a multiplicative

one: the translation of an application rule

$$\mathrm{App}_{add}\quad \frac{\begin{array}{cc}\pi_1 & \pi_2\\ \vdots & \vdots\\ \Gamma \vdash t : \kappa \to \kappa' & \Gamma \vdash u : \kappa\end{array}}{\Gamma \vdash t\ u : \kappa'}$$

First, we rename the variables of Γ occurring in t and in u, to obtain two contexts Γ_1 and Γ_2 differing only by a renaming of the variables. This renaming also affects terms, defining t' and u', and proofs, giving additive proofs π_1' of

$$\Gamma_1 \vdash t' : \kappa \to \kappa'$$

and π_2' of

$$\Gamma_2 \vdash u' : \kappa$$

Then we suppose that, by structural induction, the additive proofs π_1' and π_2' of $\Gamma_1 \vdash t' : \kappa \to \kappa'$ and of $\Gamma_2 \vdash u', : \kappa$ have been translated to multiplicative proofs $\{\!|\, \pi_1' \,|\!\}$ and $\{\!|\, \pi_2' \,|\!\}$. We then obtain a multiplicative proof of the sequent $\Gamma \vdash t\ u : \kappa'$ as follows:

$$\begin{array}{c}\mathrm{App}_{mult}\\ \text{Contractions}\end{array}\quad \cfrac{\cfrac{\begin{array}{cc}\{\!|\, \pi_1' \,|\!\} & \{\!|\, \pi_2' \,|\!\}\\ \vdots & \vdots\\ \Gamma_1 \vdash t' : \kappa \to \kappa' & \Gamma_2 \vdash u' : \kappa\end{array}}{\Gamma_1, \Gamma_2 \vdash t'\ u' : \kappa'}}{\vdots}}{\Gamma \vdash t\ u : \kappa'}$$

where "Contractions" means that we introduce a contraction for each variable of Γ, in order to equalize its occurrence in Γ_1 with its occurrence in Γ_2. So, it is the contraction rule which allows to translate an additive typing proof to a multiplicative one.

In many situations, it is helpful to leave the contractions implicit: while Γ_1, Γ_2 explicitly assumes that these contexts have different domains, we define the *sum* of contexts

$$\Gamma_1 + \Gamma_2$$

as the context obtained by contracting their common variables – this operation being only defined when these variables share the same type in both contexts. This conveniently allows to hide considerations of α-conversion of variables, which are particularly tedious when multiple applications need to be considered in a row.

To provide an additive representation of a multiplicative proof

$$\mathrm{App}_{mult}\quad \frac{\begin{array}{cc}\pi_1 & \pi_2\\ \vdots & \vdots\\ \Gamma_1 \vdash t : \kappa \to \kappa' & \Gamma_2 \vdash u : \kappa\end{array}}{\Gamma_1, \Gamma_2 \vdash t\ u : \kappa'}$$

suppose that we have translated π_1 and π_2 to additive typing proofs $[\![\pi_1]\!]$ and $[\![\pi_2]\!]$ with the same conclusion. These proofs may be weakened into additive proofs $[\![\pi'_1]\!]$ and $[\![\pi'_2]\!]$, of the sequents

$$\Gamma_1, \Gamma_2 \ \vdash \ t \ : \ \kappa \to \kappa'$$

and $\Gamma_1, \Gamma_2 \ \vdash \ u \ : \ \kappa$, by adding the necessary variables to the contexts of $[\![\pi_1]\!]$ and $[\![\pi_2]\!]$. We obtain:

$$\mathrm{App}_{add} \quad \frac{\begin{array}{cc} [\![\pi'_1]\!] & [\![\pi'_2]\!] \\ \vdots & \vdots \\ \Gamma_1, \Gamma_2 \ \vdash \ t \ : \ \kappa \to \kappa' & \Gamma_1, \Gamma_2 \ \vdash \ u \ : \ \kappa \end{array}}{\Gamma_1, \Gamma_2 \ \vdash \ t \ u \ : \ \kappa'}$$

In linear logic, the weakenings relating $[\![\pi_i]\!]$ to $[\![\pi'_i]\!]$ would be explicit, see [Cur08].

From now on, we may use any of the two presentations. The additive presentation is very useful for proof search, as it does not require to *guess how to split contexts* when typing an Application. However, from the proof-theoretic point of view, multiplicative proofs are very relevant, as they allow to focus more precisely on the hypothesis to be used locally – an information which is crucial in linear logic. Another convenience of the multiplicative presentation is that it allows, when considering proofs, to ensure that the Axiom rules are of the form

$$\mathrm{Axiom} \quad \frac{}{x \ : \ \kappa \ \vdash \ x \ : \ \kappa}$$

This will be useful, for instance, for proving Proposition 4.

In Chapter 10, we introduce a type system which needs to be presented multiplicatively, as a modality will affect the context of the argument of an application, and not the one of its head term. It is unclear whether it can be translated to an equivalent additive presentation.

3.2 Higher-order recursion schemes

Recursion schemes were first introduced by Nivat [Niv72], with the notion of *recursive applicative program scheme*. These schemes consist in a sort of grammar with parameters, a typical rule being

$$F_i \ x_1 \ \cdots \ x_n \quad = \quad t_i$$

with $x_j \ : \ o$ for $1 \leq j \leq n$, and t_i a term over a signature of constants Σ, a set of variables $\mathcal{V}$, and the set $\{ F_i \}$ of *non-terminals*, satisfying a typing condition:

$$x_1 \ : \ o, \cdots, x_n \ : \ o \ \vdash \ t_i \ : \ o$$

for every i. A distinguished non-terminal S acts as *start symbol*, representing the main function of the program to model. Rewriting non-terminals according to the rules of the recursion scheme produce a potentially infinite Σ-labeled tree, which is moreover ranked due to the restriction to simply-typed terms. Recursion schemes inherently allow mutual recursion between non-terminals, and therefore enable to abstract recursive programs.

These schemes are however too weak to compute trees as $\mathcal{T}_1$, for which *higher-order recursive calls* need to be considered. Historically, this extension was not straightforward, as discussed in [Ser13, Section 2.2.2], see also [Cou90], and led after several developments [Ind76, Dam77a, Dam77b, ES77, ES78] to the following definition of *higher-order recursion schemes* (HORS), appearing in [Dam82]:

Definition 19. A *higher-order recursion scheme* $\mathcal{G} = \langle \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$ is given by:

- a set of simply-typed variables $\mathcal{V}$,
- a signature Σ,
- a set of *non-terminals* $\mathcal{N}$,
- a function $\kappa : \mathcal{N} \to \mathcal{K}$ attributing a simple type to each non-terminal. We often naturally denote this simple typing as $F : \kappa(F)$.
- a function $\mathcal{R}$ mapping each non-terminal $F \in \mathcal{N}$ to a closed term over the set of variables $\mathcal{V}$, with constants in $\mathcal{N} \uplus \Sigma$:

$$\mathcal{R}(F) \;=\; \lambda x_1. \ldots \lambda x_n. t \tag{3.4}$$

  of simple type $\kappa(F)$, such that each of the x_i is in $\mathcal{V}$, and that t is a term without abstractions,
- and of an *axiom* $S \in \mathcal{N}$, also called *start symbol*, and such that $S : o$.

The *order* of a (higher-order) recursion scheme $\mathcal{G}$ is defined as

$$\mathit{order}(\mathcal{G}) \;=\; \max_{F \in \mathcal{N}} \; (\, \mathit{order}(\kappa(F)) \,)$$

Every HORS $\mathcal{G}$ induces a rewriting system, whose rewriting relation $\to_{\mathcal{G}}$ is defined inductively over terms by

- $F\, t_1 \cdots t_n \;\to_{\mathcal{G}}\; t[x_i := t_i]$ if $\mathcal{R}(F) = \lambda x_1 \cdots \lambda x_n. t$, where t is a term without abstractions,
- and if $s \to_{\mathcal{G}} t$ then $s\, u \to_{\mathcal{G}} t\, u$ and $u\, s \to_{\mathcal{G}} u\, t$.

Notice that the functionality of $\mathcal{R}$ makes the HORS *deterministic*: every non-terminal induces a unique rewriting rule.

Example 5 (An order-1 example)**.** Recall that we claimed that the (untyped) λ-term (3.1) normalizes to $\mathcal{T}_1$. Using recursion schemes, a more convenient representation can be given:

$$\mathcal{G}_1 \quad = \quad \begin{cases} \mathtt{S} & = & \mathtt{L\ Nil} \\ \mathtt{L} & = & \lambda x.\, \mathtt{if}\ x\ (\mathtt{L}\ (\mathtt{data}\ x)) \end{cases}$$

This recursion scheme is of order 1, as $x : o$. Its rewriting to a Σ-labeled ranked tree is depicted in Figure 3.6. Notice that this representation crucially differs from the λ-term (3.1) by the fact that *no interpretation of the recursion operator is given a priori*: we just consider a syntactic rewriting mechanism

$$F \;\to_{\mathcal{G}}\; \mathcal{R}(F)$$

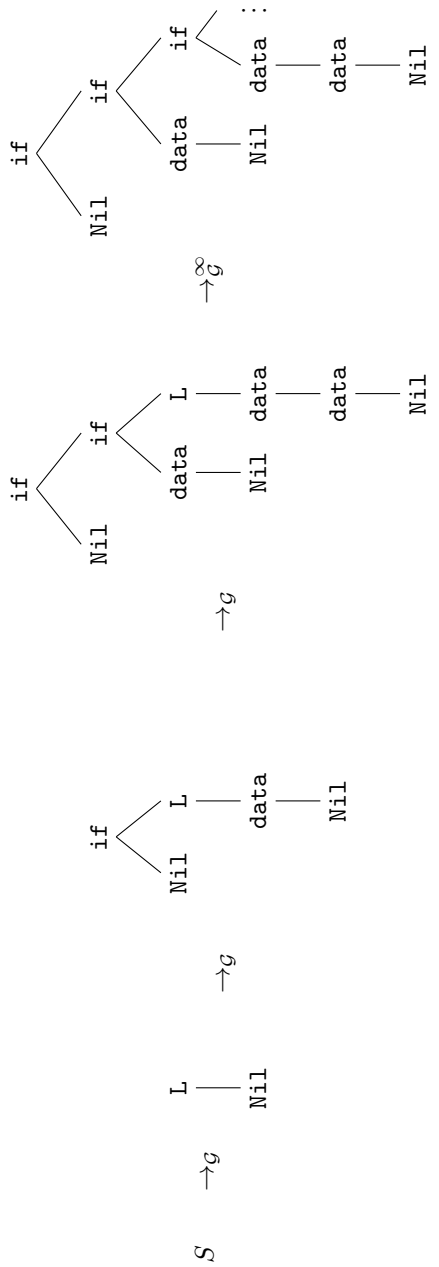


Figure 3.6: Rewriting of an order-1 recursion scheme.

The choice of a fixpoint operator nevertheless appears in the semantics of recursion schemes. Indeed, since Nivat's early work, the evaluation of recursion schemes is not only considered syntactically, but also semantically in an additional *domain* of interpretation – recall that, by Proposition 1, every domain is endowed with a least fixpoint operator, allowing the interpretation of recursion. Such semantic considerations, among which the choice of an appropriate fixpoint operator, are crucial aspects of this thesis, see notably Part III.

Example 6 (An order-2 example)**.** We now consider the (higher-order) recursion scheme considered by Serre in [Ser13]

$$\mathcal{G}_2 \quad = \quad \left\{ \begin{array}{rcl} S & = & M\ \texttt{Nil} \\ M & = & \lambda x.\, \texttt{if}\ (\ \texttt{commit}\ x\ )\ (\ A\ x\ M\ ) \\ A & = & \lambda y.\, \lambda \varphi.\, \texttt{if}\ (\ \varphi\ (\ \texttt{error end}\ )\ )\ (\ \varphi\ (\ \texttt{cons}\ y\ )\ ) \end{array} \right.$$

over the ranked alphabet

$$\Sigma_2 \quad = \quad \{\texttt{if} : 2, \texttt{commit} : 1, \texttt{error} : 1, \texttt{Nil} : 0, \texttt{end} : 0, \texttt{cons} : 1\}$$

the set of variables $\{\, x : o,\, y : o,\, \varphi : o \to o \,\}$, and the set of non-terminals

$$\{\, S : o,\, M : o \to o,\, A : o \to (o \to o) \to o \,\}$$

This recursion scheme is of order 2, since it is the order of $\kappa(A)$, and that both S and M are attributed a type of lesser order. The variable φ allows to pass *continuations* in the recursion scheme. The first steps of rewriting of $\mathcal{G}_2$ are depicted in Figure 3.7.

Unlike Example 5, reduction leads in Example 6 to trees in which *several* non-terminals may be reduced. The tree produced by a recursion scheme will be the one obtained by the "most productive" rewriting sequence, an intuition whose formalization gives the formal definition of the tree $\langle\, \mathcal{G} \,\rangle$ produced by a HORS $\mathcal{G}$.

Definition 20 (Domain of trees)**.** We consider the set $\mathit{Trees}_\perp(\Sigma)$ of $(\Sigma \uplus \{\perp\})$-labeled ranked trees, where $\perp : 0$ may only labels leaves, with a partial order over nodes:

$$\text{for } a, b \in \Sigma \uplus \{\perp\}, \qquad a \preccurlyeq b \iff a = b \quad \text{or} \quad a = \perp$$

extending to a partial order over trees: for $t, t' \in \mathit{Trees}_\perp(\Sigma)$,

$$t \preccurlyeq t' \iff \mathit{Dom}(t) \subseteq \mathit{Dom}(t') \quad \text{and} \quad \alpha \in \mathit{Dom}(t) \Rightarrow t(\alpha) \preccurlyeq t'(\alpha)$$

The partial order $(\mathit{Trees}_\perp(\Sigma), \preccurlyeq)$ is a domain.

The rewriting process induced by a recursion scheme produces $(\Sigma \uplus \mathcal{N})$-labeled trees. Note that it *progresses* in some sense: a Σ-labeled prefix of a tree obtained during the rewriting process will never be modified in the sequel of the computation. It therefore makes sense to define the greatest such prefix as the *partial production* of a rewriting sequence:

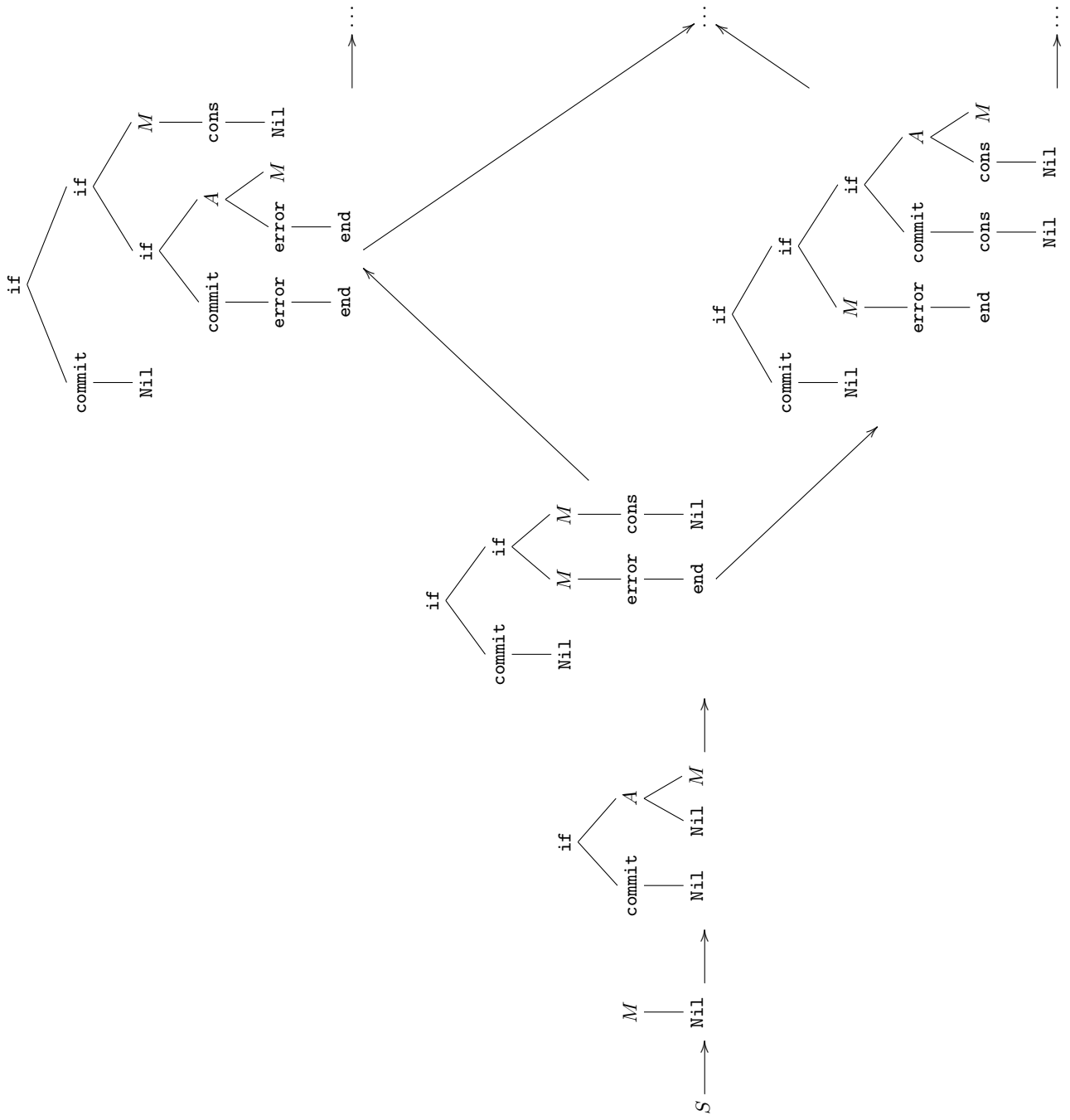


Figure 3.7: Concurrent rewriting of an order-2 recursion scheme.

Definition 21. Let t be a $(\Sigma \uplus \mathcal{N})$-labeled tree obtained via a reduction sequence

$$S \quad \to_{\mathcal{G}}^{*} \quad t$$

We define the *partial production* t^{∇} of this reduction sequence by induction on t:

- if $t = a \in \Sigma$, $t^{\nabla} = a$,
- if $t = F \in \mathcal{N}$, $t^{\nabla} = \bot$,
- if $t = t_0\ t_1 \cdots\ t_n$, $t^{\nabla} = \bot$ if $t_0^{\nabla} = \bot$, and $t^{\nabla} = t_0^{\nabla}\ t_1^{\nabla} \cdots\ t_n^{\nabla}$ else.

Figure 3.8 depicts the partial productions at every step of the rewriting sequence on $\mathcal{G}_1$ represented in Figure 3.6.

Recall that since $\mathit{Trees}_{\bot}$ is a domain, every family of $(\Sigma \uplus \{\bot\})$-labeled ranked trees whose pairs of elements always have a common supremum – that is, every *directed* family of trees – has a supremum. The rewriting relation of recursion schemes is confluent, so that the family

$$\left\{ t^{\nabla} \in \mathit{Trees}_{\bot} \mid S \to_{\mathcal{G}}^{*} t \right\}$$

is directed. The idea is that every partial production is a prefix of the value tree of $\mathcal{G}$, and that two partial productions can be "combined" to a bigger one – this being the informal meaning of *confluence*. $\mathit{Trees}_{\bot}$ being a domain means that a family of prefixes of the same tree can be combined altogether to produce this tree.

Definition 22 (Value tree of a HORS)**.** The *value tree* of a HORS $\mathcal{G}$, also called the tree *generated* by $\mathcal{G}$, is defined as the supremum

$$\langle \mathcal{G} \rangle \quad = \quad \bigvee \left\{ t^{\nabla} \in \mathit{Trees}_{\bot} \mid S \to_{\mathcal{G}}^{*} t \right\}$$

of the directed family of partial productions of all reduction sequences of $\mathcal{G}$, in the domain $\mathit{Trees}_{\bot}$.

On productivity. Notice that the value tree of a recursion scheme $\mathcal{G}$ may contain the divergence symbol $\bot$, corresponding to the fact that the infinite rewriting of the subtree rooted at this symbol never *produces* a head symbol. Consider for instance

$$\mathcal{G}_{div} \quad = \quad \left\{ \begin{array}{rcl} S & = & M\ \texttt{Nil} \\ M & = & \lambda x.\, \texttt{if}\ (\ \texttt{commit}\ x\ )\ (\ A\ x\ M\ ) \\ A & = & \lambda y.\, \lambda \varphi.\, A\ (\ M\ y\ )\ (\ A\ (\ \varphi\ y\ )\ \varphi\ ) \end{array} \right.$$

The beginning of a possible rewriting sequence for $\mathcal{G}_{div}$ is depicted in Figure 3.9. The problem is that the rule for A outputs A again as its head symbol, so that whatever we reduce below A, the partial production of the corresponding rewriting sequence will never increase, and the value tree of $\mathcal{G}_{div}$ is the partial tree

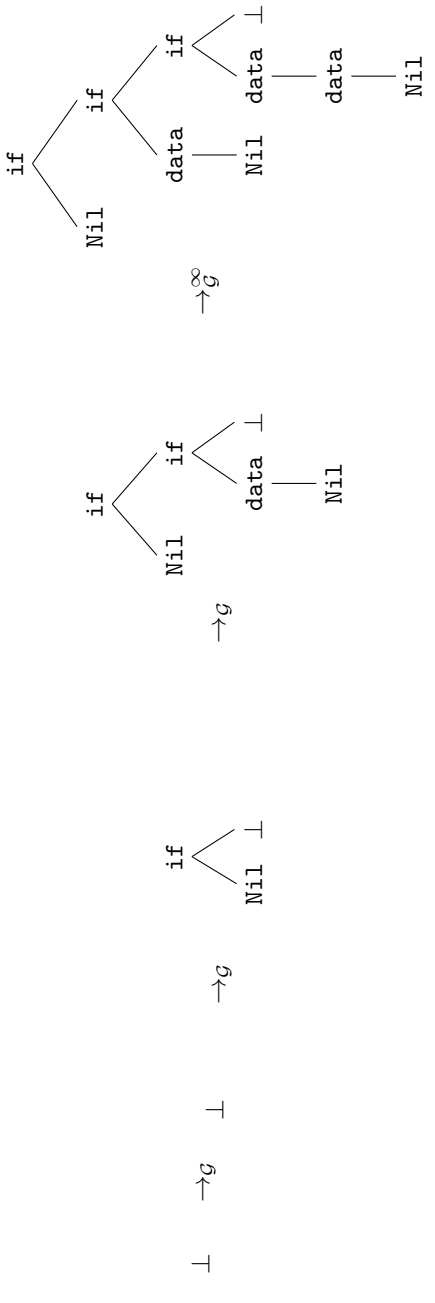
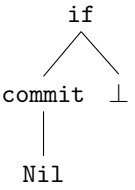


Figure 3.8: Partial productions of a rewriting sequence.

$\langle\, \mathcal{G}_{div} \,\rangle \;=$



An important fact is that higher-order recursion schemes are not Turing-complete, which would have made undecidable the question of determining whether the divergence symbol $\bot$ occurs in the value tree of a given recursion scheme. This lack of Turing-completeness can be understood by noticing that the absence of pattern-matching forbids the definition of the predecessor function of arithmetics in the language of recursion schemes. We call *productive* a recursion scheme whose value tree does not contain the divergence symbol $\bot$. As we shall see in §3.5, it is decidable whether a recursion scheme of order n is productive, and the complexity of the problem is n-EXPTIME. Even better, from a HORS $\mathcal{G}$ over a signature Σ whose value tree contains the divergence symbol $\bot$, we can effectively compute a *productive* HORS $\mathcal{G}_{prod}$, over the signature $\Sigma \uplus \{\Omega\}$, whose value tree only differs from the one of $\mathcal{G}$ by the fact that the divergence symbol $\bot$ is replaced by the distinguished divergence constant Ω. $\mathcal{G}_{prod}$ is productive, since instead of rewriting forever a term with no head normal form, it only outputs *once* the divergence constant Ω, and then stops the computation of the branch.

In fact, a consequence of the extension of the simply-typed λ-calculus to this recursive formalism that are higher-order recursion schemes is that

well-typed programs can't go too *wrong*,

since their divergence can effectively be eliminated by the construction sketched in §3.5. In the sequel, we may therefore restrict ourselves to the case where every HORS is productive.

On regularity. Notice that the regular trees defined in §2.1 correspond to the trees generated by order-0 recursion schemes, which correspond to sets of equations

$$F_i \;=\; t_i$$

where t_i does not contain variables nor abstractions, and is therefore a fully-applied term consisting of applications of elements of Σ. In other terms, it is a *tree context* containing calls to non-terminals F_j, and we can thus represent a tree generated by an order-0 recursion scheme folded as a finite graph. We extend this notion of regularity in the following way:

Definition 23 (Higher-order regularity)**.** We say that a tree t is *higher-order regular*, or *regular of order* n, if there is a higher-order recursion scheme $\mathcal{G}$ of order n such that $t \;=\; \langle\, \mathcal{G} \,\rangle$.

This notion encompasses other familiar classes of finitely generated trees: order-1 regular trees correspond to *algebraic* trees, generated by context-free grammars, while order-2 regular trees correspond to *hyperalgebraic* trees.

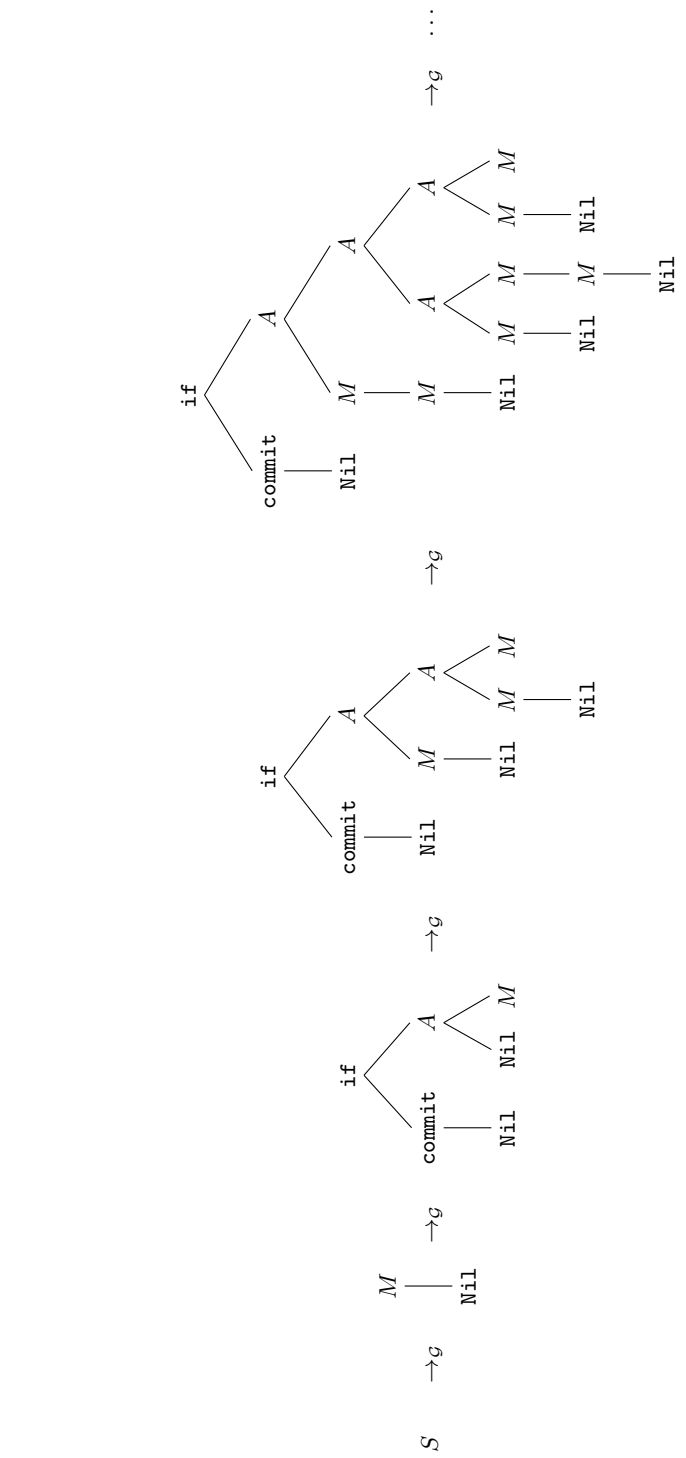


Figure 3.9: Divergent rewriting of an order-2 recursion scheme.

Reduction strategies. The definition of the value tree of a recursion scheme does not provide an *effective* strategy of computation. In his influential paper on evaluation strategies for higher-order recursion schemes, Damm [Dam82] introduced the OI (outermost-innermost) and IO (innermost-outermost) evaluation policies, which are strongly reminiscent, respectively, of the *call-by-name* and *call-by-value* strategies we discussed in the previous section for the λ-calculus. OI strategies are notably known to be more productive than IO ones, precisely for the reasons we discussed about CBN and CBV. Damm proved that the value tree of a recursion scheme can be computed using a OI policy – which corresponds, in a slightly different framework, to the fact that head reductions (or, equivalently, Böhm normalization) compute the normal forms of λ-terms.

Just as call-by-name and call-by-value evaluation strategies could be related in the λ-calculus through CPS translations, an analogous translation between OI and IO evaluation policies of recursion schemes can be designed, see Haddad's work on the subject [Had12, Had13b].

3.3 A simply-typed λ-calculus with fixpoints

Higher-order recursion schemes may be understood as an extension of the simply-typed λ-calculus where terms feature *typed recursion* of *any order*, and normalize to trees. We therefore extend the simply-typed λ-calculus with a family of typed syntactic fixpoint operators

$$Y_\kappa \quad : \quad (\,\kappa \to \kappa\,) \to \kappa$$

indexed by the simple types $\kappa \in \mathcal{K}$, and introduce a family of reduction rules

$$Y_\kappa \ (\,\lambda x.\, M\,) \quad \to_\delta \quad M[x \leftarrow Y_\kappa \ (\,\lambda x.\, M\,)]$$

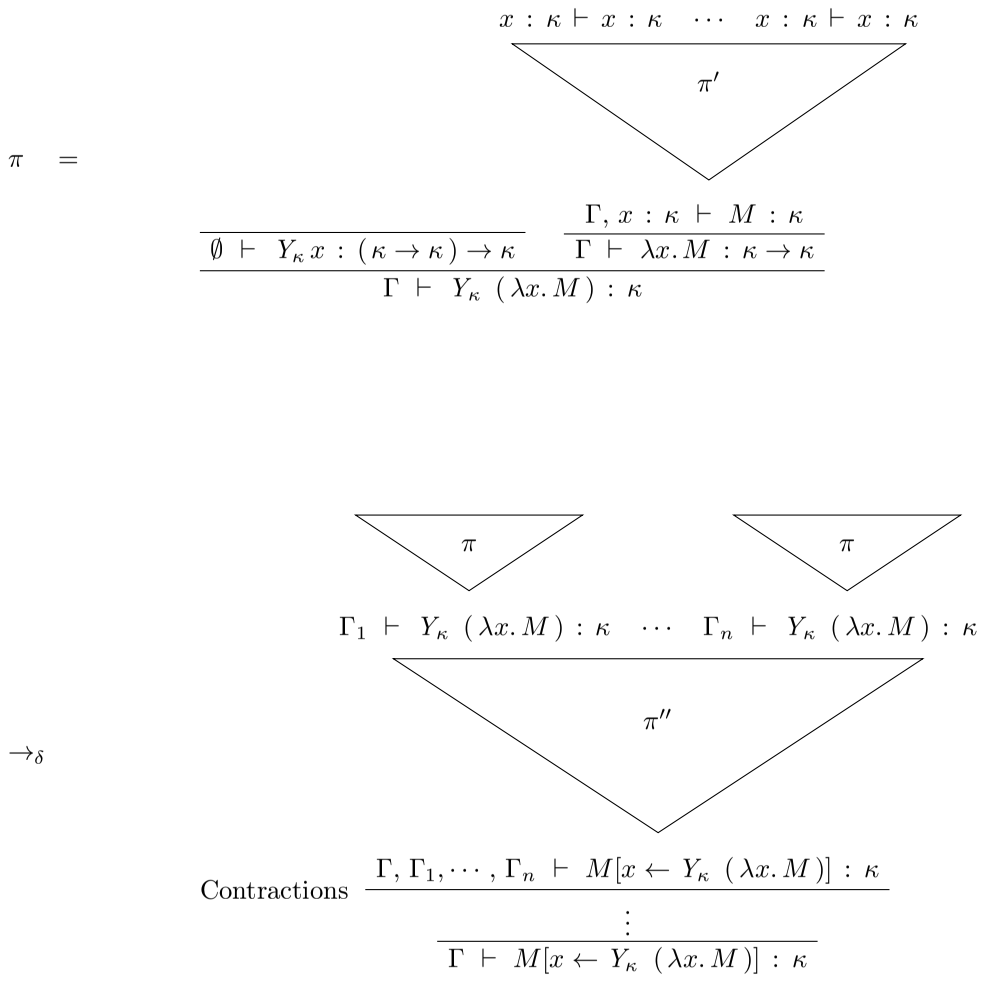
The resulting calculus is named the λY-calculus [Sta02]. Note that the combinators Y_κ are considered as *constants* extending the set of constants Σ of interest, so that the rules of Figure 3.4 provide a simple type system for the λY-calculus as well.

The *subject reduction* of the simply-typed λ-calculus can be extended by considering the rewriting of a term under the rule $\to_\delta$, in addition to the rule $\to_\beta$. We consider the multiplicative variant of the simple type system for convenience.

Proposition 4 (Subject reduction for the λY-calculus)**.** *Given a λY-term t, suppose that $\Gamma \ \vdash \ t \ : \ \kappa$. If $t \to^*_{\beta\delta} t'$, then $\Gamma \ \vdash \ t' \ : \ \kappa$.*

The preservation of the type of a term rewritten by the rule $\to_\delta$ in a given context is depicted in Figure 3.10. Notice that it can be understood as a *proof rewriting*, and that is convenient to use the multiplicative variant of the type system, as it ensures that Axiom leaves are of the shape

$$\text{Axiom} \quad \frac{}{x \ : \ \kappa \ \vdash \ x \ : \ \kappa}$$



where π'' is obtained from π' by removing x from the context, propagating the contexts Γ_i from the former Axioms leaves to the root of π', and by replacing x with $Y_\kappa\ (\lambda x.M)$ in the term we type.
Notice that we did not explicitly distinguish the *different* occurrences of x appearing bound in n different terms, to ease the representation of the operation. However, α-conversion would ensure that we can rename them all to different, unique names.

Figure 3.10: Subject reduction: stability by a step of $\to_\delta$.

HORS and the λY-calculus. *Subject reduction* ensures that the normal form, if it exists, of a λY-term of simple type o is again of type o: productive simply-typed λ-terms with recursion therefore generate trees over their signature of constants. These trees are computed by Böhm evaluation, which can be extended to handle the rule $\rightarrow_\delta$:

$$BT(Y\ (\lambda x.\, t)) \quad = \quad BT\,(t\,[x \leftarrow Y\ (\lambda x.\, t)])$$

which specifies the canonical evaluation of untyped λ-terms to a canonical evaluation for λY-terms. Using a Church encoding, we can understand the normal form of a λY-term of simple type o over the signature of constants Σ as a Σ-labeled ranked tree. Terms of ground type of the λY-calculus are then equivalent to HORS, for that they evaluate to the same trees:

Proposition 5 (HORS and λY-terms [DF80])**.** *Considering implicitly the correspondence between terms and trees obtained by a Church encoding, we have:*

- *For every closed λY-term t of type o over the signature of constants Σ, there exists a HORS $\mathcal{G}_t$ over the same signature whose value tree represents the normal form $BT(t)$ of t.*
- *For every HORS $\mathcal{G}$ over a signature Σ, there exists a closed λY-term $t_\mathcal{G}$ of type o normalizing to $\langle\, \mathcal{G}\,\rangle$, and defined over the same signature.*

The translation is described in [SW12]; we sketch on p.257 the translation from higher-order recursion schemes to λY-terms. The converse direction is more subtle.

Instead of adding explicitly constants from a signature to the grammar of λ-terms or λY-terms, we could treat them as free variables of order 0 or 1. We obtain in this case a similar proposition, where the terms are no longer closed, but with free variables of order at most one, to be thought of as the elements of Σ. This subtle difference, seen under the prism of linear logic, will allow us to disclose the dual behavior of terms and tree automata in a very precise sense, see §9.1.

Comparison with the untyped λ-calculus and HORS. Contrary to the (untyped) λ-calculus, we only consider typed terms in the λY-calculus. Moreover, the fixpoint operator we introduce is *external* to the calculus, whereas the fixpoint combinators Y_{Church} and Y_{Turing} we considered in §3.1 were part of the language – but typing them would be problematic, as they rely on the auto-application of terms. In the λY-calculus, the syntactic reduction computing the fixpoint is not only externalized, but also performed in one only step using the $\rightarrow_\delta$ rule.

While the λY-calculus and higher-order recursion schemes compute the same trees, there is more freedom in the reductions one can perform in the former, as in the latter the rewriting rule $\rightarrow_\mathcal{G}$ can be understood as a "big-step" relation

$$\rightarrow_\mathcal{G} \quad = \quad \rightarrow_\delta\ (\rightarrow_\beta)^*$$

where all the redexes introduced by the rule expansion $F \rightarrow \mathcal{R}(F)$ are reduced on-the-fly by a series of applications of the reduction rule $\rightarrow_\beta$. Moreover, $\rightarrow_\mathcal{G}$

only allows the expansion of *fully-applied terms*, while $\rightarrow_\delta$ would allow for instance to expand the occurrences of the non-terminal M occurring as leaves – that is, missing an argument – in Fig 3.7.

3.4 The higher-order model-checking problems

After these two introductory chapters, we may now state the problems related to higher-order model-checking (HOMC) we will investigate in this thesis at the light of *linear logic*, which will allow us to connect very naturally the *intersection type theory* used by Kobayashi and Ong for higher-order model-checking [KO09] with *denotational semantics*.

The first, and somehow historical problem, is to decide whether an MSO formula holds at the root of the value tree of a recursion scheme:

Definition 24 (The local HOMC problem)**.** *Given a functional program with recursion, abstracted as a Σ-labeled ranked tree of actions $\langle\, \mathcal{G} \,\rangle$ computed by a higher-order recursion scheme $\mathcal{G}$, and an MSO formula φ over the set of actions Σ, can we decide whether φ holds at the root of $\langle\, \mathcal{G} \,\rangle$?*

Since the branching in $\langle\, \mathcal{G} \,\rangle$ intends at modeling the conditional branchings of the program, checking φ at its root amounts to check whether the formula holds at the beginning of the execution of the program. The formula may state the existence of executions satisfying certain properties, or require that every execution meets some requirement.

The equivalence between the satisfaction of a formula φ and the existence of a winning execution tree of a corresponding automaton $\mathcal{A}_\varphi$ stated by Theorem 3 leads to the following reformulation of the problem, which we will favor in the sequel of this thesis:

Definition 25 (The local HOMC problem, automata-theoretically)**.** *Given a higher-order regular tree $\langle\, \mathcal{G} \,\rangle$ computed by a higher-order recursion scheme $\mathcal{G}$, and an alternating parity automaton $\mathcal{A}$, can we decide whether $\mathcal{A}$ has a winning execution over $\langle\, \mathcal{G} \,\rangle$, starting from its initial state?*

One may want to check not only the truth of a formula at the root of a higher-order regular tree, but on each of its nodes, leading to the following more difficult problem introduced in [BCOS10] — where it is called *logical reflection*:

Definition 26 (The global HOMC problem)**.** *Given a higher-order regular tree $\langle\, \mathcal{G} \,\rangle$ produced by a higher-order recursion scheme $\mathcal{G}$, and an MSO formula φ, can we compute a higher-order recursion scheme $\mathcal{G}^\varphi$ producing a lifting $\langle\, \mathcal{G}^\varphi \,\rangle$ of $\langle\, \mathcal{G} \,\rangle$, obtained by annotating with a distinguished symbol $\bullet$ every label of a node of $\langle\, \mathcal{G} \,\rangle$ where φ holds?*

Again, an automata-theoretic reformulation is possible, see [Had13b]. However, we will not need to consider this reformulation of the global HOMC problem in the sequel, as we will focus on the strictly harder problem of *MSO selection*, introduced by Carayol and Serre in [CS12]:

Definition 27 (The selection problem)**.** *Given a higher-order regular tree $\langle\, \mathcal{G} \,\rangle$ computed by a higher-order recursion scheme $\mathcal{G}$, and an existential MSO formula $\exists X.\, \varphi[X]$, can we compute a higher-order recursion scheme producing*

a lifting $\langle\, \mathcal{G}^{\varphi} \,\rangle$ *of* $\langle\, \mathcal{G} \,\rangle$ *obtained by annotating with* $\bullet$ *a set of nodes* Y *such that* $\varphi[Y]$ *holds?*

This problem encompasses the global higher-order model-checking one: for an MSO formula ψ, the selection problem for the existential MSO formula $\exists X.\ x \in X \Leftrightarrow \psi[X]$ reduces to the global HOMC of ψ. In the sequel, we will more precisely consider the automata-theoretic version of this problem introduced by Haddad [Had13a], which asks whether the existence of a winning run-tree of an alternating parity automaton over a higher-order regular tree implies the existence of a *higher-order regular* winning execution tree of this automaton:

Definition 28 (The selection problem, automata-theoretically)**.** *Given a higher-order regular tree* $\langle\, \mathcal{G} \,\rangle$ *computed by a higher-order recursion scheme* $\mathcal{G}$*, an alternating parity automaton* $\mathcal{A}$*, and a state* q *of this automaton from which* $\langle\, \mathcal{G} \,\rangle$ *is accepted, can we compute a higher-order recursion scheme* $\mathcal{G}_q$ *producing a winning run-tree* $\langle\, \mathcal{G}_q \,\rangle$ *of* $\mathcal{A}$ *from* q *over* $\langle\, \mathcal{G} \,\rangle$*?*

On decidability. These problems were all proven decidable by several independent methods:

- For the local model-checking problem, independent proofs appear in [Ong06, HMOS08, KO09, SW14, TO14, SW15a, GM15a]
- For the global model-checking problem, also called *logical reflection*, see [BCOS10, SW14, SW15a]
- For the selection problem, see [CS12, Had13a, TO14, GM15a].

Descriptions of these approaches are given in the related works, see §1.2.

Verification of higher-order terms. These problems are concerned with the verification of MSO properties over infinite *trees*, generated by the head normalization of λY-terms of simple type o. Clairambault and Murawski use a de Bruijn representation of binders to encode Böhm trees of λY-terms of *any* simple type into higher-order recursion schemes. They reduce in this way the verification of properties over the Böhm trees of λY-terms to their verification over HORS. Note that these properties must be formulated *over the de Bruijn representation* of the Böhm tree, and not directly on the Böhm tree itself. By reduction to the case of HORS, the model-checking of these properties is decidable; it would have not been the case for the general case of MSO properties over Böhm trees (without de Bruijn encoding) which, as they show, is undecidable.

3.5 Productivity and higher-order recursion schemes

As claimed earlier, it is possible to decide whether a recursion scheme $\mathcal{G}$ is productive and, if it is not, to compute effectively a *productive* HORS $\mathcal{G}_{prod}$ such that $\langle\, \mathcal{G} \,\rangle$ and $\langle\, \mathcal{G}_{prod} \,\rangle$ have the same tree domain, coincide on their Σ-labeled nodes, and such that the divergence symbol $\bot$ occurring in $\langle\, \mathcal{G} \,\rangle$ is replaced in $\langle\, \mathcal{G}_{prod} \,\rangle$ by a distinguished nullary symbol Ω. The difference comes from the

fact that, instead of being the result of an infinite, divergent computation, Ω is a symbol which will be outputted *in one step* during the evaluation of $\mathcal{G}_{prod}$.

Several constructions lead to this result, one of them consisting in evaluating the terms appearing in the rewriting of the recursion scheme in a domain, and then on modifying these rules according to their semantics, see for instance the procedure of $\bot$-elimination of [Had13b, Section 3.3.6].

We explain here an alternative and elegant construction due to Serre [Ser13, Section 9.2.4]. It is performed in several steps:

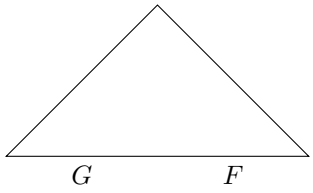
$$\mathcal{G} \quad \to \quad \mathcal{G}_\varepsilon \quad \to \quad \mathcal{G}_\varepsilon^\varphi \quad \to \quad \mathcal{G}_{prod}$$

where:

- $\mathcal{G}_\varepsilon$ is a *productive* recursion scheme, obtained from $\mathcal{G}$ by introducing a new symbol $\varepsilon : o \to o$ in the signature Σ, and by replacing every rewrite rule
$$\mathcal{R}(F) \;=\; \lambda x_1. \ldots \lambda x_n. t$$
with
$$\mathcal{R}(F) \;=\; \lambda x_1. \ldots \lambda x_n. \varepsilon\ t$$
making each rule productive,
- and by considering the MSO formula
$$\varphi \quad = \quad \exists X.\ (\, root \in X \ \wedge\ \forall x \in X.\ (\, \underline{\varepsilon}(x) \ \wedge\ (\, \exists y \in X.\, succ(x,y)\,)\,)\,)$$
adapted from the formula $\varphi^M_{1,coind}$ of Example 2 to detect infinite branches containing only the symbol ε. The application of the effective construction provided by the decidability of the global model-checking problem leads to the definition of $\mathcal{G}_\varepsilon^\varphi$,
- and $\mathcal{G}_{prod}$ is finally defined by considering each rewrite rule of $\mathcal{G}_\varepsilon^\varphi$, and by
  - replacing it with
  $$\mathcal{R}(F) \;=\; \lambda x_1. \ldots \lambda x_n. t$$
  if it is of the shape
  $$\mathcal{R}(F) \;=\; \lambda x_1. \ldots \lambda x_n. \varepsilon\ t$$
  - and replacing it with
  $$\mathcal{R}(F) \;=\; \lambda x_1. \ldots \lambda x_n. \Omega$$
  if it is of the shape
  $$\mathcal{R}(F) \;=\; \lambda x_1. \ldots \lambda x_n. \varepsilon^\bullet\ t$$

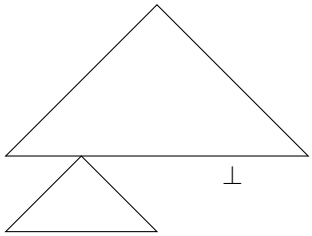
Let us illustrate this construction by a few drawings. Consider a partial rewriting of $\mathcal{G}$ in which the rewriting of some occurrence F of a non-terminal

will never actually produce anything — the most simple case being $F = F$ — while another occurrence of a non-terminal G will be productive:



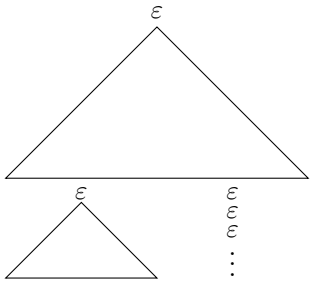
Since rewriting F does not produce anything, the value tree $\langle\, \mathcal{G} \,\rangle$ of $\mathcal{G}$ will have the shape

$\langle\, \mathcal{G} \,\rangle \quad =$



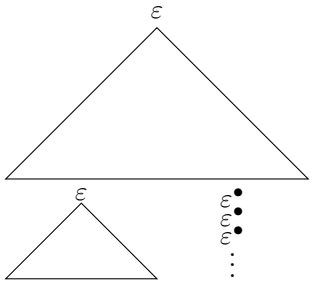
However, in $\mathcal{G}_\varepsilon$, each unproductive call to F will output a head symbol ε, so that $\langle\, \mathcal{G}_\varepsilon \,\rangle$ will be

$\langle\, \mathcal{G}_\varepsilon \,\rangle \quad =$



This tree does not contain the symbol $\bot$, so that $\mathcal{G}_\varepsilon$ is productive. By applying the construction of the global model-checking problem, we obtain G^φ_ε whose value tree is $\langle\, \mathcal{G}^\varphi_\varepsilon \,\rangle$:

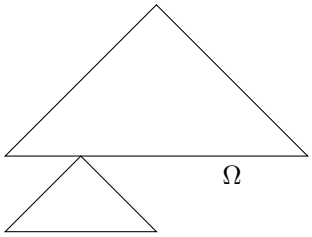
$\langle\, \mathcal{G}^\varphi_\varepsilon \,\rangle \quad =$



Since the marking only affects the symbols where φ is true, that is, from which we read an infinite sequence of ε, every marked node of this tree is precisely tracking an unproductive call to the recursion operator. Since this tree is

higher-order regular, it is enough to remove in a recursion scheme representing it every non-terminal call marked with $\varepsilon^\bullet$ to get rid of unproductive calls. By replacing them with the symbol Ω, we get the productive recursion scheme $\mathcal{G}_{prod}$, whose value tree is

$\langle\, \mathcal{G}_{prod} \,\rangle \quad =$



Chapter 4

Coinductive evaluation of infinitary λ-terms

Both higher-order recursion schemes and λY-terms are *finite* representations of infinitary objects, which they compute by successive finite approximations. In this chapter, we introduce a *coinductive* framework which encompasses them: we consider *infinitary* terms in §4.1, and their rewriting in §4.2. We also introduce an associated proof technique, *coinduction*, which has gained much attention from the λ-calculus community in the last years, for that it allows to manipulate infinitary objects without explicitly dealing with their finite approximations. We introduce a simple type system for infinitary terms, and prove its subject reduction in §4.3. We then discuss the relation with higher-order model-checking in §4.4.

4.1 Corecursive structures and infinitary λ-terms

The traditional approach in semantics, when it comes to infinite computation, is to consider them as the limit of their finite approximations. When the considered model is a domain, the notion of *continuity* is crucial: a continuous function between domains $\varphi : \mathcal{D} \to \mathcal{D}'$ is a monotonic function such that, for every directed family $D \subseteq \mathcal{D}$

$$f\left(\bigvee D\right) \quad = \quad \bigvee f(D) \tag{4.1}$$

Note that the monotonicity of f ensures that the family $f(D)$ is directed in $\mathcal{D}'$. A typical situation of continuity is the evaluation of higher-order recursion schemes in the domain $\mathit{Trees}_{\perp}(\Sigma)$ of partial productions: its set of finite approximations is enough to represent the infinite tree $\langle\, \mathcal{G} \,\rangle$. Conversely, a non-continuous behavior which is to be central in this thesis is the one of the parity condition of tree automata. Note that the execution of the alternating parity automaton itself is continuous, as it generates an infinite tree labeled with states and which can be described as the limit of a directed family of partial, finite runs. The non-continuous behavior is the one of the parity condition, which occurs after this continuous execution phase, and violates (4.1): no approximation of the execution tree suffices to determine whether it will be winning for the parity condition.

The infinite tree $\langle\, \mathcal{G} \,\rangle$ contains infinite branches, that is, *non-well founded*[1] branches, as they do not have leaves. The traditional approach uses well-founded trees to approximate a non-well founded one, and leads to reasoning over finite approximations and then on using continuity to extend the reasoning to the infinite tree of interest.

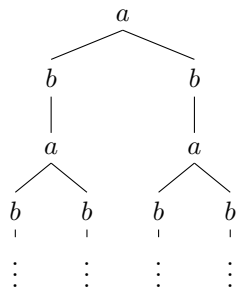
An alternative consideration of infinity comes with the notion of *coinduction*, over *corecursive* structures. As a powerful, infinitary proof technique, coinduction has received a lot of interest in the recent years, although it originated in Aczel's pioneering work on non-well founded sets [Acz88] in the late eighties. Important work on the subject is also due to Barr [Bar93]. For a historical overview of coinduction, see [San09].

Coinduction is dual to induction, and can be understood as the computation of a greatest fixpoint of a monotone endofunction f over the complete lattice $\mathcal{P}(A)$ obtained as the powerset of some set A. Recall from §2.3 that the least fixpoint $\mu X.\, f(X)$ can be computed starting from just one element – the empty set – and by iterating f over it, leading to bigger and bigger sets until the sequence eventually stabilizes. Dually, coinduction starts from the top element of $\mathcal{P}(A)$, that is, from the set of all subsets of A, and iteratively removes the ones which can not have been produced out of f, until the sequence stabilizes to $\nu X.\, f(X)$ – which can be an infinite set. Coinduction notably allows the emergence of a behavior that induction does not capture: $\nu X.\, f(X)$ may contain *self-justifying sets*, which can be understood as a form of *regular* infinite productions of the fixpoint computation. In general, coinduction offers firm foundations for the consideration of infinite objects.

Let us give an informal intuition of what coinduction allows, when it comes to infinite trees. Consider the signature

$$\Sigma \;=\; \{\, a : 2, b : 1 \,\}$$

and the Σ-labeled ranked regular tree

$T_{reg} \quad = \quad$ 

represented by the grammar

$$F \;=\; a \;\; (b \;\; F) \;\; (b \;\; F) \tag{4.2}$$

[1] We prefer this terminology to the semantically strong *ill-founded*. As it will appear in the sequel, there is no foundational problem with these trees, they just extend the usual, finite case to higher ordinal depths.

Denoting $\mathcal{T}$ the set of Σ-labeled ranked trees, this grammar represents a fixpoint of the function

$$\begin{array}{rcccl} f & : & \mathcal{T} & \to & \mathcal{T} \\ & & t & \mapsto & a\ (b\ t)\ (b\ t) \end{array}$$

which needs to contain *infinite branches*, and thus cannot be captured by an inductive construction. Dually, to compute an infinite tree such that $t = f(t)$, coinduction starts from the set of *all* trees $\mathcal{T}$. The first approximation of the fixpoint is then the set $f(\mathcal{T})$ of Σ-labeled ranked trees with tree prefix

$$a\ \ (b\ \ [\,]\,)\ \ (b\ \ [\,]\,)$$

and the second approximation is then $f(f(\mathcal{T}))$, containing trees with tree prefix

$$a\ \ (b\ \ (a\ \ (b\ \ [\,]\,)\ \ (b\ \ [\,]\,))\,)\ \ (b\ \ (a\ \ (b\ \ [\,]\,)\ \ (b\ \ [\,]\,))\,)$$

Coinduction iterates this tree discrimination until the limit ordinal ω, at which step only one solution remains, namely the Σ-labeled ranked tree T_{reg}. In general, the existence and unicity of the coinductive solution of a *guarded* equation as (4.2) — see Definition 29 — will follow from Theorem 8.

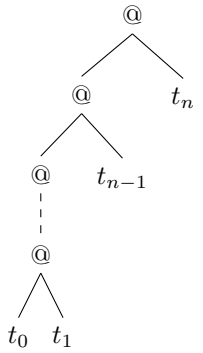
In order to ease the manipulation of coinductive structures, and the proofs over them, a subsequent work in establishing a theory of coinductive proof techniques has been done. The resulting techniques do not explicitly mention the ordinals used in the computation of the fixpoint, under some constraint of *productivity* – the result can be seen as a constrained form of transfinite induction, in which ordinal jumps do not require a particular treatment, and which notably hides conveniently questions of topological convergence to which productivity is deeply related, see for instance the introduction of [EP11]. In the sequel, we essentially follow the introduction on the subject by Czajka [Cza15] to provide the reader with elements of coinductive proof theory, and we give an alternate viewpoint on the λY-calculus, seen as a fragment of infinitary simply-typed λ-terms. Another interesting reference for the reader new to coinduction is [KS12].

Infinitary terms. Terms of the λY-calculus may indeed be understood as *infinitary* terms with a finite representation: given a term t of the λY-calculus, we define t^∞ as the *infinitary* λ-term obtained by an infinite iteration of the rule $\to_\delta$, without performing any substitution using $\to_\beta$. Similarly, a higher-order recursion scheme $\mathcal{G}$ produces an infinitary term $t^\infty(\mathcal{G})$ by unfolding its rules infinitely, without performing any substitution. Such infinitary terms will correspond to ranked trees of potentially countable depth, over the signature

$$\Sigma_{terms} \quad = \quad \Sigma \ \uplus\ \{\, x : \mathrm{ar}(x) \mid x \in \mathcal{V} \,\} \ \uplus\ \{\, \lambda x. : 1 \mid x \in \mathcal{V} \,\} \ \uplus\ \{\, @ : 1 \,\}$$

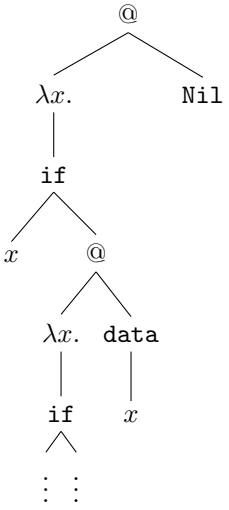
where Σ is the signature of constants of the term, $\mathcal{V}$ its set of variables, and @ is a distinguished application symbol used to mark explicitly the application of two terms: the application $t_0\ t_1$ is represented as

and more generally $t_0\ t_1\ \cdots\ t_n$ is represented as



For instance, the recursion scheme $\mathcal{G}_1$ of Example 5 corresponds to the infinite Σ_{terms}-labeled ranked tree

$t^\infty(\mathcal{G}_1)$ =



Following [Cza15], when working in a coinductive framework, we call *coterm* over the signature Σ a Σ-labeled ranked tree; we denote their set with $Coterms(\Sigma)$. The choice of this word stresses the fact that such infinite definitions can be obtained coinductively, that is, using greatest fixpoints of appropriate functions, see [Cza15] for details and for an extension of the notion of coterm to many-sorted signatures.

Example 7 (Streams)**.** The most canonical example of coinductive structure is probably *streams*: given a set A of actions, we consider the signature

$$\Sigma_A \quad = \quad \{\, a : 1 \mid a \in A \,\}$$

and define streams over A as Σ-labeled coterms. We denote their set $Streams(A)$. A stream over A can alternatively be understood as an infinite word of elements of A; it typically comes with two associated functions:

- the *head* function $\mathrm{hd} : Streams(A) \to A$, which maps a stream to the label of its root,

- and the *tail* function tail : $Streams(A) \to Streams(A)$, which *erases* the root and returns the resulting stream.

A stream s satisfies the equation

$$s \quad = \quad \mathrm{hd}(s) :: \mathrm{tail}(s)$$

where :: is the concatenation operator, adding a finite prefix to the beginning of a stream.

This example, in spite of its simplicity, illustrates a crucial fact:

induction is dual to coinduction

with the important consequence that while inductive data is *constructed* by *adding* symbols, coinductive data is *accessed* using *destructors*. Note that accessing a stream and removing its head symbol results in another stream: the infinitary nature of the object is preserved. Infinite trees should be thought of just in the same way – in a sense, they are just *branching streams*, whose head can be read and removed, resulting in a family of infinite trees.

Streams over A are defined by the equation

$$Streams(A) \quad = \quad A \times Streams(A) \tag{4.3}$$

and the existence and unicity of this set will follow from Theorem 8. Denoting the greatest fixpoint as ν, the type of streams is often denoted as

$$\nu X.\, A \times X$$

in the literature, which is another way to say that they are the greatest solution of (4.3).

Equality and bisimilarity. Equality of coterms can be defined using finite approximations: two coterms t and t' are equal if and only if they have the same domain, and that their restrictions to the same finite subdomain coincide. A coinductive characterization of equality can be expressed, via the notion of *bisimilarity* introduced in Definition 6.

Given a set S, a relation $\mathcal{R} \subseteq S^n$ is a *coinductive relation* if it is the greatest fixpoint of a monotone function $F_{\mathcal{R}} : \mathcal{P}(S) \to \mathcal{P}(S)$. We typically define such relations by a set of coinductive rules; for instance, the bisimilarity relation $\equiv$ on coterms over Σ is defined by the set of rules

$$\frac{t_i \;\equiv\; t'_i \quad \text{for every } i \in \{1, \ldots, n\}}{a(t_1, \ldots, t_n) \;\equiv\; a(t'_1, \ldots, t'_n)} \quad a \in \Sigma$$

Formally, it is defined as the greatest fixpoint of the monotone function

$$\begin{array}{rccl} F_{\equiv} : & Coterms(\Sigma)^2 & \to & Coterms(\Sigma)^2 \\ & T \times T' & \mapsto & \{\, (t,\, t') \in Coterms(\Sigma)^2 \mid \exists a \in \Sigma \;\; \forall i \in \{1, \ldots, n\} \\ & & & \qquad \exists (t_i,\, t'_i) \in T \times T' \quad t_i \;\equiv\; t'_i \\ & & & \qquad\qquad \wedge\; t \;=\; a(t_1, \ldots, t_n) \\ & & & \qquad\qquad \wedge\; t' \;=\; a(t'_1, \ldots, t'_n)\, \} \end{array}$$

Informally, two terms t and t' are bisimilar if there exists a *potentially infinite* derivation of $t \equiv t'$ using the family of rules defining $\equiv$. More generally, when defining a coinductive relation $\mathcal{R}$ using deduction rules, we have that $t, t' \in \mathcal{R}$ if and only if there exists a proof of potentially infinite depth of $t \mathcal{R} t'$.

Bisimulation is in fact an infinitary, coinductive presentation of equality:

Proposition 6. *Given a signature Σ and $t, t' \in Coterms(\Sigma)$, we have that $t = t'$ if and only if $t \equiv t'$.*

Cogrammars. The equation (4.3) suggests yet another presentation of streams, using a *coinductive* grammar

$$S \quad ::= \quad a_1 :: S \;\mid\; \cdots \;\mid\; a_n :: S$$

where $A = \{ a_1, \ldots, a_n \}$. More generally, given a signature

$$\Sigma \quad = \quad \{ a_i : \mathrm{ar}(a_i) \mid i \in I \}$$

the set of coterms t over Σ can be represented by the coinductive grammar

$$t \quad ::= \quad a_1 \underbrace{t \cdots t}_{\mathrm{ar}(a_1)} \;||\; \cdots \;||\; a_n \underbrace{t \cdots t}_{\mathrm{ar}(a_n)}$$

For instance, considering the signature $\Sigma_1 = \{ \texttt{if} : 2, \texttt{data} : 1, \texttt{Nil} : 0 \}$ of Example 1, the coinductive grammar

$$t \quad ::= \quad \texttt{if}\ t\ t \;||\; \texttt{data}\ t \;||\; \texttt{Nil}$$

generates a set of coterms which precisely correspond to Σ_1-labeled ranked trees, under the usual term-tree correspondence.

A cogrammar for infinitary λ-terms. Given a signature Σ providing a set of constants, and an *infinite* set of variables $\mathcal{V}$, *infinitary λ-terms* can be alternatively defined by the *cogrammar*

$$t, u \quad ::= \quad x \;||\; a \;||\; \lambda x.\, t \;||\; t\ u$$

where $x \in \mathcal{V}$ is a variable and $a \in \Sigma$ a constant. Note that this representation slightly differs from the one we previously gave for infinitary λ-terms, as we implicitly hide the application symbol @ in this representation. The result being equivalent, we will favor this representation in the sequel.

Regarding variables, usual notions of the λ-calculus are preserved by this infinitary extension, see [Cza15] for details:

- the notion of *free variable* is defined similarly,
- although it is not as straightforward, we may consider only α-renamed terms, that is, terms in which each variable name corresponds to a unique variable. This typically requires an uncountable set of variables $\mathcal{V}$.

The infinite rewriting system associated to these infinitary λ-terms is introduced in §4.2.

Corecursive definitions; productivity. In order to justify *corecursive* definitions such as (4.2), we need to check the *productivity* of the function whose greatest fixpoint is computed. A simple requirement is to restrict our attention to *guarded* functions:

Definition 29. Given a signature Σ, a function $f : Coterms(\Sigma)^n \to Coterms(\Sigma)$ is *constructor-guarded* if it can be decomposed as

$$f(t_1, \ldots, t_n) \quad = \quad a\left(g_1\left(t_1, \ldots, t_n\right), \ldots, g_{\mathrm{ar}(a)}\left(t_1, \ldots, t_n\right)\right)$$

where $a \in \Sigma$ is a constant, and for each $i \in \{1, \ldots, \mathrm{ar}(a)\}$:

- g_i is constructor-guarded,
- or g_i is a constant function,
- or g_i is a projection: there is $j \in \{1, \ldots, n\}$ such that

$$g_i\left(t_1, \ldots, t_n\right) \quad = \quad t_j$$

More generally, given a set S of parameters and a function $f : S \times Coterms(\Sigma)^n \to Coterms(\Sigma)$, we say that f is constructor-guarded if, for every $x \in S$, the function

$$f_x : (t_1, \ldots, t_n) \mapsto f(x, t_1, \ldots, t_n)$$

is constructor-guarded.

Such guard conditions ensure productivity; yet they are quite restrictive. More general classes of productive functions can be captured using for instance *sized types*, which are beyond the scope of this thesis, the idea being that some measure has to increase during the computation of the solution, in order to guarantee (topological) convergence to a unique solution. Constructor-guarded functions allow the coinductive definition of objects:

Definition 30 (Guarded corecursion)**.** A functional equation

$$\forall x \in S \quad f(x) \quad = \quad h\left(x, f(g_1(x)), \ldots, f(g_n(x))\right) \tag{4.4}$$

is said to be *guarded corecursive* if $f : S \to Coterms(\Sigma)$ and

- $h : S \times Coterms(\Sigma)^n \to Coterms(\Sigma)$ is a constructor-guarded function,
- $\forall i \in \{1, \ldots, n\}$, g_i is an endofunction over S.

A folklore theorem in the coalgebra community ensures the definition of such a function f:

Theorem 8. *Each guarded corecursive functional equation (4.4) has a unique solution $f : S \to Coterms(\Sigma)$.*

In a sense, these equations generalize the concept of *regularity* for infinite trees to all coterms. We can therefore see the λY-calculus as the *regular* fragment of the infinitary λ-calculus.

Example 8 (Corecursive functions over streams)**.** On $Streams(A)$, the guarded corecursive equation

$$even(t) \quad = \quad hd(t) :: even(tl(tl(t)))$$

defines a unique function *even*, mapping a stream to the stream obtained by keeping its first element, dropping its second, keeping its third, and so on. Considering the streams *one*, *zero* and *alt* defined by guarded corecursion:

$$one \quad = \quad 1 :: one$$

$$zero \quad = \quad 0 :: zero$$

$$alt \quad = \quad 1 :: 0 :: alt$$

we have that $even(alt) = one$, and $even(tl(alt)) = zero$. Another example is the function *zip*, again defined by guarded corecursion:

$$zip(x :: t, s) \quad = \quad x :: zip(s, t)$$

We see intuitively that

$$alt \quad = \quad zip(one, zero)$$

and that, for any stream t,

$$zip(even(t), even(tl(t))) \quad = \quad t$$

but we lack a proof principle to firmly establish it – this is where *coinductive proofs* will be useful.

To conclude this example, let us mention two more tricky corecursive "definitions" from [BH15]: first of all, considering the set $Streams(\mathbb{B})$ of streams over booleans, we naturally extend the negation operation to streams:

$$\neg(\, x :: s \,) \quad = \quad (\, \neg x \,) :: (\, \neg s \,)$$

where $x \in \mathbb{B}$ and $s \in Streams(\mathbb{B})$. We may then define the stream $alt_{\mathbb{B}}$ by

$$\begin{array}{rcl} hd(alt_{\mathbb{B}}) & = & \bot \\ tl(alt_{\mathbb{B}}) & = & \neg\, alt_{\mathbb{B}} \end{array}$$

This definition is not constructor-guarded, as the construction of the tail of the stream does not output a symbol, but refers directly to a function call on the stream itself. Yet it is productive: the reader may check that

$$alt_{\mathbb{B}} \quad = \quad \bot :: \top :: \bot :: \top :: alt_{\mathbb{B}}$$

since

$$hd(tl(alt_{\mathbb{B}})) \;=\; hd(\neg\, alt_{\mathbb{B}}) \;=\; \neg\, hd(alt_{\mathbb{B}}) \;=\; \neg \bot \;=\; \top$$

and so on: there are productive, yet non guarded objects. There also are unproductive equations, leading to ill-defined "streams": consider for instance the stream *evil* such that

$$\begin{array}{rcl} hd(evil) & = & \bot \\ tl(evil) & = & even(evil) \end{array}$$

Coinductive proofs. We now introduce the notion of *coinductive proof*, allowing us to prove statements on corecursively-generated objects. We will write these proofs in a quite informal style, which hides most of the underlying theoretic machinery. We follow in this way the approach of Czajka [Cza15], who provides firm foundations – using sized CPOs – for this seemingly informal style. As we said earlier, *coinductive proofs* allow, under some conditions, to reason on infinite objects defined as greatest fixpoints, without having to explicitly consider their approximations at all ordinal heights lesser than their closure ordinals. The resulting framework is very convenient and, even if it is not subtle enough to prove all properties on a general object defined as the greatest fixpoint of some productive endofunction, it will be sufficient for the purpose of this thesis.

Roughly speaking, a coinductive proof proceeds by supposing that the result we want to prove is true up to some ordinal α, and to prove that it holds for the next ordinal as well – but all that by keeping the ordinals implicit. Some productivity is required in the proof, in order to increase the ordinal we consider. Let us prove by coinduction that

$$zip(even(t),\, even(tl(t))) \quad = \quad t \tag{4.5}$$

Suppose that, for every stream t over A, the equation (4.5) holds. This will be our *coinductive hypothesis*. Consider then a given stream $x :: y :: t$, with $x, y \in A$; we have that

$$\begin{array}{lcll} zip(even(x :: y :: t),\, even(tl(x :: y :: t))) & = & zip(x :: even(tl(y :: t)),\, even(y :: t)) & \\ & = & x :: zip(even(y :: t),\, even(tl(y :: t))) & \\ & = & x :: y :: t & \\ & & \text{(by coinductive hypothesis)} & \end{array}$$

and the coinduction principle permits us to conclude that Equation (4.5) is true for every stream. Note two crucial facts, which will be required by coinductive proofs:

- the proof has to be *productive* in a sense: we need to call the coinductive hypothesis on a strictly lower ordinal than the depth of the coterm we want to prove the equation valid for. Here, the proof is made on the stream $x :: y :: t$, while the coinductive hypothesis is applied to $y :: t$, so that this condition is satisfied,
- we do not assume anything on the stream $y :: t$ we apply the coinductive hypothesis to – we would not be allowed, for instance, to perform case reasoning over it. In general, no assumption may be taken on the object the coinductive hypothesis is applied to.

4.2 Coinductive normalization of infinitary λ-terms

The normalization of infinitary λ-terms raises concerns about productivity again: while it seems quite natural to extend β-reduction and the other rewriting relations presented in §3.1 to this infinitary setting, one has to take care that all redexes are at *finite depth*: the reduction should not explore the tree

$$\frac{s \;\to^* a}{s \;\to^\infty a} \qquad \frac{s \;\to^* \lambda x.\, t \qquad t \;\to^\infty t'}{s \;\to^\infty \lambda x.\, t'}$$

$$\frac{s \;\to^* t_1\; t_2 \qquad t_1 \;\to^\infty t_1' \qquad t_2 \;\to^\infty t_2'}{s \;\to^\infty t_1'\; t_2'}$$

Figure 4.1: Coinductive closure $\to^\infty$ of a relation $\to$ on infinitary λ-terms.

infinitely without finding any. In fact, we define the β-reduction of infinitary terms *inductively*, simply by replacing finite terms with infinitary ones in Figure 3.1. In the same spirit, we extend the inductive definition of $\to_w$ and $\to_h$ to infinitary terms. Since these terms contain a potentially infinite amount of redexes, their reduction will require a *coinductive* use of $\to_\beta$. Interestingly enough, such constructions using coinductive calls of inductive constructions to enforce productivity appear in other areas of theoretical computer science, as in type theory for instance [MV05].

A first temptation is therefore to define the relations $\to_\beta^\infty$, $\to_w^\infty$ and $\to_h^\infty$ by replacing $\to$ in Figure 4.1 with each of them. This relation allows to compute coinductively the normal form of *productive* infinitary λ-terms, that is, of terms in which an inductive iteration of the β-reduction relation always outputs a symbol. To properly define an analogous of the Böhm reduction (Definition 18), we need to add the rule

$$\frac{t \text{ has no } \to\text{-normal form}}{t \;\to_\perp^\infty \perp}$$

and this defines $\to_{\beta\perp}^\infty$, $\to_{w\perp}^\infty$ and $\to_{h\perp}^\infty$. Just as in the λ-calculus, normal forms, when they exist, are computed by head reductions:

Theorem 9 ([EP11])**.** *Given two infinitary λ-terms t and t', we have that*

$$t \;\to_\beta^\infty t' \quad \textit{if and only if} \quad t \;\to_h^\infty t'$$

To capture the whole Böhm reduction, Czajka [Cza15] introduces the parallel coinductive head reduction $\to_N^\infty$:

$$\frac{t \;\to_h^* \lambda x_1 \cdots \lambda x_m.\, a\; t_1 \cdots t_n \qquad t_i \;\to_N^\infty t_i' \;\; (\forall i) \qquad a \not\equiv \perp}{t \;\to_N^\infty \lambda x_1 \cdots \lambda x_m.\, a\; t_1' \cdots t_n'}$$

$$\frac{t \text{ has no hnf}}{t \;\to_N^\infty \perp}$$

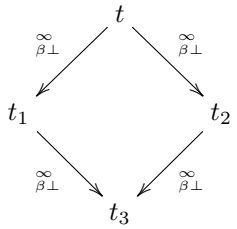
Note that the idea of the second rule, reducing terms without a head normal form to the divergence symbol $\perp$, was earlier considered in the setting of infinitary λ-calculus by Severi and de Vries [SdV02], who also obtained the theorems of confluence and of normalization we are about to state. However, their work relies on a presentation of reduction using transfinite sequences of ordinals, while Czajka's reformulation is purely coinductive.

Theorem 10 ([Cza15]). *Given two infinitary λ-terms t and t', we have that*

$$t \quad \to_{\beta\perp}^{\infty} \quad t' \quad \text{if and only if} \quad t \quad \to_{N}^{\infty} \quad t'$$

Confluence and normalization. The theorem of confluence of the (untyped) λ-calculus extends to the infinitary λ-calculus:

Theorem 11 ([Cza15]). *If t, t_1 and t_2 are three infinitary λ-terms such that $t \quad \to_{\beta\perp}^{\infty} \quad t_1$ and $t \quad \to_{\beta\perp}^{\infty} \quad t_2$, then there exists t_3 such that*



Infinitary λ-terms also normalize in some sense: there is a lambda-term without redexes, but potentially containing the additional divergence symbol $\perp$, to which they reduce. Note that this relation extends the Böhm reduction for the finitary λ-calculus, as it precisely computes the Böhm trees of finite λ-terms. More formally, considering that a $\beta\perp$-normal form is an infinitary term t to which the rewriting relation $\to_{\beta\perp}$ can not be applied, we have:

Theorem 12 ([Cza15]). *For every infinitary λ-term t, there exists a unique infinitary λ-term t_{nf} in $\beta\perp$-normal form such that $t \quad \to_{\beta\perp}^{\infty} \quad t_{nf}$.*

By Theorem 10, we can compute this normal form using the coinductive parallel head reduction $\to_N^{\infty}$.

4.3 Simply-typed infinitary terms

Proposition 5 enables us to consider higher-order recursion scheme as a particular kind of λY-terms, which in turn correspond to a subset of infinitary λ-terms: they are terms admitting a finite representation, and a simple type. In this section, we extend the simple type system of λ-calculus to infinitary terms, and prove coinductively subject reduction, as an illustration of this proof technique. We obtain the simple type system for infinitary λ-terms, both in its additive and multiplicative presentations, by considering a coinductive version of the systems of Figure 3.4 and of Figure 3.5. The use of coinduction allows to build proofs of infinite depth, which in turn implies that the contexts Γ occurring in derivations may contain infinitely many variables.

Note that there is an additional subtlety when considering the multiplicative presentation of the system: the spirit of this formulation is to track explicitly the introduction and usage of variables; for instance, the Weakening rule makes explicit the introduction of a variable the term we type does not actually use. When considering coinductive proofs, such variables may appear in the context without having been introduced by an Axiom nor by a Weakening rule. This point will not matter for our purpose in this current section. However, to restore this purely multiplicative spirit, one may add an additional *guard* condition to

coinductive proofs: for every variable occurring in a context, there should be at finite depth an Axiom or Weakening rule introducing it[2].

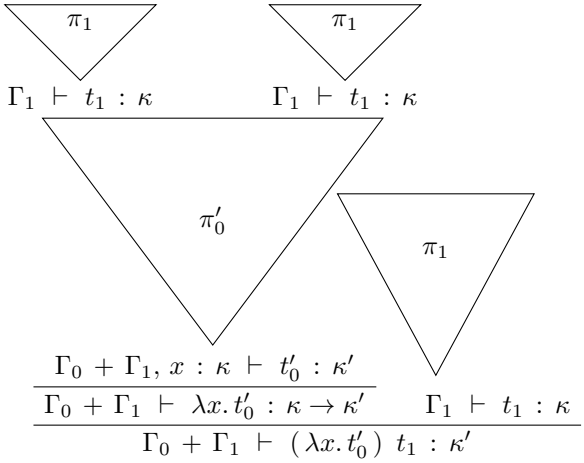
The type system enjoys subject reduction:

Proposition 7 (Subject reduction, infinitary version)**.** *Given two infinitary λ-terms t and t' such that $t \to_\beta^* t'$ or $t \to_\beta^\infty t'$, if there is a possibly infinite context Γ such that $\Gamma \vdash t : \kappa$, then $\Gamma \vdash t' : \kappa$.*

To prove this result, we first define the linear substitution in a proof of an occurrence of a variable $x : \kappa$ with a term $t_1 : \kappa$ in Figure 4.2. Given an enumeration of the occurrences of a variable x in a proof π of a sequent

$$\Gamma_0 + \Gamma_1 \vdash (\lambda x.\, t_0)\ t_1 : \kappa'$$

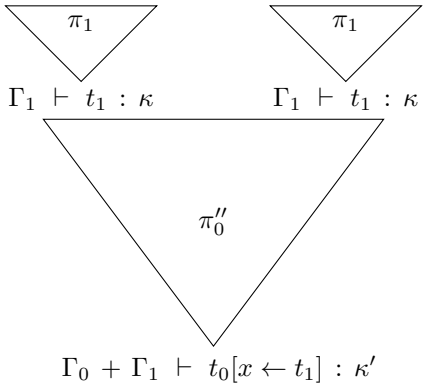
(where context addition was defined on p. 66), we define its *β-reduction* corecursively by applying in a first step a linear substitution to each occurrence, following the order provided by the enumeration of these occurrences. The proof obtained is of the form



where by construction

$$t'_0 \quad = \quad t_0[x \leftarrow t_1]$$

so that $x : \kappa$ appears in contexts of π'_0, but is never introduced by an Axiom rule. We therefore rewrite this proof to



[2]I would like to thank Pierre Vial for interesting discussions on this point.

where π_0'' is obtained from π_0' by removing from the context the useless variable x – recall that we only consider proofs in which each variable name corresponds to a unique variable, thanks to α-renaming of variables. This resulting proof is a typing proof of the sequent

$$\Gamma_0 + \Gamma_1 \vdash t_0[x \leftarrow t_1] : \kappa'$$

typing the term obtained from $(\lambda x.\, t_0)\ t_1$ by contracting its outermost redex.

The proof of the subject reduction property then follows from the (co)inductive application of this proof rewriting procedure. If $t \to_\beta^* t'$, where $\Gamma \vdash t : \kappa$, we suppose by induction that every sequence of length at most n of reductions preserves typing, and consider a reduction sequence of length $n+1$:

$$t_0 \quad \to_\beta \quad \cdots \quad t_n \quad \to_\beta \quad t_{n+1}$$

By induction hypothesis, we obtain a rewritten proof of typing of the sequent $\Gamma \vdash t_n : \kappa$, and an application of the proof rewriting procedure to the redex contracted by the reduction $t_n \to_\beta t_{n+1}$ allows to conclude.

If $t \to_\beta^\infty t'$, with $\Gamma \vdash t : \kappa$, the proof is by coinduction. We suppose that every reduction sequence preserves typing – implicitly, up to some ordinal length α. We then consider a reduction sequence, of implicit length $\alpha + 1$:

$$t \quad \to_\beta \quad t_1 \quad \to_\beta^\infty \quad t_\alpha$$

Since we have proved subject reduction for one step of β-reduction, and that $\Gamma \vdash t : \kappa$, we obtain $\Gamma \vdash t_1 : \kappa$. Since $t_1 \quad \to_\beta^\infty \quad t_\alpha$ is of length α, we can apply it the coinductive hypothesis, and conclude that $\Gamma \vdash t_\alpha : \kappa$.

By application of the coinduction principle, subject reduction holds for every coinductive rewriting sequence.

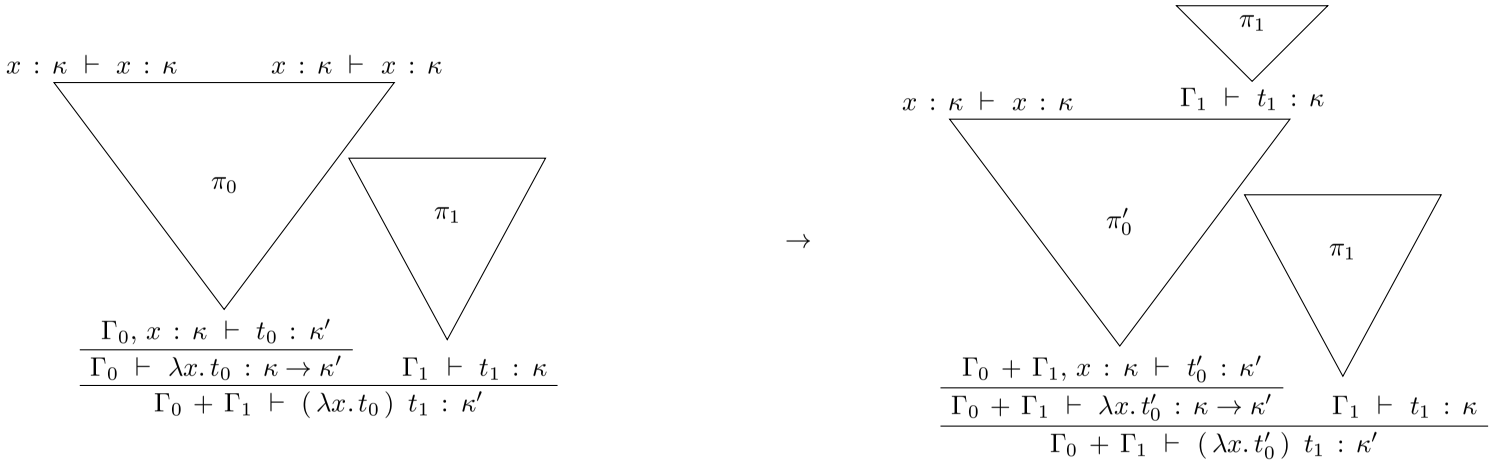
4.4 The higher-order model-checking problem, again

Proposition 5 enables us to consider higher-order recursion schemes as a particular kind of λY-terms, which in turn correspond to finitely presentable infinitary simply-typed λ-terms.

We may therefore compute their value trees no longer using the supremum of the partial productions of all finite reductions, but directly using infinitary, coinductive rewriting. Adapting the reduction $\to_N^\infty$ to the particular setting of *productive* recursion schemes, to which we may restrict thanks to the results of §3.5, we define the inductive weak head rewriting relation $\to_{\mathcal{G},w}$, and the coinductive rewriting relation $\to_{\mathcal{G}}^\infty$ in Figure 4.3. The relation is defined according to a precise recursion scheme $\mathcal{G}$, which is used to unfold the fixpoint calls. From the previous sections, we obtain:

Proposition 8. *The value tree $\langle\, \mathcal{G} \,\rangle$ of a productive recursion scheme $\mathcal{G}$ is its unique normal form for the coinductive rewriting relation $\to_{\mathcal{G}}^\infty$.*

Note that $\langle\, \mathcal{G} \,\rangle$ can be obtained equivalently as the unique normal form of $t^\infty(\mathcal{G})$ for the coinductive rewriting relation $\to_N^\infty$. An advantage of this presentation, compared to the traditional one recalled in §3.2, is that it allows coinductive reasoning on rewriting. We believe that this would allow,



In this figure, we depict the *linear* substitution of *one* occurrence of the variable x – the one occurring on the right of π_0. As the figure suggests, every other instance of x is left unchanged by this reduction.
The proof π'_0 is obtained by substituting *only* the rewritten occurrence of x with π_1. The corresponding term is denoted t'_0.

Figure 4.2: Substitution of *one* occurrence of a variable in an infinitary term.

$$\frac{}{(\lambda x.\, s)\ t \ \ \to_{\mathcal{G},w} \ \ s[x \leftarrow t]} \qquad \frac{s \ \ \to_{\mathcal{G},w} \ \ s'}{s\ t \ \ \to_{\mathcal{G},w} \ \ s'\ t} \qquad \frac{}{F \ \ \to_{\mathcal{G},w} \ \ \mathcal{R}(F)}$$

$$\frac{t \ \ \to^*_{\mathcal{G},w} \ \ a\ t_1 \cdots t_n \qquad t_i \ \ \to^\infty_{\mathcal{G}} \ \ t'_i \ \ (\forall i)}{t \ \ \to^\infty_{\mathcal{G}} \ \ a\ t'_1 \cdots t'_n}$$

Figure 4.3: Coinductive evaluation of productive higher-order recursion schemes.

as discussed in §6.5, to rephrase the proof of the soundness-and-completeness theorem presented in §6.3 in a purely coinductive setting, using coinductive subject reduction – and expansion, which will hold in this new setting – to relate properties of $\mathcal{G}$ with properties of $\langle\, \mathcal{G} \,\rangle$. This would also allow a natural extension of the theorem to infinitary λ-terms.

As a step towards such more expressive type systems, let us consider the *synchronization* of the evaluation of a higher-order recursion scheme with the execution of an alternating parity tree automaton over it. Indeed, we do not need to fully evaluate a HORS to its value tree before running an APT over it: every time the evaluation outputs a new head symbol, we may execute a transition of the APT over it. This leads to the coinductive definition of the relation $\to^\infty_{\mathcal{G},\mathcal{A}}$, based on the earlier-defined relation $\to_{\mathcal{G},w}$, and on the set of rules

$$\frac{t \ \ \to^*_{\mathcal{G},w} \ \ a\ t_1 \ \cdots \ t_n \qquad\qquad t_i : q_{i,j} \ \ \to^\infty_{\mathcal{G},\mathcal{A}} \ \ t'_i : q_{i,j}}{t : q \ \ \to^\infty_{\mathcal{G},\mathcal{A}} \ \ (a\ (t'_1 : (1, q_{1,1}))\ \cdots (t'_1 : (1, q_{1,k_1}))\ \cdots\ (t'_n : (n, q_{n,1}))\ \cdots\ (t'_n : (n, q_{n,k_n}))) : q}$$

for every conjunctive clause (recall p. 48)

$$\varphi \ \ = \ \ \bigwedge_{i \in \{1,\ldots,n\}} \ \bigwedge_{j \in \{1,\ldots,k_i\}} (i,\, q_{i,j})$$

of a transition $\delta(q,\, a)$ of the alternating parity automaton of interest. Note that there is non-determinism hidden in the definition of this coinductive rewriting relation, due to the choice of a conjunctive clause of the transition, but also that we needed to *parametrize* the terms *with a state*. Non-determinism is a hurdle to the definition of a unique notion of *normal form* for this reduction; however, generalizing the deterministic case, we call a normal form of $\to^\infty_{\mathcal{G},\mathcal{A}}$ an infinitary term which can not be further reduced.

Remark also that S being of ground type o by definition of $\mathcal{G}$, coinductive subject reduction implies that $\to^\infty_{\mathcal{G}}$ computes a normal form of ground type again – that is, a Σ-labeled ranked tree. Therefore $\to^\infty_{\mathcal{G},\mathcal{A}}$ computes a Σ-labeled *unranked* tree. We easily obtain the following proposition:

Proposition 9. *There is a correspondence between the run-trees of $\mathcal{A}$ over $\langle\, \mathcal{G} \,\rangle$ and the normal forms of $\to^\infty_{\mathcal{G},\mathcal{A}}$.*

This proposition suggests to refine the simple typing of terms with state annotations, to propagate in terms the behavior of alternating tree automata, and to relate the typing of a term with the one of its normal form using subject reduction and expansion. This will be the point of Part II.

Head reduction and Krivine machines. The reduction we presented is strongly related to the evaluation by Krivine machine [Kri07], which implements weak-head reduction of (untyped) λ-terms. An alternate way to understand $\to^{\infty}_{\mathcal{G},\mathcal{A}}$ would therefore be to introduce an infinitary version of the Krivine machine, dealing with *simply-typed* coterms, and with a notion of *internal state* storing the current state of the automaton we synchronize the machine with. The machine computes the branches of an execution of the automaton over the normal form of the term it evaluates: when a symbol is outputted by the head reduction, the machine chooses non-deterministically a conjunctive clause of the associated transition function, and then an external choice picks which direction of the branch should be explored.

This idea is very similar to the one developed by Salvati and Walukiewicz in [SW14]. Their approach differs by the fact that it is not formulated in a coinductive framework, and by the fact that it considers non-deterministic tree automata over binary trees — but this is not a restriction, as their expressiveness is the same as the one of alternating tree automata by Theorem 4.

Part II

Colored intersection types and higher-order model-checking

Chapter 5

Syntax and semantics of tree automata

As emphasized by Proposition 9, the computation of an execution tree of an alternating tree automaton $\mathcal{A}$ over the value tree of a *productive* higher-order recursion scheme $\mathcal{G}$ can be performed using the non-deterministic rewriting relation

$$\frac{t \;\to^*_{\mathcal{G},w}\; a\ t_1\ \cdots\ t_n \qquad\qquad t_i : q_{i,j} \;\to^\infty_{\mathcal{G},\mathcal{A}}\; t'_i : q_{i,j}}{t : q \;\to^\infty_{\mathcal{G},\mathcal{A}}\; (a\ (t'_1 : (1, q_{1,1}))\ \cdots (t'_1 : (1, q_{1,k_1}))\ \cdots\ (t'_n : (n, q_{n,1}))\ \cdots\ (t'_n : (n, q_{n,k_n}))) : q}$$

which requires to remember the current state of the computation, in addition to the term we rewrite. This suggests to *refine* the simple type of trees o with states of the automaton of interest. These refinements indicate from which states the normal form of a term of ground type is accepted, and this construction is then lifted to higher-order types. In addition to this refinement, we observe that some *duplication* – or *erasure* – of terms occurs during the rewriting, a fact which is reminiscent of what happens in *intersection type systems*. In such systems, we may give *several* refinements to a given simple type, and have for instance

$$\texttt{if} : \emptyset \to (q_0 \wedge q_1) \to q_0$$

whose application to arguments T_1 and T_2 leads to a typing proof

$$\dfrac{\dfrac{\dfrac{\delta\ \dfrac{}{\emptyset \vdash \texttt{if} : \emptyset \to (q_0 \wedge q_1) \to q_0}\quad \emptyset}{\text{App}\ \ \emptyset \vdash \texttt{if}\ T_1 : (q_0 \wedge q_1) \to q_0}\quad \dfrac{\vdots}{\Gamma_{21} \vdash T_2 : q_0}}{\text{App}\ \ \Gamma_{21}, \Gamma_{22} \vdash \texttt{if}\ T_1\ T_2 : q_0}\quad \dfrac{\vdots}{\Gamma_{22} \vdash T_2 : q_1}}{} \tag{5.1}$$

in which the first argument T_1 is *not* derived, while T_2 is derived *twice* – just as when an alternating tree automaton executes the transition function

$$\delta(q_0, \texttt{if}) \;=\; (2, q_0) \wedge (2, q_1)$$

on the tree $\texttt{if}\ T_1\ T_2$. This connection of tree automata with type systems originates in Aehlig's work on a finitary semantics for recursion schemes [Aeh06], in which the interpretation of a HORS describes the acceptation of its value

tree by a *trivial* tree automaton, that is, a Büchi tree automaton with trivial acceptance condition – a class strictly weaker than APT. Kobayashi then gave a type-theoretic presentation of Aehlig's semantics in [Kob09b], which was extended to an intersection type system accounting for APT, and thus for all MSO properties, by Kobayashi and Ong in [KO09].

The purpose of this thesis is to reinvestigate this line of work using the clarifying prism of *linear logic*. As a preliminary step to the next chapters, we explain in §5.1 how an intersection type system, differing from the one of Kobayashi by its *non-idempotency*, can be used to type a term in a way which reflects the behavior of a given alternating tree automaton over its normal form. After recalling the relational semantics of linear logic in §5.2, we use a bridge between them and non-idempotent intersection types, due to Bucciarelli and Ehrhard [BE00, BE01] and carried on further by de Carvalho [dC09], to restate in §5.3 this type-theoretic approach in the relational semantics. We obtain that the interpretation of a term of simple type contains a given state if and only if the tree it generates by normalization is accepted from this state by the alternating automaton of interest. Using a result of extensional collapse by Ehrhard, we give a similar theorem in the qualitative Scott semantics of linear logic in §5.4.

5.1 Refining simply-typed λ-calculus with non-idempotent intersection types

Refinement of simple types with intersection types. Given a finite set of states Q, we define intersection types by the grammar

$$\begin{array}{lcll} \theta & ::= & q \mid \tau \to \theta & (q \in Q) \\ \tau & ::= & \bigwedge_{i \in I} \theta_i & (I \text{ finite}) \end{array} \tag{5.2}$$

We usually keep implicit the parenthesis around the intersection operator. The intersection operator we consider here is not idempotent: we do not have $\sigma \wedge \sigma = \sigma$. As such, it may be understood as the representation of a *list* of elements indexed by a family I, which may appear several times, and in which the order is irrelevant. In other terms, these types can be represented as *multisets*, as we explain more in details in §5.3 – and not as sets, in which the elements do not have multiplicities. This differs from Kobayashi and Ong's approach, in which the intersection is idempotent, and corresponds to a set of elements. We discuss idempotency in §5.4.

Another difference with intersection type systems considered in semantics, notably for defining filter models, is that we do not allow a term to be typed as $q \wedge (q \to q)$ for instance: the terms we consider are simply-typed, and our intersection type system aims at *refining* this simple type. We therefore define the proper refinement relation :: by the rules of Figure 5.1; this relation ensures that all the types occurring in an intersection are of the same shape.

The intersection type system. We define contexts as finite sequences

$$\Gamma \quad = \quad x_1 : \bigwedge_{i_1 \in I_1} \theta_{1,i_1} :: \kappa_1, \ \ldots, \ x_n : \bigwedge_{i_n \in I_n} \theta_{n,i_n} :: \kappa_n$$

$$\frac{q \in Q}{q :: o} \qquad \frac{\tau :: \kappa_1 \qquad \theta :: \kappa_2}{\tau \to \theta :: \kappa_1 \to \kappa_2} \qquad \frac{\forall i \in I \quad \theta_i :: \kappa}{\bigwedge_{i \in I} \boxdot_{m_i} \theta_i :: \kappa}$$

Figure 5.1: The proper refinement relation ::

of variables together with their simple type and an intersection type refining this simple type. As earlier, we call the *domain* $dom(\Gamma)$ of a context Γ the set $\{x_1, \ldots, x_n\}$ of variable names occurring in it. We write

$$x : \bigwedge_{i \in I} \theta_i :: \kappa \quad \in \quad \Gamma$$

when x occurs in Γ with these simple and intersection types.

The *sum* $\Gamma_1 + \Gamma_2$ of two contexts is the context of domain $dom(\Gamma_1) \cup dom(\Gamma_2)$, such that for every $x \in dom(\Gamma_1) \cup dom(\Gamma_2)$:

- if $x \in dom(\Gamma_1)$ but $x \notin dom(\Gamma_2)$, x occurs with the same intersection and simple type in Γ_1 and in $\Gamma_1 + \Gamma_2$,
- if $x \in dom(\Gamma_2)$ but $x \notin dom(\Gamma_1)$, x occurs with the same intersection and simple type in Γ_2 and in $\Gamma_1 + \Gamma_2$,
- if $x \in dom(\Gamma_1) \cap dom(\Gamma_2)$, with $x : \bigwedge_{i \in I_1} \theta_i :: \kappa_1 \in \Gamma_1$ and $x : \bigwedge_{i \in I_2} \theta_i :: \kappa_2 \in \Gamma_2$
  - if $I_1 \cap I_2 \neq \emptyset$, or if $\kappa_1 \neq \kappa_2$, the operation is undefined,
  - else we set $\kappa = \kappa_1 = \kappa_2$, and

$$x : \bigwedge_{i \in I_1 \uplus I_2} \theta_i :: \kappa \quad \in \quad \Gamma_1 + \Gamma_2$$

When $\Gamma_1 + \Gamma_2$ is well-defined, we say that these contexts are *compatible*.

Note that compatibility is intrinsically related to the non-idempotency of the type system, which itself corresponds to the notion of *quantitativity*: there should be *precisely* one type in the context for each occurrence of an Axiom rule introducing a variable in the term we type.

Considering a variable

$$x : \bigwedge_{i \in I} \theta_i :: \kappa \quad \in \Gamma$$

the *subtraction* of $x : \theta_j$ from the context Γ, denoted $\Gamma - (x : \theta_j)$, is the context where all variables other than x keep the same type as in Γ, while x receives type

$$x : \bigwedge_{i \in I \setminus \{j\}} \theta_i :: \kappa$$

$$\text{Axiom}\quad \frac{}{x : \bigwedge_{\{i\}}\ \theta_i :: \kappa \vdash x : \theta_i :: \kappa} \qquad (x \in \mathcal{V} \cup \mathcal{N})$$

$$\delta\quad \frac{\{\,(i, q_{ij}) \mid 1 \leq i \leq \mathrm{ar}(a), 1 \leq j \leq k_i\}\ \text{ satisfies }\ \delta_A(q, a)}{\emptyset \vdash a : \bigwedge_{j=1}^{k_1}\ q_{1j} \to \ldots \to \bigwedge_{j=1}^{k_n}\ q_{nj} \to q :: o^n \to o} \qquad (a \in \Sigma)$$

$$\text{App}\quad \frac{\Gamma \vdash t : \big(\bigwedge_{i\in I}\ \theta_i\big) \to \theta' :: \kappa \to \kappa' \qquad \Gamma_i \vdash u : \theta_i :: \kappa \quad \text{for every } i \in I}{\Gamma + \sum_{i\in I} \Gamma_i \vdash t\ u : \theta' :: \kappa'}$$

$$\lambda\quad \frac{\Gamma, x : \bigwedge_{i\in I}\ \theta_i :: \kappa \vdash t : \theta' :: \kappa'}{\Gamma \vdash \lambda x\,.\,t : \big(\bigwedge_{i\in I}\ \theta_i\big) \to \theta' :: \kappa \to \kappa'}$$

Figure 5.2: The non-idempotent intersection type system $\mathcal{H}(\mathcal{A})$.

We extend naturally this operation to intersections: $\Gamma - (x : \bigwedge_{j\in J} \theta_j)$ contains $x : \bigwedge_{i\in I\setminus J}\ \theta_i :: \kappa$, and we extend similarly this operation to the subtraction of contexts.

Given an alternating tree automaton $\mathcal{A}$, the type system $\mathcal{H}(\mathcal{A})$ is defined inductively in Figure 5.2. The intersection types denoted with θ are required to match the shape they have in (5.2). In the Axiom rule, the context contains only x, with a *unique* type θ_i in the intersection. The index i can be chosen freely. In the δ rule, we use the notion of set satisfying a transition function from p. 47. Notice that the Application rule is of finite, yet variable width, since it depends on the size of the intersection type the argument is required to have; and that the use of the sum operator on contexts implicitly assumes that they are all compatible – a way to ensure compatibility being to introduce a *unique* index i in the intersection operator at each Axiom leaf.

Notice that the typing proof (5.1) becomes an example of derivation in this type system if we understand Γ_{21}, Γ_{22} as $\Gamma_{21} + \Gamma_{22}$ in its conclusion.

Typings and automata executions. Consider a simply-typed λ-term t of ground type o. By subject reduction (Proposition 3), its normal form t_{nf} is of type o as well. Since a normal form does not contain any redex and that t_{nf} is of ground type, it can not contain any λ. Suppose that t is closed, then t_{nf} does not contain variables, and is only an application of symbols of the signature of constants Σ – that is, a Σ-labeled tree, moreover ranked due to the simple typing constraints.

Now, if we consider a proof of typing of $t_{nf} = a\ t_1\ \cdots\ t_{\mathrm{ar}(a)}$ in the system $\mathcal{H}(\mathcal{A})$, its typical shape is as in Figure 5.3, where each π_{ij} is again of the same shape. Note that the contexts are empty since t_{nf} is closed. Using the usual correspondence between trees and terms over a same signature, we easily obtain that:

Proposition 10. *Given an alternating automaton $\mathcal{A}$ and a closed finite λ-term t_{nf} of simple type o and in normal form, there is a correspondence between the executions of $\mathcal{A}$ from the initial state q_0 over t_{nf} and the typings of*

$$\emptyset \vdash t_{nf} : q_0 :: o$$

in the intersection type system $\mathcal{H}(\mathcal{A})$.

Since we may now capture acceptation by a given alternating tree automaton as a type-theoretic property, it is relevant to study the preservation of the intersection types admitted by a term along reduction: this property would allow to deduce *directly from a term normalizing to a tree* the acceptation of this tree by an alternating automaton. The key properties are therefore *subject reduction* and *subject expansion*, which we prove right after defining a notion of rewriting on derivation trees.

Definition 31.

- Consider two simply-typed λ-terms $(\lambda x.\, t)\ u$ and $t[x \leftarrow u]$, and a proof π of conclusion $\Gamma \vdash (\lambda x.\, t)\ u : \tau :: \kappa'$. We define the proof π' of conclusion $\Gamma \vdash t[x \leftarrow u] : \tau :: \kappa'$ in Figure 5.4, and we say that $(\lambda x.\, t)\ u \to_\beta t[x \leftarrow u]$ *lifts* to $\pi \to_\beta \pi'$ at the level of typing derivations.
- Let t and t' be two simply-typed terms such that $t \to_\beta t'$, and let π be a proof of conclusion $\Gamma \vdash t : \tau :: \kappa'$. We extend the notion of proof rewriting by defining the proof π' of conclusion $\Gamma \vdash t' : \tau :: \kappa'$ obtained by applying the transformation of Figure 5.4 to the subproof of π whose conclusion is the redex contracted by $t \to_\beta t'$, leaving the remaining proof context unchanged, and we say as well in this more general situation that $t \to_\beta t'$ *lifts* to $\pi \to_\beta \pi'$ at the level of typing derivations.

By noticing that the lifting to the level of typing derivations of $t \to_\beta t'$ preserves their conclusion, and by applying inductively the process, we obtain:

Proposition 11 (Subject reduction)**.** *Let t and t' be simply-typed λ-terms such that $t \to_\beta^* t'$ and that $\Gamma \vdash t : \theta :: \kappa$. Then $\Gamma \vdash t' : \theta :: \kappa$.*

This allows to conclude that if $t : q_0 :: o$ in $\mathcal{H}(\mathcal{A})$, then its normal form is accepted by $\mathcal{A}$ from the initial state q_0. To obtain an equivalence, we need:

Proposition 12 (Subject expansion)**.** *Let t and t' be simply-typed λ-terms such that $t \to_\beta^* t'$ and that $\Gamma \vdash t' : \theta :: \kappa$. Then $\Gamma \vdash t : \theta :: \kappa$.*

This property follows from the definition of the proof rewriting relation $\to_\beta^{-1}$ in Figure 5.5, applied inductively on the redexes contracted by $t \to_\beta^* t'$. The essential difference between subject reduction and subject expansion is that we need in the latter to reconstruct the context for π_t. Thanks to the quantitativity of the system, this is easily done by context subtraction.

A consequence of these two results and of Proposition 10 is the following "preliminary" higher-order model-checking theorem, where we use the notion of acceptation from a state of p. 50:

Theorem 13. *Given a term t of simple type o, there is a proof in $\mathcal{H}(\mathcal{A})$ of the sequent*

$$\emptyset \vdash t : q_0 :: o$$

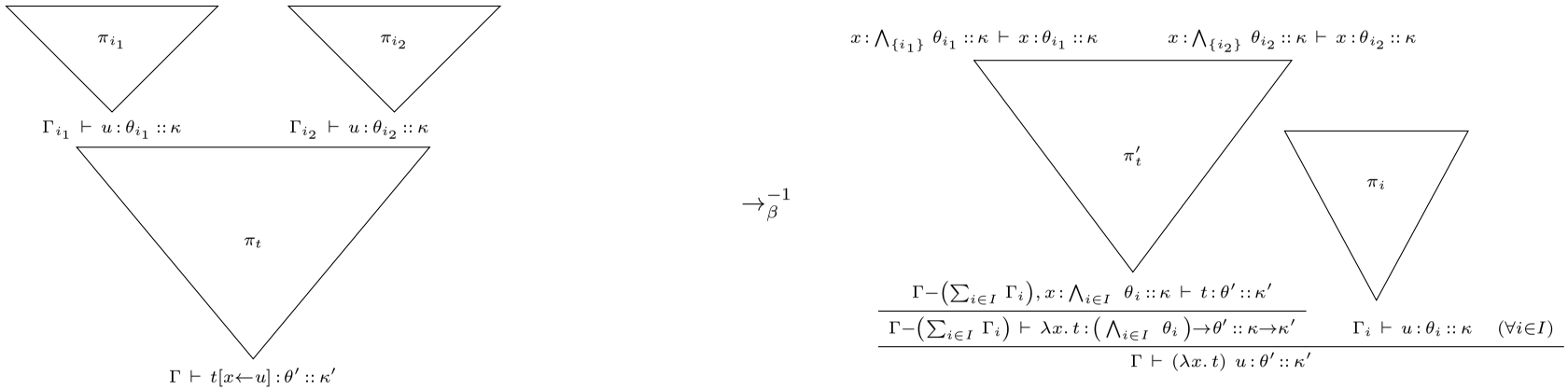
$$
\dfrac{\dfrac{\{(i,q_{ij}) \mid 1 \le i \le \mathrm{ar}(a), 1 \le j \le k_i\} \text{ satisfies } \delta_A(q,a)}{\emptyset \vdash a : \bigwedge_{j=1}^{k_1} q_{1j} \to \cdots \to \bigwedge_{j=1}^{k_n} q_{nj} \to q :: o^n \to o}\,\delta \qquad \begin{array}{c} \pi_{1j} \\ \vdots \\ \emptyset \vdash t_1 : q_{1j} :: o \ (\forall j \in \{1,\ldots,k_1\}) \end{array}}{\emptyset \vdash a\ t_1 : q :: o}\,\mathrm{App}
$$

$$
\dfrac{\begin{array}{c} \vdots \\ \emptyset \vdash a\ t_1 : q :: o \end{array} \qquad \begin{array}{c} \pi_{nj} \\ \vdots \\ \emptyset \vdash t_{\mathrm{ar}(a)} : q_{\mathrm{ar}(a)j} :: o \ (\forall j \in \{1,\ldots,k_{\mathrm{ar}(a)}\}) \end{array}}{\emptyset \vdash a\ t_1 \cdots t_{\mathrm{ar}(a)} : q :: o}\,\mathrm{App}
$$

Figure 5.3: Typical derivation of the normal form of a term of simple type o.

In this figure, we depict the substitution of the variable x occurring in t during a step of β-reduction at the level of typing derivations. The substitution process replaces the Axiom leaf $x : \bigwedge_{\{i\}} \theta_i :: \kappa \vdash x : \theta_i :: \kappa$ in π_t with the proof π_i, for every $i \in I$ – here we only depicted two such occurrences. Notice that since intersection types are non-idempotent, there is no duplication nor erasure of the proofs π_i: each one of them is substituted only once, precisely to the corresponding occurrence of x in π_t.
The proof π'_t is obtained by propagating for every $i \in I$ the context Γ_i from the former Axiom leaf $x : \bigwedge_{\{i\}} \theta_i :: \kappa \vdash x : \theta_i :: \kappa$ to the root of π_t, and by replacing the term t with $t[x \leftarrow u]$ – and similarly on its subterms.
This defines the rewriting relation $\pi \to_\beta \pi'$, which is extended to all terms t, t' such that $t \to_\beta t'$ by rewriting locally the redexes in typing proofs.

Figure 5.4: Lifting of $\to_\beta$ to typing trees.



In this figure, we depict the "de-substitution" of the variable x occurring in t during a reverse step of β-reduction at the level of typing derivations. The process reintroduces the Axiom leaves of t, and recomputes the proof π'_t by removing for every $i \in I$ the context Γ_i on the path from the Axiom leaf $x : \bigwedge_{\{i\}} \theta_i :: \kappa \vdash x : \theta_i :: \kappa$ to the root of π_t, and by replacing each occurrence of u in π_t with x. This defines the rewriting relation $\pi \rightarrow_\beta^{-1} \pi'$, which is extended to more general proofs by local rewriting.

Figure 5.5: Inversing $\rightarrow_\beta$ over typing trees: a step of subject expansion.

if and only if the tree generated by the normalization of t is accepted from q_0 by the alternating tree automaton $\mathcal{A}$.

Thanks to this theorem, we no longer need to reduce a term to determine whether the tree it generates is accepted by a given tree automaton: we just need to *type* it with the initial state of the automaton.

5.2 The relational model of linear logic

As we explain in §5.3, the non-idempotency of the intersection operator connects it to the interpretation of the λ-calculus in the relational semantics of linear logic. Before that, we find useful to briefly recall the definition of the relational model of linear logic, and of a Seely category, which is one of the possible axiomatizations of a model of linear logic, see [Mel09] for details.

The category *Rel* is defined as the category with finite or countable sets as objects, and with binary relations between A and B as morphisms $A \to B$. The category *Rel* is symmetric monoidal closed, with tensor product defined as (set-theoretic) cartesian product, and tensorial unit defined as singleton:

$$A \otimes B \quad = \quad A \times B \qquad\qquad 1 \quad = \quad \{\star\}.$$

In other words, the notion of tensor in the category of relations meets the one of cartesian product of sets. As such, its unit is (up to isomorphism) the singleton set $1 = \{\star\}$. Its internal hom (also called linear implication) $X \multimap Y$ simply defined as $X \otimes Y$, the evaluation morphism being

$$ev \quad = \quad \{(((a,\, b),\, a),\, b) \mid (a,\, b) \in X \times Y\} \in Rel((X \multimap Y) \times X, Y)$$

Since the object $\bot = 1 = \{\star\}$ is dualizing, the category *Rel* is moreover $*$-autonomous. The category *Rel* has also finite products defined as

$$A \& B \qquad = \qquad \{(1, a) \mid a \in A\} \cup \{(2, b) \mid b \in B\}$$

with the empty set as terminal object $\top$. As in any category with finite products, there is a diagonal morphism $\Delta_A : A \to A \,\&\, A$ for every object A, defined as

$$\Delta_A \quad = \quad \{(a,\, (i,\, a)) \mid i \in \{1,\, 2\} \text{ and } a \in A\}$$

Note that the category *Rel* has finite sums as well, since the negation $A^\bot = A \multimap \bot$ of any object A is isomorphic to the object A itself. Note that this degeneracy can be addressed: *Rel* can be turned into a model where dual connectives are no longer isomorphic, see [BE01] for details. All this makes *Rel* a model of multiplicative additive linear logic. In order to establish that it defines a model of propositional linear logic, we find convenient to check that it satisfies the axioms of a Seely category, as originally axiomatized by Seely [See89] and then revisited by Bierman [Bie95], see the survey [Mel09] for details.

Definition 32 (Seely category)**.** A *Seely category* is defined as a symmetric monoidal closed category $(\mathscr{L}, \otimes, 1)$ with binary products $A \,\&\, B$, a terminal object $\top$, and:

1. a comonad $(!, \mathbf{dig}, \mathbf{der})$,
2. two natural isomorphisms

$$m^2_{A,B} \quad : \quad !A \otimes !B \cong !(A \,\&\, B) \qquad\qquad m^0 \quad : \quad 1 \cong !\top$$

making

$$(!, m) \quad : \quad (\mathscr{L}, \&, \top) \quad \longrightarrow \quad (\mathscr{L}, \otimes, 1)$$

a symmetric monoidal functor.

One also asks that the coherence diagram

$$\begin{array}{ccc} !A \otimes !B & \xrightarrow{\quad m \quad} & !(A \,\&\, B) \\ \Big\downarrow{\scriptstyle \mathbf{dig}_A \otimes \mathbf{dig}_B} & & \Big\downarrow{\scriptstyle \mathbf{dig}_{A\&B}} \\ !!A \otimes !!B & & !!(A \,\&\, B) \\ & & \Big\downarrow{\scriptstyle !\langle !\pi_1, !\pi_2\rangle} \\ !!A \otimes !!B & \xrightarrow{\quad m \quad} & !(!A \& !B) \end{array} \tag{5.3}$$

commutes in the category $\mathscr{L}$ for all objects A and B, and that the four following diagrams expressing the fact that the functor $(!, m)$ is symmetric monoidal:

$$\begin{array}{ccc} (!A \otimes !B) \otimes !C & \xrightarrow{\quad \alpha \quad} & !A \otimes (!B \otimes !C) \\ \Big\downarrow{\scriptstyle m \otimes !C} & & \Big\downarrow{\scriptstyle !A \otimes m} \\ !(A \,\&\, B) \otimes !C & & !A \otimes !(B \,\&\, C) \\ \Big\downarrow{\scriptstyle m} & & \Big\downarrow{\scriptstyle m} \\ !((A \,\&\, B) \,\&\, C) & \xrightarrow{\quad !\alpha \quad} & !(A \,\&\, (B \,\&\, C)) \end{array} \tag{5.4}$$

$$\begin{array}{ccc} !A \otimes 1 & \xrightarrow{\quad \rho \quad} & !A \\ \Big\downarrow{\scriptstyle !A \otimes m} & & \Big\uparrow{\scriptstyle !\rho} \\ !A \otimes !\top & \xrightarrow{\quad m \quad} & !(A \,\&\, \top) \end{array} \qquad\qquad \begin{array}{ccc} 1 \otimes !B & \xrightarrow{\quad \lambda \quad} & !B \\ \Big\downarrow{\scriptstyle m \otimes !B} & & \Big\uparrow{\scriptstyle !\lambda} \\ !\top \otimes !B & \xrightarrow{\quad m \quad} & !(\top \,\&\, B) \end{array} \tag{5.5}$$

$$\begin{array}{ccc} !A \otimes !B & \xrightarrow{\quad \gamma \quad} & !B \otimes !A \\ \Big\downarrow{\scriptstyle m} & & \Big\downarrow{\scriptstyle m} \\ !(A \,\&\, B) & \xrightarrow{\quad !\gamma \quad} & !(B \,\&\, A) \end{array} \tag{5.6}$$

commute in the category $\mathscr{L}$ for all objects A, B and C.

To define *Rel* as a Seely category, recall that a *finite multiset* over a set A is a (set-theoretic) function $w : A \to \mathbb{N}$ with finite support, where the support of w is the set of elements of A whose image is not equal to 0. Alternatively, it can be seen as a finite sequence $[a_1, \cdots, a_n]$ of elements of A up to commutation of elements (it is in fact the abelianized of the sequence). The addition of two

finite multisets m_1 and m_2 can equivalently be seen as their pointwise addition as functions, or as the finite result of the concatenation of both sequences. Denote $\mathcal{M}_{fin}(A)$ the set of finite multisets of elements of A. The functor $! : Rel \to Rel$ is defined as

$$\begin{array}{lcl} !\,A & = & \mathcal{M}_{fin}(A) \\ !\,f & = & \{([a_1, \cdots, a_n], [b_1, \cdots, b_n]) \mid \forall i, (a_i, b_i) \in f\} \end{array}$$

The comultiplication and counit of the comonad are defined as the digging and dereliction morphisms below:

$$\begin{array}{lcl} \mathbf{dig}_A & = & \{(w_1 + \cdots + w_k, [w_1, \cdots, w_k]) \mid \forall i, w_i \in !\,A\} \ \in \ Rel(!A, !!A) \\ \mathbf{der}_A & = & \{([a], a) \mid a \in A\} \ \in \ Rel(!A, A) \end{array}$$

In order to define a Seely category, one also needs the family of isomorphisms

$$\begin{array}{lcccl} m^0 & : & 1 & \longrightarrow & !\top \\ m^2_{A,B} & : & !\,A \otimes !\,B & \longrightarrow & !\,(\,A \,\&\, B\,) \end{array}$$

which are defined as $m^0 = \{(\star, [])\}$ and

$$m^2_{A,B} \quad = \quad \{(([a_1, \cdots, a_m], [b_1, \cdots, b_n]), [(1, a_1), \cdots, (1, a_m), (2, b_1), \cdots, (2, b_n)])\}$$

One then carefully checks that the coherence diagrams expected of a Seely category commute. From this follows that

Proposition 13. *The category Rel together with the finite multiset interpretation of the exponential modality ! defines a model of propositional linear logic.*

5.3 Non-idempotent intersection types and the relational semantics of linear logic

The relational semantics of linear logic gives birth to a denotational model of the λ-calculus by the *Kleisli construction*, which is in a sense the semantic counterpart to the linear decomposition of the intuitionnistic arrow

$$A \ \Rightarrow \ B \quad = \quad !\,A \ \multimap \ B$$

This decomposition is a fundamental property of linear logic, and means that every intuitionnistic arrow building a formula $A \Rightarrow B$ can be understood as a two-step operation

- which performs in a first time *duplications* or *erasures* of its argument A: this is the effect of the modality ! on the formula A,
- followed by a *linear* use of each element of $!\,A$: each copy made in the previous step is used only once in the program of type $!\,A \ \multimap \ B$ we consider.

In the relational semantics, the Kleisli construction accordingly leads to the interpretation of each term of simple type $\kappa \to \kappa'$ as a subset of the interpretation

$$[\![\kappa \to \kappa']\!] \quad = \quad \mathcal{M}_{fin}([\![\kappa]\!]) \times [\![\kappa']\!]$$

of its type, in which the duplication step is interpreted as the finite multiset construction $\mathcal{M}_{fin}$.

By interpreting the ground type o as $[\![o]\!] = Q$, we obtain for instance that the binary symbol

$$\texttt{if} \ :: \ o \to o \to o$$

is interpreted as a subset

$$[\![\texttt{if}]\!] \ \subseteq \ \mathcal{M}_{fin}(Q) \times \mathcal{M}_{fin}(Q) \times Q$$

The idea is that every element

$$([q_1, \ldots, q_n], [q'_1, \ldots, q'_m], q'') \quad \in \quad [\![\texttt{if}]\!]$$

describes the fact that the application of `if` to a term T_1 accepted from each state q_i and to another term T_2 accepted from each q'_j gives a term `if` T_1 T_2 accepted from q''. The *alternating* transition

$$\delta(q_0, \texttt{if}) \ = \ (2, q_0) \wedge (2, q_1)$$

can accordingly be reflected in the semantics by setting that

$$([\,], [q_0, \, q_1], q_0) \quad \in \quad [\![\texttt{if}]\!]$$

where $[\,]$ is the empty multiset. More generally, given an alternating automaton $\mathcal{A}$ *without parity condition*, we consider an interpretation of λ-terms over a signature of constants Σ such that, for every constant a of arity n of Σ,

$$[\![a]\!] \ = \ \bigcup \{ ([q_{11}, \ldots, q_{1k_1}], \ldots, [q_{n1}, \ldots, q_{nk_n}], \, q) \ \mid \ \{ (i, q_{ij}) \mid 1 \leq i \leq n, 1 \leq j \leq k_i \} \ \text{ satisfies } \ \delta_A(q, a) \}$$

Recall that the non-idempotent type system $\mathcal{H}(\mathcal{A})$ introduced in §5.1 uses the rule δ to provide intersection types to the constants of Σ in a very similar way: the only two differences between

$$([q_{11}, \ldots, q_{1k_1}], \ldots, \, [q_{n1}, \ldots, q_{nk_n}], \, q) \quad \in \quad [\![a]\!]$$

and

$$a \ : \ \bigwedge_{j=1}^{k_1} q_{1j} \ \to \ \ldots \ \to \ \bigwedge_{j=1}^{k_n} q_{nj} \to q$$

lie in the fact that we use a multiset notation in the first case, and a non-idempotent intersection in the second, and in the fact that a tuple is used in the semantics to represent a succession of arrows in types.

This observation extends to a more general connection between the denotations of λ-terms in the relational semantics and associated non-idempotent intersection type systems. In particular, the interpretation of terms we just

defined in the relational semantics maps a closed term to a set isomorphic to the set of the non-idempotent intersection types it admits in the system $\mathcal{H}(\mathcal{A})$:

$$[\![t]\!] \;\cong\; \{\, \theta \mid \emptyset \vdash t : \theta :: \kappa \,\}$$

where κ is the simple type of t. A similar result holds for terms with free variables, by reducing to the case of closed terms by λ-abstraction.

This connection between the relational model of linear logic and non-idempotent intersection types originates in Bucciarelli and Ehrhard's work on *indexed linear logic* [BE00, BE01], in which the exponential modality is constrained so as to reflect the denotational interpretation of terms. Indeed, the relational semantics contains more than just interpretations of λ-terms: for instance, suppose that $([q_0], [q_1], q_0) \in [\![a]\!]$, and that

$$([q_0, q_1], q_0) \;\in\; [\![\lambda x.a\ x\ x]\!] \;\subseteq\; [\![o \to o]\!] \tag{5.7}$$

is an element of the relational semantics of $\lambda x.a\ x\ x$. From two different terms t_0 and t_1 such that $q_0 \in [\![t_0]\!]$ and $q_1 \in [\![t_1]\!]$, we can form a multiset $[q_0, q_1]$ and compose it with (5.7), even if this does not correspond to the fact that the substitution of x in the λ-calculus shall be performed twice with the *same* term. With this respect, the relational model corresponds to a wider calculus named *λ-calculus with resources*, defined for instance in [Ehr14]. The indexation incorporated to the exponential modality in indexed linear logic is a way to ensure that the denotations used to build a multiset come from the same term - or, equivalently, from a proof of the same structure. The idea can be understood from the rule for Application in intersection type systems: in

$$\frac{\Gamma \vdash t : \left(\bigwedge_{i\in I} \theta_i\right) \to \theta' :: \kappa \to \kappa' \qquad \Gamma_i \vdash u : \theta_i :: \kappa \quad \text{for every } i \in I}{\Gamma + \sum_{i\in I} \Gamma_i \vdash t\ u : \theta' :: \kappa'}$$

we open a new proof typing u for each type it should have. In indexed linear logic, the idea is similarly to superpose the family of proofs of typing for u to a unique proof indexed by the set I. The connection between relational semantics and non-idempotent intersection types, inspired by indexed linear logic, was carried on further by de Carvalho [dC09] to study the complexity of reduction of λ-terms. The relation between the intersection type system of Kobayashi for higher-order model-checking, the type system $\mathcal{H}(\mathcal{A})$ and indexed linear logic is detailed by Grellois and Melliès in [GM15b].

Now that we explained how the interpretation of λ-terms in the relational model of linear logic precisely corresponds to the computation of its set of intersection types in $\mathcal{H}(\mathcal{A})$, we find useful to point out that a semantic counterpart to Theorem 13 is provided directly by the fact that *Rel* is *denotational*:

$$\text{if } t \to_\beta t' \text{ , then } [\![t]\!] = [\![t']\!]$$

This stability of the semantics under β-reduction is a counterpart to the type-theoretic properties of subject reduction and expansion. We therefore obtain:

Theorem 14. *Given a closed term t of simple type o, we have that*

$$q_0 \in [\![t]\!]$$

in the relational semantics of linear logic if and only if the tree generated by the normalization of t is accepted from q_0 by the alternating tree automaton $\mathcal{A}$.

5.4 Idempotent types and extensional collapses

Non-idempotent intersection types have several advantages:

- they connect naturally to the interpretation of λ-terms in the relational semantics of linear logic, that are in a sense its most canonical semantics,
- this connection does not require subtyping,
- and it is particularly clear to prove subject expansion and reduction, as it is just a linear manipulation of subproofs, since each occurrence of a variable corresponds precisely to such a subproof.

However, from a more practical point of view, to design algorithms as well as to prove decidability of higher-order model-checking problems, the precise account of the multiplicity of uses of a type in an intersection is a hurdle. The set of intersection types refining a given simple type is indeed infinite. An alternative is to switch to *idempotent* intersection types, in which the equation

$$\sigma \wedge \sigma \quad = \quad \sigma$$

holds. More generally, intersections are considered modulo surjective reindexing: for every surjection $f : J \twoheadrightarrow I$,

$$\bigwedge_{j \in J} \sigma_{f(j)} \quad = \quad \bigwedge_{i \in I} \sigma_i \tag{5.8}$$

This amounts to considering the intersection operator as the representation of a set of intersection types:

$$x : \bigwedge_{i \in I} \theta_i :: \kappa \quad \cong \quad \{ x : \theta_i :: \kappa \mid i \in I \} \tag{5.9}$$

instead of the multiset-based representation related to non-idempotent types. In the same spirit, contexts correspond to sets obtained as unions of such sets of refined typings. From this definition of contexts of idempotent types as sets follows immediately the definition of the union

$$\Gamma_1 \cup \Gamma_2$$

of two contexts, as the union of the sets they represent. Notably, if x occurs in Γ with the idempotent intersection type $\bigwedge_{i \in I} \theta_i$, and in Γ' with the intersection type $\bigwedge_{j \in J} \theta'_j$ refining the same simple type, then it appears in $\Gamma \cup \Gamma'$ with the intersection type corresponding to the set

$$\{ x : \theta_i :: \kappa \mid i \in I \} \cup \{ x : \theta'_j :: \kappa \mid j \in J \}$$

modulo the identification (5.9). The notion of belonging to a context is also derived from this set-theoretic interpretation: we say that $x : \theta \in \Gamma$ if and only if x appears in Γ with the idempotent intersection type $\bigwedge_{i \in I} \theta_i$ and that there exists $i \in I$ such that $\theta = \theta_i$. The inclusion of contexts is defined accordingly. We sometimes forget the simple type in contexts when it makes the reading clearer, or the intersection operator when it is taken over a singleton set: in

this case, we may write $x : \theta :: \kappa$ for $x : \bigwedge_{\{1\}} \theta :: \kappa$.

As pointed out by Ehrhard in [Ehr12a] and by Terui in [Ter12], *idempotent* intersection types are a type-theoretic counterpart to the interpretation of the λ-calculus in the Scott model of linear logic, if we consider a type system with *subtyping*. As in the non-idempotent case, the set of intersection types a term admits in the associated idempotent type system is isomorphic to the set of elements appearing in the semantics of the term. We recall the Scott model of linear logic, before connecting it to the relational semantics. We do not make the idempotent intersection type system of [Ter12] explicit here, but we will present it extended with a coloring modality in §10.2.

5.4.1 The Scott model of linear logic

We present here the finitary Scott semantics of linear logic, where formulas are interpreted as partial orders. The resulting semantics of linear logic is *qualitative* in the technical sense that its exponential modality ! is interpreted using the finite *powerset* construction, which transports finite sets into finite sets, in contrast to the finite multiset construction used in the traditional and *quantitative* relational semantics. The terminology of Scott semantics comes from the fact that in the derived semantics of the simply-typed λ-calculus, every type is interpreted as a prime algebraic complete lattice, and every simply-typed λ-term as a Scott-continuous function.

However, we find convenient to develop here the formulation of this model of linear logic due to Winskel [Win98], and obtained from the former by Stone duality, see for example [Ter12]. So, let **ScottL** denote the category with preorders $\mathbf{A} = (A, \leq_A)$ as objects and downward-closed binary relations $R \subseteq A \times B$ as morphisms $(A, \leq_A) \to (B, \leq_B)$. Here, by a downward-closed relation, we mean a binary relation R such that for all $a, a' \in A$ and $b, b' \in B$, one has :

$$(a, b) \in R \quad \text{and} \quad a \leq_A a' \quad \text{and} \quad b' \leq_B b \qquad \Rightarrow \qquad (a', b') \in R.$$

The binary relation R is thus downward closed in the partial order $(A, \leq_A)^{op} \times (B, \leq_B)$ interpreting the formula $(A, \leq_A) \multimap (B, \leq_B)$ in the Scott semantics. The intuition guiding this property is that if a binary relation R interpreting a proof of linear logic can produce an output b from an input a, then the same binary relation can also produce a less informative output b' from a more informative input a'. It is well-known in the literature on linear logic that this "saturation property" is essential in order to obtain a relational semantics of linear logic with a qualitative (that is, based on finite sets instead of finite multisets) interpretation of the exponential modality. This remark is generally attributed to Ehrhard, see [Mel09] for details. When considered type-theoretically, this downward closure corresponds to the subtyping rule, see [Ter12] for details. The composition in **ScottL** is relational, since relational composition preserves the property of being downward-closed. The identity morphism over $(A, \leq_A)$ is

$$id_A \quad = \quad \{ (a', a) \mid a \leq_A a' \}$$

ScottL is a compact closed category with products, with

$$\begin{array}{rcl@{\qquad}rcl} (A, \leq_A) \otimes (B, \leq_B) & = & (A \times B, \leq_A \times \leq_B) & 1 & = & (\{\star\}, =) \\ (A, \leq_A) \,\&\, (B, \leq_B) & = & (A \uplus B, \leq_A \uplus \leq_B) & \top & = & (\emptyset, \emptyset) \\ (A, \leq_A)^{\perp} & = & (A, \geq_A) & & & \end{array}$$

The exponential modality

$$! \;:\; A \;\mapsto\; !A \;:\; \mathbf{ScottL} \;\longrightarrow\; \mathbf{ScottL}$$

is then defined by associating to the ordered set $(A, \leq_A)$ the set $\mathcal{P}_{fin}(A)$ of finite subsets of A, where two finite subsets u and v are ordered in the following way:

$$u \leq_{!A} v \qquad \Longleftrightarrow \qquad \forall a \in u,\, \exists b \in v, \quad a \leq_A b.$$

The endofunctor ! transports every morphism $R : A \to B$ of the category **ScottL** to the following morphism:

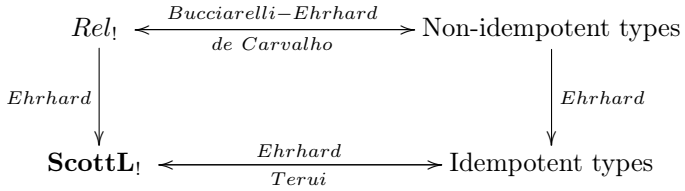
$$!R \;=\; \{\, (u, v) \;\in !A \times !B \;\mid\; \forall b \in v \;\; \exists a \in u \;\; (a, b) \in R \,\} \;:\; !A \to !B$$

The endofunctor ! is in fact a comonad and defines a Seely category, so that:

Proposition 14. *The category **ScottL** together with the finite powerset interpretation of the exponential modality* ! *defines a model of propositional linear logic.*

5.4.2 Extensional collapse and idempotency

The Scott model of linear logic is deeply related to its relational model, as Ehrhard proved that the former is the *extensional collapse* of the latter [Ehr12b]. In a sense, quotienting the relational model by extensionality makes the precise count of multiplicities disappear, as this is an intrinsically internal property of the term: a context can not observe how many times a program uses the arguments it provides it (of course, the contexts considered here are without effects). If we focus on the models of the λ-calculus obtained from these two models of linear logic, we can translate this result to an equivalent type-theoretic setting, which is explicited by Ehrhard in [Ehr12a] (essentially in his Theorem 18), and obtain a correspondence between the non-idempotent type system $\mathcal{H}(\mathcal{A})$ and its idempotent counterpart, enriched with subtyping. The general picture we obtain is:



where the categories $Rel_!$ and $\mathbf{ScottL}_!$ are the models of the λ-calculus induced by the relational and Scott models of linear logic.

A consequence of Ehrhard's extensional collapse result is the following counterpart to Theorem 14:

Theorem 15. *Given a term t of simple type o, we have that*

$$q_0 \in [\![t]\!]$$

in the Scott semantics of linear logic if and only if the tree generated by the normalization of t is accepted from q_0 by the alternating tree automaton $\mathcal{A}$.

This relation between a quantitative and a qualitative semantics of linear logic will serve as a guide towards an extension of these theorems with recursion and an account for the parity condition of APT. We will use the conceptual clarity of the relational semantics to investigate in a first time the necessary extensions of the semantics to capture the whole higher-order model-checking problem, and then adapt them to the finitary Scott semantics to obtain a decidability result.

Chapter 6

A type system for higher-order model-checking

In order to study the higher-order model-checking problems of §3.4, we need to extend the constructions of Chapter 5 with two new ingredients:

- *recursion*, in order to handle higher-order recursion schemes,
- *colors*, in order to reflect the parity condition of alternating parity automata.

Our starting point is the idempotent intersection type system introduced by Kobayashi and Ong in [KO09], which can be understood as an extension of the idempotent intersection type system of Chapter 5 with coloring annotations. Given a HORS $\mathcal{G}$ and an APT $\mathcal{A}$, a parity game $\mathit{Adamic}(\mathcal{G}, \mathcal{A})$ is defined, and handles at the same time the recursion of the recursion scheme $\mathcal{G}$ and the parity condition of the tree automaton $\mathcal{A}$. Kobayashi and Ong prove in [KO] a soundness-and-completeness theorem which states that $\mathcal{A}$ accepts $\langle\, \mathcal{G} \,\rangle$ if and only if Eve has a winning strategy in $\mathit{Adamic}(\mathcal{G}, \mathcal{A})$.

In this chapter, we start by recalling their approach in §6.1, and by connecting it further to proof theory in §6.2. The original type system formulated by Kobayashi and Ong has a treatment of colors which is a bit ankward from a logical and semantic point of view. For that reason, motivated by the desire to extend the semantic constructions of Chapter 5 to include parities, we alter their coloring policy, and obtain in §6.3 a type system in which this coloring operation defines a *modal operator*, akin to a S4 modality. Our reformulation, in spite of its similarity with the original type system, is not at all equivalent to it. The theorem of soundness-and-completeness for this alternative type system, which relates winning typing derivations for $\mathcal{G}$ with respect to an APT $\mathcal{A}$ with winning executions of $\mathcal{A}$ over $\langle\, \mathcal{G} \,\rangle$, is stated in §6.4. We give an outline of the proof – to which Chapter 7 and Chapter 8 are devoted – in §6.5. We also formulate important remarks about the consequences of this proof, and discuss some potential extensions. To conclude the chapter, we extend in §6.6 the colored intersection type system we designed with a general weakening rule, which was absent from Kobayashi and Ong's original type system and which will be necessary to establish the correspondence with the finitary semantics of linear logic in Chapter 10.

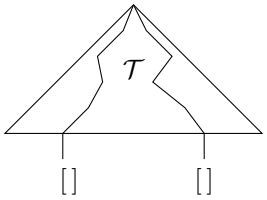
6.1 The Kobayashi-Ong type system

After designing in [Kob09b] a type-theoretic approach to alternating tree automata based on *idempotent* intersection types, Kobayashi carried on and generalized his approach with Ong [KO09] to the larger class of alternating *parity* automata. The basic idea of their work is to incorporate coloring annotations in the intersection types, in order to reflect the parity conditions of the tree automata. Suppose for instance that a binary terminal $a \in \Sigma$ induces a transition $\delta(q, a) = (1, q_1) \wedge (2, q_2)$ in an alternating parity tree automaton with coloring function $\Omega : Q \to \mathbb{N}$. In that case, the terminal a is assigned in [KO09] the intersection type

$$a \quad : \quad (q_1, m_1) \to (q_2, m_2) \to q \tag{6.1}$$

where $m_1 = max(\Omega(q_1), \Omega(q))$ and $m_2 = max(\Omega(q_2), \Omega(q))$ are colors indicating to the type system the colors of the states q, q_1, q_2 of the parity tree automaton.

In the case of an alternating automaton without parity condition described in the previous chapter, recall that a typing derivation of the value tree $\langle\, \mathcal{G} \,\rangle$ of the recursion scheme $\mathcal{G}$ is isomorphic to a run-tree of $\mathcal{A}$ over $\langle\, \mathcal{G} \,\rangle$. The idea is to extend this situation with coloring information. Consider a partial run-tree with two holes



Both paths leading from the root to a hole are labeled with elements of the signature Σ, and with states of the automaton describing the use of the transition function in the run. Moreover, each of these states has a color. So, denote by m_1 (resp. m_2) the maximal color encountered on the path leading to the first (resp. second) hole. In the intersection type system, this partial run tree corresponds to a derivation tree of a term $\lambda x.\, t$, in which the variable x occurs twice, and whose intersection types $\tau \to \theta$ refine the kind $o \to o$. The idea underlying Kobayashi and Ong's colored approach is to incorporate the maximal colors m_1 and m_2 seen along the path as annotations in the intersection types. Accordingly, the term $\lambda x.\, t$ corresponding to the partial run-tree depicted above will admit the colored intersection type $(m_1, \theta_1) \wedge (m_2, \theta_2) \to \theta$. This idea is then lifted to any order.

In this section, we recall the original type system of Kobayashi and Ong, before simplifying its coloring policy in the next section – a simplification which reveals the *modal* nature of colors (or priorities) in higher-order model-checking. In particular, we find clarifying to write $\boxdot_m \theta$ instead of (θ, m) as originally done in [KO09]. So, given a set of states Q and a coloring function $\Omega : Q \to \mathbb{N}$, we define the set of colors

$$Col \quad = \quad \{\, \Omega(q) \mid q \in Q \,\}$$

which contains the colors used by Ω. The intersection types are then generated by the grammar

$$\begin{array}{lcll} \theta & ::= & q \mid \tau \to \theta & (q \in Q) \\ \tau & ::= & \bigwedge_{i \in I} \boxdot_{m_i} \theta_i & (I \text{ finite}, m_i \in Col) \end{array}$$

where the intersection operator is *idempotent*, as defined in §5.4. This implies in particular the expected idempotency equation

$$\sigma \wedge \sigma \;=\; \sigma.$$

The refinement relation between intersection types and kinds is defined by the inductive rules below:

$$\frac{q \in Q}{q :: o} \qquad \frac{\tau :: \kappa_1 \qquad \theta :: \kappa_2}{\tau \to \theta :: \kappa_1 \to \kappa_2} \qquad \frac{\forall i \in I \quad \theta_i :: \kappa}{\bigwedge_{i \in I} \boxdot_{m_i} \theta_i :: \kappa}$$

Recall that contexts correspond to sets of variables, where each variable comes together with a simple type and a *set* of intersection types refining this simple type. The operation of union of contexts is defined in the obvious way, extending the non-colored case of §5.4: if x appears with the set of intersection types

$$\{\boxdot_{c_1} \theta_1, \ldots, \boxdot_{c_n} \theta_n\} \qquad \left(\text{denoted as } \bigwedge_{i=1}^{n} \boxdot_{c_i} \theta_i \right)$$

in Γ and with the set of intersection types

$$\left\{\boxdot_{c'_1} \theta'_1, \ldots, \boxdot_{c'_m} \theta'_m\right\} \qquad \left(\text{denoted as } \bigwedge_{j=1}^{m} \boxdot_{c'_j} \theta'_j \right)$$

in Γ', it appears in $\Gamma \cup \Gamma'$ with the set of intersection types

$$\left\{\boxdot_{c_1} \theta_1, \ldots, \boxdot_{c_n} \theta_n, \boxdot_{c'_1} \theta'_1, \ldots, \boxdot_{c'_m} \theta'_m\right\}$$

Note that this set has at most $n+m$ elements, but may have a smaller number of elements if an intersection type occurs with the same color in both contexts. The color modality acts on intersection types and contexts in the following way:

$$\boxdot_m \left(\bigwedge_{i \in I} \boxdot_{m_i} \theta_i \right) \;=\; \bigwedge_{i \in I} \boxdot_{\max(m, m_i)} \theta_i \qquad\qquad \boxdot_m (x : \tau, \Gamma) \;=\; x : \boxdot_m \tau \,,\; \boxdot_m \Gamma$$

An important aspect of Kobayashi and Ong's construction is that every intersection type should be assigned a color. So, given a coloring function $\Omega : Q \to \mathbb{N}$ on the states of the automaton $\mathcal{A}$, an intersection type $\tau \to \sigma$ has color:

$$\Omega(\tau \to \sigma) \;=\; \Omega(\sigma) \tag{6.2}$$

This means that every intersection type inherits the color of its output. As we will see, one benefit of our modal approach to higher-order model-checking is that we can avoid that slightly arbitrary definition, to replace it by a more careful treatment of colors. The original type system of Kobayashi and Ong is recast in Figure 6.1. We find useful to emphasize the fact that the rule λ allows a restricted form of *weakening*: since the intersection types we consider are idempotent, contexts are sets containing sets of intersection types refining

$$\text{Axiom}\quad \frac{}{x : \bigwedge_{\{\star\}} \boxdot_{\Omega(\theta)}\, \theta :: \kappa \;\vdash\; x : \theta :: \kappa} \qquad (x \in \mathcal{V} \cup \mathcal{N})$$

$$\delta\quad \frac{\{\,(i, q_{ij}) \mid 1 \le i \le n, 1 \le j \le k_i\} \quad \text{satisfies} \quad \delta_A(q,a)}{\emptyset \vdash a : \bigwedge_{j=1}^{k_1} \boxdot_{m_{1j}}\, q_{1j} \;\to\; \ldots \;\to\; \bigwedge_{j=1}^{k_n} \boxdot_{m_{nj}}\, q_{nj} \to q :: o^n \to o} \qquad \text{for } a \in \Sigma \text{ and } m_{ij} = \max(\Omega(q_{ij}), \Omega(q))$$

$$\text{App}\quad \frac{\Gamma \vdash t : (\boxdot_{m_1}\, \theta_1 \;\wedge \cdots \wedge\; \boxdot_{m_k}\, \theta_k) \to \theta :: \kappa \to \kappa' \qquad \Gamma_1 \vdash u : \theta_1 :: \kappa \quad \cdots \quad \Gamma_k \vdash u : \theta_k :: \kappa}{\Gamma \cup \boxdot_{m_1}\Gamma_1 \cup \ldots \cup \boxdot_{m_k}\Gamma_k \;\vdash\; t\,u : \theta :: \kappa'}$$

$$\lambda\quad \frac{\Gamma\,,\, x : \bigwedge_{i\in I} \boxdot_{m_i}\, \theta_i :: \kappa \;\vdash\; t : \theta :: \kappa' \qquad I \subseteq J}{\Gamma \;\vdash\; \lambda x\,.\,t : \left(\bigwedge_{j\in J} \boxdot_{m_j}\, \theta_j\right) \to \theta :: \kappa \to \kappa'} \qquad (x \in \mathcal{V})$$

Figure 6.1: The Kobayashi-Ong type system $KO(\mathcal{A})$ associated to the alternating parity tree automaton $\mathcal{A}$

the simple type of a variable. In particular, a variable with an empty set of intersection types is always considered by Kobayashi and Ong as appearing in a context, and taking $I = \emptyset$ in the rule λ allows to apply weakening – but only on variables which are captured by an abstraction. The interested reader will see in §6.6 how to extend the modal variant of this system with general weakening.

The resulting intersection type system [KO09] enables one to type the rewriting rules of a higher-order recursion scheme

$$\Gamma \;\vdash\; \mathcal{R}(F) \;:\; \theta \;::\; \kappa(F) \tag{6.3}$$

where the non-terminals occurring in the λ-term $\mathcal{R}(F)$ appear as variables in the context Γ of the typing judgment. Note that every rule of a recursion scheme admits finitely many colored intersection types (6.3), where the context consists of intersection types refining the simple types of the non-terminals occurring in $\mathcal{R}(F)$. In a context Γ, a non-terminal G typically occurs as

$$G : \bigwedge_{i\in I} \boxdot_{m_i}\, \theta_i :: \kappa(G) \tag{6.4}$$

Recursion as a parity game. In order to account for recursion, Kobayashi and Ong introduce the finite parity game $\mathit{Adamic}(\mathcal{G}, \mathcal{A})$ in which Adam incrementally tries to disprove Eve's typing by picking the appropriate non-terminal to unfold.

More specifically, Eve's vertices correspond to colored typings for non-terminals, while Adam's vertices are typing contexts. There is an edge (potentially played by Eve) from a typing $F : \boxdot_m\, \theta :: \kappa$ to a context Γ if and only the sequent

$$\Gamma \vdash \mathcal{R}(F) \;:\; \theta \;::\; \kappa(F) \tag{6.5}$$

is provable in the colored type system $KO(\mathcal{A})$, and there is an edge (potentially played by Adam) from a context Γ to a typing $G : \boxdot_m\, \theta :: \kappa$ if and only

if $G : \boxdot_m \theta :: \kappa$ is an element of the context Γ. Note that the resulting game is finite, thanks to the idempotency of the intersection operator. Vertices $F : \boxdot_m \theta :: \kappa$ receive color m; other vertices receive the neutral color 0.

A *play* is winning for Adam if and only if it ends on a node $F : \boxdot_m \theta :: \kappa$ from which Eve has no move - that is, if she made a typing assumption she can not prove - or if the play is infinite and the maximal color played infinitely often by Adam is odd. Here, by the color played by Adam at any point of the game, we simply mean the color of the non-terminal picked by Adam. Therefore, Eve has a winning strategy in this game if and only if she can ensure the existence of a winning sequence of typings along every possible branch of reductions in the scheme. This leads to the main result by Kobayashi and Ong, which may be seen as some kind of soundness-and-completeness theorem:

Theorem 16 ([KO09])**.** *Given a* productive *higher-order recursion scheme $\mathcal{G}$ and an alternating* parity *tree automaton $\mathcal{A}$, Eve has a winning strategy in the parity game Adamic$(\mathcal{G}, \mathcal{A})$ iff $\mathcal{A}$ has a winning run-tree over $\langle\, \mathcal{G} \,\rangle$.*

Thanks to the positional determinacy of parity games (Theorem 5), an immediate consequence of this theorem is the decidability of the local higher-order model-checking problem:

Corollary 1 (Decidability of the local HOMC problem)**.** *Given an alternating parity automaton $\mathcal{A}$ and a higher-order recursion scheme $\mathcal{G}$, it is decidable whether $\mathcal{A}$ has a winning execution over $\langle\, \mathcal{G} \,\rangle$.*

In §6.3, we show how the coloring policy of the type system $KO(\mathcal{A})$ underlying the parity game *Adamic*$(\mathcal{G}, \mathcal{A})$ can be modified in a way which reveals the modal nature of the coloration box $\boxdot$. Besides the questionable equation (6.2) already mentioned, the fundamental obstruction to interpret the original type system by Kobayashi and Ong as a modal proof system is the pair of Axiom and δ rules, which do not respect the basic principles of a modal logic like S4. In particular, the original intersection type (6.1) of the Kobayashi-Ong intersection type system

$$a \quad : \quad (q_1, m_1) \to (q_2, m_2) \to q$$

does not carry any clear modal meaning since the coloring annotations $m_i = \max(\Omega(q), \Omega(q_i))$ depend on the color $\Omega(q)$ of the return state q as well as on the color $\Omega(q_i)$ of the argument q_i.

The discovery of the modal nature of coloring in our variant of the Kobayashi-Ong type system plays a fundamental rôle in our unification of higher-order model-checking and contemporary semantics. In particular, it leads us in Part III to the very natural definition of denotational models of the λY-calculus, based on the construction of a coloring comonad and of an inductive-coinductive interpretation of the fixpoint operator Y, based on the parity condition. We establish moreover that the resulting proof system satisfies a soundness-and-completeness theorem (Theorem 19) similar to Theorem 16. As such, it provides a denotational semantics of higher-order model-checking where the interpretation of a higher-order recursion scheme reflects its acceptance by a given alternating parity tree automaton.

6.2 A proof-theoretic reformulation

Before introducing in §6.3 our modal variant of the Kobayashi-Ong type system, we find useful to replace the parity game $Adamic(\mathcal{G}, \mathcal{A})$ by a proof system $KO_{fix}(\mathcal{G}, \mathcal{A})$ with typing derivations of possibly infinite depth. To that purpose, we introduce the intermediate parity game $Edenic(\mathcal{G}, \mathcal{A})$.

From *Adamic*$(\mathcal{G}, \mathcal{A})$ to *Edenic*$(\mathcal{G}, \mathcal{A})$. Despite its intuitive connection with type theory, the game $Adamic(\mathcal{G}, \mathcal{A})$ does not describe the on-the-fly construction of a branch of a typing proof, for two essential reasons:

- Eve does not provide a *witness* of the typing proof of the sequent (6.5) which builds Γ, so that proofs can not be extracted from plays, and that no distinction is made between different derivations with the same conclusion,
- and Adam does not play an *occurrence* of a non-terminal in $\mathcal{R}(F)$, but a *typing* which could, due to idempotency, correspond to several Axiom leaves of a derivation tree.

In order to understand the game $Adamic(\mathcal{G}, \mathcal{A})$ from a purely type-theoretic point of view, we introduce the parity game $Edenic(\mathcal{G}, \mathcal{A})$, which only differs on these two points:

- Every time Eve plays a move Γ such that

  $$\Gamma \vdash \mathcal{R}(F) : \theta :: \kappa(F)$$

  she needs to produce a typing derivation π of this sequent,
- Once Eve has played in this way, and every time Adam plays after it a move

  $$G : \boxdot_m \theta :: \kappa(G)$$

  appearing inside the context Γ just played by Eve, Adam should also play an occurrence of this non-terminal in the typing derivation π of $\mathcal{R}(F)$. Equivalently, Adam should pick an Axiom rule introducing a non-terminal in the typing derivation π provided by Eve at the previous turn.

Note that the resulting game is bigger than the original game $Adamic(\mathcal{G}, \mathcal{A})$, yet finite. We prove the following correspondence:

Proposition 15. *The parity games $Adamic(\mathcal{G}, \mathcal{A})$ and $Edenic(\mathcal{G}, \mathcal{A})$ are equivalent, in the sense that a player Eve or Adam has a winning strategy in one game if and only if she or he has a winning strategy in the other game.*

Note that the translation of strategies of $Edenic(\mathcal{G}, \mathcal{A})$ to strategies of $Adamic(\mathcal{G}, \mathcal{A})$ may be seen as a collapse which relies on a particular uniformization property, reminiscent of the proof of positional determinacy of parity games: from a winning strategy for Eve in $Edenic(\mathcal{G}, \mathcal{A})$, one can extract a strategy such that, given a colored type, any occurrence of a non-terminal with this type will be mapped to the same typing proof.

From *Edenic*$(\mathcal{G},\mathcal{A})$ to the proof system $KO_{fix}(\mathcal{G},\mathcal{A})$. In order to give a purely type-theoretic account of the parity game *Edenic*$(\mathcal{G},\mathcal{A})$, we accommodate recursion in the intersection type system $KO(\mathcal{A})$. We proceed by extending $KO(\mathcal{A})$ with a rule *fix* whose purpose is to expand the non-terminals $F \in \mathcal{N}$ of the original recursion scheme $\mathcal{G}$. The original intersection type system $KO(\mathcal{A})$ is a refinement system based on the simply-typed λ-calculus: its typing derivations are finite. Adding the *fix* rule to the proof system thus enables one to obtain possibly infinitary derivation trees. So, given a higher-order recursion scheme $\mathcal{G}$ and an alternating parity automaton $\mathcal{A}$, we define the intersection type system $KO_{fix}(\mathcal{G},\mathcal{A})$ as the system $KO(\mathcal{A})$ extended with the recursion rule below:

$$\textit{fix} \quad \frac{\Gamma \vdash \mathcal{R}(F) \,:\, \theta \,::\, \kappa}{F \,:\, \boxdot_{\Omega(\theta)}\, \theta \,::\, \kappa \;\vdash\; F \,:\, \theta \,::\, \kappa} \qquad \mathrm{dom}(\Gamma) \subseteq \mathcal{N}$$

An important novelty of the resulting type system, as we have just said, is that derivations with possibly infinite depth are allowed in the system $KO_{fix}(\mathcal{G},\mathcal{A})$. Note moreover that the *fix* rule depends on the underlying higher-order recursion scheme $\mathcal{G}$. The type system $KO_{fix}(\mathcal{G},\mathcal{A})$ thus depends of both $\mathcal{G}$ and $\mathcal{A}$.

We now explain how the parity game *Edenic*$(\mathcal{G},\mathcal{A})$ and the proof system $KO_{fix}(\mathcal{G},\mathcal{A})$ are related. The key observation is that the total strategies σ for Eve in *Edenic*$(\mathcal{G},\mathcal{A})$ are in one-to-one correspondence with the derivation trees of the initial sequent

$$S \,:\, \boxdot_{\Omega(q_0)}\, q_0 \,::\, o \;\vdash\; S \,:\, q_0 \,::\, o \tag{6.6}$$

in the type system $KO_{fix}(\mathcal{G},\mathcal{A})$. The translation of a total strategy σ for Eve in *Edenic*$(\mathcal{G},\mathcal{A})$ into a proof $\pi(\sigma)$ in $KO_{fix}(\mathcal{G},\mathcal{A})$ proceeds by incrementally plugging to every *fix* rule the finite proof-tree Eve plays when she follows the total strategy σ, starting from the unfolding of the axiom S of the scheme:

$$\textit{fix} \quad \frac{\dfrac{\vdots}{\Gamma \vdash \mathcal{R}(S) \,:\, q_0 \,::\, o}}{S \,:\, \boxdot_{\Omega(q_0)}\, q_0 \,::\, o \;\vdash\; S \,:\, q_0 \,::\, o}$$

The converse process can be defined in a similar way. It enables one to associate to every derivation tree π in the proof system $KO_{fix}(\mathcal{G},\mathcal{A})$ a total strategy $\sigma(\pi)$ played by Eve in *Edenic*$(\mathcal{G},\mathcal{A})$. Moreover, the translations are mutually inverse, in the sense that

$$\sigma(\pi(\sigma)) \;=\; \sigma \quad \text{and} \quad \pi(\sigma(\pi)) \;=\; \pi.$$

The proof system $KO_{fix}(\mathcal{G},\mathcal{A})$ is slightly biased towards Eve's strategies in *Edenic*$(\mathcal{G},\mathcal{A})$. However, a strategy by Adam in *Edenic*$(\mathcal{G},\mathcal{A})$ has also a meaning (although less obvious) in the proof system $KO_{fix}(\mathcal{G},\mathcal{A})$. Indeed, suppose that τ is a strategy for Adam in *Edenic*$(\mathcal{G},\mathcal{A})$, and let π be a derivation tree of the base sequent (6.6). It appears that the interaction of Adam's strategy τ and Eve's strategy $\sigma(\pi)$ in *Edenic*$(\mathcal{G},\mathcal{A})$ corresponds to the exploration of exactly on branch of the derivation tree π. In particular, the infinite plays of *Edenic*$(\mathcal{G},\mathcal{A})$ where Eve follows the strategy $\sigma(\pi)$ are in one-to-one correspondence with the infinite branches of π. Note that the finite plays where Eve

follows $\sigma(\pi)$ are not exactly the finite branches of π, but paths leading to an occurrence of the *fix* rule unfolding a non-terminal to a term. In particular, the maximal finite plays lead to a *fix* rule unfolding a non-terminal to a term without non-terminals.

Accommodating colors in $KO_{fix}(\mathcal{G},\mathcal{A})$. One main benefit of this formulation compared to the original formulation by Kobayashi and Ong using the parity game $Adamic(\mathcal{G},\mathcal{A})$ [KO09] is that the parity condition of the alternating parity tree automaton $\mathcal{A}$ may be reformulated as a parity condition on the infinite paths of the derivation trees of $KO_{fix}(\mathcal{G},\mathcal{A})$. The key idea to perform this transfer of colors from automata theory to proof-theory is to color the *fix* rules of the typing derivation tree π. A subtle point is that we need to color the *fix* rules of the derivation tree π in the same way as the corresponding moves are colored in the game $Edenic(\mathcal{G},\mathcal{A})$. As we will see, the resulting coloring policy is far from obvious from a proof-theoretic point of view. This has to do with the original coloring policy of the game $Adamic(\mathcal{G},\mathcal{A})$ designed by Kobayashi and Ong. The situation will become much clearer and nicer when we shift to our modal coloring policy in §6.3.

Let us explain how this coloring policy works for $KO_{fix}(\mathcal{G},\mathcal{A})$. Every *fix* rule $(\star)$ expands a non-terminal G, typed with $\boxdot_{\Omega(\theta)}\,\theta$, which occurs in a term $\mathcal{R}(F)$ itself introduced by a *fix* rule noted $(\star\star)$. The only exception is the *fix* rule $(\star)$ of the root of the derivation tree. The *fix* $(\star\star)$ rule is precisely the previous *fix* rule on the path between the root of the derivation tree and the non-terminal G unfolded by the rule $(\star)$. The *fix* rule $(\star\star)$ introduces a context Γ typing a sequent

$$\Gamma \vdash \mathcal{R}(F) \,:\, \theta \,::\, \kappa(F)$$

By definition of $(\star\star)$, the non-terminal G occurs in the context Γ with several colored intersection types, as follows:

$$G \,:\, \bigwedge_{i\in I} \boxdot_{m_i}\,\theta_i \,::\, \kappa(G)$$

One of these colored intersection types $\boxdot_{m_i}\,\theta_i$ corresponds to the occurrence of the non-terminal G of interest in the term $\mathcal{R}(F)$. We color the *fix* rule $(\star)$ unfolding this occurrence of G with the color m_i. Note that, by construction, the color m_i is greater than the color $\Omega(\theta)$. Every *fix* rule of the tree is assigned a color in this way, except the *fix* rule at the root which receives the color $\Omega(q_0)$. All the other rules, which are not *fix* rules, are assigned the neutral color 0.

At this stage, we are ready to define the notion of *winning* derivation tree. Every infinite branch in a derivation tree π of $KO_{fix}(\mathcal{G},\mathcal{A})$ encounters an infinite number of *fix* rules. The color of an infinite branch is thus defined as the maximal color $m \in Col$ encountered infinitely often along it. An infinite branch of a derivation tree is declared *winning* precisely when its color is an even integer.

Definition 33. A derivation tree of $KO_{fix}(\mathcal{G},\mathcal{A})$ is *winning* when all its infinite branches are winning.

As we have just said, the coloring policy of the *fix* rules in the derivation trees of $KO_{fix}(\mathcal{G}, \mathcal{A})$ is designed to reflect precisely the coloring policy of the parity game $Edenic(\mathcal{G}, \mathcal{A})$. We will see in §6.3 how to define an alternative notion of coloring, this time on derivation trees of our modal type system. In this alternative formulation, the colors will be transmitted and tagged on the Application rules, and no longer on the *fix* rules. This change of perspective comes from the discovery that the coloring operation can be defined in a modal way, and therefore behaves in the spirit of linear logic, as a calculus of functorial boxes [Mel06b]. Each Application rule will open coloring boxes, and the color of a branch will be precisely the maximal color crossed infinitely often along it.

Since we were careful to define the coloring policy of the *fix* rules in $KO_{fix}(\mathcal{G}, \mathcal{A})$ in harmony with the color policy of the parity game $Edenic(\mathcal{G}, \mathcal{A})$, a derivation tree π is winning if and only if the associated strategy $\sigma(\pi)$ for Eve is winning in the parity game:

Theorem 17.

- *A total strategy σ for Eve is winning in $Edenic(\mathcal{G}, \mathcal{A})$ if and only if $\pi(\sigma)$ is a winning derivation tree of $KO_{fix}(\mathcal{G}, \mathcal{A})$.*
- *A derivation tree π of $KO_{fix}(\mathcal{G}, \mathcal{A})$ is winning if and only if Eve's total strategy $\sigma(\pi)$ is winning in $Edenic(\mathcal{G}, \mathcal{A})$.*

As a consequence, Theorem 16 can be rephrased as follows:

> Given a productive recursion scheme $\mathcal{G}$ and an alternating parity automaton $\mathcal{A}$, there exists a winning run-tree of $\mathcal{A}$ over $\langle\, \mathcal{G} \,\rangle$ with initial state q if and only if the sequent
>
> $$S : \bigwedge_{\{1\}} \boxdot_{\Omega(q)}\ q :: o \ \vdash\ S : q :: o$$
>
> has a winning derivation tree in the type system $KO_{fix}(\mathcal{G}, \mathcal{A})$.

6.3 A proof-theoretic and modal reformulation

One problem with the original type system by Kobayashi and Ong is that it can not be interpreted as a modal proof system (at least in the style of S4), because of the definition of the Axiom and δ rules in their original system. For instance, the Axiom rule

$$\text{Axiom} \qquad \frac{}{x : \boxdot_{\Omega(\theta)}\ \theta :: \kappa \ \vdash\ x : \theta :: \kappa} \qquad (x \in \mathcal{V} \cup \mathcal{N})$$

requires to tag the intersection type θ it introduces in the context by its color $\Omega(\theta)$. From a modal point of view, this rule would mean that

$$\Box_m\ \theta \ \vdash\ \theta$$

holds when θ is of the form $\tau \to q$ for a state q of color m. For a very long time we tried to understand the modal meaning of this rule, but it seemed to lead nowhere serious. So, in order to address this obstruction, we will need to revise drastically the coloring system as we found it in [KO09] in such a way

that the soundness-and-completeness theorem (Theorem 16) still holds in our setting. Moreover, we believe that by doing this we were able to capture the essence of the original type system.

It appears that the purpose of the coloring annotation of a non-terminal F occurring in a term t is to carry coloring information on the context C generated by a reduction of t putting F in head position. Let t be a term of simple type o appearing during the reduction of a higher-order recursion scheme $\mathcal{G}$. Suppose that t is such that an inductive application of the weak head reduction computes[1]:

$$t \quad \rightarrow^* \quad C\,[F\ t_1 \cdots t_n] \tag{6.7}$$

where the branch of C leading to the hole filled with $F\ t_1 \cdots t_n$ only contains terminals of the signature Σ of interest. To ease reasoning, we will consider that there is only one occurrence of the non-terminal F in $C\,[F\ t_1 \cdots t_n]$. Then the context $C[\,]$ generated by this reduction contains a subtree of the normal form $\langle\, t \,\rangle$ of t. This subtree is the partial production of $C\,[F\ t_1 \cdots t_n]$, obtained by the transformation

$$C\,[F\ t_1 \cdots t_n] \ \mapsto\ (C\,[F\ t_1 \cdots t_n])^{\nabla}$$

of Definition 21 (on p.72). Due to the assumption we made on $C[\,]$, the partial production contains the whole branch leading to the hole $[\,]$.

Let us now consider the lifting of this reduction to a derivation of the sequent

$$\Gamma,\, F : \bigwedge_{i\in I} \boxdot_{m_i}\, \theta_i \,::\, \kappa(F) \ \vdash\ t : q :: o \tag{6.8}$$

The application of the β-reduction to the term t can be seen as a rewriting procedure on proofs, associated with the subject reduction property, as explained in Chapter 5 for intersection types without coloring annotations and parametrized by an alternating automaton $\mathcal{A}$. Recall also from Chapter 5 that while the β-reduction of t computes the tree $\langle\, t \,\rangle$ it represents, the normalization of typing derivations in the associated intersection type system computes a representation of a run-tree of $\mathcal{A}$ over $\langle\, t \,\rangle$. In this way, the rewriting of the proof of the sequent (6.8) along the reduction (6.7) produces a proof of

$$\Gamma,\, F : \bigwedge_{i\in I} \boxdot_{m_i}\, \theta_i \,::\, \kappa(F) \ \vdash\ C\,[F\ t_1 \cdots t_n] : q :: o \tag{6.9}$$

which represents a run of the alternating automaton $\mathcal{A}$ over the subtree $C^{\nabla}[\,]$ of $\langle\, t \,\rangle$. Note that the notion of execution of an automaton is slightly extended here, as the tree we consider has a hole. This hole is just treated as a unlabeled leaf: it inherits a state from the execution of the automaton over its predecessor, and we assume that the automaton accepts the leaf. In the resulting run, the hole may appear several times, as the execution of the automaton may have duplicated it. It follows that the derivation of (6.9) may contain several occurrences of F, even though we assumed that F occurs only once in the

[1] In general, we would have to consider also the unfolding of non-terminals, regulated by the rewrite rules of $\mathcal{G}$. We prefer to stick to a situation which does not require it, as this would make our explanation unnecessary complex. For the general case, see the proof of soundness in Chapter 8.

term $C\,[F\ t_1 \cdots t_n]$. Each of these occurrences in the proof is introduced by an Axiom rule, with an intersection type $\boxdot_{m_i}\ \theta_i$. The color m_i is precisely the maximal color seen from the root of this run-tree over $C^{\nabla}[\,]$ to the copy of the hole where the occurrence of F of interest is inserted.

What we have just explained treats the situation when the context $C[\,]$ is non-empty. We also need to consider the case of a rule whose rewriting does not produce any tree context, as for instance $G\ =\ F$: we have

$$G \ \rightarrow\ C\,[F]$$

with $C\ =\ [\,]$ the empty tree context. In this case, if F is typed with the state q, the original type system of Kobayashi and Ong attributes it the color $\Omega(q)$. However, the execution of any alternating automaton $\mathcal{A}$ over the empty tree context $C\ =\ [\,]$ produces this empty tree context again. A hurdle to a proper definition of the color annotation which should be attributed to F is the fact that the maximal color encountered from the root of C to its hole is ill-defined, as there is no color on this empty path. We therefore introduce a new, *neutral* color ε, indicating that the branch of the run-tree is empty. This neutral color will allow us to design an Axiom rule following the principles of a modal logic. Note that this neutral color is only introduced here to allow a uniform definition of types and contexts; it should be understood as the *absence* of a coloring annotation, as it does not affect the coloring of types. We could thus write

$$\Box_{\varepsilon}\ \theta\ =\ \theta.$$

Due to the introduction of this neutral color, we redefine the set of colors as:

$$Col\ \ =\ \ \{\,\Omega(q)\mid q\in Q\,\}\ \uplus\ \{\,\varepsilon\,\}$$

The intersection types, the refinement relation and the action of a color modality on intersection types and contexts are defined just as in the previous section, setting

$$\max(c,\,\varepsilon)\ \ =\ \ c\quad \text{for every color } c\in Col$$

From this, one obtains an intersection type system $\gimel(\mathcal{A})$ parametrized by the alternating tree automaton $\mathcal{A}$, whose rules are given in Figure 6.2. Here we use the Hebrew letter $\gimel$ which should be read "tsadi". This type system is a variant of the original intersection type system $KO(\mathcal{A})$, with the following changes:

- in the Axiom rule

$$\text{Axiom}\qquad \frac{}{x\ :\ \boxdot_{\varepsilon}\ \theta\ ::\ \kappa\ \ \vdash\ \ x\ :\ \theta\ ::\ \kappa}\qquad (x\in\mathcal{V}\cup\mathcal{N})$$

  the variable x is introduced with the neutral color ε, as its normalization produces an empty context. In the formulation of Figure 6.2, we write

$$x\ :\ \bigwedge_{\{\star\}}\ \boxdot_{\varepsilon}\ \theta\ ::\ \kappa$$

  as we did for the Kobayashi-Ong type system, in order to allow a more uniform formulation of contexts.

- in the δ rule, the change is more subtle. The idea is to replace the typing

$$a \quad : \quad \boxdot_{\max(\Omega(q),\Omega(q_1))}\ q_1 \to \boxdot_{\max(\Omega(q),\Omega(q_2))}\ q_2 \to q$$

of a terminal a with the typing

$$a \quad : \quad \boxdot_{\Omega(q_1)}\ q_1 \to \boxdot_{\Omega(q_2)}\ q_2 \to q$$

in which the coloring by the state q does not appear on arguments anymore.

This change in the δ rule is inspired by the fact that, on a branch of a partial run-tree or on an infinite branch

$$a_1 \cdot a_2 \cdots a_n \cdots$$

every symbol a_{i+1} occurs as argument of its predecessor a_i, and therefore inherits the color of its return state q_{i+1} from the fact that it is taken as argument by the symbol a_i. For this reason, it is not necessary to propagate in the typing the color of the return state q_{i+1} of a symbol a_{i+1} of the signature Σ. This leads us to a lighter treatment of colors, moreover nicely connected to modal logic. This discussion is carried out in more details in the proofs of soundness and of completeness of the type system $\boldsymbol{\gamma}_{fix}(\mathcal{G},\mathcal{A})$ in Chapter 7 and Chapter 8. This alternative point of view enables us to understand semantically coloring as a family of modal boxes $(\boxdot_m)_{m\in Col}$ which formally defines what Melliès calls a *parametric comonad* in [Mel14b, Mel06b], paving the way for the semantic constructions for higher-order model-checking defined in Part III. The notion of parametric comonad is discussed in §6.5.

Just as $KO(\mathcal{A})$, the resulting type system $\boldsymbol{\gamma}(\mathcal{A})$ enables us to type the rewriting rules of a higher-order recursion scheme

$$\Gamma \ \vdash\ \mathcal{R}(F)\ :\ \sigma\ ::\ \kappa \tag{6.10}$$

where the non-terminals occurring in the λ-term $\mathcal{R}(F)$ appear as variables in the context Γ of the typing judgment, but it does not include a fixpoint operator Y and for that reason does not accommodate recursion.

Interpretation of recursion. We define $\mathsf{Adamic}(\mathcal{G},\mathcal{A})$ and $\mathsf{Edenic}(\mathcal{G},\mathcal{A})$ as the counterparts of $\mathit{Adamic}(\mathcal{G},\mathcal{A})$ and $\mathit{Edenic}(\mathcal{G},\mathcal{A})$ where Eve builds derivations in the type system $\boldsymbol{\gamma}(\mathcal{A})$, and where Adam picks typings for non-terminals from these proofs. In the same way as Proposition 15, we can establish that the games $\mathsf{Adamic}(\mathcal{G},\mathcal{A})$ and $\mathsf{Edenic}(\mathcal{G},\mathcal{A})$ are equivalent. Given a higher-order recursion scheme $\mathcal{G}$ and an alternating parity automaton $\mathcal{A}$, we define the intersection type system $\boldsymbol{\gamma}_{fix}(\mathcal{G},\mathcal{A})$ as the system $\boldsymbol{\gamma}(\mathcal{A})$ extended with the recursion rule below:

$$\mathit{fix} \qquad \frac{\Gamma \vdash \mathcal{R}(F)\ :\ \theta\ ::\ \kappa}{F\ :\ \boxdot_\varepsilon\ \theta\ ::\ \kappa\ \vdash\ F\ :\ \theta\ ::\ \kappa} \qquad \mathrm{dom}(\Gamma) \subseteq \mathcal{N}$$

and we restrict at the same time the Axiom rule to variables $x \in \mathcal{V}$, so that it can not be applied to non-terminals any more. The rule *fix* enables the construction of derivation trees of countable depth. In the same way as Theorem 17 there is a one-to-one correspondence between the total strategies for Eve

$$\text{Axiom}\qquad \frac{}{x \,:\, \bigwedge_{\{\star\}} \boxdot_{\varepsilon}\, \theta \,::\, \kappa \;\vdash\; x \,:\, \theta \,::\, \kappa} \qquad (x \in \mathcal{V} \cup \mathcal{N})$$

$$\delta\qquad \frac{\{\,(i, q_{ij}) \mid 1 \leq i \leq n, 1 \leq j \leq k_i\} \quad \text{satisfies} \quad \delta_A(q,a)}{\emptyset \vdash a \,:\, \bigwedge_{j=1}^{k_1} \boxdot_{\Omega(q_{1j})}\, q_{1j} \;\to\; \ldots \;\to\; \bigwedge_{j=1}^{k_n} \boxdot_{\Omega(q_{nj})}\, q_{nj} \to q \,::\, o \to \cdots \to o \to o} \qquad a \in \Sigma$$

$$\text{App}\qquad \frac{\Gamma \vdash t : (\boxdot_{m_1}\, \theta_1 \;\wedge \cdots \wedge\, \boxdot_{m_k}\, \theta_k) \to \theta \,::\, \kappa \to \kappa' \qquad \Gamma_1 \vdash u \,:\, \theta_1 \,::\, \kappa \quad \cdots \quad \Gamma_k \vdash u \,:\, \theta_k \,::\, \kappa}{\Gamma \cup \boxdot_{m_1}\Gamma_1 \cup \ldots \cup \boxdot_{m_k}\Gamma_k \;\vdash\; t\,u \,:\, \theta \,::\, \kappa'}$$

$$\lambda\qquad \frac{\Gamma\,,\, x \,:\, \bigwedge_{i\in I} \boxdot_{m_i}\, \theta_i \,::\, \kappa \;\vdash\; t \,:\, \theta \,::\, \kappa' \qquad I \subseteq J}{\Gamma \;\vdash\; \lambda\, x\,.\, t \,:\, \left(\bigwedge_{j\in J} \boxdot_{m_j}\, \theta_j\right) \to \theta \,::\, \kappa \to \kappa'} \qquad (x \in \mathcal{V})$$

Figure 6.2
The type system $\gamma(\mathcal{A})$ associated to the alternating parity tree automaton $\mathcal{A}$.

in $\mathit{Edenic}(\mathcal{G}, \mathcal{A})$ and the derivation trees of $\gamma_{fix}(\mathcal{G}, \mathcal{A})$. This correspondence notably maps winning strategies to winning derivation trees and conversely winning derivation trees to winning strategies, for the notion of winning derivation tree we define just below.

First coloring policy. A first manner to reflect the parity condition of $\mathsf{Edenic}(\mathcal{G}, \mathcal{A})$ over the derivation trees of the system $\gamma_{fix}(\mathcal{G}, \mathcal{A})$ is to proceed in the same spirit as we did for $\mathit{Edenic}(\mathcal{G}, \mathcal{A})$ and $KO_{fix}(\mathcal{G}, \mathcal{A})$ in §6.2. We attribute the neutral color ε to the *fix* rule occurring as root of the tree, and to every non-fix rule. We attribute to every other *fix* rule the color m_i introduced by the preceding *fix* rule for the occurrence of the non-terminal it rewrites.

Second coloring policy. A second way, which leads to an equivalent and more *local* definition of the color of a branch, consists in assigning a color to each node of the derivation tree as follows:

- the node $\Gamma_i \vdash u \,:\, \theta_i \,::\, \kappa$ is assigned the color m_i in every Application rule

$$\frac{\Gamma \vdash t : (\boxdot_{m_1}\, \theta_1 \;\wedge \cdots \wedge\, \boxdot_{m_k}\, \theta_k) \to \theta \,::\, \kappa \to \kappa' \qquad \cdots \quad \Gamma_i \vdash u \,:\, \theta_i \,::\, \kappa \quad \cdots}{\Gamma \;+\; \boxdot_{m_1}\Gamma_1 \;+\; \ldots \;+\; \boxdot_{m_k}\Gamma_k \;\vdash\; t\,u \,:\, \theta \,::\, \kappa'}$$

  of the derivation tree,

- all the other nodes of the derivation tree are assigned the neutral color ε, which means in some sense that they are not colored.

We are now ready to extend the usual parity condition with a neutral color ε, according to both coloring policies. This leads to a natural definition of the color of an infinite branch of a given derivation tree: it is

- the neutral color ε if no other color $m \in \mathit{Col}$ occurs infinitely often in the branch,
- otherwise, the maximal non-neutral color $m \in \mathit{Col} \setminus \{\varepsilon\}$ seen infinitely often.

We call « *fix* color » the color of an infinite branch according to the first coloring policy, and « *App* color » the color of a branch according to the second policy. An infinite branch of the derivation tree is declared *winning* for the *fix* (resp. *App*) policy precisely when its *fix* (resp. *App*) color is an even integer, and in particular is different from the neutral color. A winning derivation tree for the *fix* (resp. *App*) coloring policy is then defined as a derivation tree whose infinite branches are all winning in the sense just explained.

The two coloring policies are equivalent, as stated by the following theorem:

Theorem 18. *Let π be a $\boldsymbol{\gamma}_{fix}(\mathcal{G}, \mathcal{A})$ derivation tree, and b an infinite branch of this tree. Then the fix color of b is equal to is App color.*

Corollary 2. *Let π be a $\boldsymbol{\gamma}_{fix}(\mathcal{G}, \mathcal{A})$ derivation tree. Then π is winning for the fix policy if and only if it is winning for the App policy.*

We will thus say that π is winning when it is winning for any of the two policies. Theorem 18 is a consequence of the following proposition, which implies that the *fix* color of an infinite branch of π is equal to its *App* color:

Proposition 16. *Consider a proof π in $\boldsymbol{\gamma}(\mathcal{A})$ of the sequent*

$$\Gamma,\, F \,:\, \bigwedge_{i \in I} \boxdot_{m_i}\, \theta_i \,::\, \kappa(F) \;\vdash\; t \,:\, \theta' \,::\, \kappa \tag{6.11}$$

and a branch leading from the root of π to a leaf introducing an occurrence of the non-terminal F whose type in the context of the sequent (6.11) is $\boxdot_{m_i}\, \theta_i$. Then then the App color of this branch is m_i.

Proof. We proceed by induction on t:

- If $t = x$ is a variable, then π simply consists of an Axiom leaf and the context does not contain any non-terminal. Such a case can not occur on a branch leading to a leaf introducing a non-terminal.
- If $t = F$ is a non-terminal, then π simply consists of an Axiom leaf

  $$\text{Axiom}\quad \frac{}{F \,:\, \bigwedge_{\{i\}} \boxdot_{\varepsilon}\, \theta_i \,::\, \kappa \;\vdash\; F \,:\, \theta_i \,::\, \kappa}$$

  The maximal color seen on this branch consisting of a single node is the color of the Axiom rule, that is ε, which coincides with the color of F in the context.
- If $t = \lambda x.\, u$ is an abstraction, since $x \in \mathcal{V}$, the proof π is of the shape

$$\dfrac{\begin{array}{c}\pi' \\ \vdots \\ \Gamma,\, F : \bigwedge_{i\in I} \boxdot_{m_i}\, \theta_i :: \kappa(F)\,,\, x : \bigwedge_{j\in J} \boxdot_{m'_j}\, \theta'_j :: \kappa' \;\vdash\; t : \theta'' :: \kappa'' \end{array} \qquad J \subseteq K}{\Gamma,\, F : \bigwedge_{i\in I} \boxdot_{m_i}\, \theta_i :: \kappa(F) \;\vdash\; \lambda\, x\,.\, t : \left(\bigwedge_{k\in K} \boxdot_{m'_k}\, \theta'_k\right) \to \theta'' :: \kappa' \to \kappa''}$$

where by induction hypothesis the maximal color seen from the root of π' to the occurrence of F of interest is m_i. Since the rule λ is colored with ε, the maximal color seen from the root of π to the occurrence of F of interest is $\max(m_i, \varepsilon) = m_i$, which is the color of the corresponding occurrence in the context of the conclusion of π.

- If $t = u\ v$, then π is of the shape

$$\dfrac{\begin{array}{ccc}\pi' & & \pi'_j \\ \vdots & & \vdots \\ \Gamma \vdash u : \left(\boxdot_{m'_1}\, \theta_1 \;\wedge \cdots \wedge \boxdot_{m'_k}\, \theta_k\right) \to \theta :: \kappa \to \kappa' & \cdots & \Gamma_j \vdash v : \theta_j :: \kappa \quad \cdots \end{array}}{\Gamma \,+\, \boxdot_{m'_1}\Gamma_1 \,+\, \ldots \,+\, \boxdot_{m'_k}\Gamma_k \;\vdash\; u\,v : \theta :: \kappa'}$$

where the occurrence of F of interest is either a leaf of π' or a leaf of π'_j for some j.

  - If it is a leaf of π', then $F : \boxdot_{m_i}\, \theta_i :: \kappa(F) \in \Gamma$. We apply the induction hypothesis to this occurrence of π', and the color of the branch leading to it in π is the same as in π', as the node

$$\Gamma \vdash u : \left(\boxdot_{m'_1}\, \theta_1 \;\wedge \cdots \wedge \boxdot_{m'_k}\, \theta_k\right) \to \theta :: \kappa \to \kappa'$$

    is colored with ε. So the maximal color seen along the branch is m_i and we can conclude.
  - If it is a leaf of π'_j for some j, then the occurrence of F of interest appears in Γ_j with a color m'' which is, by induction hypothesis, the maximal color seen on the path from the root of π_j to the leaf introducing this occurrence. By definition of the Application rule, $m_i = \max(m'_j, m'')$. Consider now the maximal color seen from the root of π to the leaf introducing the occurrence of interest: since the node

$$\Delta_j \vdash v : \theta_j :: \kappa$$

    is attributed color m'_j, it is $\max(m'_j, m'') = m_i$ as well.

□

6.4 Soundness and completeness of our modal system

Once the notion of infinite winning derivation tree explicated, there remains to relate this winning condition to the acceptance condition of alternating parity

automata, by adapting Theorem 16 to the modal coloring policy of the type system $\boldsymbol{\gamma}(\mathcal{A})$. The following theorem establishes a soundness and completeness theorem which relates the winning condition on the infinite derivation trees of $\boldsymbol{\gamma}_{fix}(\mathcal{G}, \mathcal{A})$ to the parity acceptance condition of the automaton $\mathcal{A}$ during its exploration of the infinite tree $\langle\, \mathcal{G} \,\rangle$ generated by the recursion scheme $\mathcal{G}$:

Theorem 19 (Soundness and completeness)**.** *Suppose given a productive recursion scheme $\mathcal{G}$ and an alternating parity automaton $\mathcal{A}$. There exists a winning run-tree of $\mathcal{A}$ over $\langle\, \mathcal{G} \,\rangle$ with initial state q if and only if the sequent*

$$S : \bigwedge_{\{1\}} \boxdot_{\varepsilon}\, q :: o \;\vdash\; S : q :: o \tag{6.12}$$

has a winning derivation tree in the type system $\boldsymbol{\gamma}_{fix}(\mathcal{G}, \mathcal{A})$.

The two following sections are devoted to a summary of the proof together with some remarks about possible extensions or variants, followed by an extension of the type system with a general form of weakening. This general weakening will be required for later connections of the type system with a finitary, colored model of linear logic.

We give the proof of the soundness-and-completeness theorem in two steps: the completeness proof appears in Chapter 7, and Chapter 8 is devoted to the soundness proof. These proofs are based on the ones appearing in Kobayashi and Ong's unpublished journal version [KO] of their original article [KO09]. We give essentially the same proof, but with our modified, modal coloring policy, and we add some explanations to ease the understanding of this complex proof. Although we follow essentially the notations used in the unpublished proof of [KO], our reconstruction and adaptation of the proof of soundness benefited from the nice account given by Haddad in his PhD thesis [Had13b]. We notably emphasize the main challenge of the completeness proof: introducing *optimal* typings for non-terminals, in a way which is deeply related to the fact that the head normalization computes the normal form of higher-order recursion schemes.

6.5 Remarks on soundness and completeness

Let us start this section by giving the outline of the proof of soundness and completeness of the idempotent intersection type system $\boldsymbol{\gamma}_{fix}(\mathcal{G}, \mathcal{A})$ for the higher-order model-checking problem. For technical convenience, the proof is formulated in the game $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$, in which two players exchange typing information about the rules of a higher-order recursion scheme $\mathcal{G}$.

- **Completeness** consists in building a winning strategy for Eve from an accepting run-tree of $\mathcal{A}$ over $\langle\, \mathcal{G} \,\rangle$. This run-tree can be computed by a rewriting relation similar in the spirit to $\rightarrow^{\infty}_{\mathcal{G},\mathcal{A}}$, but which is not considered coinductively – although the relation may be iterated countably to generate the run-tree. From a sequence computing this run-tree, we extract information allowing to type the rules of recursion schemes, so as to generate a winning strategy for Eve in $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$. The point is, to type each symbol in head position, to extract from the reduction sequence the set of types its arguments will actually *need* in the future

– that is, the set of intersection types they will have when they appear in head position in the reduction. The reason of this quest for *minimality*, emphasized by the example of p.147, is that Eve may introduce typings for non-terminals of the recursion scheme which are never actually accessed in the reduction, and introduce in this way loosing plays in $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$, unrelated to the run-tree of $\mathcal{A}$ over $\langle\, \mathcal{G} \,\rangle$. These minimal typings and the associated contexts allow to define a strategy for Eve. This strategy is winning because the color of the type of a non-terminal in a context – and thus, the color associated to Adam's move picking it – is precisely the color of the fragment of the branch of interest the head reduction putting it in head position generates. This allows to relate the maximal color occurring infinitely often along any infinite branch of the run-tree with the maximal color seen infinitely often in the interaction of $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$ which explores it.

- **Soundness** consists in generating a winning run-tree of $\mathcal{A}$ over $\langle\, \mathcal{G} \,\rangle$ from a winning strategy for Eve in $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$. The point is to introduce a rewriting relation whose spirit is

  – to rewrite non-terminals according to the rules of $\mathcal{G}$ and to perform subject reduction on the resulting derivations provided by Eve's strategy,

  – and to extract from these derivations information on the transitions to use to generate the run-tree.

  The fact that the run-tree is winning comes, again, from the fact that coloring is in a sense invariant under β-reduction: the color of an infinite branch may be computed from the colors occurring in the infinite play exploring it. An important lemma, which was missing in the original proof of Kobayashi and Ong [KO09] but is proved in the extended version [KO], states that every infinite branch comes indeed from an infinite play of $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$.

Non-idempotent types. Following the discussion of Chapter 5, a first natural question is to design a non-idempotent variant of $\boldsymbol{\gamma}_{fix}(\mathcal{G}, \mathcal{A})$, and to establish a soundness-and-completeness theorem for it. This would not change the structure of the proof, and in particular not the soundness proof, in which subject reduction would be in fact cleaner. However, as it appears in the completeness proof, when the types of prefixes are defined in §7.4, intersection types with *countable* intersections would be required in the non-idempotent case. The reason is that an argument of a head symbol f may be put in head position infinitely often, as in the example of p.153. It follows that the intersection type of this head symbol f would be described in that case by an infinite intersection. The model of linear logic we introduce in Chapter 9 needs to be infinitary for the same reason: the need for a countable multiplicity for intersection types. Without this infinite multiplicity, the non-idempotent variant of the type system only captures *finite* run-trees of $\mathcal{A}$ over $\langle\, \mathcal{G} \,\rangle$. The infinitary multiplicities occurring in the non-idempotent type system account for the use of a potentially infinite number of subterms in the computation of the run-tree.

Subtyping. Another natural question coming from the discussion on denotational models of Chapter 5 is the one of *subtyping*. Its presence in a denotational model associated to an idempotent type system is related to the fact that while typability is preserved by β-reduction and expansion in such a type system without subtyping, it is not preserved by η-reduction, see for instance Salvati [Sal10] in the context of higher-order model-checking. The addition of subtyping allows to solve this issue, and to obtain a type system invariant modulo both β and η. This is why subtyping is necessary in the intersection type systems corresponding to the semantics of the λ-calculus in the Scott model of linear logic given by Ehrhard in [Ehr12a] and by Terui in [Ter12] – as this denotational model is closed under both β and η reductions and expansions. The direct extension of the theorem of soundness and completeness to a setting with subtyping would probably be technically involved – while the extension of the completeness is immediate, the one of the soundness would require to track precisely occurrences of variables and non-terminals. Indeed, the presence of subtyping implies that an occurrence of a variable or of a non-terminal may appear with a different intersection type in the Axiom leaf that introduces it, and in the context of the initial sequent of the proof of interest. In the sequel, we will circumvent this difficulty by adapting another result of Salvati [Sal10] which states that subtyping can be eliminated in η-long forms, and in particular on the β-normal η-long ones.

It appears, to our knowledge, that subtyping has only been considered marginally in the context of higher-order model-checking, and only as a tool leading to optimizations, as explained by Kobayashi in [Kob13]. The soundness of the extension of the *uncolored* version of the Kobayashi-Ong type system with subtyping was studied by Ong and Ramsay, and appears in the latter's PhD thesis [Ram13].

The lack of subtyping does not affect the equivalence of the existence of a winning strategy for Eve in one of the games, and of the existence of a winning run-tree of the alternating automaton of interest. Indeed, the lack of subtyping breaks the stability of typing modulo η-reduction: if $\Gamma \vdash t : \theta :: \kappa$ and $t \to_\eta t'$, it may be the case that $\Gamma \vdash t' : \theta :: \kappa$ does not hold. However, it is always possible to introduce a different context Γ' such that $\Gamma' \vdash t' : \theta :: \kappa$. The idea is that this context contains types related by subtyping to the ones occurring in Γ. So, if Eve has a winning strategy for a given recursion scheme, she should also has one for a recursion scheme obtained by η-reduction of some rules: informally, she will play some Γ' instead of Γ.

More investigations on subtyping and η-conversion are carried in §10.3, where we clarify the connection of $\mathfrak{T}(\mathcal{A})$ with a type system with idempotent intersections and subtypings.

Relating plays of $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$ and branches of the run-tree over $\langle \mathcal{G} \rangle$. In general, a winning strategy for Eve in $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$ contains plays corresponding to the branches of the run-tree computed by the proof of soundness, but also "unnecessary" plays in which Eve gave to some non-terminals intersection types they can not have when they arrive in head position during the computation. In particular, she may give a non-empty intersection type to a non-terminal which will never occur in head position. In the soundness proof, this does not matter since as long as Eve wins on all her plays, she wins on

the ones corresponding to actual branches of the run-tree. In the completeness proof, we compute a *minimal* strategy, in which only the necessary types are played. In a sense, some order relation on strategies of Eve could be defined to account for this situation. This also means that the soundness proof allows to handle interactions in which the context contains typings of non-terminals which are unnecessary from the point of view of the computation of the run-tree: we may therefore consider, as we do in the next section, a more general form in weakening in which such unnecessary typing information may be added to proofs. The addition of such a general weakening policy is necessary to connect the type system to a finitary model of linear logic.

Towards a coinductive proof. As explained in Chapter 4, we can see λY-terms, and thus higher-order recursion schemes, as finitary representations of infinite *regular* terms. This infinitary nature is rooted in the derivations of the type system $\boldsymbol{\gamma}_{fix}(\mathcal{G},\mathcal{A})$, in which the *fix* rule allows to formulate derivation trees with infinite branches. As such, a proper treatment of these trees would require to define them coinductively, and to see the soundness as a coinductive subject reduction property. In other words, the proof of soundness would compute the winning run-tree of $\mathcal{A}$ over $\langle\,\mathcal{G}\,\rangle$ by an infinitary, coinductive rewriting process on coinductively-defined derivation trees. In this setting, the completeness proof would be of interest: the author conjectures that a form of coinductive subject expansion along $\rightarrow^{\infty}_{\mathcal{G},\mathcal{A}}$ would produce exactly the *minimal* types and environments that the original completeness proof extracts from a reduction sequence. Intuitively, the idea is that by going "backwards", the coinductive process has access to the result of the reduction, and thus the branches it manipulates are only branches appearing in the run-tree over $\langle\,\mathcal{G}\,\rangle$. The use of coinduction would be particularly sensible in the setting of non-idempotent types, where it would probably ease the manipulation of the countable multiplicities appearing in intersection types.

A modal coloring policy. The most important aspect of this soundness-and-completeness theorem in ourself quest for denotational models for higher-order model-checking is that it reveals the hidden, deeply *comonadic* behavior of the coloring of alternating parity tree automata. In Chapter 5, we explained how the intersection operator on types is related to the exponential of linear logic by the fundamental decomposition of the intuitionnistic arrow in linear logic:

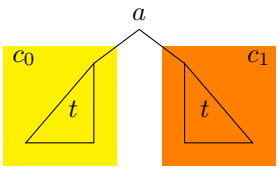
$$A \Rightarrow B \;\; = \;\; !\,A \multimap B$$

In the colored system $\boldsymbol{\gamma}_{fix}(\mathcal{G},\mathcal{A})$, the Application rule is

$$\frac{\Gamma \vdash t : (\boxdot_{m_1}\,\theta_1 \;\wedge \cdots \wedge \boxdot_{m_k}\,\theta_k) \to \theta \;::\; \kappa \to \kappa' \qquad\qquad \Gamma_i \vdash u \;:\; \theta_i \;::\; \kappa}{\Gamma \,\cup\, \boxdot_{m_1}\Gamma_1 \,\cup\, \ldots \,\cup\, \boxdot_{m_k}\Gamma_k \;\;\vdash\;\; t\,u \;:\; \theta :: \kappa'}$$

In addition to the duplication of typings of u the exponential modality of linear logic realizes in the non-colored case, we see that in this colored framework the coloring operation $\boxdot$ acts on each copy, and affects accordingly the contexts contained in these copies. The semantic idea behind the reformulation of $KO_{fix}(\mathcal{G},\mathcal{A})$ into $\boldsymbol{\gamma}_{fix}(\mathcal{G},\mathcal{A})$ is that every symbol of the signature Σ opens indeed a *coloring box* which affects the behavior of the boxed term in the same

way as an exponential modality ! of Girard, or more precisely as a comonadic capability or coeffect [POM14]. In the case of colors and priorities, these coeffects regulate the inductive/coinductive evaluation policy of fixed points in the term. Consider a unary symbol $a \in \Sigma$ such that $\delta(q, a) = (1, q_0) \wedge (1, q_1)$ and set $c_i = \Omega(q_i)$. The application of a term t of appropriate type to a can be understood pictorially as



It should be noted that this figure depicts a very precise semantic concept: the coloring modality is a *parametric comonad* [Mel14b], and the boxes we depict here correspond to Mellies' *functorial boxes* [Mel06b].

This structure of parametric comonad formalizes the fact that these coloring boxes have good algebraic properties: a box of color $\max(c_1, c_2)$ can be duplicated into two nested boxes of respective colors c_1 and c_2, and there is a neutral color ε which intuitively corresponds to the absence of a coloring box, so that such boxes can be removed.

This observation has consequences at the logical level, as well as at the semantic one. It induces that we can extend linear logic with a family of modalities $(\square_c)_{c \in Col}$, such that the sequents

$$\begin{array}{ccc} \square_\varepsilon\ A & \vdash & A \\ \square_{\max(m_1,\, m_2)}\ A & \vdash & \square_{m_1}\ \square_{m_2}\ A \\ \square_m\ A \otimes \square_m\ B & \vdash & \square_m\ (A \otimes B) \end{array}$$

are canonically provable in this extension of linear logic with colors. As we briefly explain in §9.11, it is enough and in fact more accurate for our purpose to consider a colored extension of *tensorial* logic [MT10], in which the coloring modality behaves in the same way.

In order to extend the connection between intersection types, indexed linear logic and the relational semantics, we can extend indexed linear logic with the family of modalities we just mentioned, but in an indexed way. In the resulting logic, the intuitionnistic arrow can be translated as

$$A \Rightarrow B \quad = \quad !_u \square_{\vec{m}}\ A \multimap B$$

where $u : I \to \{1\}$ is a function and $\vec{m} \in Col^I$ is a vector of $|I|$ colors. The right-hand part of this equality can be precisely connected to the colored intersection types of the shape

$$\bigwedge_{i \in I} \boxdot_{m_i}\ \theta_i \ \to\ \theta$$

with additional conditions on the simple types the θ_i and θ refine. While we keep colored indexed linear logic implicit in this work, it acts as a theoretic bridge between colored intersection types and a colored extension of the relational semantics of linear logic, which we define in Chapter 9. Just as in the

non-colored case, we can as well consider the finitary Scott semantics of linear logic, and extend them in a similar way with a coloring modality, as we do in Chapter 10.

In both cases, the point is that $\Box$ is a parametric comonad which can be composed with the usual exponential of linear logic – thanks to a *distributive law* – to give birth to a new, colored exponential, which allows to interpret λ-terms in colored models of linear logic.

It is important to stress that this algebraic structure of the coloring modality would allow to reflect colored intersection types in more than just these two models of linear logic, and notably in models of *game semantics*. More generally, the modal nature of the coloring annotation paves the way towards an extension of dialogue categories [Mel09, Mel16a, Mel12, Mel16b] with a coloring modality and infinite interactions regulated by an inductive-coinductive discipline. Moreover, the existence of a colored extension of tensorial logic is suggesting a connection between the colored, infinitary relational semantics of Chapter 9 and a colored, infinitary game semantics based on dialogue games, extending the connection between tensorial logic and dialogue games investigated by Melliès [Mel12].

Another extension would be to try to accommodate the soundness-and-completeness proof to other effects, in order to describe other winning conditions on alternating tree automata than the parity condition. For the parity condition, an external winning condition is devised on elements of the set Col, whose finite composition is handled by a parametric comonad, and whose infinite composition is performed by a fixpoint operator – the rule *fix* in the type system, and an external fixpoint combinator in the colored models we are to consider in Part III. It would be interesting, and probably challenging, to try to mimic the situation with coeffects such as counters and probabilities – and to try to excavate more generally the conditions the modality $\Box$ and the fixpoint operator need to satisfy for the soundness-and-completeness proof to hold.

6.6 Final remarks on weakening

In this section, we consider the addition of general weakening to the type system $\mathfrak{T}(\mathcal{A})$, and the associated parity game $\mathsf{Adamic}^{st}(\mathcal{G}, \mathcal{A})$. The type system $\mathfrak{T}^{st}(\mathcal{A})$ is obtained from the one of Figure 6.2, by adding two extra rules:

$$Weak \qquad \frac{\Gamma \ \vdash \ t : \theta :: \kappa'}{\Gamma\,,\, x : \emptyset :: \kappa \ \vdash \ t : \theta :: \kappa'} \qquad (x \notin dom(\Gamma))$$

$$Weak_c \qquad \frac{\Gamma\,,\, x : \bigwedge_{i\in I} \boxdot_{m_i} \theta_i :: \kappa \ \vdash \ t : \theta :: \kappa' \qquad I \subseteq J}{\Gamma\,,\, x : \bigwedge_{j\in J} \boxdot_{m_j} \theta_j :: \kappa \ \vdash \ t : \theta :: \kappa'}$$

We do not add an explicit contraction rule, as this would be redundant: contraction is already performed by the Application rule, by the $Weak_c$ rule, and can be eliminated at other nodes of proofs.

We also modify the rule λ of $KO(\mathcal{A})$ and of $\mathfrak{T}(\mathcal{A})$: there is no longer need to introduce weakening in this rule in $\mathfrak{T}^{st}(\mathcal{A})$, as it is handled by the general rules we added. We therefore consider a usual abstraction rule in $\mathfrak{T}^{st}(\mathcal{A})$:

$$\lambda \qquad \frac{\Gamma\,,\, x : \bigwedge_{i\in I} \boxdot_{m_i} \theta_i :: \kappa \;\vdash\; t : \theta :: \kappa'}{\Gamma \;\vdash\; \lambda x\,.\,t : \left(\bigwedge_{i\in I} \boxdot_{m_i} \theta_i\right) \to \theta :: \kappa \to \kappa'} \qquad (x \in \mathcal{V})$$

In comparison with Kobayashi and Ong's approach, the addition of explicit weakening rules, and especially of the rule $Weak$, leads to a cleaner system. Indeed, the rule λ of $KO(\mathcal{A})$ (and of $\boldsymbol{\gamma}(\mathcal{A})$) can abstract a variable which does not occur in the context, simply because this is considered to be equivalent to occurring with an empty set of typings in the context. In $\boldsymbol{\gamma}^{st}(\mathcal{A})$, we make explicit the introduction of such variables, and allow a more general form of weakening, as it does not necessarily require to abstract the variables which were weakened. We call $\mathsf{Adamic}^{st}(\mathcal{G},\mathcal{A})$ the variant of $\mathsf{Adamic}(\mathcal{G},\mathcal{A})$ where Eve plays by constructing proofs in $\boldsymbol{\gamma}^{st}(\mathcal{A})$. We prove the following soundness and completeness theorem by reduction to the one for $\mathsf{Adamic}(\mathcal{G},\mathcal{A})$:

Theorem 20. *Given a productive higher-order recursion scheme $\mathcal{G}$ and an alternating parity automaton $\mathcal{A}$, there is a winning execution of $\mathcal{A}$ over $\mathcal{G}$ if and only if Eve has a winning strategy in the game* $\mathsf{Adamic}^{st}(\mathcal{G},\mathcal{A})$.

Again, an alternative statement of this theorem can be given using the type system $\boldsymbol{\gamma}^{st}_{fix}(\mathcal{G},\mathcal{A})$ where $\boldsymbol{\gamma}^{st}(\mathcal{A})$ is extended with the *fix* rule. The completeness proof follows directly from Theorem 19, since it constructs a proof in $\boldsymbol{\gamma}(\mathcal{A})$ which is easily translated into a proof of $\boldsymbol{\gamma}^{st}(\mathcal{A})$ by adding the necessary structural rules before using the rule λ. For the soundness proof, consider a winning strategy σ for Eve in $\mathsf{Adamic}^{st}(\mathcal{G},\mathcal{A})$. We define a winning strategy $\sigma^\dagger$ in $\mathsf{Adamic}(\mathcal{G},\mathcal{A})$ as follows: for every answer of Eve to a move of Adam, the context she plays is justified by a proof π in $\boldsymbol{\gamma}^{st}(\mathcal{A})$, whose conclusion is a sequent of the form

$$\Gamma \;\vdash\; \mathcal{R}(F) : \theta :: \kappa \tag{6.13}$$

If this proof does not contain weakenings, nothing has to be done. Else, the proof can be rewritten by commutation of structural rules: weakenings can be brought to the bottom of the proof, unless they affect a variable which is abstracted, in which case they can not be brought below the rule which abstracts them. For instance, we can locally rewrite

$$Weak_c\;\frac{\lambda\;\dfrac{\Gamma\,,\, x : \bigwedge_{i\in I} \boxdot_{m_i} \theta_i :: \kappa\,,\, y : \bigwedge_{k\in K} \boxdot_{m'_k} \theta'_k :: \kappa' \;\vdash\; t : \theta :: \kappa'' \qquad I \subseteq J}{\Gamma\,,\, x : \bigwedge_{j\in J} \boxdot_{m_j} \theta_j :: \kappa\,,\, y : \bigwedge_{k\in K} \boxdot_{m'_k} \theta'_k :: \kappa' \;\vdash\; t : \theta :: \kappa''}}{\Gamma\,,\, x : \bigwedge_{j\in J} \boxdot_{m_j} \theta_j :: \kappa \;\vdash\; \lambda y\,.\,t : \left(\bigwedge_{k\in K} \boxdot_{m'_k} \theta'_k\right) \to \theta :: \kappa' \to \kappa''}$$

by exchanging these two rules, as it leads to a block with the same hypothesis and conclusion:

$$\lambda\;\frac{Weak_c\;\dfrac{\Gamma\,,\, x : \bigwedge_{i\in I} \boxdot_{m_i} \theta_i :: \kappa\,,\, y : \bigwedge_{k\in K} \boxdot_{m'_k} \theta'_k :: \kappa' \;\vdash\; t : \theta :: \kappa''}{\Gamma\,,\, x : \bigwedge_{i\in I} \boxdot_{m_i} \theta_i :: \kappa \;\vdash\; \lambda y\,.\,t : \left(\bigwedge_{k\in K} \boxdot_{m'_k} \theta'_k\right) \to \theta :: \kappa' \to \kappa''} \qquad I \subseteq J}{\Gamma\,,\, x : \bigwedge_{j\in J} \boxdot_{m_j} \theta_j :: \kappa \;\vdash\; \lambda y\,.\,t : \left(\bigwedge_{k\in K} \boxdot_{m'_k} \theta'_k\right) \to \theta :: \kappa' \to \kappa''}$$

but we can not commute the rules in the following situation:

$$Weak_c\ \lambda\ \dfrac{\dfrac{\Gamma\,,\,x\,:\,\bigwedge_{i\in I}\boxdot_{m_i}\,\theta_i\,::\,\kappa\ \vdash\ t\,:\,\theta\,::\,\kappa' \qquad I\subseteq J}{\Gamma\,,\,x\,:\,\bigwedge_{j\in J}\boxdot_{m_j}\,\theta_j\,::\,\kappa\ \vdash\ t\,:\,\theta\,::\,\kappa'}}{\Gamma\ \vdash\ \lambda x\,.\,t\,:\,\left(\bigwedge_{j\in J}\boxdot_{m_j}\,\theta_j\right)\to\theta\,::\,\kappa\to\kappa'}$$

In the same way, a rule $Weak$ can be brought down in the proof, but never lower than a λ rule abstracting it, or than a $Weak_c$ applied to the variable it introduces. When two $Weak_c$ rules affecting the same variable occur in a row, they can be contracted to a unique one. This flexibility of the position of structural rules in a proof allows us to rewrite the proof π of conclusion (6.13) to a proof π' of same conclusion, and therefore justifying the same move for Eve, but in which the structural rules occur

- at the bottom of the proof, if they apply to variables which are not abstracted,
- or just before the λ rule abstracting the variable they weaken.

To translate π' to a proof of $\gamma(\mathcal{A})$, we

- remove all the weakenings occurring at the bottom of the proof, obtaining in this way a proof of typing of a sequent

$$\Gamma'\ \vdash\ \mathcal{R}(F)\,:\,\theta\,::\,\kappa \tag{6.14}$$

  typing the same term as (6.13), and with the same type, but from a subcontext Γ' of Γ,
- and we use the fact that in $\gamma(\mathcal{A})$ the abstraction rule contains weakening on the abstracted variable to translate a block as

$$Weak_c\ \lambda\ \dfrac{\dfrac{\Gamma\,,\,x\,:\,\bigwedge_{i\in I}\boxdot_{m_i}\,\theta_i\,::\,\kappa\ \vdash\ t\,:\,\theta\,::\,\kappa' \qquad I\subseteq J}{\Gamma\,,\,x\,:\,\bigwedge_{j\in J}\boxdot_{m_j}\,\theta_j\,::\,\kappa\ \vdash\ t\,:\,\theta\,::\,\kappa'}}{\Gamma\ \vdash\ \lambda x\,.\,t\,:\,\left(\bigwedge_{j\in J}\boxdot_{m_j}\,\theta_j\right)\to\theta\,::\,\kappa\to\kappa'}$$

  in $\gamma^{st}(\mathcal{A})$ to

$$\lambda\ \dfrac{\Gamma\,,\,x\,:\,\bigwedge_{i\in I}\boxdot_{m_i}\,\theta_i\,::\,\kappa\ \vdash\ t\,:\,\theta\,::\,\kappa' \qquad I\,\subseteq\,J}{\Gamma\ \vdash\ \lambda x\,.\,t\,:\,\left(\bigwedge_{j\in J}\boxdot_{m_j}\,\theta_j\right)\to\theta\,::\,\kappa\to\kappa'} \qquad (x\in\mathcal{V})$$

  in $\gamma(\mathcal{A})$.

This process defines a proof $\pi^\dagger$ in $\gamma(\mathcal{A})$ of the sequent (6.14). In σ, Eve plays Γ according to the proof π; we define $\sigma^\dagger$ by letting her play the subcontext Γ' of Γ, according on the proof $\pi^\dagger$. This potentially reduces the set of moves Adam could play, but does not introduce any new move, so that we can iterate the construction of $\sigma^\dagger$ by considering all potential moves of Adam and extracting from σ a strategy playing without weakening. This defines $\sigma^\dagger$, which is a winning strategy as every play in which Eve follows this strategy can be seen as a play where she follows the winning strategy σ.

Chapter 7

Completeness of the type system

In this chapter, adapting the original proof of Kobayashi and Ong [KO], we establish the completeness part of the soundness-and-completeness theorem (Theorem 19). For convenience, we express it in an equivalent way using the parity game $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$:

> Let $\mathcal{A}$ be an alternating parity automaton and $\mathcal{G}$ a productive higher-order recursion scheme. If $\mathcal{A}$ has a winning execution over $\langle\, \mathcal{G} \,\rangle$, then Eve has a winning strategy in $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$.

We start by explaining the main idea of the proof completeness in §7.1, using representations of plays as infinitary typing derivations in $\boldsymbol{\gamma}_{fix}(\mathcal{G}, \mathcal{A})$. The crucial idea is that Eve should play in an « optimal » or « parsimonious » way, in order to let Adam pick only non-terminals unfolding to subterms which will actually be explored during the evaluation of the recursion scheme of interest. In other words, Eve should proceed in such a way that Adam does not have too much freedom. We then formalize the proof by adapting the original proof of Kobayashi and Ong to the new modal coloring policy underlying $\boldsymbol{\gamma}_{fix}(\mathcal{G}, \mathcal{A})$ and $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$. We begin by defining in §7.2 a rewriting system (with rewriting relation noted $\blacktriangleright$) which computes a run-tree $\mathcal{T}_{run}$ of $\mathcal{A}$ over $\langle\, \mathcal{G} \,\rangle$ by performing the normalization of $\mathcal{G}$ at the same time as the execution of $\mathcal{A}$. The rewriting relation $\blacktriangleright$ on $\mathcal{G}$ works in the same spirit as the relation $\rightarrow^{\infty}_{\mathcal{G},\mathcal{A}}$ designed in §4.4, since it also depends on the automaton $\mathcal{A}$. We carry on and define in §7.3 a notion of color on the tree contexts produced during this rewriting process. We explain in §7.4 how to deduce from a rewriting sequence $\blacktriangleright$ intersection types θ for the prefixes t of the terms occurring in all the reduction, as well as contexts Γ such that $\Gamma \vdash t : \theta :: \kappa$ holds in $\boldsymbol{\gamma}(\mathcal{A})$. After introducing these necessary definitions, we give an overview of the proof in §7.5. We prove three preliminary lemmas in §7.6, and then the completeness theorem in §7.7.

7.1 Optimal typings in an infinitary framework

While it is technically useful to prove the completeness of the type system $\boldsymbol{\gamma}(\mathcal{A})$ in the parity game $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$, we first give some informal explanations about the proof in the type system $\boldsymbol{\gamma}_{fix}(\mathcal{G}, \mathcal{A})$, as it allows a convenient representation of strategies. In $\boldsymbol{\gamma}_{fix}(\mathcal{G}, \mathcal{A})$, we can see (winning) run-trees of $\mathcal{A}$ over $\langle\, \mathcal{G} \,\rangle$ as

(winning) derivations typing the sequent

$$\emptyset \ \vdash \ \langle\, \mathcal{G} \,\rangle \,:\, q_0 \,::\, o$$

using only the rules Application and δ, in the spirit of Proposition 10. So the idea is, as in Chapter 5, to use the information contained in such a derivation to reconstruct a derivation of the sequent

$$S \,:\, \bigwedge_{\{1\}} \boxdot_{\varepsilon} \, q_0 \,::\, o \ \vdash \ S \,:\, q_0 \,::\, o$$

using subject expansion. However, we need to take good care of the types we introduce for non-terminals. Consider indeed the signature $\Sigma = \{\, a \,:\, 1,\, b \,:\, 1 \,\}$, and the recursion scheme

$$\mathcal{G} \quad = \quad \left\{ \begin{array}{lcl} S & = & F\ H \\ F & = & \lambda x.\, a\ (\, F\ x\,) \\ H & = & b\ H \end{array} \right.$$

over this signature Σ. Its normalization produces the infinite tree (which we represent as a word, as it is defined using only constants of arity one):

$$\langle\, \mathcal{G} \,\rangle \ = \ a\, a\, a \cdots$$

Using the rewriting relation $\rightarrow_{\mathcal{G}}$ a finite number of times produces

$$S \ \rightarrow_{\mathcal{G}}^{*} \ a \cdots a\ F\ (\, b \cdots b\ H\,)$$

but when the rewriting is iterated infinitely, the sequence of b is "postponed to infinity". Notice that the head reduction never rewrites the « useless » non-terminal H. We need to reflect in the type of F the fact that it does not actually use its argument, else we could obtain a loosing derivation even in a case where $\langle\, \mathcal{G} \,\rangle$ is accepted by the automaton of interest, as we shall explain now. Consider the alternating parity automaton $\mathcal{A}$ with two states q_0 and q_1, whose transition function is

$$\begin{array}{lcl} \delta(q_0, a) & = & (1, q_0) \\ \delta(q_0, b) & = & (1, q_1) \\ \delta(q_1, b) & = & (1, q_1) \end{array}$$

and whose coloring function is defined by $\Omega(q_0) = 0$ and $\Omega(q_1) = 1$. We depict in Figure 7.1 a derivation of $\mathfrak{Y}_{fix}(\mathcal{G}, \mathcal{A})$ in which the typing of F indicates that it uses its argument once; this enables us to introduce in the derivation a loosing branch (of color 1) expanding H infinitely, while this is unnecessary as the head reduction will never expand H. To compute the color of the two infinite branches, note that we have two choices by Theorem 18:

- we can read the colors of all the boxes we cross on Application rules, and take the maximal one among the ones occurring infinitely often,
- or we can read the colors introduced by the *fix* rules expanding the occurrences of the non-terminals appearing in the branch, and take the maximal one occurring infinitely often.

Along the leftmost branch, corresponding to the *fix* rules applied to S and then infinitely to F, two colors occur infinitely often: the neutral color ε, and the even color 0. It follows that this branch has color 0, and is therefore winning. Along the rightmost branch, unfolding S and then infinitely H, we see the color 0 only once, and then infinitely often the color 1. The color of this branch is thus 1, and the branch is loosing. As a consequence, the derivation of Figure 7.1 is loosing, even though the recursion scheme $\mathcal{G}$ it types generates a tree $\langle\, \mathcal{G} \,\rangle$ which is accepted by $\mathcal{A}$.

In fact, we should *not* explore the rightmost branch in the derivation. We should somehow be aware that F *never* uses its argument, and design accordingly the *optimal*, *winning* derivation presented in Figure 7.2. The whole point of the completeness proof is to introduce types only for non-terminals which will occur in head position at some step of the rewriting process, and thus be actually unfolded using the rules of the recursion scheme.

7.2 Partial run-trees and their computation

After these preliminary considerations, we may now adapt the proof of Kobayashi and Ong [KO] to our modal variant $\mathbf{\gamma}(\mathcal{A})$ of their original type system $KO(\mathcal{A})$.

In the sequel, we consider a *productive* higher-order recursion scheme $\mathcal{G} = \langle \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$, an alternating parity automaton $\mathcal{A} = \langle \Sigma,\, Q,\, \delta,\, q_0,\, \Omega \rangle$ having a winning run-tree over $\langle\, \mathcal{G} \,\rangle$, and we fix such a winning run-tree $\mathcal{T}_{run}$. We also set $\mathcal{N} = \{F_1, \ldots, F_n\}$.

Recall that the run-tree $\mathcal{T}_{run}$ of $\mathcal{A}$ over $\langle\, \mathcal{G} \,\rangle$ can be computed by the coinductive relation

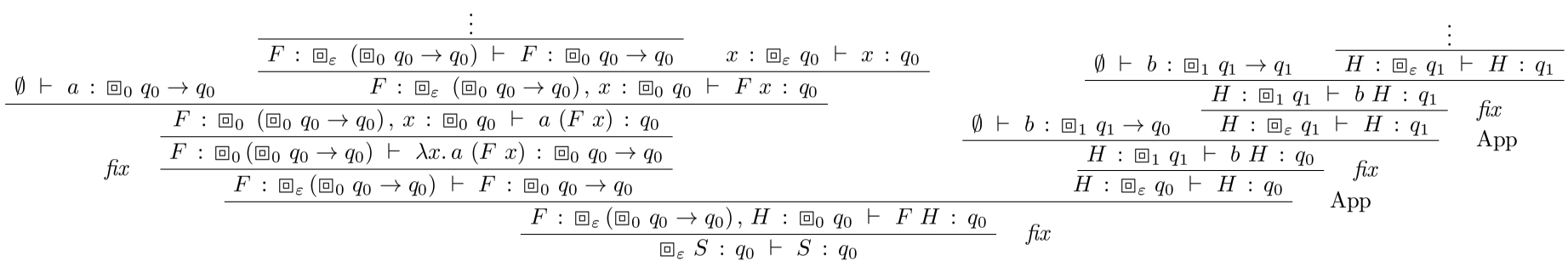
$$\frac{t \;\to^{*}_{\mathcal{G},w}\; a\; t_1\; \cdots\; t_n \qquad\qquad t_i : q_{i,j} \;\to^{\infty}_{\mathcal{G},\mathcal{A}}\; t'_i : q_{i,j}}{t : q \;\to^{\infty}_{\mathcal{G},\mathcal{A}}\; (a\; (t'_1 : (1, q_{1,1}))\; \cdots (t'_1 : (1, q_{1,k_1}))\; \cdots\; (t'_n : (n, q_{n,1}))\; \cdots\; (t'_n : (n, q_{n,k_n}))) : q}$$

introduced in §4.4. For technical convenience, and following [KO], we define a set of labels Σ_{comp} and a rewrite relation $\blacktriangleright$ on Σ_{comp}-labeled unranked trees which is essentially a sequential counterpart to $\to^{\infty}_{\mathcal{G},\mathcal{A}}$. The set of labels Σ_{comp} also contains more than just terms and states, in order to ease the localization of occurrences of non-terminals during the rewriting process.

Let n be the maximal arity of symbols of the signature Σ. The set of labels Σ_{comp} contains the symbols:

- $\langle\, \alpha,\, q \,\rangle$, where $\alpha \in \{1, \ldots, n\}$ and $q \in Q$. Recall the definition of a run-tree (Def. 11 on page 49): in the rewriting process, these labels will come from $\mathcal{T}_{run}$.

- $\langle\, \beta,\, l,\, t,\, q \,\rangle$, where $\beta \in (\{1, \ldots, n\} \times \mathbb{N})^*$ is a sequence of pairs of natural numbers used to identify uniquely each leaf introduced in the reduction, l is a natural number counting the rewriting steps, t is a λ-term without abstractions over $\Sigma \uplus \mathcal{N}$, and $q \in Q$ is a state of the automaton.



To ease reading, we omit the singleton intersection $\bigwedge_{\{1\}}$ in types and contexts, as well as the simple types as they are immediate to reconstruct.

In this derivation, we assume that the *fix* rules justifying coinductively the typings $F : \boxdot_\varepsilon\, (\boxdot_0\, q_0 \to q_0) \vdash F : \boxdot_0\, q_0 \to q_0$ and $H : \boxdot_\varepsilon\, q_1 \vdash H : q_1$ always reintroduce the same "finite piece of derivation" in $\mathfrak{Y}(\mathcal{A})$.

Figure 7.1: A loosing derivation for a recursion scheme whose value tree is accepted by the automaton of interest.

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\emptyset \vdash a : \boxdot_0\, q_0 \to q_0 \qquad \dfrac{\dfrac{\vdots}{F : \boxdot_\varepsilon\, (\emptyset \to q_0) \vdash F : \emptyset \to q_0}}{F : \boxdot_\varepsilon\, (\emptyset \to q_0)\, ,\, x : \emptyset \vdash F\, x : q_0}}{F : \boxdot_0\, (\emptyset \to q_0)\, ,\, x : \emptyset \vdash a\, (F\, x) : q_0}}{F : \boxdot_0\, (\emptyset \to q_0) \vdash \lambda x.\, a\, (F\, x) : \emptyset \to q_0}\;\textit{fix}}{F : \boxdot_\varepsilon\, (\emptyset \to q_0) \vdash F : \emptyset \to q_0}\;\text{App}}{F : \boxdot_\varepsilon\, (\emptyset \to q_0) \vdash F\, H : q_0}\;\textit{fix}}{\boxdot_\varepsilon\, S : q_0 \vdash S : q_0}
$$

To ease reading, we omit the singleton intersection $\bigwedge_{\{1\}}$ in types and contexts, as well as the simple types as they are immediate to reconstruct. We write $\emptyset$ in the type of F and of x to indicate the intersection of an empty set of refined types.
In this derivation, we assume that the *fix* rules justifying coinductively the typings $F : \boxdot_\varepsilon\, (\boxdot_0\, q_0 \to q_0) \vdash F : \boxdot_0\, q_0 \to q_0$ always reintroduce the same "finite piece of derivation" in $\Upsilon(\mathcal{A})$. Note that the Application rule used to deconstruct $F\ H$ does *not* explore H, as the typing of F indicates that it should not be considered – and similarly for $F\ x$.

Figure 7.2: A winning derivation, obtained by typing only subterms reached by the head reduction of the recursion scheme.

Σ_{comp}-labeled unranked trees will be used to compute finite prefixes of $\mathcal{T}_{run}$. The inner nodes will be of the form $\langle\, \alpha,\, q\,\rangle$, and the leaves will be of either shape:

- a leaf of the form $\langle\, \alpha,\, q\,\rangle$ corresponds to a leaf of $\mathcal{T}_{run}$, so that the Σ_{comp}-labeled tree we consider contains the whole finite branch of $\mathcal{T}_{run}$ ending on this leaf,
- a leaf of the form $\langle\, \beta,\, l,\, t,\, q\,\rangle$ does not correspond to a node of $\mathcal{T}_{run}$, but its head normalization (as defined below) will produce one.

Given a sequence of pairs $\beta = (m_1, n_1) \cdots (m_k, n_k) \in (\{1, \ldots, n\} \times \mathbb{N})^*$, we define $\mathit{fst}(\beta) = m_1 \cdots m_k$. We then define the head rewriting relation $\blacktriangleright$ on Σ_{comp}-labeled unranked trees by induction:

1. If $F \in \mathcal{N}$ is a non-terminal which rewrites to $\mathcal{R}(F) = \lambda x_1 \ldots \lambda x_n.\, t$ then

   $$\langle\, \beta,\, l,\, F\, t_1 \ldots t_n,\, q\,\rangle \quad \blacktriangleright \quad \langle\, \beta,\, l+1,\, t[x_i \leftarrow t_i],\, q\,\rangle$$

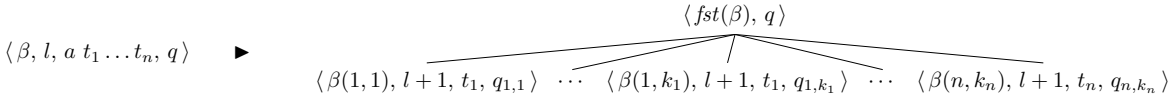
   In other words, to compute the label of the node of $\mathcal{T}_{run}$ identified by β, we rewrite F and perform the necessary substitutions; and we increment the counter l.

2. When a leaf is of the form $\langle\, \beta,\, l,\, a\, t_1 \ldots t_n,\, q\,\rangle$, the head rewriting process computed the symbol a labeling the node located by β, and can therefore forget the additional information about it since it will not be reduced anymore: it only keeps $\langle\, \alpha,\, q\,\rangle$, where $\alpha = \mathit{fst}(\beta)$.

Then it starts the computation of the children of this node, by reading from $\mathcal{T}_{run}$ which transition of $\mathcal{A}$ it should use. So, if the children of $\langle\, fst(\beta),\, q\,\rangle$ in $\mathcal{T}_{run}$ are

$$\langle\, \alpha 1,\, q_{1,1}\,\rangle,\ \ldots,\ \langle\, \alpha 1,\, q_{1,k_1}\,\rangle,\ \ldots,\ \langle\, \alpha n,\, q_{n,1}\,\rangle,,\ \ldots,\ \langle\, \alpha n,\, q_{n,k_n}\,\rangle$$

we open accordingly new leaves to be rewritten:



3. Defining tree contexts in the usual way by the grammar

$$C \quad ::= \quad [\,] \mid \langle\, \alpha,\, q\,\rangle\ T_1 \cdots T_{i-1}\ C\ T_{i+1} \cdots T_k$$

where the T_j are Σ_{comp}-labeled unranked trees and $[\,]$ is a distinguished hole symbol, we extend the rewriting relation to contexts: if $t \blacktriangleright t'$, then for any context C we have $C[t] \blacktriangleright C[t']$.

Note that this does not break the fact that the rewriting is restricted to terms occurring in head position, due to the structure of the trees and contexts we consider.

For a Σ_{comp}-labeled unranked tree T, we define the Σ_{comp}-labeled unranked tree $T^\sharp$ by relabeling the leaves of T of the form $\langle\, \beta,\, l,\, t,\, q\,\rangle$ with $\langle\, fst(\beta),\, q\,\rangle$. Since $\blacktriangleright$ is essentially an inductive presentation of a sequentialization of the coinductive rewriting relation $\rightarrow^{\infty}_{\mathcal{G},\mathcal{A}}$, it is easy to see that if

$$\langle\, \varepsilon,\, 0,\, S,\, q_0\,\rangle \quad \blacktriangleright^* \quad T$$

then $T^\sharp$ is a subtree of $\mathcal{T}_{run}$. Moreover, there is a fair (that is, eventually reducing every occurrence that can be rewritten) and possibly infinite sequence generating $\mathcal{T}_{run}$:

$$T_0 \ = \ \langle\, \varepsilon,\, 0,\, S,\, q_0\,\rangle \ \blacktriangleright\ T_1 \ \blacktriangleright\ T_2 \ \blacktriangleright\ \cdots \tag{7.1}$$

We fix such a sequence in the sequel, and will use it to define the appropriate types for variables and non-terminals. This sequence will provide us with the necessary information on the subterms which eventually occur in head position during the head reduction, preventing us to consider typings exploring unnecessary – and thus potentially loosing – branches as in Figure 7.1.

In the sequel, all the applications of the rewriting rule $\blacktriangleright$ we consider come from the fixed rewriting sequence (7.1). More precisely, if we write

$$\langle\, \beta,\, l,\, t,\, q\,\rangle \quad \blacktriangleright^* \quad C'[\,\langle\, \beta',\, l',\, t',\, q'\,\rangle\,]$$

then we mean that there exists two integers i, j and a context C such that

$$T_i \quad = \quad C[\,\langle\, \beta,\, l,\, t,\, q\,\rangle\,]$$

and that

$$T_j \quad = \quad C[\,C'[\,\langle\, \beta',\, l',\, t',\, q'\,\rangle\,]\,]$$

where T_i and T_j appear in (7.1).

For technical reasons, we consider that each subterm is identified by some implicit label, so that we can distinguish between different occurrences of a same term. The key point is that when we write

$$\langle\, \beta,\, l,\, t_0\ t_1\, q \,\rangle \quad \blacktriangleright^* \quad C'[\langle\, \beta',\, l',\, t_1\ t_2,\, q' \,\rangle]$$

we assume that t_1 is the *same* subterm in both leaves, that is, that the reduction of $t_0\ t_1$ put its argument t_1 in head position.

7.3 Color of a context and modal boxes

We consider contexts in which the hole is attributed a state: we write $[\,]_q$ for a hole which is to receive either a tree of root $\langle\, \alpha,\, q \,\rangle$, or a leaf $\langle\, \beta,\, l,\, t,\, q \,\rangle$. We then define the color $\Omega(C[\,]_q)$ of the context $C[\,]_q$ in the following way:

- if $C[\,]_q \;=\; [\,]_q$, then $\Omega(C[\,]_q) \;=\; \varepsilon$,
- if $C[\,]_q \;=\; \langle\, \alpha,\, q'' \,\rangle\ T_1 \cdots T_{i-1}\ [C'[\,]_q]_{q'}\ T_{i+1} \cdots T_n$, then

$$\Omega(C[\,]_q) \;=\; \max\left(\Omega(q'),\, \Omega\left(C'[\,]_q\right)\right)$$

  Note that in the case where $C'[\,]_q \;=\; [\,]_q$ we have $q \;=\; q'$ and $\Omega(C[\,]_q) \;=\; \Omega(q)$.

This definition differs from the original one by Kobayashi and Ong in [KO09], formalized more precisely in [KO], in which $\Omega([\,]_q) \;=\; \Omega(q)$. This is a direct consequence of our change of coloring policy: just as emphasized by Proposition 16, the color we compute on a finite branch of a tree is the maximal color we *cross*. Each leaf $\langle\, \alpha,\, \beta \,\rangle$ of the run-tree $\mathcal{T}_{run}$, whose children are

$$\langle\, \alpha 1,\, q_{1,1} \,\rangle,\ \ldots,\ \langle\, \alpha 1,\, q_{1,k_1} \,\rangle,\ \ldots,\ \langle\, \alpha n,\, q_{n,1} \,\rangle,,\ \ldots,\ \langle\, \alpha n,\, q_{n,k_n} \,\rangle$$

should be very precisely understood in our approach as opening a *coloring box* of color $\Omega(q_{i,j})$ on its successor in direction (i, j). Since the empty context does not create any box, we consider it of neutral color. This allows us to consider a type system in which the coloring of the Axiom and δ rules obeys the principles of a modal logic. Note also that our definition does not take into account the color of the state of the root of the tree: consider for instance the color of the branch

$$\begin{array}{lcl}
\Omega\left(\langle\, \varepsilon,\, q_0 \,\rangle\ \left(\langle\, 1,\, q_1 \,\rangle\ \left(\langle\, 11,\, q_2 \,\rangle\left([\,]_{q_3}\right)\right)\right)\right) & = & \max\left(\Omega(q_1),\, \Omega\left(\langle\, 1,\, q_1 \,\rangle\ \left(\langle\, 11,\, q_2 \,\rangle\left([\,]_{q_3}\right)\right)\right)\right) \\
& = & \max\left(\Omega(q_1),\, \Omega(q_2),\, \Omega\left(\langle\, 11,\, q_2 \,\rangle\left([\,]_{q_3}\right)\right)\right) \\
& = & \max\left(\Omega(q_1),\, \Omega(q_2),\, \Omega(q_3),\, \Omega\left([\,]_{q_3}\right)\right) \\
& = & \max\left(\Omega(q_1),\, \Omega(q_2),\, \Omega(q_3),\, \varepsilon\right) \\
& = & \max\left(\Omega(q_1),\, \Omega(q_2),\, \Omega(q_3)\right)
\end{array}$$

So, *the color of the state of the root is not taken into account.* The deep idea, underlying the developments of Part III, is that coloring is a *coeffect* – or semantically what Melliès calls a *parametric comonad* in [Mel14b, Mel06b]. This means that *only symbols occurring as arguments* can receive a color. However, this does not matter thanks to the following easy proposition:

Proposition 17. *Let $C_1[C_2[\,]_q]_{q'}$ be a context, then*

$$\Omega\left(C_1[C_2[\,]_q]_{q'}\right) \;=\; \max\left(\Omega\left(C_1[\,]_{q'}\right), \Omega\left(C_2[\,]_q\right)\right)$$

In fact, when the context C_1 opens the hole $[\,]_{q'}$, it opens a box of color $\Omega(q')$ which is taken into account when computing the color of the context. Therefore it does not matter not to consider the color of the root of $C_2[\,]_q$: we already counted it in $C_1[\,]_{q'}$. When considering an infinite branch factorized as infinitely many finite contexts, the only color that is ignored is the one of the root – but this has no impact on the parity condition. This is the key point why the proof by Kobayashi and Ong can be adapted so smoothly to our new coloring policy.

7.4 Using prefixes to reconstruct the contexts

In the previous sections, we defined a rewriting system computing the run-tree $\mathcal{T}_{run}$ of $\mathcal{A}$ over $\langle\, \mathcal{G} \,\rangle$ by the possibly infinite reduction sequence (7.1), defined as:

$$T_0 \;=\; \langle\, \varepsilon,\, 0,\, S,\, q_0 \,\rangle \;\blacktriangleright\; T_1 \;\blacktriangleright\; T_2 \;\blacktriangleright\; \cdots$$

This sequence computes $\mathcal{T}_{run}$ by considering the sequence of head reductions of $\mathcal{G}$, performed as the same time as the exploration of the tree by the automaton $\mathcal{A}$. Recall from our remarks in §7.1 that defining an optimal strategy for Eve requires to type the rules of $\langle\, \mathcal{G} \,\rangle$ in a « optimal » or « parsimonious » way. This means that a subterm appearing as an argument of an application should only receive the intersection types it will encounter when it appears in head position during the sequence of head reductions computing $\langle\, \mathcal{G} \,\rangle$. To pave the way for the definition of an optimal strategy for Eve, we formally define the « parsimonious » types and contexts for subterms occurring in prefix position in the rewriting sequence (7.1) computing $\mathcal{T}_{run}$.

Type of a prefix. These preliminary definitions being given, our goal is now to extract from the fixed rewriting sequence (7.1) an appropriate set of typings for the rules of the higher-order recursion scheme $\mathcal{G}$. We start by attributing an intersection type to each term t_0 occurring as a *prefix* of a term t, that is to each term t_0 such that $t \;=\; t_0\; t_1 \cdots t_k$. Note that, for $k \;=\; 0$, this will allow to type t as well. For each leaf $\langle\, \beta,\, l,\, t,\, q \,\rangle$ of a tree occurring in the reduction sequence (7.1), we attribute to every prefix t_0 of t the type $\theta_{(t_0,\beta,l)}$ refining the simple type of t_0, by induction on this simple type:

1. If $t_0 \;::\; o$, we set $\theta_{(t_0,\beta,l)} \;=\; q$, as the leaf we consider is $\langle\, \beta,\, l,\, t_0,\, q \,\rangle$.
2. If $t_0 \;::\; \kappa_1 \;\to\; \cdots \;\to\; \kappa_n \;\to\; o$, the leaf we consider is of the form $\langle\, \beta,\, l,\, t_0\; t_1 \cdots t_n,\, q \,\rangle$. As emphasized by the examples given in Figure 7.1 and Figure 7.2, the point is to precisely give to every argument the set of types it will need to have when appearing in head position later in the reduction, and no more, as this may introduce unnecessary loosing branches.

   We therefore consider the set S_j of types $\boxdot_{\Omega\left(C'[\,]_{q'}\right)}\, \theta_{(t_j,\beta',l')}$ such that

$$\langle\, \beta,\, l,\, t_0\; t_1 \cdots t_n,\, q \,\rangle \;\blacktriangleright^*\; C'[\,\langle\, \beta',\, l',\, t_j\; \widetilde{t'},\, q' \,\rangle\,] \tag{7.2}$$

where $\widetilde{t'}$ is a vector of terms we do not need to explicitly consider. The set S_j is well-defined, as the simple type κ_j is lesser than the simple type κ, so that we can use the induction hypothesis to obtain each $\theta_{(t_j,\beta',l')}$.

Note also that there may be infinitely many trees of the form

$$C'[\,\langle\,\beta',\,l',\,t_j\;\widetilde{t'},\,q'\,\rangle\,]$$

obtained via the rewriting (7.2). This may occur, for instance, when evaluating a recursion scheme

$$\left\{\begin{array}{lcl} S & = & F\;t \\ F & = & \lambda x.\,a\;x\;\;(F\;x\,) \end{array}\right.$$

with t some term of ground type and a a binary symbol. The set S_j is however finite, thanks to the idempotency of intersection types and to the finiteness of the set of colors. In a non-idempotent type system, the precise account of the multiplicity of use of a type would require to consider intersections of countably many types. This motivates the introduction of the infinitary relational semantics of linear logic in Chapter 9.

We can now define the type of the prefix t_0 as

$$\theta_{(t_0,\beta,l)} = \bigwedge S_1 \;\to\; \cdots \;\to\; \bigwedge S_n \;\to q$$

Context of a prefix. Now that we attributed a type $\theta_{(t_0,\beta,l)}$ to every prefix t_0 of a leaf $\langle\,\beta,\,l,\,t_0\;\,t_1\cdots t_n,\,q\,\rangle$ of a tree appearing in (7.1), we define an associated context $\Gamma_{(t_0,\beta,l)}$ such that

$$\Gamma_{(t_0,\beta,l)} \;\vdash\; t_0 \;:\; \theta_{(t_0,\beta,l)} \tag{7.3}$$

holds in the type system $\boldsymbol{\gamma}(\mathcal{A})$, by induction on the structure of the term:

1. If $t_0 \;=\; a \in \Sigma$ is a terminal, we set $\Gamma_{(t_0,\beta,l)} \;=\; \emptyset$. By definition of the rule δ, (7.3) holds.

2. If $t_0 \;=\; F \in \mathcal{N}$ is a non-terminal, we set $\Gamma_{(t_0,\beta,l)} \;=\; \bigwedge_{\{1\}} \boxdot_\varepsilon\, \theta_{(F,\beta,l)}$. By definition of the Axiom rule, (7.3) holds.

3. If $t_0 \;=\; u\;v$, consider the set S of triples

$$(\beta',\,l',\,\Omega(C'[\,]_{q'}))$$

such that

$$\langle\,\beta,\,l,\,t_0\;t_1\cdots t_n,\,q\,\rangle \quad\blacktriangleright^*\quad C'[\,\langle\,\beta',\,l',\,v\;\widetilde{t'},\,q'\,\rangle\,]$$

We consider a subset $S' \subseteq S$ such that for every $(\beta'',\,l'',\,m) \in S$ there is exactly one $(\beta',\,l',\,m) \in S'$ such that $\theta_{(v,\beta',l')} \;=\; \theta_{(v,\beta'',l'')}$. We then define

$$\Gamma_{(t_0,\beta,l)} \quad=\quad \Gamma_{(u,\beta,l)} \cup \left(\bigcup \left\{\, \boxdot_m\, \Gamma_{(v,\beta',l')} \;\mid\; (\beta',l',m) \;\in\; S' \,\right\}\right)$$

7.5 Summary of the proof of completeness

The proof of completeness proceeds in two steps:

1. We first prove that we can use the "optimal" types and environments we just defined to appropriately type the terms to which non-terminals rewrite: for each occurrence of a non-terminal F in head position in the reduction, occurring in a leaf identified by β and l, we can define a context Γ such that $\Gamma \vdash \mathcal{R}(F) : \theta_{(F,\beta,l)}$ holds. This is a consequence of Lemma 2, which itself relies on Lemma 1.

2. We then show that it is a winning strategy for Eve in $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$ to play the context Γ from the node $F : \boxdot_m \theta_{(F,\beta,l)} :: \kappa(F)$. This relies on Lemma 3, which relates the color m attributed to F with the color of contexts generated from the leaf identified by β and l and in which F appears in head position.

These two steps allow to prove the completeness theorem (Theorem 21).

7.6 Three preliminary lemmas

Lemma 1. *Suppose that*

$$\langle\, \varepsilon,\, 0,\, S,\, q_0 \,\rangle \quad \blacktriangleright^* \quad C[\,\langle\, \beta,\, l,\, t_0\ t_1 \cdots t_n,\, q \,\rangle\,]$$

with $t_0 = [x_1 \leftarrow s_1, \ldots, x_n \leftarrow s_n]u$. *Then there exists a context* Γ_0, *sets* $J_1, \ldots, J_k$ *and intersection types* $\boxdot_{m_{i,j}}\ \theta_{i,j}$ *for* $i \in \{1, \ldots, k\}$ *and* $j \in J_i$ *that satisfy the three following properties:*

$$\Gamma_0,\, \ldots,\, x_i : \bigwedge_{j \in J_i} \boxdot_{m_{i,j}}\ \theta_{i,j},\, \ldots \vdash u : \theta_{(t_0,\beta,l)}$$

$$\left\{\boxdot_{m_{i,j}}\ \theta_{i,j} \,\middle|\, j \in J_i\right\} \subseteq \left\{\boxdot_{\Omega(C'[\,]_{q'})}\ \theta_{(s_i,\beta',l')} \,\middle|\, \langle\, \beta,\, l,\, t_0\ t_1 \cdots t_n,\, q \,\rangle \ \blacktriangleright^*\ C'[\langle\, \beta',\, l',\, s_i\ \widetilde{t'},\, q' \,\rangle]\right\}$$

$$\Gamma_0 \ \subseteq\ \Gamma_{(t_0,\beta,l)}$$

Note that we do not mention simple types to ease reading; we will often do the same implicitly in the sequel. For convenience, we will also write $[\widetilde{x} \leftarrow \widetilde{s}]$ for the combined substitution $[x_1 \leftarrow s_1, \ldots, x_n \leftarrow s_n]$. Remark that the second condition formalizes the intuition we gave earlier: all the types occurring in a context need to be used to type the corresponding argument when it is put in head position by the rewriting process.

Proof. The proof proceeds by induction on the structure of u. The induction is not very complicated, although the case where u is an application requires heavy notations to apply the induction hypothesis.

- If $u = a \in \Sigma$ or $u = F \in \mathcal{N}$, taking $\Gamma_0 = \Gamma_{(t_0,\beta,l)}$ and $J_i = \emptyset$ for each i satisfies the three conditions expressed in the lemma.

- If $u = x_i$, then $t_0 = s_i$. The three conditions hold for $\Gamma_0 = \emptyset$, $J_k = \emptyset$ for every $k \neq i$, $J_i = \{1\}$ and $\boxdot_{i,1}\ \theta_{i,1} = \boxdot_\varepsilon\ \theta_{(t_0,\beta,l)}$.

- If $u = u_0\ u_1$, then $t_0 = t_{0,0}\ t_{0,1}$ with $t_{0,0} = [\widetilde{x} \leftarrow \widetilde{s}]u_0$ and $t_{0,1} = [\widetilde{x} \leftarrow \widetilde{s}]u_1$. By definition of $\Gamma_{(t_0,\beta,l)}$, there exists a finite set H such that

$$\Gamma_{(t_0,\beta,l)} \ = \ \Gamma_{(t_{0,0},\beta,l)} \cup \left(\bigcup_{h\in H} \boxdot_{m_h} \Gamma_{(t_{0,1},\beta_h,l_h)} \right)$$

$$\theta_{(t_{0,0},\beta,l)} \ = \ \left(\bigwedge_{h\in H} \boxdot_{m_h}\ \theta_{(t_{0,1},\beta_h,l_h)} \right) \to \theta_{(t_0,\beta,l)}$$

and for each $h \in H$,

$$\langle\, \beta,\, l,\, t_0\ t_1 \cdots t_n,\, q \,\rangle \ \blacktriangleright^* \ C_h[\,\langle\, \beta_h,\, l_h,\, t_{0,1}\ \widetilde{t_h},\, q_h \,\rangle\,]$$

with $m_h = \Omega(C_h[\,]_{q_h})$. This allow us, for each $h \in H$, to apply the induction hypothesis to $t_{0,1}$, and we obtain in this way a family indexed by $h \in H$ of types and contexts:

$$\Gamma_{0,h},\, \ldots,\, x_i \,:\, \bigwedge_{j\in J_{h,i}} \boxdot_{m_{h,i,j}}\ \theta_{h,i,j},\, \ldots \ \vdash \ u_1 \,:\, \theta_{(t_{0,1},\beta_h,l_h)}$$

$$\left\{ \boxdot_{m_{h,i,j}}\ \theta_{h,i,j} \,|\, j \in J_{h,i} \right\} \subseteq \left\{ \boxdot_{\Omega(C'[\,]_{q'})}\ \theta_{(s_i,\beta',l')} \,|\, \langle\, \beta_h,\, l_h,\, t_{0,1}\ \widetilde{t_h},\, q_h \,\rangle \ \blacktriangleright^* \ C'[\langle\, \beta',\, l',\, s_i\ \widetilde{t'},\, q' \,\rangle] \right\}$$

$$\Gamma_{0,h} \ \subseteq \ \Gamma_{(t_{0,1},\beta_h,l_h)}$$

Applying the induction hypothesis to $t_{0,0}$ provides:

$$\Gamma_{0,0},\, \ldots,\, x_i \,:\, \bigwedge_{j\in J_{0,i}} \boxdot_{m_{0,i,j}}\ \theta_{0,i,j},\, \ldots \ \vdash \ u_0 \,:\, \theta_{(t_{0,0},\beta,l)}$$

$$\left\{ \boxdot_{m_{0,i,j}}\ \theta_{0,i,j} \,|\, j \in J_{0,i} \right\} \subseteq \left\{ \boxdot_{\Omega(C'[\,]_{q'})}\ \theta_{(s_i,\beta',l')} \,|\, \langle\, \beta,\, l,\, t_{0,0}\ t_{0,1}\ t_1 \cdots t_n,\, q \,\rangle \ \blacktriangleright^* \ C'[\langle\, \beta',\, l',\, s_i\ \widetilde{t'},\, q' \,\rangle] \right\}$$

$$\Gamma_{0,0} \ \subseteq \ \Gamma_{(t_{0,0},\beta,l)}$$

We now have all we need to use the Application rule to type $u_0\ u_1$. We set

$$\Gamma_0 \ = \ \Gamma_{0,0} \cup \left(\bigcup_{h\in H} \boxdot_{m_h}\ \Gamma_{0,h} \right)$$

and for $h \in H$, $i \in \{1, \ldots, k\}$, $j \in J_{h,i}$ we set $m'_{h,i,j} = \max(m_{h,i,j}, m_h)$. For $i \in \{1, \ldots, k\}$ and $j \in J_{0,i}$, we set $m'_{0,i,j} = m_{0,i,j}$. Using the Application rule on the sequents provided by the induction hypothesis gives a proof of the sequent

$$\Gamma_0,\, \ldots,\, x_i \,:\, \bigwedge_{h\in\{0\}\cup H,\, j\in J_{h,i}} \boxdot_{m'_{h,i,j}}\ \theta_{h,i,j},\, \ldots \ \vdash \ u \,:\, \theta_{(t_0,\beta,l)}$$

Moreover:

$$\Gamma_0 \ \subseteq \ \Gamma_{(t_{0,0},\beta,l)} \cup \left(\bigcup_{h\in H} \boxdot_{m_h} \Gamma_{(t_{0,1},\beta_h,l_h)} \right) \ = \ \Gamma_{(t_0,\beta,l)}$$

To complete the demonstration, remark that

$$\left\{ \boxdot_{m'_{h,i,j}} \theta_{(h,i,j)} \mid h \in \{0\} \uplus H,\, j \in J_{h,i} \right\}$$

consists only of elements $\boxdot_{\Omega(C'[\,]_{q'})} \theta_{(s_i,\beta',l')}$ such that

$$\langle\, \beta,\, l,\, t_{0,0}\; t_{0,1}\; t_1 \cdots t_n,\, q \,\rangle \;\blacktriangleright^*\; C'[\langle\, \beta',\, l',\, s_i\; \widetilde{t'},\, q' \,\rangle\,] \tag{7.4}$$

For $h = 0$, this is immediate: it is what the induction hypothesis provides. For the other values $h \in H$, this is obtained from the fact that by definition of $\Gamma_{(t_0,\beta,l)}$ every h corresponds to a leaf where $t_{0,1}$ occurs in head position, from which we can then apply the induction hypothesis for the corresponding h. We obtain (7.4) as:

$$\langle\, \beta,\, l,\, t_{0,0}\; t_{0,1}\; t_1 \cdots t_n,\, q \,\rangle \;\blacktriangleright^*\; C_h[\langle\, \beta_h,\, l_h,\, t_{0,1}\; \widetilde{t_h},\, q_h \,\rangle] \;\blacktriangleright^*\; C_h[C'[\langle\, \beta',\, l',\, s_i\; \widetilde{t'},\, q' \,\rangle\,]]$$

and the fact that $m'_{h,i,1} = \Omega(C_h[C'[\,]_{q'}]_{q_h})$ follows from Proposition 17.

□

From this technical lemma, we obtain the result which will guide Eve's strategy to answer Adam's questions with appropriate typing environments:

Lemma 2. *Denote $\mathcal{R}(F) = \lambda\widetilde{x}.\, t$, and suppose that*

$$\langle\, \varepsilon,\, 0,\, S,\, q_0 \,\rangle \quad \blacktriangleright^* \quad C[\,\langle\, \beta,\, l,\, F\; \widetilde{s},\, q \,\rangle\,] \quad \blacktriangleright \quad C[\,\langle\, \beta,\, l+1,\, t[\widetilde{x} \leftarrow \widetilde{s}],\, q \,\rangle\,]$$

Then there exists $\Gamma \subseteq \Gamma_{(t[\widetilde{x}\leftarrow\widetilde{s}],\beta,l+1)}$ such that

$$\Gamma \quad \vdash \quad \mathcal{R}(F) \,:\, \theta_{(F,\beta,l)} \,::\, \kappa(F)$$

in the type system $\boldsymbol{\gamma}(\mathcal{A})$.

Proof. Since t is of simple type o, $\theta_{(t[\widetilde{x}\leftarrow\widetilde{s}],\beta,l+1)} = q$. An immediate application of Lemma 1 gives a context Γ such that:

$$\Gamma,\, \ldots,\, x_i \,:\, \bigwedge_{j \in J_i} \boxdot_{m_{i,j}} \theta_{i,j},\, \ldots \;\vdash\; t \,:\, q$$

$$\left\{ \boxdot_{m_{i,j}} \theta_{i,j} \,|\, j \in J_i \right\} \subseteq \left\{ \boxdot_{\Omega(C'[\,]_{q'})} \theta_{(s_i,\beta',l')} \,|\, \langle\, \beta,\, l,\, t[\widetilde{x} \leftarrow \widetilde{s}],\, q \,\rangle \;\blacktriangleright^*\; C'[\langle\, \beta',\, l',\, s_i\; \widetilde{t'},\, q' \,\rangle] \right\}$$

$$\Gamma \;\subseteq\; \Gamma_{(t[\widetilde{x}\leftarrow\widetilde{s}],\beta,l+1)}$$

By definition, $\theta_{(F,\beta,l)}$ is of the form

$$\bigwedge_{j \in J'_1} \boxdot_{m_{1,j}} \theta_{1,j} \;\to\; \cdots \;\to\; \bigwedge_{j \in J'_n} \boxdot_{m_{n,j}} \theta_{n,j} \;\to\; q$$

where for every i we have $J_i \subseteq J'_i$, as J'_i contains by definition of $\theta_{(F,\beta,l)}$ all the types that s_i can have when it appears in head position.

To conclude, we apply n times the λ rule, using at each step the weakening $J_i \subseteq J'_i$. □

Now that we defined contexts to be used as answers by Eve, and which do not suffer from the defect leading to the loosing proof of Figure 7.1, we need to relate the colors occurring in typing environments with the ones occurring in paths of partial approximations of the accepting run-tree.

Lemma 3. *Suppose that*

$$\langle\, \varepsilon,\, 0,\, S,\, q_0 \,\rangle \;\; \blacktriangleright^* \;\; C[\,\langle\, \beta,\, l,\, t,\, q \,\rangle\,]$$

and that $F : \boxdot_m\, \theta \in \Gamma_{(t,\beta,l)}$. *Then there exists* C', β', l', $\widetilde{t'}$, q' *such that*

- $\langle\, \beta,\, l,\, t,\, q \,\rangle \;\; \blacktriangleright^* \;\; C'[\langle\, \beta',\, l',\, F\ \widetilde{t'},\, q' \,\rangle]$
- $\theta \;=\; \theta_{(F,\beta',l')}$
- $m \;=\; \Omega(C'[\,]_{q'})$

Proof. We prove by induction on the structure of the term t a stronger version of this property, where the only change is that we allow t to have arguments: suppose that $\langle\, \varepsilon,\, 0,\, S,\, q_0 \,\rangle \;\; \blacktriangleright^* \;\; C[\,\langle\, \beta,\, l,\, t\ \widetilde{u},\, q \,\rangle\,]$ and that $F : \boxdot_m\, \theta \in \Gamma_{(t,\beta,l)}$. Then there exists C', β', l', $\widetilde{t'}$, q' such that $\langle\, \beta,\, l,\, t\ \widetilde{u},\, q \,\rangle \;\; \blacktriangleright^* \; C'[\langle\, \beta',\, l',\, F\ \widetilde{t'},\, q' \,\rangle]$ with $m \;=\; \Omega(C'[\,]_{q'})$ and $\theta \;=\; \theta_{(F,\beta',l')}$.

- Since $F : \boxdot_m\, \theta \in \Gamma_{(t,\beta,l)}$, t can not be a terminal $a \in \Sigma$ or a non-terminal other than F, by definition of this typing environment.
- If $t = F$, since F is already in head position in t, the property holds for $C' = [\,]_q$, $\beta' = \beta$, $l' = l$. Note that we have $m = \Omega([\,]_q) = \varepsilon$.
- If $t = t_0\ t_1$, we have by definition

  $$\Gamma_{(t,\beta,l)} \;\; = \;\; \Gamma_{(t_0,\beta,l)} \;\cup\; \left(\bigcup_{i=1}^{k} \boxdot_{m_i}\, \Gamma_{(t_1,\beta_k,l_i)} \right)$$

  where for each $i \in \{1, \ldots, k\}$ we have that

  $$\langle\, \beta,\, l,\, t_0\ t_1\ \widetilde{u},\, q \,\rangle \;\; \blacktriangleright^* \;\; C_i[\,\langle\, \beta_i,\, l_i,\, t_1\ \widetilde{s_i},\, q_i \,\rangle\,]$$

  and $m_i \;=\; \Omega(C_i[\,]_{q_i})$. If $F : \boxdot_m\, \theta \in \Gamma_{(t_0,\beta,l)}$, we can conclude by using the induction hypothesis. Else there exists $i \in \{1, \ldots, k\}$ such that $F : \boxdot_m\, \theta \in \boxdot_{m_i}\, \Gamma_{(t_1,\beta_i,l_i)}$. This implies the existence of a color m' such that $F : \boxdot_{m'}\, \theta \in \Gamma_{(t_1,\beta_i,l_i)}$ with $m = \max(m', m_i)$.

  Since $\langle\, \beta,\, l,\, t_0\, t_1\ \widetilde{u},\, q \,\rangle \;\; \blacktriangleright^* \;\; C_i[\,\langle\, \beta_i,\, l_i,\, t_1\ \widetilde{s_i},\, q_i \,\rangle\,]$, we have $\langle\, \varepsilon,\, 0,\, S,\, q_0 \,\rangle \;\; \blacktriangleright^* \; C[C_i[\,\langle\, \beta_i,\, l_i,\, t_1\ \widetilde{s_i},\, q_i \,\rangle\,]]$ which together with $F : \boxdot_{m'}\, \theta \in \Gamma_{(t_1,\beta_i,l_i)}$ allows to use the induction hypothesis on t_1. We obtain in this way C'_i, β'_i, l'_i, $\widetilde{t'_i}$, q'_i such that

  $$\langle\, \beta_i,\, l_i,\, t_1\ \widetilde{s_i},\, q_i \,\rangle \;\; \blacktriangleright^* \;\; C'_i[\,\langle\, \beta'_i,\, l'_i,\, F\ \widetilde{t'_i},\, q'_i \,\rangle\,]$$

  with $m' \;=\; \Omega(C'_i[\,]_{q'_i})$ and $\theta \;=\; \theta_{(F,\beta'_i,l'_i)}$. We conclude by setting $C' \;=\; C_i[C'_i]$, $\beta' \;=\; \beta'_i$, $l' \;=\; l'_i$, $\widetilde{t'} \;=\; \widetilde{t'_i}$ and $q' \;=\; q'_i$.

  Note that we indeed have

  $$m \;=\; \max(m', m_i) \;=\; \max(C_i[\,]_{q_i}, C'_i[\,]_{q'_i}) \;=\; \Omega(C_i[C'_i[\,]_{q'_i}]_{q_i}$$

  by Proposition 17.

□

7.7 The completeness theorem

The idea behind Lemma 3 is that Eve plays typing environments included in environments of the form $\Gamma_{(t,\beta,l)}$, so that Adam will answer by playing typed non-terminals $F : \boxdot_m \theta$ as in the lemma. The color played by this interaction is therefore the one of the tree context that is built by the rewriting sequence putting F in head position. Incrementally, the interaction between Adam and Eve explores a branch of the winning run-tree $\mathcal{T}_{run}$ we fixed. When the interaction is infinite, the explored branch of $\mathcal{T}_{run}$ is also infinite, due to the productivity of the higher-order recursion scheme $\mathcal{G}$. Each pair of a move of Adam followed by a move by Eve constructs a finite tree context $C[\,]$, containing a finite part of the infinite branch of interest. The color played by Adam is the maximal one encountered along this finite part of the branch – excluding the color of the root but including the one of the hole, as we explain in the proof of the next theorem. It follows that the color of the infinite play in $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$ is the same as the one of the infinite branch of $\mathcal{T}_{run}$ it explores. Since $\mathcal{T}_{run}$ is winning, the corresponding play is winning as well. These are the key ingredients of the completeness theorem.

Theorem 21 (Completeness)**.** *Let $\mathcal{A}$ be an alternating parity automaton and $\mathcal{G}$ a higher-order recursion scheme. If $\mathcal{A}$ has a winning execution over $\langle\, \mathcal{G} \,\rangle$, then Eve has a winning strategy in* $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$.

Proof. Since $\mathcal{A}$ has a winning run-tree $\mathcal{T}_{run}$ over $\langle\, \mathcal{G} \,\rangle$, there is a fair rewriting sequence as (7.1):

$$T_0 \;=\; \langle\, \varepsilon,\, 0,\, S,\, q_o \,\rangle \;\blacktriangleright\; T_1 \;\blacktriangleright\; T_2 \;\blacktriangleright\; \cdots \tag{7.5}$$

computing it. We construct a winning strategy $\mathcal{W}$ for Eve in $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$ using this sequence. The idea is that Eve will only provide non-terminals that occur in head position at some point of the reduction, that is, non-terminals which will actually be reduced. Then, when Adam picks one of these non-terminals, Eve moves in the reduction sequence to the point where it occurs in head position, and computes a new typing context from there. The sequence of Adam's moves will iteratively explore a branch of $\mathcal{T}_{run}$.

On the initial vertex

$$S \,:\, \boxdot_\varepsilon\, q_0$$

Eve has to move by picking a typing environment Γ such that

$$\Gamma \;\vdash\; \mathcal{R}(S) \,:\, q_0 \,::\, o \tag{7.6}$$

is provable in $\boldsymbol{\gamma}(\mathcal{A})$. We obtain this context using Lemma 2, applied to

$$\langle\, \varepsilon,\, 0,\, S,\, q_0 \,\rangle \;\blacktriangleright^0\; \langle\, \varepsilon,\, 0,\, S,\, q_0 \,\rangle \;\blacktriangleright\; \langle\, \varepsilon,\, 1,\, \mathcal{R}(S),\, q_0 \,\rangle$$

which provides $\Gamma \subseteq \Gamma_{(\mathcal{R}(S),\varepsilon,1)}$ satisfying (7.6). We denote Γ as $\Gamma^{[\mathcal{R}(S),\varepsilon,1]}$.

In the sequel of the play, Eve will keep answering Adam using contexts provided by Lemma 2. In order to keep track of them, we label explicitly the

typing contexts played by Eve as $\Gamma^{[t,\beta,l]}$. We will always have, by construction, that $\Gamma^{[t,\beta,l]} \subseteq \Gamma_{(t,\beta,l)}$.

Consider now that Eve played a context $\Gamma^{[t,\beta,l]}$, and that Adam answered by picking $\boxdot_m\ \theta \in \Gamma^{[t,\beta,l]}$. By Lemma 3, there exists C', β', l', $\widetilde{s'}$ and q' associated with a reduction sequence putting F in head position from the leaf identified by β and l:

$$\langle\, \beta,\, l,\, t,\, q_t \,\rangle \ \blacktriangleright^*\ C'[\,\langle\, \beta',\, l',\, F\ \widetilde{s'},\, q' \,\rangle\,] \ \blacktriangleright\ C'[\,\langle\, \beta',\, l'+1,\, t_F[\widetilde{x} \leftarrow \widetilde{s'}],\, q' \,\rangle\,]$$

with $\Omega(C'[\,]_{q'}) = m$, $\theta = \theta_{(F,\beta',l')}$ and $\mathcal{R}(F) = \lambda\widetilde{x}.\, t_F$. We can now apply Lemma 2 and obtain a typing context Γ' such that

- $\Gamma' \vdash \mathcal{R}(F) : \theta_{(F,\beta',l')}$
- and $\Gamma' \subseteq \Gamma_{(t_F[\widetilde{x}\leftarrow\widetilde{s'}],\beta',l'+1)}$

We set $\Gamma^{[t_F[\widetilde{x}\leftarrow\widetilde{s'}],\beta',l'+1]} = \Gamma'$ and define it as Eve's answer to Adam's move picking $F : \boxdot_m\ \theta$. Note that the strategy $\mathcal{W}$ we define here is *not* memoryless, as we use β, l and t from Eve's previous move to define its next one. However, the existence of a winning strategy for a player in a parity game implies the existence of a memoryless such strategy.

By construction, $\mathcal{W}$ is winning for Eve on finite plays, as it always provide an answer to Adam's moves. It remains to prove that it is satisfies the parity condition on infinite plays. Let us consider such an infinite play, where Eve plays according to the strategy $\mathcal{W}$:

$$(\, F_0 \,:\, \boxdot_{m_0}\ \theta_0) \quad \Gamma_0^{[t_0,l_0,\beta_0]} \quad (\, F_1 \,:\, \boxdot_{m_1}\ \theta_1) \quad \Gamma_1^{[t_1,l_1,\beta_1]} \quad \cdots \tag{7.7}$$

where by construction $F_0 : \boxdot_{m_0}\ \theta_0$ is $S : \boxdot_\varepsilon\ q_0$ and $\Gamma_0^{[t_0,l_0,\beta_0]}$ is $\Gamma^{[\mathcal{R}(S),\varepsilon,1]}$. The definition of $\mathcal{W}$ induces a factorization of the fair rewriting sequence (7.5) as

$$\begin{array}{lll}
 & \langle\, \varepsilon,\, 0,\, S,\, q_0 \,\rangle & \blacktriangleright\ \ \langle\, \varepsilon,\, 1,\, \mathcal{R}(S),\, q_0 \,\rangle \\
\blacktriangleright^* & C_1[\,\langle\, \beta_1,\, l_1 - 1,\, F_1\ \widetilde{s_1},\, q_1 \,\rangle\,] & \blacktriangleright\ \ C_1[\,\langle\, \beta_1,\, l_1,\, t_1,\, q_1 \,\rangle\,] \\
\blacktriangleright^* & C_1[C_2[\,\langle\, \beta_2,\, l_2 - 1,\, F_2\ \widetilde{s_2},\, q_2 \,\rangle\,]] & \blacktriangleright\ \ C_1[C_2[\,\langle\, \beta_2,\, l_2,\, t_2,\, q_2 \,\rangle\,]] \\
\blacktriangleright^* & \cdots &
\end{array}$$

with $\Omega\,(C_i[\,]_{q_i}) = m_i$. Note that since the rewriting sequence (7.5) is fair and generates $\mathcal{T}_{run}$, it does not only explore a branch, but computes the whole tree. However, the interaction of Adam with Eve only plays colors from a given infinite branch of $\mathcal{T}_{run}$, characterized by the holes occurring in the increasing sequence

$$[\,]_{q_0} \ ,\ C_1[\,]_{q_1} \ ,\ C_1[C_2[\,]_{q_2}] \ \cdots \tag{7.8}$$

Since $\mathcal{T}_{run}$ is winning, the maximal color seen infinitely often along this branch is even. Note that since the branch of $\mathcal{T}_{run}$ associated with the sequence (7.8) is infinite, there are infinitely many *non-empty* contexts C_{j_0}, C_{j_1}, ..., and

$$C_{j_0}[C_{j_1}[C_{j_2}[\cdots]]]$$

builds the infinite branch of interest – note, however, that the resulting Σ_{comp}-labeled unranked tree is only a subtree of $\mathcal{T}_{run}$, as we removed computation steps generating other infinite branches.

Note that the integers i such that $m_i = \varepsilon$ are precisely the ones such that $\exists k \;\; j_k < i < j_{k+1}$: they correspond to computation steps which do not increase the branch of interest. The maximal color seen infinitely often on the sequence $(m_i)_{i\in\mathbb{N}}$ coincides with the one of the extracted sequence $(m_{j_k})_{k\in\mathbb{N}}$, as we only remove neutral colors ε. Recall now the discussion following Proposition 17:

- m_{j_0} is the maximal color seen from the root of C_{j_0} (excluded) to its hole $[\,]_{q_{j_0}}$ (included),
- m_{j_1} is the maximal color seen from the root of C_{j_1} (excluded, but taken into account in m_{j_0} as the color of the hole we plug the root of C_{j_1} in) to its hole $[\,]_{q_{j_1}}$ (included),
- and so on.

To summarize, $(m_{j_k})_{k\in\mathbb{N}}$ is a sequence of maximums of finite parts of the infinite branch of interest, each color occurring along this infinite branch being counted exactly once, except for the root of the tree which is dismissed. As a consequence, the maximal color appearing infinitely often in the sequence $(m_i)_{i\in\mathbb{N}}$ is the color of the infinite branch of interest. This color is even since $\mathcal{T}_{run}$ is winning. Since $(m_i)_{i\in\mathbb{N}}$ is the sequence of colors appearing in the play (7.7), this play is winning. By considering all such plays, we obtain that $\mathcal{W}$ is a winning strategy for Eve. □

Chapter 8

Soundness of the type system

In this chapter, we adapt the soundness part of the original proof of Kobayashi and Ong [KO] to our new, modal coloring policy. We prove the following equivalent formulation of the soundness part of Theorem 19:

> Let $\mathcal{A}$ be an alternating parity automaton and $\mathcal{G}$ be a productive higher-order recursion scheme. If Eve has a winning strategy $\mathcal{W}$ in $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$, then $\mathcal{A}$ has a winning execution over $\langle\, \mathcal{G} \,\rangle$.

As in the completeness proof, it is more convenient to use the parity game $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$ to prove the statement, and to use the infinitary type system $\boldsymbol{\gamma}_{fix}(\mathcal{G}, \mathcal{A})$ to explain informally the idea of the proof. In $\boldsymbol{\gamma}_{fix}(\mathcal{G}, \mathcal{A})$, the proof can be understood as an infinitary rewriting process on derivations, using subject reduction and non-terminal unfoldings to incrementally compute from a proof of

$$S : \bigwedge_{\{1\}} \boxdot_{\varepsilon}\, q :: o \;\vdash\; S : q :: o \tag{8.1}$$

a proof of

$$\emptyset \;\vdash\; \langle\, \mathcal{G} \,\rangle : q :: o \tag{8.2}$$

which corresponds to a run-tree of $\mathcal{A}$ over $\langle\, \mathcal{G} \,\rangle$, and which only consists of Application and δ rules. We then have to prove that this run-tree is winning for the parity condition – or, equivalently, that the derivation we computed of the sequent (8.2) is winning. We thus have to prove that the color of any infinite branch of this derivation is even, by relating it to an infinite branch of the winning derivation tree proving (8.1).

The proof of soundness starts in §8.1 with a series of preliminary definitions. We then prove in §8.2 a subject reduction property for the intersection type system $\boldsymbol{\gamma}(\mathcal{A})$. This subject reduction property enables us to define in §8.3 a rewriting system (with rewriting relation noted $\rhd$) parametrized by a winning strategy for Eve in $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$. The remaining of the proof of soundness consists in proving that $\rhd$ computes a winning run-tree of $\mathcal{A}$ over the infinite tree $\langle\, \mathcal{G} \,\rangle$. The main ingredients of the proof are explained in §8.4. We start by establishing in §8.5 a progress lemma for the rewriting relation $\rhd$. We then establish in §8.6 a « fair production lemma », stating that fair rewriting

sequences for $\triangleright$ generate a run-tree of $\mathcal{A}$ over $\langle\, \mathcal{G} \,\rangle$, and not just a partial run-tree. We finally obtain in §8.7 the soundness theorem by showing that this run-tree is moreover winning. The proof of this theorem relies on a crucial lemma, called the « infinite ancestor lemma », which states that every infinite branch of the run-tree comes from an infinite play of the same color in $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$, in which Eve follows her winning strategy $\mathcal{W}$. We conclude the chapter by a proof of a technical lemma, necessary to establish the infinite ancestor lemma.

8.1 Preliminary definitions

Following [KO], we prove soundness in $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$ rather than in $\boldsymbol{\gamma}_{fix}(\mathcal{G}, \mathcal{A})$, as it is technically more convenient. Let $\mathcal{A} = \langle \Sigma,\, Q,\, \delta,\, q_0,\, \Omega \rangle$ be an alternating parity automaton and $\mathcal{G} = \langle \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$ be a productive higher-order recursion scheme with $\mathcal{N} = \{F_1, \ldots, F_n\}$. Let also $\mathcal{W}$ be Eve's winning strategy; we can assume it to be *memoryless* since we consider a parity game. Therefore, $\mathcal{W}$ maps nodes $F : \boxdot_m\, \theta$ to typing environments we denote $\Gamma_{(F\,:\,\boxdot_m\ \theta)}$.

We define a set of labels Σ_{sound} and a rewrite relation $\triangleright$ on Σ_{sound}-labeled unranked trees which models the effect of the normalization of $\mathcal{G}$ on the typing derivations of $\boldsymbol{\gamma}_{fix}(\mathcal{G}, \mathcal{A})$. The set of labels Σ_{sound} contains more than just terms and states, in order to ease the localization of occurrences of non-terminals during the rewriting process, and to keep track of the colors appearing during the construction of the run-tree.

Let n be the maximal arity of the symbols of the signature Σ. The set of labels Σ_{sound} contains the symbols:

- $\langle\, \alpha,\, q \,\rangle$, where $\alpha \in \{1, \ldots, n\}$ and $q \in Q$. These nodes correspond to nodes of the run-tree generated by the rewriting process.
- $\langle\, \alpha,\, l,\, \Lambda,\, \Gamma \vdash t : q \,\rangle$, where $\alpha \in \{1, \ldots, n\}^*$ is the node computed by the head reduction of t, l is a natural number counting the rewriting steps, $\Lambda : \mathbb{N} \to \mathit{Col}$ is a partial function mapping some of the rewriting steps to the maximal color seen since them, and $\Gamma \vdash t : q$ is a sequent provable in $\boldsymbol{\gamma}(\mathcal{A})$. Note that this sequent comes implicitly together with a proof, a fact which is not explicited in [KO].

Σ_{sound}-labeled unranked trees will be used to compute finite prefixes of the run-tree we want to generate. The inner nodes will be of the form $\langle\, \alpha,\, q \,\rangle$, and the leaves can be of either shapes:

- a leaf of the form $\langle\, \alpha,\, q \,\rangle$ corresponds to a leaf of the run-tree, from which no further computation will be performed,
- and a leaf of the form $\langle\, \alpha,\, l,\, \Lambda,\, \Gamma \vdash t : q \,\rangle$ is to generate the label of the node α by head normalization of t; the information contained in the proof of $\Gamma \vdash t : q$ will allow to determine which transition function should be applied once the head symbol is computed, but also to carry on coloring information.

For a Σ_{sound}-labeled unranked tree T, we define the Σ_{sound}-labeled unranked tree $T^\sharp$ by relabeling the leaves of T of the form $\langle\, \alpha,\, l,\, \Lambda,\, \Gamma \vdash t : q \,\rangle$ to $\langle\, \alpha,\, q \,\rangle$. This corresponds to the finite run-tree prefix computed by T.

Consider a type judgment of the form $\Gamma \vdash F\ \widetilde{t} : q$ occurring in a Σ_{sound}-label. As we said, it implicitly comes together with a proof of its validity, which is of the form:

$$\text{App}\ \dfrac{\dfrac{}{F : \boxdot_\varepsilon\ \theta \vdash F : \theta} \quad \begin{matrix}\vdots\end{matrix} \quad \begin{matrix}\vdots\end{matrix}}{\Gamma \vdash F\ \widetilde{t} : q}$$

We sometimes write F^θ to indicate that F is used with this refinement type in the proof.

8.2 Subject reduction

Before we define the rewriting relation on Σ_{sound}-trees in §8.3, we prove a crucial lemma of subject reduction:

Lemma 4 (Subject reduction)**.** *If* $\Gamma \vdash (\lambda x.\, t_0)\ t_1 : \theta :: \kappa$ *in* $\boldsymbol{\gamma}(\mathcal{A})$*, then there exists* $\Gamma' \subseteq \Gamma$ *such that* $\Gamma' \vdash t_0[x \leftarrow t_1] : \theta :: \kappa$*.*

The proof of this lemma is a consequence of the following result:

Lemma 5. *If* $\Gamma_0, x : \bigwedge_{i\in I} \boxdot_{m_i} \theta_i \vdash t_0 : \theta$ *and* $\forall i \in I \quad \Gamma_i \vdash t : \theta_i$ *holds in* $\boldsymbol{\gamma}(\mathcal{A})$*, then* $\Gamma_0 \cup \left(\bigcup_{i\in I} \boxdot_{m_i} \Gamma_i\right) \vdash t_0[x \leftarrow t] : \theta$ *holds in* $\boldsymbol{\gamma}(\mathcal{A})$ *as well.*

Proof. We proceed by induction on the structure of the proof of the sequent $\Gamma_0, x : \bigwedge_{i\in I} \boxdot_{m_i} \theta_i \vdash t_0 : \theta$.

- If the last rule is δ, then $t_0 \in \Sigma$ is a terminal: x does not appear in it, so that $I = \emptyset$ and $t_0[x \leftarrow t] = t_0$. The result therefore holds, by considering the same proof.
- If the last rule is an Axiom applied to a symbol $y \in \mathcal{V} \cup \mathcal{N}$ other than x, then $t_0[x \leftarrow t] = t_0$ and $I = \emptyset$: again, the result follows immediately.
- If the last rule is an Axiom applied to the variable x, then we have:

  $$x : \bigwedge_{\{1\}} \boxdot_\varepsilon \theta \vdash x : \theta$$

  so that $t_0[x \leftarrow t] = t$, $\Gamma_0 = \emptyset$, $m_1 = \varepsilon$, $\theta_1 = \theta$. By hypothesis, we have a proof of $\Gamma_1 \vdash t : \theta_1$ from which we deduce the result, as $\Gamma_0 \cup \boxdot_\varepsilon \Gamma_1 = \Gamma_1$.
- If the last rule used is a λ, then $t_0 = \lambda y.\, t_1$. We suppose that the term is α-renamed so that $x \neq y$ and that y does not occur in t. So, we have $\theta = \left(\bigwedge_{j\in J} \boxdot_{n_j} \theta'_j\right) \to \theta'$, and

  $$\dfrac{\dfrac{\begin{matrix}\vdots\end{matrix}}{\Gamma_0, y : \bigwedge_{j\in J'} \boxdot_{n_j} \theta'_j, x : \bigwedge_{i\in I} \boxdot_{m_i} \theta_i \vdash t_1 : \theta'} \quad J' \subseteq J}{\Gamma_0, x : \bigwedge_{i\in I} \boxdot_{m_i} \theta_i \vdash \lambda y.\, t_1 : \left(\bigwedge_{j\in J} \boxdot_{n_j} \theta'_j\right) \to \theta'}$$

  Applying the induction hypothesis gives

  $$\Gamma_0 \cup \bigcup_{i\in I} \boxdot_{m_i} \Gamma_i,\ y : \bigwedge_{j\in J'} \boxdot_{n_j} \theta'_j \vdash t_1[x \leftarrow t] : \theta'$$

from which we deduce by a rule λ

$$\dfrac{\dfrac{\vdots}{\Gamma_0 \ \cup \ \bigcup_{i\in I} \boxdot_{m_i} \Gamma_i,\ y : \bigwedge_{j\in J'} \boxdot_{n_j} \theta'_j \ \vdash \ t_1[x \leftarrow t] : \theta'} \qquad J' \subseteq J}{\Gamma_0 \ \cup \ \bigcup_{i\in I} \boxdot_{m_i} \Gamma_i \ \vdash \ (\lambda y.\, t_1)[x \leftarrow t] : \left(\bigwedge_{j\in J} \boxdot_{n_j} \theta'_j\right) \to \theta'}$$

- If the last rule is an Application, $t_0 = t_1\ t_2$ and the proof is of the shape

$$\dfrac{\dfrac{\vdots}{\Gamma_{0,0},\, x : \bigwedge_{i\in I, m\in S_{0,i}} \boxdot_m \theta_i \ \vdash \ t_1 : \bigwedge_{j\in J} \boxdot_{n_j} \theta'_j \to \theta} \qquad \dfrac{\vdots}{\Gamma_{0,j},\, x : \bigwedge_{i\in I, m\in S_{j,i}} \boxdot_m \theta_i \ \vdash \ t_2 : \theta'_j \ \text{ for each } j \in J}}{\Gamma_0,\, x : \bigwedge_{i\in I} \boxdot_{m_i} \theta_i \ \vdash \ t_1\ t_2 : \theta}$$

where $\Gamma_0 \ = \ \Gamma_{0,0} \cup \bigcup_{j\in J} \boxdot_{n_j} \Gamma_{0,j}$ and

$$\text{for each } i \in I, \quad \{m_i\} \ = \ S_{0,i} \cup \bigcup_{j\in J} \{\max(m, n_j) \mid m \in S_{j,i}\} \qquad (8.3)$$

Applying the induction hypothesis, we obtain

$$\Gamma_{0,0} \cup \bigcup_{i\in I, m\in S_{0,i}} \boxdot_m \Gamma_i \ \vdash \ t_1[x \leftarrow t] : \bigwedge_{j\in J} \boxdot_{n_j} \theta'_j \to \theta$$

$$\text{for each } j \in J, \qquad \Gamma_{0,j} \cup \bigcup_{i\in I, m\in S_{j,i}} \boxdot_m \Gamma_i \ \vdash \ t_2[x \leftarrow t] : \theta'_j$$

which combine using the Application rule to a proof of

$$\Gamma_{0,0} \ \cup \bigcup_{i\in I, m\in S_{0,i}} \boxdot_m \Gamma_i \ \cup \left(\bigcup_{j\in J} \boxdot_{n_j} \left(\Gamma_{0,j} \ \cup \bigcup_{i\in I, m\in S_{j,i}} \boxdot_m \Gamma_i\right)\right) \ \vdash \ t_0[x \leftarrow t] : \theta$$

To conclude, we need to prove that the context of this sequent is equal to $\Gamma_0 \ \cup \ \bigcup_{i\in I} \boxdot_{m_i} \Gamma_i$. First of all, notice that it rewrites to

$$\Gamma_{0,0} \cup \bigcup_{j\in J} \boxdot_{n_j} \Gamma_{0,j} \cup \bigcup_{i\in I, m\in S_{0,i}} \boxdot_m \Gamma_i \cup \left(\bigcup_{j\in J} \bigcup_{i\in I, m\in S_{j,i}} \boxdot_{\max(m,n_j)} \Gamma_i\right)$$

which is equal to $\Gamma_0 \ \cup \ \bigcup_{i\in I} \boxdot_{m_i} \Gamma_i$ by definition of Γ_0 and by (8.3).

□

Proof of Lemma 4. Suppose that $\Gamma \ \vdash \ (\lambda x.\, t_0)\ t_1 \ : \ \theta \ :: \ \kappa$ in $\boldsymbol{\gamma}(\mathcal{A})$, then there exists a derivation of the shape:

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma_0,\, x : \bigwedge_{i\in I} \boxdot_{m_i} \theta_i \ \vdash \ t_0 : \theta} \qquad I \subseteq J}{\Gamma_0 \ \vdash \ \lambda x.\, t_0 : \left(\bigwedge_{i\in J} \boxdot_{m_i} \theta_i\right) \to \theta} \qquad \dfrac{\vdots}{\Gamma_i \ \vdash \ t_1 : \theta_i \quad (i \in J)}}{\Gamma_0 \cup \bigcup_{i\in J} \boxdot_{m_i} \Gamma_i \ \vdash \ (\lambda x.t_0)\ t_1 \ : \ \theta}$$

We apply Lemma 5 to Γ_0, $x : \bigwedge_{i\in I} \boxdot_{m_i} \theta_i \vdash t_0 : \theta$ and obtain the desired context

$$\Gamma' \;=\; \Gamma_0 \cup \bigcup_{i\in I} \boxdot_{m_i} \Gamma_i$$

8.3 Definition of the rewriting system

We define the rewriting system which will compute a run-tree of $\mathcal{A}$ over $\langle\, \mathcal{G} \,\rangle$. The initial tree of the rewriting sequence will be

$$\langle\, \varepsilon,\, 1,\, \{0 \mapsto \varepsilon\}\,,\, S^0 : \bigwedge_{\{1\}} \boxdot_{\varepsilon} q_0 \vdash S^0 : q_0 \,\rangle$$

meaning that to compute the run-tree, we start

- by computing the label of its root ε,
- that it is the first rewriting step, identified by the integer 1,
- that the maximal color seen from the rewriting step 0 (introduced for technical convenience, and to be understood as the step which introduces the start symbol S) is ε,
- and that the label of the node will be computed by head reduction of S, currently in state q_0.

Note that we write S^0 to indicate that S was introduced by the rewriting step 0 (that is, the initial one). We will usually annotate non-terminals with the label of the rewriting step introducing them: F^l will identify an occurrence of F introduced by a rule rewriting at step l, as we shall see more precisely in the definition of $\rhd$. This annotation is crucial in Lemma 8.
We then define by induction the head rewriting relation $\rhd$ on Σ_{sound}-labeled unranked trees:

1. If $F \in \mathcal{N}$ is a non-terminal which rewrites to $\mathcal{R}(F) \;=\; \lambda x_1 \ldots \lambda x_n.\, t'$, such that $\Gamma \vdash F^{l',\theta}\, \widetilde{t} : q$ holds in $\boldsymbol{\gamma}(\mathcal{A})$ and that $\Gamma_{(F:\boxdot_{\Lambda(l')}\,\theta)}$ is defined, then

   $$\langle\, \alpha,\, l,\, \Lambda,\, \Gamma \vdash F^{l',\theta}\, \widetilde{t} : q \,\rangle \;\rhd\; \langle\, \alpha,\, l{+}1,\, \Lambda\{l \mapsto \varepsilon\},\, \Gamma' \vdash \rho(t')[\widetilde{x} \leftarrow \widetilde{t}] : q \,\rangle$$

   Here $\rho(t')$ is the term obtained by renaming each non-terminal F_i occurring in t as F_i^l, expliciting in this way the rewriting step which created the occurrence of this non-terminal. The context Γ' is computed as follows: we start from the proof π of

   $$\text{App}\;\; \dfrac{F^{l'} : \boxdot_{\varepsilon}\, \theta \vdash F^{l'} : \theta}{\dfrac{\vdots \qquad\qquad \vdots}{\Gamma \vdash F^{l',\theta}\, \widetilde{t} : q}}$$

Since we assumed $\Gamma_{(F:\boxdot_{\Lambda(l')}\,\theta)}$ to be defined, there is a derivation of

$$\Gamma_{(F:\boxdot_{\Lambda(l')}\,\theta)} \;\vdash\; \mathcal{R}(F) \,:\, \theta$$

We apply it the renaming induced by ρ, which labels with l every non-terminal occurring in this proof. Then, we substitute the Axiom leaf

$$F^{l'} \,:\, \boxdot_{\varepsilon}\,\theta \;\vdash\; F^{l'} \,:\, \theta$$

in π with this proof of $\rho(\Gamma_{(F:\boxdot_{\Lambda(l')}\,\theta)}) \;\vdash\; \rho(\mathcal{R}(F)) \,:\, \theta$. The resulting derivation π' is a proof of

$$\Gamma_1 \;\cup\; \rho(\Gamma_{(F:\boxdot_{\Lambda(l')}\,\theta)}) \;\vdash\; \rho(\mathcal{R}(F))\;\widetilde{t} \,:\, \theta$$

where $\Gamma_1 \cup \{F \,:\, \boxdot_{\varepsilon}\,\theta\} = \Gamma$ — notice that due to idempotency, $F \,:\, \boxdot_{\varepsilon}\,\theta$ may either disappear from the context if there was only one Axiom leaf typing F with $\boxdot_{\varepsilon}\,\theta$, or stay if there were several. By application of the subject reduction lemma (Lemma 4), there exists $\Gamma' \subseteq \Gamma_1 \uplus \rho(\Gamma_{(F:\boxdot_{\Lambda(l')}\,\theta)})$ such that $\Gamma' \;\vdash\; \rho(t')[\widetilde{x} \leftarrow \widetilde{t}]$. This provides the typing context Γ' introduced by the reduction.

2. We now have to deal with the case in which the typing judgment occurring in the leaf of interest types a term which contains a terminal as its head symbol. In this case, the leaf contains $\Gamma \;\vdash\; a\;t_1 \cdots t_n \,:\, q$, and the derivation it implicitly comes together with derives it from a set of proofs of conclusions $\Gamma_{i,j} \;\vdash\; t_i \,:\, q_{i,j}$, such that

$$\{(i, q_{i,j}) \mid 1 \leq i \leq n,\; 1 \leq j \leq k_i\} \quad \text{satisfies} \quad \delta(q, a).$$

In this case, defining $\boxdot_m\,\Lambda$ as the partial function of same domain as Λ, and whose values on this domain are given by

$$(\boxdot_m\,\Lambda) \;=\; \max(\Lambda(l),\, m)$$

the relation $\rhd$ rewrites

$$\langle\, \alpha,\, l,\, \Lambda,\, \Gamma \;\vdash\; a\;t_1 \cdots t_n \,:\, q \,\rangle$$

to

$$\langle\, \alpha,\, q \,\rangle \quad \text{over} \quad \langle\, \alpha 1,\, l+1,\, \boxdot_{\Omega(q_{1,1})}\,\Lambda,\, \Gamma_{1,1} \;\vdash\; t_1 \,:\, q_{1,1} \,\rangle \;\cdots\; \langle\, \alpha n,\, l+1,\, \boxdot_{\Omega(q_{n,k_n})}\,\Lambda,\, \Gamma_{n,k_n} \;\vdash\; t_n \,:\, q_{n,k_n} \,\rangle$$

3. Defining tree contexts – as in the completeness proof – by the grammar

$$C \;::=\; [\,] \mid \langle\, \alpha,\, q \,\rangle\; T_1 \cdots T_{i-1}\; C\; T_{i+1} \cdots T_k$$

where the T_j are Σ_{sound}-labeled unranked trees and $[\,]$ is a distinguished hole symbol, we extend the rewriting relation to contexts: if $t \rhd t'$, then for any context C we have $C[t] \rhd C[t']$.

Note that this does not break the fact that the rewriting is restricted to terms occurring in head position, due to the structure of the trees and contexts we consider.

We define the color of a tree context just as in the completeness proof: writing $[\,]_q$ for a hole which is to receive either a tree of root $\langle\, \alpha,\, q \,\rangle$, or a leaf $\langle\, \alpha,\, l,\, \Lambda,\, \Gamma \;\vdash\; t \;:\; q \,\rangle$, we define the color $\Omega(C[\,]_q)$ of the context $C[\,]_q$ in the following way:

- if $C[\,]_q \;=\; [\,]_q$, then $\Omega(C[\,]_q) \;=\; \varepsilon$,
- if $C[\,]_q \;=\; \langle\, \alpha,\, q'' \,\rangle\; T_1 \cdots T_{i-1}\; [C'[\,]_q]_{q'}\; T_{i+1} \cdots T_n$, then

$$\Omega(C[\,]_q) \;=\; \max\left(\Omega(q'),\, \Omega\left(C'[\,]_q\right)\right)$$

  Note that in the case where $C'[\,]_q \;=\; [\,]_q$ we have $q \;=\; q'$ and $\Omega(C[\,]_q) \;=\; \Omega(q)$.

For more discussion about this definition and its comparison with the original one of Kobayashi and Ong, see p.151.

8.4 Overview of the soundness proof

Now that we have established the subject reduction property of the type system $\boldsymbol{\gamma}(\mathcal{A})$, and that we have used it to define the rewriting relation $\rhd$ over Σ_{sound}-labeled trees, we give a short overview of the remaining parts of the soundness proof. The guiding idea is to apply the rewriting relation $\rhd$ starting from the initial tree

$$\langle\, \varepsilon,\, 1,\, \{0 \mapsto \varepsilon\}\,,\, S^0 \;:\; \bigwedge_{\{1\}} \boxdot_\varepsilon\; q_0 \;\vdash\; S^0 \;:\; q_0 \,\rangle \tag{8.4}$$

We compute in this way a possibly infinite Σ_{sound}-labeled tree T. Our goal is to prove that this tree T is a winning run-tree of $\mathcal{A}$ over $\langle\, \mathcal{G} \,\rangle$. We proceed in three steps.

Progress. We prove that the rewriting relation $\rhd$ can always progress, so that we can consider maximal rewriting sequences. In order to establish the progress lemma (Lemma 6), it suffices to show that Eve's winning strategy $\mathcal{W}$ always enables us to define the context $\Gamma_{(F:\boxdot_{\Lambda(l')}\,\theta)}$ necessary to the first case of the definition of $\rhd$. The proof of the progress lemma (Lemma 6) relies on three technical lemmas: Lemmas 7, 8 and 9. We devote §8.5 to the proof of the progress lemma and of its three companions. As we will see, the proof of the progress lemma contains an interesting technique relating on the one hand the finite plays in $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$ where Eve follows $\mathcal{W}$, and in the other hand the computation of finite prefixes of branches of T. More specifically, consider the tree

$$C[\,\langle\, \alpha,\, l,\, \Lambda,\, \Gamma \;\vdash\; t \;:\; q \,\rangle\,]$$

obtained after a finite number of applications of the rewriting relation $\rhd$, starting from the initial tree (8.4):

$$\langle\, \varepsilon,\, 1,\, \{0 \mapsto \varepsilon\}\,,\, S^0 \;:\; \bigwedge_{\{1\}} \boxdot_\varepsilon\; q_0 \;\vdash\; S^0 \;:\; q_0 \,\rangle \;\; \rhd^* \;\; C[\,\langle\, \alpha,\, l,\, \Lambda,\, \Gamma \;\vdash\; t \;:\; q \,\rangle\,]$$

The proof of the progress lemma contains a construction relating the finite branch leading from the root of $C[\,]$ to its hole $[\,]$ filled with the leaf

$$\langle\, \alpha,\, l,\, \Lambda,\, \Gamma \;\vdash\; t \;:\; q \,\rangle$$

with a finite play where Eve follows her winning strategy $\mathcal{W}$ and where Adam plays an appropriate sequence of non-terminals. We will see this idea reappear a bit later, when we prove the infinite ancestor lemma.

Fair production. Consider a maximal fair rewriting sequence

$$T_0 \;=\; \langle\, \varepsilon,\, 1,\, \{0 \mapsto \varepsilon\}\,,\, S^0 \,:\, \bigwedge_{\{1\}} \boxdot_\varepsilon\, q_0 \;\vdash\; S^0 \,:\, q_0 \,\rangle \;\rhd\; T_1 \;\rhd\; T_2 \;\rhd\; \cdots$$

This sequence computes a Σ_{sound}-labeled tree $T = \bigvee_{i\in\mathbb{N}} T_i^\sharp$. The « fair production lemma » (Lemma 10) proves that this tree is a run-tree of $\mathcal{A}$ over $\langle\, \mathcal{G} \,\rangle$, rather than simply a partial run-tree. This fundamental property is proved in §8.6.

The infinite ancestor lemma. The last ingredient necessary to the proof of the soundness theorem (Theorem 22) is the infinite ancestor lemma (Lemma 12), which states that every infinite branch of the run-tree T generated by the rewriting relation $\rhd$ has the same color as an infinite play of $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$ where Eve follows her winning strategy $\mathcal{W}$. Since $\mathcal{W}$ is winning for Eve, it follows that the run-tree T is winning (Corollary 3). The infinite ancestor lemma is established in §8.7, together with the soundness theorem (Theorem 22). The idea of the proof of the infinite ancestor lemma is to extend to the infinitary case the connection (mentioned above) appearing in the proof of the progress lemma between finite plays of $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$ and finite prefixes of branches of T. This previous connection shows that any finite branch of T is computed by a rewriting sequence for $\rhd$ which unfolds the non-terminals played by Adam (as well as other intermediate non-terminals). The definition of the game $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$ implies moreover that when Adam plays a non-terminal F, he should play his next move by picking a non-terminal G introduced by the unfolding $F \mapsto \mathcal{R}(F)$. In order to relate an infinite branch b of the run-tree T with an infinite play of $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$, we must therefore ensure that there exists an infinite sequence of non-terminals

$$S \;=\; F_0^b,\, F_1^b,\, F_2^b, \ldots$$

having the following property: the unfolding of F_i^b during the rewriting by $\rhd$ creates the occurrence F_{i+1}^b. We should moreover require that this sequence of non-terminals computes the branch b in the following sense: for every integer i, the tree context $C_i^b[\,]$ generated by the rewriting relation $\rhd$ putting the occurrence F_i^b in head position:

$$\langle\, \varepsilon,\, 1,\, \{0 \mapsto \varepsilon\}\,,\, S^0 \,:\, \bigwedge_{\{1\}} \boxdot_\varepsilon\, q_0 \;\vdash\; S^0 \,:\, q_0 \,\rangle \;\rhd^*\; C_i^b[\,\langle\, \alpha,\, l,\, \Lambda,\, \Gamma \;\vdash\; F_i^b \,:\, q \,\rangle\,]$$

is such that the path from the root of $C_i^b[\,]$ to its hole is a finite prefix of the infinite branch b of the run-tree T. The existence of such a sequence of non-terminals is stated by a technical lemma (Lemma 11. As the proof of Lemma 11 is rather long, it is presented in a separate section (§8.8). This lemma then enables us to relate each infinite branch of the run-tree T with an infinite play of $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$ where Eve follows her winning strategy $\mathcal{W}$. Since this strategy

is winning, all the infinite branches of T also are. This result is stated in §8.7 by Lemma 12, whose proof relates infinite branches of T with infinite plays of $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$ where Eve follows the winning strategy $\mathcal{W}$. The soundness theorem (Theorem 22) follows immediately.

8.5 Progress lemma

The « progress » lemma for the rewriting relation $\triangleright$ is stated as follows:

Lemma 6 (Progress)**.** *Suppose that*

$$\langle\, \varepsilon,\, 1,\, \{0 \mapsto \varepsilon\}\,,\, S^0 \,:\, \bigwedge_{\{1\}} \boxdot_\varepsilon\, q_0 \;\vdash\; S^0 \,:\, q_0 \,\rangle \quad \triangleright^* \quad C[\,\langle\, \alpha,\, l,\, \Lambda,\, \Gamma \;\vdash\; t \,:\, q \,\rangle\,]$$

Then there exists T such that

$$\langle\, \alpha,\, l,\, \Lambda,\, \Gamma \;\vdash\; t \,:\, q \,\rangle \quad \triangleright \quad T$$

It requires some companion lemmas:

Lemma 7. *Suppose that*

$$\langle\, \varepsilon,\, 1,\, \{0 \mapsto \varepsilon\}\,,\, S^0 \,:\, \bigwedge_{\{1\}} \boxdot_\varepsilon\, q_0 \;\vdash\; S^0 \,:\, q_0 \,\rangle \quad \triangleright \quad C[\,\langle\, \alpha,\, l,\, \Lambda,\, \Gamma \;\vdash\; t \,:\, q \,\rangle\,]$$

then $\Gamma \;\vdash\; t \,:\, q$ holds in $\boldsymbol{\gamma}(\mathcal{A})$.

Proof. Easy induction on the length of the rewrite sequence, coming from the fact that by definition the case (1) of the definition of $\triangleright$ introduces appropriate typing contexts obtained by subject reduction, and that the case (2) introduces judgments which already hold. □

The following lemma formalizes the discussion of the beginning of §6.3: the coloring annotation of an occurrence of a non-terminal in a context is precisely the maximal color seen, after normalization, from the root to the leaf where this occurrence is in head position.

Lemma 8. *Suppose that*

$$\langle\, \alpha_0,\, l_0,\, \Lambda_0,\, \Gamma_0 \;\vdash\; s_0 \,:\, q_0 \,\rangle \quad \triangleright^* \quad C[\,\langle\, \alpha,\, l,\, \Lambda,\, \Gamma \;\vdash\; F^{l',\theta}\; \widetilde{t} \,:\, q \,\rangle\,]$$

where F is not introduced by the intermediate steps – in other words, the integer l' labeling the occurrence of F of interest is strictly lesser than l_0. Then $F \,:\, \boxdot_{\Omega(C[\,]_q)}\, \theta \;\in\; \Gamma_0$.

Proof. The proof is by induction on the length of the reduction sequence

$$\langle\, \alpha_0,\, l_0,\, \Lambda_0,\, \Gamma_0 \;\vdash\; s_0 \,:\, q_0 \,\rangle \quad \triangleright^* \quad C[\,\langle\, \alpha,\, l,\, \Lambda,\, \Gamma \;\vdash\; F^{\theta}\; \widetilde{t} \,:\, q \,\rangle\,]$$

If it is 0, then $q = q_0$, $\Gamma_0 = \Gamma$ and $C[\,]_q = [\,]_q$. The proof of $\Gamma_0 \;\vdash\; F^\theta\; \widetilde{t} \,:\, q$ must use an Axiom leaf

$$F \,:\, \bigwedge_{\{1\}} \boxdot_\varepsilon\, \theta \;\vdash\; F \,:\, \theta$$

to type F, together with applications of $\widetilde{t}$ which do not update the color of $F : \boxdot_\varepsilon \theta$ as it is in head position. So we have $F : \boxdot_\varepsilon \theta \in \Gamma_0$, with $\varepsilon = \Omega(C[\,]_q) = \Omega([\,]_q)$.

Let us now consider the inductive step, distinguishing between the cases (1) and (2) of the definition of $\rhd$.

1. Suppose the first rewrite step is of the form (1)

$$\langle\, \alpha_0,\, l_0,\, \Lambda_0,\, \Gamma_0 \;\vdash\; F_k^{l'}\, \widetilde{t} : q_0 \,\rangle \;\rhd\; \langle\, \alpha_0,\, l_0{+}1,\, \Lambda',\, \Gamma' \;\vdash\; \rho(t')[\widetilde{x} \leftarrow \widetilde{t}] : q_0 \,\rangle$$

with $\mathcal{R}(F_k) = \lambda \widetilde{x}.\, t'$ and ρ the relabeling operation of non-terminals F_i to $F_i^{l_0}$. Since F is not introduced by intermediate reduction steps, it is not labeled with l_0. By the induction hypothesis, we obtain that $F : \boxdot_{\Omega(C[\,]_q)} \theta \in \Gamma' \setminus \{G^{l_0} \mid G \in \mathcal{N}\}$. The case (1) in the definition of $\rhd$ implies that $\Gamma' \setminus \{G^{l_0} \mid G \in \mathcal{N}\} \subseteq \Gamma_0$, so that $F : \boxdot_{\Omega(C[\,]_q)} \theta \in \Gamma_0$. Since the first step of rewriting we considered did not increase the context $C[\,]_q$, the result holds.

2. If the first rewrite step is of the form (2), we have

$$\langle\, \alpha_0,\, l_0,\, \Lambda_0,\, \Gamma_0 \;\vdash\; a\, t_1 \cdots t_n : q_0 \,\rangle \;\rhd\; \langle\, \alpha_0,\, q_0 \,\rangle \left(\cdots \langle\, \alpha_0 i,\, l_0 + 1,\, \boxdot_{q_{i,j}} \Lambda_0,\, \Gamma_{i,j} \;\vdash\; t_i : q_{i,j} \,\rangle \cdots \right)$$

for $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, k_i\}$. So there exists a family $T_{i,j}$ of Σ_{sound}-labeled trees such that

$$C[\langle\, \alpha,\, l,\, \Lambda,\, \Gamma \;\vdash\; F^\theta\, \widetilde{t} : q \,\rangle] \;=\; \langle\, \alpha_0,\, q_0 \,\rangle\; T_{1,1} \;\cdots\; T_{n,k_n}$$

Moreover, there exists precisely one such couple (i, j) such that $T_{i,j} = C'[\langle\, \alpha,\, l,\, \Lambda,\, \Gamma \;\vdash\; F^\theta\, \widetilde{t} : q \,\rangle]$, and by definition of $\rhd$ we have that

$$\langle\, \alpha_0 i,\, l_0{+}1,\, \boxdot_{q_{i,j}} \Lambda_0,\, \Gamma_{i,j} \;\vdash\; t_i : q_{i,j} \,\rangle \;\rhd^*\; C'[\langle\, \alpha,\, l,\, \Lambda,\, \Gamma \;\vdash\; F^\theta\, \widetilde{t} : q \,\rangle]$$

Note that

$$C[\,]_q \;=\; \langle\, \alpha_0,\, q_0 \,\rangle\; T_{1,1} \;\cdots\; [C'[\,]_q]_{q_{i,j}} \;\cdots\; T_{n,k_n}$$

so that $\Omega(C[\,]_q) = \max(\Omega(q_{i,j}), \Omega(C'[\,]_q))$.

By induction hypothesis, $F : \boxdot_{\Omega(C'[\,]_q)} \theta \in \Gamma_{i,j}$. In the derivation of $\Gamma_0 \;\vdash\; a\; t_1 \cdots t_n : q_0$, the use of the Application rule colors $\Gamma_{i,j}$ with $\Omega(q_{i,j})$. Since $\Omega(C[\,]_q) = \max(\Omega_(q_{i,j}), \Omega(C'[\,]_q))$, $F : \boxdot_{\Omega(C[\,]_q)} \theta \in \Gamma_0$, so that the result holds.

□

The next lemma clarifies the technical rôle of the partial function Λ occurring in leaves of the second form. Recall that from a leaf

$$\langle\, \alpha_0,\, l_0,\, \Lambda_0,\, \Gamma_0 \;\vdash\; F\, \widetilde{t} : q_0 \,\rangle$$

the reduction relation $\rhd$ will use the case (1) to expand F, and the newly-introduced non-terminals will be labeled with l_0. The purpose of Λ is to store the maximal color seen since every such introduction of non-terminals.

Lemma 9. *If* $\langle\, \alpha_0, l_0, \Lambda_0, \Gamma_0 \;\vdash\; F\ \widetilde{t} : q_0 \,\rangle \;\; \rhd^+ \;\; C[\langle\, \alpha, l, \Lambda, \Gamma \;\vdash\; s : q \,\rangle]$, *then* $\Lambda(l_0) = \Omega(C[\,]_q)$.

Proof. The proof is by induction on the length of the reduction sequence. For the base case, the rewriting is

$$\langle\, \alpha_0, l_0, \Lambda_0, \Gamma_0 \;\vdash\; F\ \widetilde{t} : q_0 \,\rangle \;\; \rhd \;\; \langle\, \alpha_0, l_0 + 1, \Lambda, \Gamma' \;\vdash\; \rho(F)[\widetilde{x} \leftarrow \widetilde{t}] : q_0 \,\rangle$$

with $\Lambda = \Lambda_0\{l_0 \mapsto \varepsilon\}$, $q = q_0$ and $C = [\,]_{q_0}$, such that $\Lambda(l_0) = \varepsilon = \Omega([\,]_{q_0})$.

For the inductive step, we write

$$\begin{array}{lll} \langle\, \alpha_0, l_0, \Lambda_0, \Gamma_0 \;\vdash\; F\ \widetilde{t} : q_0 \,\rangle & \rhd^+ & C'[\langle\, \alpha', l', \Lambda', \Gamma' \;\vdash\; s' : q' \,\rangle] \\ & \rhd & C[\langle\, \alpha, l, \Lambda, \Gamma \;\vdash\; s : q \,\rangle] \end{array}$$

and distinguish two cases, depending on whether the last step of the reduction comes from the case (1) or (2) of the definition of $\rhd$:

- if it comes from (1), then $C = C'$, $q = q'$ and $\Lambda = \Lambda'\{l' \mapsto \varepsilon\}$. The induction hypothesis gives $\Lambda'(l_0) = \Omega(C'[\,]_q)$, from which we deduce that $\Lambda(l_0) = \Omega(C[\,]_q)$.
- if it comes from (2), then $C[\,]_q = C'[\,\langle \alpha', q' \rangle \;\cdots [\,]_q \cdots]_{q'}$, and $\Lambda = \boxdot_{\Omega(q)}\, \Lambda'$. By induction hypothesis, $\Lambda'(l_0) = \Omega(C'[\,]_{q'})$. By composition of contexts (Proposition 17), we obtain

$$\Lambda(l_0) \;=\; \max(\Lambda'(l_0), \Omega(q)) \;=\; \max\left(\Omega\left(C'[\,]_{q'}\right)\right), \Omega\left(\langle \alpha', q' \rangle \;\cdots [\,]_q \cdots\right)) \;=\; \Omega(C[\,]_q)$$

from which we conclude.

□

Proof of Lemma 6. Suppose that

$$\langle\, \varepsilon, 1, \{0 \mapsto \varepsilon\}, S^0 : \bigwedge_{\{1\}} \boxdot_\varepsilon\, q_0 \;\vdash\; S^0 : q_0 \,\rangle \;\; \rhd^* \;\; C[\langle\, \alpha, l, \Lambda, \Gamma \;\vdash\; t : q \,\rangle] \tag{8.5}$$

Then by Lemma 7 we have that $\Gamma \;\vdash\; t : q$ holds in $\gamma(\mathcal{A})$. If $t = a\ t_1 \cdots t_n$ is in head normal form, then we can apply the case (2) of the definition of $\rhd$ and obtain T as required.

Otherwise, $t = F_i^{l',\theta}\ \widetilde{s}$, and our aim is to apply the case (1) of the definition of $\rhd$. However, this requires to ensure that the context $\Gamma_{(F_i : \boxdot_{\Lambda(l')}\ \theta)}$ is defined. The goal is to extract from the rewriting sequence (8.5) a play in $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$ in which Eve conforms to $\mathcal{W}$, and in which Adam raises the question $F_i : \boxdot_{\Lambda(l')}\ \theta$.

The rewriting sequence (8.5) factors as

$$\begin{array}{rl} & \langle\, \varepsilon, 1, \{0 \mapsto \varepsilon\}, S^0 : \bigwedge_{\{1\}} \boxdot_\varepsilon\, q_0 \;\vdash\; S^0 : q_0 \,\rangle \\ \rhd^* & C_1[\langle\, \alpha_1, l_1, \Lambda_1, \Gamma_1 \vdash\; F_{i_1}^{l_0,\theta_1}\ \widetilde{t_1} : q_1 \,\rangle] \\ \rhd^* & C_1[C_2[\langle\, \alpha_2, l_2, \Lambda_2, \Gamma_2 \vdash\; F_{i_2}^{l_1,\theta_2}\ \widetilde{t_2} : q_2 \,\rangle]] \\ \rhd^* & C_1[C_2[\cdots C_n[\langle\, \alpha_n, l_n, \Lambda_n, \Gamma_n \vdash\; F_{i_n}^{l_{n-1},\theta_n}\ \widetilde{t_n} : q_n \,\rangle] \cdots] \\ = & C[\langle\, \alpha, l, \Lambda, \Gamma \;\vdash\; F_i^{l',\theta}\ \widetilde{s} : q \,\rangle] \end{array}$$

with $l_0 = 1$ and $C = C_1[C_2[\cdots[C_n]\cdots]]$. Note that in this factorization of the rewriting sequence (8.5) we focus at each step on a non-terminal which occurs in head position and has been introduced by the previous one. The idea is that this computes a finite prefix of a branch of the run-tree we want to build, and that we can see this process as an interaction between Adam and Eve, the former's moves corresponding to the exploration of this branch. We can indeed extract from this sequence of rewritings the partial play of $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$

$$(S : \boxdot_\varepsilon\ q_0)\quad \Gamma_{(S:\boxdot_\varepsilon\ q_0)}\quad \Big(F_{i_1} : \boxdot_{\Omega(C_1[]_{q_1})}\ \theta_1\Big)\quad \Gamma_{\Big(F_{i_1}:\boxdot_{\Omega(C_1[]_{q_1})}\ \theta_1\Big)}\quad \cdots\quad \big(F_{i_n} : \boxdot_{\Omega(C_n[]_{q_n})}\ \theta_n\big)$$

as follows: for every k, the reduction

$$\langle\, \alpha_k,\, l_k,\, \Lambda_k,\, \Gamma_k \vdash\ F_{i_k}^{l_{k-1},\theta_k}\ \widetilde{t_k} : q_k \,\rangle \rhd^*\quad C_{k+1}[\,\langle\, \alpha_{k+1},\, l_{k+1},\, \Lambda_{k+1},\, \Gamma_{k+1} \vdash\ F_{i_{k+1}}^{l_k,\theta_{k+1}}\ \widetilde{t_{k+1}} : q_{k+1} \,\rangle\,]$$

factors as

$$\begin{array}{ll} & \langle\, \alpha_k,\, l_k,\, \Lambda_k,\, \Gamma_k \vdash\ F_{i_k}^{l_{k-1},\theta_k}\ \widetilde{t_k} : q_k \,\rangle \\ \rhd & \langle\, \alpha_k,\, l_k+1,\, \Lambda_k\{l_k \mapsto \varepsilon\},\, \Gamma'_k \vdash\ \rho(t')[\widetilde{x} \leftarrow \widetilde{t_k}] : q_k \,\rangle \\ \rhd^* & C_{k+1}[\,\langle\, \alpha_{k+1},\, l_{k+1},\, \Lambda_{k+1},\, \Gamma_{k+1} \vdash\ F_{i_{k+1}}^{l_k,\theta_{k+1}}\ \widetilde{t_{k+1}} : q_{k+1} \,\rangle\,] \end{array}$$

where ρ relabels a non-terminal G to G^{l_k}, and where $\mathcal{R}(F_{i_k}) = \lambda\widetilde{x}.t'$. By definition of $\rhd$, the context $\Gamma_{(F_{i_k}:\boxdot_{\Lambda_k(l_{k-1})}\ \theta_k)}$ is well-defined, and

$$\Gamma'_k \subseteq \Gamma_k \cup \rho\Big(\Gamma_{(F_{i_k}:\boxdot_{\Lambda_k(l_{k-1})}\ \theta_k)}\Big)$$

Lemma 8 then implies that $F_{i_{k+1}}^{l_k} : \boxdot_{C_{k+1}[]_{q_{k+1}}}\ \theta_{k+1} \in \Gamma'_k$. Since $F_{i_{k+1}}^{l_k}$ was introduced at the step l_k, it can not belong to Γ_k, and we have

$$F_{i_{k+1}} : \boxdot_{C_{k+1}[]_{q_{k+1}}}\ \theta_{k+1} \in \Gamma_{(F_{i_k}:\boxdot_{\Lambda_k(l_{k-1})}\ \theta_k)} \tag{8.6}$$

Moreover, by Lemma 9, $\Lambda_k(l_{k-1}) = \Omega(C_k[]_{q_k})$. This means that, for every $k < n$ and every node $\Big(F_{i_k} : \boxdot_{\Omega(C_k[]_{q_k})}\ \theta_k\Big)$, and starting for $k = 0$ from

$$\Big(F_0 : \boxdot_{\Omega(C_0[]_{q_0})}\ \theta_0\Big) = (S : \boxdot_\varepsilon\ q_0)$$

Eve can answer with $\Gamma_{\Big(F_{i_k}:\boxdot_{\Omega(C_k[]_{q_k})}\ \theta_k\Big)}$, from which Adam can play $F_{i_{k+1}} : \boxdot_{C_{k+1}[]_{q_{k+1}}}\ \theta_{k+1}$ since (8.6) holds.

As Eve follows her winning strategy $\mathcal{W}$ in this partial play, she can answer to

$$\big(F_{i_n} : \boxdot_{\Omega(C_n[]_{q_n})}\ \theta_n\big)$$

with

$$\Gamma_{(F_{i_n}:\boxdot_{\Lambda(l')}\ \theta)} = \mathcal{W}\big(F_{i_n} : \boxdot_{\Omega(C_n[]_{q_n})}\ \theta_n\big)$$

otherwise she would loose on this finite play. So $\Gamma_{(F_{i_n}:\boxdot_{\Lambda(l')}\ \theta)}$ is well-defined, and the case (1) of the definition of the rewrite rule $\rhd$ can be applied to $t = F_i^{l',\theta}\ \widetilde{s}$.

8.6 Fair production lemma

Now that we proved that the rewriting relation $\rhd$ does not get stuck on leaves of the form

$$\langle\, \alpha,\, l,\, \Lambda,\, \Gamma \;\vdash\; t \,:\, q \,\rangle$$

we can show that maximal fair[1] rewriting sequences compute a run-tree of $\mathcal{A}$ over $\langle\, \mathcal{G} \,\rangle$. Recall from Definition 20 that we can consider ranked trees over a signature as a domain, and therefore take the supremum $\bigvee$ of a directed family of Σ-labeled *ranked* trees. The set of Σ_{sound}-labeled *unranked* trees is a domain as well when we consider the prefix order:

$$t \preccurlyeq t' \iff \mathit{Dom}(t) \subseteq \mathit{Dom}(t') \quad \text{and} \quad \alpha \in \mathit{Dom}(t) \Rightarrow t(\alpha) = t'(\alpha)$$

Consider a family $(T_j)_{j\in J}$ of Σ_{sound}-labeled trees such that $\left(T_j^\sharp\right)_{j\in J}$ is directed. Then this directed family has a supremum $\bigvee_{j\in J} T_j^\sharp$. This supremum enables us to compute a run-tree from all the finite prefixes obtained by finite reduction sequences.

Lemma 10 (Fair production lemma)**.** *Consider a fair maximal rewriting sequence*

$$T_0 \;=\; \langle\, \varepsilon,\, 1,\, \{0 \mapsto \varepsilon\},\, S^0 \,:\, \bigwedge_{\{1\}} \boxdot_\varepsilon\, q_0 \;\vdash\; S^0 \,:\, q_0 \,\rangle \;\rhd\; T_1 \;\rhd\; T_2 \;\rhd\; \cdots$$

Then the tree $T \;=\; \bigvee_{i\in\mathbb{N}} T_i^\sharp$ *is a run-tree of* $\mathcal{A}$ *over* $\langle\, \mathcal{G} \,\rangle$.

Proof. The family of Σ_{sound}-labeled trees

$$T_0^\sharp \;=\; \langle\, \varepsilon,\, q_0 \,\rangle \;\preccurlyeq\; T_1^\sharp \;\preccurlyeq\; T_2^\sharp \;\preccurlyeq\; \cdots$$

is directed by definition of $\rhd$ and $(\cdot)^\sharp$, so that it has a supremum $T \;=\; \bigvee_{i\in\mathbb{N}} T_i^\sharp$. Since every $T_i^\sharp$ is labeled by an element of $\mathit{Dom}(\langle\, \mathcal{G} \,\rangle) \times Q$, the limit tree T is as well a $(\mathit{Dom}(\langle\, \mathcal{G} \,\rangle) \times Q)$-labeled unranked tree. From the definition of an execution tree of $\mathcal{A}$ over $\langle\, \mathcal{G} \,\rangle$ (Definition 11 on p. 49), we need to check that:

- $T(\varepsilon) \;=\; \langle\, \varepsilon,\, q_0 \,\rangle$: this follows from $T_0^\sharp \;\preccurlyeq\; T$.
- and that for every $\beta \in \mathit{dom}(T)$, denoting $T(\beta) \;=\; \langle\, \alpha,\, q \,\rangle$ and $a \;=\; \langle\, \mathcal{G} \,\rangle(\alpha)$, there exists $S \subset \mathbb{N} \times Q$ satisfying $\delta(q,\, a)$ and such that

  $$\forall (i, q') \in S,\; \exists j \in \mathbb{N},\; (\beta j \in \mathit{dom}(T)) \wedge (T(\beta j) \;=\; \langle\, \alpha i,\, q' \,\rangle)$$

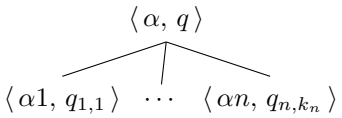
  Since T is computed by an increasing sequence of $(\mathit{Dom}(\langle\, \mathcal{G} \,\rangle) \times Q)$-labeled trees, there exists $k \in \mathbb{N}$ such that $T_k^\sharp(\beta) \;=\; \langle\, \alpha,\, q \,\rangle$. Two different situations arise:

  - If $T_k(\beta) \;=\; \langle\, \alpha,\, q \,\rangle$, then the node β was generated by the case (2) of the definition of $\rhd$. It therefore exists

    $$\{(i, q_{i,j}) \mid 1 \leq i \leq n,\; 1 \leq j \leq k_i\} \quad \text{satisfying} \quad \delta(q, a)$$

    such that the children of β in $T_k^\sharp$ are

[1] Recall that a sequence is *fair* when it eventually rewrites every leaf of the form $\langle\, \alpha,\, l,\, \Lambda,\, \Gamma \;\vdash\; t \,:\, q \,\rangle$.



Since $T_k^\sharp \preccurlyeq T$, $T(\beta)$ has the same children as $T_k^\sharp(\beta)$, so that their set satisfies $\delta(q, a)$ as well.

– If $T_k(\beta) \neq \langle\, \alpha, q \,\rangle$, then it is of the shape $\langle\, \alpha, l, \Lambda, \Gamma \vdash t : q \,\rangle$. In a sense, it means that we did not pick k "late enough" in the reduction sequence: by the progress lemma (Lemma 6), there exists $k' > k$ such that $T'_k(\beta) = \langle\, \alpha, q \,\rangle$. We apply it the argument just above.

□

8.7 Soundness theorem

A crucial technical lemma in the proof of the soundness theorem is the infinite ancestor lemma (Lemma 12), which requires first a preliminary lemma (Lemma 11, proved in §8.8). The infinite ancestor lemma states that every infinite branch b of the run-tree T constructed in the fair production lemma (Lemma 10) is computed from an infinite chain of unfoldings

$$S^0,\ F^{i_1}_{k_{i_1}},\ F^{i_2}_{k_{i_2}}, \dots$$

in which each non-terminal is introduced by the unfolding of the previous one. This enables us to relate the construction of the infinite branch b of the run-tree T with the infinite play of $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$ where Adam picks these non-terminals and Eve plays her winning strategy $\mathcal{W}$. Lemma 12 then ensures that every infinite branch of the run-tree constructed in the fair production lemma (Lemma 10) is winning, by relating the colors occurring along it with the colors played in the corresponding play in $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$.

As we explained in our overview of the proof (§8.4), the proof of the progress lemma (Lemma 6) explicates a construction which relates a finite play in $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$ where Eve follows her winning strategy $\mathcal{W}$, with the computation of a finite branch of the run-tree T computed by $\rhd$. The relation between infinite plays in $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$ where Eve follows $\mathcal{W}$ and infinite branches of T extends this construction to the infinitary case. As we will see in the proof of Lemma 12, this infinitary extension requires to use the following technical lemma:

Lemma 11. *Consider an infinite rewriting sequence computing an infinite branch of the run-tree of $\mathcal{A}$ over $\langle\, \mathcal{G} \,\rangle$ constructed in the fair production lemma (Lemma 10):*

$$\begin{array}{lcl} \langle\, \varepsilon, 1, \{0 \mapsto \varepsilon\}, S^0 : \bigwedge_{\{1\}} \boxdot_\varepsilon q_0 \vdash S^0 : q_0 \,\rangle & = & \langle\, \alpha_1, 1, \Lambda_1, \Gamma_1 \vdash t_1 : q_1 \,\rangle \\ & \rhd & C_1[\,\langle\, \alpha_2, 2, \Lambda_2, \Gamma_2 \vdash t_2 : q_2 \,\rangle\,] \\ & \rhd & C_1[C_2[\,\langle\, \alpha_3, 3, \Lambda_3, \Gamma_3 \vdash t_3 : q_3 \,\rangle\,]] \\ & \rhd & \cdots \end{array}$$

We write $j \prec i$ *when*

$$t_j \;=\; F^i_{k_j}\ \widetilde{s_j}$$

meaning that the non-terminal $F^i_{k_j}$ *introduced at the* i^{th} *step is to be unfolded at the* j^{th}*. Then there exists an infinite sequence of integers* $(i_n)_{n\in\mathbb{N}}$ *such that*

$$\cdots \prec i_3 \prec i_2 \prec i_1 \prec i_0 = 0$$

Note that $i_1 = 1$ *by construction.*

Proof. See §8.8. □

This technical lemma enables us to prove the infinite ancestor lemma, which relates the infinite branches of the run-tree T of $\mathcal{A}$ over $\langle\, \mathcal{G} \,\rangle$ computed by the fair production lemma (Lemma 10) to infinite plays of $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$ where Eve plays following her winning strategy $\mathcal{W}$.

Lemma 12 (Infinite ancestor lemma)**.** *Consider a fair maximal rewriting sequence*

$$T_0 \;=\; \langle\, \varepsilon,\, 1,\, \{0 \mapsto \varepsilon\}\,,\, S^0 : \bigwedge_{\{1\}} \boxdot_\varepsilon\ q_0 \;\vdash\; S^0 : q_0 \,\rangle \;\rhd\; T_1 \;\rhd\; T_2 \;\rhd\; \cdots$$

For every infinite branch b *of the run-tree* $T = \bigvee_{i\in\mathbb{N}} T^\sharp_i$ *of* $\mathcal{A}$ *over* $\langle\, \mathcal{G} \,\rangle$*, there exists an infinite play in* $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$ *with the same color.*

Proof. Consider an infinite branch b of T, identified by the sequence of nodes $(\alpha_i)_{i\in\mathbb{N}}$ it contains. Note that $\alpha_0 = \varepsilon$. We can extract from the maximal fair rewriting sequence

$$T_0 \;=\; \langle\, \varepsilon,\, 1,\, \{0 \mapsto \varepsilon\}\,,\, S^0 : \bigwedge_{\{1\}} \boxdot_\varepsilon\ q_0 \;\vdash\; S^0 : q_0 \,\rangle \;\rhd\; T_1 \;\rhd\; T_2 \;\rhd\; \cdots \tag{8.7}$$

an infinite rewriting sequence

$$\begin{array}{lcl}
\langle\, \varepsilon,\, 1,\, \{0 \mapsto \varepsilon\}\,,\, S^0 : \bigwedge_{\{1\}} \boxdot_\varepsilon\ q_0 \;\vdash\; S^0 : q_0 \,\rangle & = & \langle\, \alpha_1,\, 1,\, \Lambda_1,\, \Gamma_1 \;\vdash\; t_1 : q_1 \,\rangle \\
& \rhd & C_1[\,\langle\, \alpha_2,\, 2,\, \Lambda_2,\, \Gamma_2 \;\vdash\; t_2 : q_2 \,\rangle\,] \\
& \rhd & C_1[C_2[\,\langle\, \alpha_3,\, 3,\, \Lambda_3,\, \Gamma_3 \;\vdash\; t_3 : q_3 \,\rangle\,]] \\
& \rhd & \cdots
\end{array}$$

computing the branch b, by removing from (8.7) every rewriting step applied to a leaf

$$\langle\, \alpha',\, l,\, \Lambda,\, \Gamma \vdash t : q \,\rangle$$

where α' does not belong to b. By Lemma 11, there exists an infinite sequence

$$\cdots \prec i_3 \prec i_2 \prec i_1 \prec i_0 = 0$$

of indexes such that $t_{i_j} \;=\; F^{i_{j-1}}_{k_{i_j}}\ \widetilde{s_{i_j}}$. This means that the infinite rewriting sequence computing b factors as

$$
\begin{array}{ll}
& \langle\, \varepsilon,\, 1,\, \{0 \mapsto \varepsilon\}\,,\, S^0 \,:\, \bigwedge_{\{1\}} \boxdot_\varepsilon\, q_0 \;\vdash\; S^0 \,:\, q_0 \,\rangle \\
= & \langle\, \alpha_{i_1},\, i_1,\, \Lambda_{i_1},\, \Gamma_{i_1} \;\vdash\; F^{i_0}_{k_{i_1}}\; \widetilde{s_{i_1}} \,:\, q_{i_1} \,\rangle \\
\rhd^+ & C'_1[\,\langle\, \alpha_{i_2},\, i_2,\, \Lambda_{i_2},\, \Gamma_{i_2} \;\vdash\; F^{i_1}_{k_{i_2}}\; \widetilde{s_{i_2}} \,:\, q_{i_2} \,\rangle\,] \\
\rhd^+ & C'_1[C'_2[\,\langle\, \alpha_{i_3},\, i_3,\, \Lambda_{i_3},\, \Gamma_{i_3} \;\vdash\; F^{i_2}_{k_{i_3}}\; \widetilde{s_{i_3}} \,:\, q_{i_3} \,\rangle\,]] \\
\rhd^+ & \cdots \\
\rhd^+ & C'_1[C'_2 \cdots [C'_n[\,\langle\, \alpha_{i_{n+1}},\, i_{n+1},\, \Lambda_{i_{n+1}},\, \Gamma_{i_{n+1}} \;\vdash\; F^{i_n}_{k_{i_{n+1}}}\; \widetilde{s_{i_{n+1}}} \,:\, q_{i_{n+1}} \,\rangle\,]] \cdots]] \\
\rhd^+ & \cdots
\end{array}
$$

Note that, for every $n \in \mathbb{N}$, we can consider only the finite prefix of this infinite rewriting sequence ending on

$$C'_1[C'_2 \cdots C'_n[\,\langle\, \alpha_{i_{n+1}},\, i_{n+1},\, \Lambda_{i_{n+1}},\, \Gamma_{i_{n+1}} \;\vdash\; F^{i_n}_{k_{i_{n+1}}}\; \widetilde{s_{i_{n+1}}} \,:\, q_{i_{n+1}} \,\rangle\,] \cdots]]$$

The situation is then precisely the same as in the proof of the progress lemma (Lemma 6), so that we can similarly use Lemma 8 and Lemma 9 to prove the existence of a play

$$(S \,:\, \boxdot_\varepsilon\, q_0)\;\; \Gamma_{(S \,:\, \boxdot_\varepsilon\, q_0)}\;\; \left(F_{k_{i_1}} \,:\, \boxdot_{\Omega(C'_1[\,]_{q_{i_1}})}\, \theta_{i_1}\right)\;\; \Gamma_{\left(F_{k_{i_1}} \,:\, \boxdot_{\Omega(C'_1[\,]_{q_{i_1}})}\, \theta_{i_1}\right)}\;\; \cdots\;\; \left(F_{k_{i_n}} \,:\, \boxdot_{\Omega(C'_n[\,]_{q_{i_n}})}\, \theta_{i_n}\right)$$

in $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$. By doing this for increasing values of n, we obtain an infinite play in $\mathsf{Adamic}(\mathcal{G}, \mathcal{A})$ where Eve plays according to her winning strategy $\mathcal{W}$, so that the maximal color occurring infinitely often in the sequence

$$\Omega(C'_1[\,]_{q_{i_1}})\;\;,\;\; \Omega(C'_2[\,]_{q_{i_2}})\;\;,\;\; \cdots$$

is even. The sequence

$$C'_1[C'_2[\cdots C'_n[\,]_{q_{i_n}} \cdots]_{q_{i_2}}]_{q_{i_1}} \tag{8.8}$$

converges to b, in the sense that for every finite prefix of b there exists n such that (8.8) contains this finite prefix. Note that the sequence of contexts $\left(C'_j[\,]_{q_{i_j}}\right)_{j \in \mathbb{N}}$ can be understood as a factorization of the infinite branch b into infinitely many finite parts. As discussed at the end of the completeness proof, $\left(\Omega(C'_j[\,]_{q_{i_j}})\right)_{j \in \mathbb{N}}$ is the sequence containing the maximal color seen along every such finite part $C'_j[\,]_{q_{i_j}}$ of the infinite branch b. The color $\Omega(C'_j[\,]_{q_{i_j}})$ is precisely the maximal one seen from the root of $C'_j[\,]_{q_{i_j}}$ – excluded – to its hole $[\,]_{q_{i_j}}$ – included. Note that C'_j may be empty; this is precisely the case when $\Omega(C'_j[\,]_{q_{i_j}}) = \varepsilon$. However, the productivity of the recursion scheme ensures that no infinite sequence of ε may occur in a branch. It follows that every color occurring along b, except the one of the root, is taken into account in the computation of this sequence of colors. This implies that the color of b is the maximal color occurring infinitely often in the sequence $\left(\Omega(C'_j[\,]_{q_{i_j}})\right)_{j \in \mathbb{N}}$. □

Since $\mathcal{W}$ is a winning strategy, every infinite branch of T is winning. It follows that T is a winning run-tree of $\mathcal{A}$ over $\langle\, \mathcal{G} \,\rangle$:

Corollary 3. *Consider a fair maximal rewriting sequence*

$$T_0 \;=\; \langle\, \varepsilon,\, 1,\, \{0 \mapsto \varepsilon\}\,,\, S^0 \,:\, \bigwedge_{\{1\}} \boxdot_\varepsilon\, q_0 \;\vdash\; S^0 \,:\, q_0 \,\rangle \;\rhd\; T_1 \;\rhd\; T_2 \;\rhd\; \cdots$$

The run-tree $T \;=\; \bigvee_{i\in\mathbb{N}} T_i^\sharp$ *over* $\mathcal{A}$ *over* $\langle\, \mathcal{G} \,\rangle$ *is winning.*

We may now prove the soundness theorem.

Theorem 22 (Soundness)**.** *Let* $\mathcal{A}$ *be an alternating parity automaton and* $\mathcal{G}$ *be a higher-order recursion scheme. If Eve has a winning strategy in* $\mathsf{Adamic}(\mathcal{G},\mathcal{A})$*, then* $\mathcal{A}$ *has a winning execution over* $\langle\, \mathcal{G} \,\rangle$*.*

Proof. Let $\mathcal{W}$ be a winning strategy for Eve. Its existence allows to define $\rhd$ and to obtain the lemmas proved in this section. Using the progress lemma (Lemma 6), we build a maximal fair rewriting sequence

$$T_0 \;=\; \langle\, \varepsilon,\, 1,\, \{0 \mapsto \varepsilon\}\,,\, S^0 \,:\, \bigwedge_{\{1\}} \boxdot_\varepsilon\, q_0 \;\vdash\; S^0 \,:\, q_0 \,\rangle \;\rhd\; T_1 \;\rhd\; T_2 \;\rhd\; \cdots$$

which is potentially infinite. By the fair production lemma (Lemma 10), this sequence computes a run-tree of $\mathcal{A}$ over $\langle\, \mathcal{G} \,\rangle$. If the sequence is finite, then the run-tree is finite and therefore winning. Else, it has infinite branches, which are winning by Lemma 12, so that $\mathcal{A}$ accepts $\langle\, \mathcal{G} \,\rangle$. □

8.8 Proof of Lemma 11

In this section, we present the proof of Lemma 11 due to Kobayashi and Ong [KO, Appendix B]. This lemma is crucial in the original proof of soundness of Kobayashi and Ong's theorem (Theorem 16) as well as in our modal reformulation in the type system $\boldsymbol{\gamma}(\mathcal{A})$ (Theorem 19), as it ensures that every infinite branch of the run-tree computed by the relation $\rhd$ is associated to an infinite play of the corresponding parity game $\mathit{Adamic}(\mathcal{G},\mathcal{A})$ or $\mathsf{Adamic}(\mathcal{G},\mathcal{A})$ where Eve follows her winning strategy. Equivalently, we can relate the infinite branches of the run-tree with infinite branches of the infinite derivation of $KO_{fix}(\mathcal{G},\mathcal{A})$ or $\boldsymbol{\gamma}_{fix}(\mathcal{G},\mathcal{A})$ corresponding to Eve's winning strategy.

Since this lemma is only concerned with rewriting, and does not mention coloring, most of the technical annotations of the Σ_{sound}-labels can be dropped. Moreover, we focus here on the exploration of an infinite branch, and not on the generation of a whole infinite tree. This leads us to redefine the rewriting relation $\rhd$ for this section. We set:

1. $\langle\, l,\, F^{l'}\ \widetilde{t} \,\rangle \;\rhd\; \langle\, l+1,\, \rho(t')[\widetilde{x} \leftarrow \widetilde{t}] \,\rangle$
   for $\mathcal{R}(F) \;=\; \lambda\widetilde{x}.\, t$ and ρ the function rewriting every non-terminal G to its annotated counterpart G^l.
2. and $\langle\, l,\, a\ t_1 \cdots t_n \,\rangle \;\rhd\; \langle\, l+1,\, t_i \,\rangle$.

Recall that we write $j \prec i$ when

$$t_j \quad = \quad F^i_{k_j} \ \widetilde{s_j}$$

meaning that the non-terminal $F^i_{k_j}$ introduced at the i^{th} step is to be unfolded at the j^{th}. Lemma 11 translates to this redefinition of $\rhd$ as:

Lemma 13. *Consider an infinite rewriting sequence*

$$\langle\, 1,\, S^0 \,\rangle \;=\; \langle\, 1,\, t_1 \,\rangle \quad \rhd \quad \langle\, 2,\, t_2 \,\rangle \quad \rhd \quad \langle\, 3,\, t_3 \,\rangle \quad \rhd \quad \cdots$$

Then there exists an infinite sequence of integers $(i_n)_{n\in\mathbb{N}}$ such that

$$\cdots \;\prec\; i_3 \;\prec\; i_2 \;\prec\; i_1 \;\prec\; i_0 \;=\; 0$$

Note that $i_1 = 1$ by construction. This subtle proof proceeds in two steps:

1. First, an intersection type system is defined, in which a higher-order recursion scheme has the refinement type ∞ if and only if it admits an infinite reduction sequence for $\rhd$.

2. Then, considering a higher-order recursion scheme $\mathcal{G}$ which has an infinite rewriting sequence, we define two variants $\mathcal{G}'$ and $\mathcal{G}''$.

   - The first one only differs from $\mathcal{G}$ by the fact that every of its non-terminals is explicitly labeled with the integer corresponding to the rewriting step it was introduced at. It has therefore an infinite reduction sequence. Note that we need infinitely many non-terminals for this purpose: we will consider the appropriate notion of *non-deterministic* higher-order recursion scheme in this goal.
   - The second one only allows to use the rewriting rules of $\mathcal{G}$ until a fixed depth N of rewritings, but no more: $\mathcal{G}''$ will not have an infinite reduction sequence, and consequently will not admit the type ∞ in the intersection type system previously defined. N will be taken "big enough": it will be the maximal size of the set of types for $\mathcal{G}$ in this type system.

   We then proceed by contradiction, supposing that $\preccurlyeq$ is a well-founded relation. This allows us to use well-founded induction (a principle recalled on p. 43) on $\preccurlyeq$ to prove that the non-terminals of $\mathcal{G}'$ can not have types their counterparts in $\mathcal{G}''$ do not have, and to reach the contradictory statement that $\mathcal{G}'$ does not have an infinite reduction sequence.

8.8.1 An intersection type system detecting infinite reduction sequences

The construction of $\mathcal{G}'$ we just sketched requires an extension of the notion of higher-order recursion scheme with *non-determinism*:

Definition 34 (Non-deterministic HORS)**.** A *non-deterministic* higher-order recursion scheme is a tuple $\mathcal{G} = \langle \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$, which differs from deterministic higher-order recursion schemes by the fact that

- the set $\mathcal{N}$ of non-terminals may be either finite or countable,
- the function $\mathcal{R}$ now maps a non-terminal F to a *set* of closed terms $\lambda x_1. \ldots \lambda x_n. t \ :: \ \kappa(F)$ over the set of variables $\mathcal{V}$, with constants in $\mathcal{N} \uplus \Sigma$, and such that $t :: o$ is a term without abstractions.

Generalizing the deterministic case, we write $F \ \to \ \lambda x_1. \ldots \lambda x_n. t$ when $\lambda x_1. \ldots \lambda x_n. t \ \in \ \mathcal{R}(F)$.

We define over non-deterministic HORS the rewriting relation $\rhd$, which explores branches obtained by reduction just as $\triangleright$ would, with the minor alteration that the labels, which are not necessary in this subsection, are omitted:

$$\text{if } F \ \to \ \lambda\widetilde{x}. t' \text{ then } F \ \widetilde{t} \ \rhd \ t'[\widetilde{x} \leftarrow \widetilde{t}]$$

$$a \ t_1 \cdots t_n \ \rhd \ t_i$$

The grammar of intersection types we define is similar to the one considered in Chapter 5, with $Q = \{\infty\}$, but the type system itself will be defined differently. In other terms, intersection types are defined by the grammar:

$$\delta \ ::= \ \infty \mid \bigwedge_{i \in I} \delta_i \to \delta$$

The intersection operator is idempotent. Notice for instance that the type

$$\emptyset \to \infty \to \emptyset \to \infty$$

is obtained using empty intersections. The well-foundedness relation is defined inductively by the rules

$$\infty :: o \qquad \frac{\delta :: \kappa' \qquad \delta_i :: \kappa \ \text{ for every } i \in I}{\bigwedge_{i \in I} \ \delta_i \to \delta \ :: \ \kappa \to \kappa'}$$

The typing rules of the associated system are presented in Figure 8.1. The contexts, managed additively, are sequences of variables together with a set of intersection types and the unique simple type these intersection types refine. Contexts are denoted using the letter Δ. Due to the additive management of contexts, weakening is admissible: if $\Delta \subseteq \Delta'$ and $\Delta \ \vdash \ t \ : \ \delta \ :: \ \kappa$, then $\Delta' \ \vdash \ t \ : \ \delta \ :: \ \kappa$ holds as well. Three rules deserve to be commented:

- In the Const rule, the typical intersection type for a non-terminal $a \in \Sigma$ of arity $n \geq 1$ is
$$a \ : \ \alpha_1 \to \cdots \to \alpha_n \to \infty$$
where α_i is either $\emptyset$ or ∞, and at least one α_i is ∞. Informally, this means that the computation of the direction i is to take an infinite amount of rewriting steps. Note that a symbol of arity 0 can not have the type ∞: this corresponds to the fact that its head reduction will not take any further step.

$$\text{Axiom} \quad \frac{\delta \ \in \ \{\, \delta_i \mid i \in I \,\}}{\Delta,\, x \,:\, \bigwedge_{i\in I} \ \delta_i \ ::\ \kappa \ \vdash \ x \,:\, \delta \ ::\ \kappa} \quad (x \in \mathcal{N} \cup \mathcal{V})$$

$$\text{Const} \quad \frac{\infty \ \in \ \bigcup \ \{\, \delta_{i,j} \mid i \in \{1, \ldots, n\}, \, j \in S_i \,\}}{\Delta \ \vdash \ a \,:\, \bigwedge_{j\in S_1} \ \delta_{1,j} \ \to \ \cdots \ \to \ \bigwedge_{j\in S_n} \ \delta_{n,j} \ \to \ \infty \ ::\ o^n \to o} \quad a \in \Sigma$$

$$\text{App} \quad \frac{\Delta \ \vdash \ t \,:\, \bigwedge_{i\in I} \ \delta_i \to \delta \ ::\ \kappa \to \kappa' \qquad \Delta \ \vdash \ u \,:\, \delta_i \ ::\ \kappa \ \text{ for every } i \in I}{\Delta \ \vdash \ t \ u \,:\, \delta \ ::\ \kappa'}$$

$$\lambda \quad \frac{\Delta,\, x \,:\, \bigwedge_{i\in I} \ \delta_i \ ::\ \kappa \ \vdash \ t \,:\, \delta \ ::\ \kappa'}{\Delta \ \vdash \ \lambda x.\, t \,:\, \bigwedge_{i\in I} \ \delta_i \to \delta \ ::\ \kappa \to \kappa'}$$

$$\mathit{fix} \quad \frac{\forall F \,:\, \delta \ ::\ \kappa \ \in \ \Delta, \quad \exists t \in \mathcal{R}(F) \quad \text{such that} \quad \Delta \ \vdash \ t \,:\, \delta \ ::\ \kappa(F)}{\vdash \ \mathcal{G} \,:\, \Delta}$$

$$\mathit{ini} \quad \frac{\vdash \ \mathcal{G} \,:\, \Delta \qquad \Delta \ \vdash \ t \,:\, \delta \ ::\ \kappa}{\vdash \ (\mathcal{G}, t) \,:\, \delta}$$

Figure 8.1: An intersection type system for detecting infinite rewriting sequences.

- In the *fix* rule, we check that the context Δ only provides *sound* typings for the non-terminals $F \in \mathcal{N}$, in the sense that every intersection type for F must be admitted by some term F rewrites to. This is a non-deterministic adaptation of the stability of types ensured by the *fix* rules of the type systems we considered earlier.

- The *ini* rule is the one we intend to use as conclusion of the typing derivations: it checks that there exists a "sound" typing environment for the non-terminals in which t has the appropriate type.

This intersection type system is related to the rewriting of non-deterministic HORS by the following theorem:

Theorem 23. *Let $\mathcal{G}$ be a* non-deterministic *higher-order recursion scheme of start symbol S. Then*

$$\vdash \quad (\mathcal{G},\, S) \ :\ \infty$$

if and only if there exists an infinite rewriting sequence

$$S \quad \rhd \quad t_1 \quad \rhd \quad t_2 \quad \rhd \quad \cdots$$

The proof of this theorem is divided in companion lemmas. The existence of an infinite rewriting sequence is obtained from Lemma 14 and Lemma 16, while the existence of a typing derivation follows from Lemma 18 applied to $S = t_0$.

8.8.2 Existence of an infinite rewriting sequence

Lemma 14 (Progress)**.** *If* $\vdash\ (\mathcal{G}, t) : \infty$ *then there exists a term* t' *such that* $t \rhd t'$.

Proof. Suppose that $\vdash\ (\mathcal{G}, t) : \infty$ but that there is no such t'. By definition, there exists a context Δ such that $\mathcal{G} : \Delta$ and that $\Delta \vdash t : \infty :: o$. As t is irreducible and of ground type, it is

- either a non-terminal $a \in \Sigma$ of arity 0, which can not have type ∞ as we remarked earlier,
- or $t = F\ \widetilde{s}$ for a non-terminal F such that $\mathcal{R}(F) = \emptyset$. Since $\Delta \vdash F\ \widetilde{s} : \infty :: o$, F must have at least one intersection type in Δ. But then $\mathcal{G} : \Delta$ implies that there exists at least one term to which F rewrites, which is contradictory.

□

The following lemma is required to prove Lemma 16:

Lemma 15 (Substitution lemma)**.** *If*

$$\Delta, x : \bigwedge_{i \in I} \delta_i :: \kappa \vdash t : \delta :: \kappa'$$

and that, for every $i \in I$, $\Delta \vdash s : \delta_i :: \kappa$, *then* $\Delta \vdash t[x \leftarrow s] : \delta :: \kappa'$ *holds.*

Proof. The proof is by induction on t. We omit simple types to ease reading.

- If $t = a \in \Sigma$ is a terminal such that $\Delta, x : \bigwedge_{i \in I} \delta_i \vdash a : \delta$, then $a[x \leftarrow s] = a$ and $\Delta \vdash a : \delta$ follows immediately from the Const rule.
- If $t = F \in \mathcal{N}$ is a non-terminal, then $\Delta, x : \bigwedge_{i \in I} \delta_i \vdash F : \delta$ was obtained from the Axiom rule, with $F : \delta \in \Delta$, so that $\Delta \vdash F : \delta$ holds as well.
- If $t = y \neq x$ is a variable other than x, the situation is the same as in the previous point.
- If $t = x$, then there exists $j \in I$ such that $\delta = \delta_j$ since $\Delta, x : \bigwedge_{i \in I} \delta_i :: \kappa \vdash x : \delta$. It follows immediately that $\Delta \vdash s : \delta_j$ holds.
- If $t = \lambda y.\, u$, we have that $\delta = \bigwedge_{j \in J} \delta'_j \to \delta''$ and that

  $$\Delta, x : \bigwedge_{i \in I} \delta_i, y : \bigwedge_{j \in J} \delta'_j \vdash u : \delta''$$

  We apply the induction hypothesis to u, which provides a derivation of

  $$\Delta, y : \bigwedge_{j \in J} \delta'_j \vdash u[x \leftarrow s] : \delta''$$

  We obtain the desired result by applying the rule λ.

- If $t = t_1\ t_2$ is an application of terms, the typing of t is derived from the rule

$$\frac{\Delta, x : \bigwedge_{i\in I}\ \delta_i \vdash t_1 : \bigwedge_{j\in J}\ \delta'_j \to \delta'' \qquad \Delta, x : \bigwedge_{i\in I}\ \delta_i \vdash t_2 : \delta'_j \quad (\forall j \in J)}{\Delta, x : \bigwedge_{i\in I}\ \delta_i \vdash t_1\ t_2 : \delta''}$$

  We apply the induction hypothesis to the typing derivation of t_1 and to each derivation of t_2, use the Application rule on the result, and obtain in this way a derivation of $\Delta \vdash t[x \leftarrow s] : \delta$.

□

The following lemma shows that if progress can be made in the reduction of a term of type ∞, then a reduction preserving this type exists. The point is, depending on the nature of the head symbol, either to choose an appropriate rewriting if it is a non-terminal, or an appropriate direction to explore if it is a terminal.

Lemma 16. *If* $\vdash (\mathcal{G}, t) : \infty$ *and* $t \rhd t'$*, then there exists a term* t'' *such that* $t \rhd t''$ *and that* $\vdash (\mathcal{G}, t'') : \infty$*.*

Proof. 1. Suppose that $t \rhd t'$ comes from

$$t = F\ s_1 \cdots s_n \rhd u[x_i \leftarrow s_i] = t'$$

where $F \to \lambda x_1 \cdots x_n.\, u$. Since $\vdash (\mathcal{G}, t) : \infty$, we have that $\vdash \mathcal{G} : \Delta$ and that $\Delta \vdash F\ \widetilde{s} : \infty :: o$. The latter necessarily follows from the existence of a type

$$F : \bigwedge_{j\in S_1} \delta_{1,j} \to \cdots \to \bigwedge_{j\in S_n} \delta_{n,j} \to \infty \tag{8.9}$$

and of derivations of the sequents $\Delta \vdash s_i : \delta_{i,j}$ for $i \in \{1, \ldots, n\}$ and $j \in S_i$.

Since $\vdash \mathcal{G} : \Delta$ and that (8.9) holds, the rule *fix* ensures the existence of a term u' such that $F \to \lambda x_1 \cdots x_n.\, u'$ and that

$$\Delta \vdash \lambda x_1 \cdots x_n.\, u' : \bigwedge_{j\in S_1} \delta_{1,j} \to \cdots \to \bigwedge_{j\in S_n} \delta_{n,j} \to \infty$$

This must have been obtained by applying n times the λ rule, so that there exists a proof of

$$\Delta, x_1 : \bigwedge_{j\in S_1} \delta_{1,j}, \cdots x_n : \bigwedge_{j\in S_n} \delta_{n,j} \vdash u' : \infty$$

to which we can apply the substitution lemma for each variable x_i and term s_i, obtaining a proof of

$$\Delta \vdash u'[\widetilde{x} \leftarrow \widetilde{s}] : \infty$$

We set $t'' = u'[\widetilde{x} \leftarrow \widetilde{s}]$.

2. Suppose that $t \rhd t'$ comes from

$$t \;=\; a\; s_1 \cdots s_n \quad \rhd \quad s_i \;=\; t'$$

Then $\vdash\ (\mathcal{G}, t) : \infty$ implies that there exists Δ such that $\Delta \vdash a\; s_1 \cdots s_n$. In the derivation proving this statement, a is introduced with a type

$$a : \alpha_1 \to \cdots \to \alpha_n \to \infty$$

such that there exists j with $\alpha_j = \infty$. We set $t'' = s_j$ and obtain the desired property.

□

8.8.3 Existence of a typing derivation

Consider an infinite rewriting sequence

$$S \quad \rhd \quad t_1 \quad \rhd \quad t_2 \quad \rhd \quad \cdots \tag{8.10}$$

We shall prove that $\vdash\ (\mathcal{G}, t_i) : \infty$ holds for every $i \in \mathbb{N}$. The idea is to extract typing information from the reduction sequence, in the spirit of the completeness proof. We define similarly types for prefixes of the terms t_i, and associated typing contexts.

Type of a prefix. For each prefix s of t_i, that is, each term s such that $t_i = s\ \widetilde{u}$, we define the intersection type $\delta_{s,i}$ by induction on the simple type of s:

- If $s :: o$, then $\delta_{s,i} = \infty$.
- If $s : \kappa \to \kappa'$, then $t_i = s\, u'\, \widetilde{u}$ for some term $u' :: \kappa$, so that $s\, u' :: \kappa'$. As in the completeness proof, we focus on the types u' has when it appears in head position in the rewriting sequence (8.10), and consider the set

$$S \;=\; \{\, j \in \mathbb{N} \mid u' \text{ is a prefix of } t_j \,\}$$

We then define

$$\delta_{s,i} \quad = \quad \bigwedge_{j \in S} \delta_{u',j} \;\to\; \delta_{s\,u',i}$$

Typing environment associated to a prefix. For each prefix s of t_i, we define the typing environment $\Delta_{s,i}$ by induction on the structure of s:

- If $s = a \in \Sigma$, then $\Delta_{s,i} = \emptyset$.
- If $s = F \in \mathcal{N}$, then $\Delta_{s,i} = F : \delta_{s,i}$.
- If $s = s_1\ s_2$, consider the set $S = \{\, j \in \mathbb{N} \mid s_2 \text{ is a prefix of } t_j \,\}$, and set $\Delta_{s,i} = \Delta_{s_1,i} \cup \bigcup_{j \in S} \Delta_{s_2,j}$.

These typing environments allow to type a prefix s of t_i with $\delta_{s,i}$:

Lemma 17. *For every $i \in \mathbb{N}$ and every prefix s of t_i, we have that*

$$\Delta_{s,i} \vdash s : \delta_{s,i}$$

Moreover, this judgment can be derived only from judgments of the form $\Delta_{s,i} \vdash u : \delta_{u,j}$.

Proof. The proof is by induction on the structure of the prefix s of t_i:

- If $s = a \in \Sigma$, then $s \neq t_i$ as it would imply $a :: o$ and t_i could not rewrite to t_{i+1}. So a has simple type $o^n \to o$, with $n \geq 1$, and its refined type $\delta_{a,i}$ is of the shape

  $$a : \alpha_1 \to \cdots \to \alpha_n \to \infty$$

  where each α_k is either $\emptyset$ or ∞. To prove that $\Delta_{a,i} = \emptyset \vdash a : \delta_{a,i}$, we need to use the Const rule, and thus to show that there is some k such that $\alpha_k = \infty$. But since the head of t_i is a, the rewriting step $t_i \rhd t_{i+1}$ will put in head position one of the arguments of a, which is equal to t_{i+1} and is of base type. By definition, $\delta_{t_{i+1},i+1} = \infty$, so that in $\delta_{a,i}$ one of the arguments has type ∞ and we can use Const to conclude that $\Delta_{a,i} \vdash a : \delta_{a,i}$.

- If $s = F \in \mathcal{N}$, then $F : \bigwedge_{\{1\}} \delta_{F,i} \vdash F : \delta_{F,i}$ is obtained immediately by the Axiom rule.

- If $s = s_1\ s_2$, we have by induction hypothesis that

  $$\Delta_{s_1,i} \vdash s_1 : \bigwedge_{j \in S} \delta_{s_2,j} \to \delta_{s,i}$$

  since s_1 is a prefix of t_i as well and, by definition of $\delta_{s_1,i}$, S is the set of indexes j such that s_2 is a prefix of t_j. So we can apply the induction hypothesis to s_2 for every $j \in S$, and obtain derivations of

  $$\Delta_{s_2,j} \vdash s_2 : \delta_{s_2,j}$$

  Note that, by induction hypothesis, all the derivations introduced here are themselves derived from judgments of the form $\Delta_{s,i} \vdash u : \delta_{u,j}$. Since $\Delta_{s,i} = \Delta_{s_1,i} \cup \bigcup_{j \in S} \Delta_{s_2,j}$, every context appearing in one of these derivations is included in $\Delta_{s,i}$, and we can use the admissibility of weakening to obtain proofs in this context. We can then use the Application rule to conclude that $\Delta_{s,i} \vdash s_1\ s_2 : \delta_{s,i}$.

□

It follows that we can extract from the rewriting sequence (8.10) a typing derivation of $(\mathcal{G}, t_i)$ for every $i \in \mathbb{N}$, and notably for $i = 0$ which implies the converse direction of Theorem 23:

Lemma 18. *If there is an infinite rewriting sequence*

$$S = t_0 \rhd t_1 \rhd t_2 \rhd \cdots$$

then for every $i \in \mathbb{N}$ we have that $\vdash (\mathcal{G}, t_i) : \infty$ holds.

Proof. Let $\Delta = \bigcup_{i\in\mathbb{N}} \Delta_{t_i,i}$. By Lemma 17, followed by context weakening, we obtain that, for every $i \in \mathbb{N}$, $\Delta \vdash t_i : \infty$. It remains to prove that $\vdash \mathcal{G} : \Delta$ to conclude, using the *ini* rule. Let us consider $F : \delta :: \kappa(F) \in \Delta$, we need to prove that there exists $u \in \mathcal{R}(F)$ such that $\Delta \vdash u : \delta :: \kappa(F)$. We will obtain this term from the reduction sequence. By definition of Δ and of the types of prefixes $\delta_{t,i}$, there exists $i \in \mathbb{N}$ such that $t_i = F\ s_1 \cdots s_n$, and that

$$\delta = \delta_{F,i} = \bigwedge_{j\in S_1} \delta_{s_1,j} \to \cdots \to \bigwedge_{j\in S_n} \delta_{s_n,j} \to \infty$$

The reduction $t_i \rhd t_{i+1}$ uses a rewrite rule $F \to \lambda x_1 \cdots x_n.\, u'$, such that $t_{i+1} = u'[x_k \leftarrow s_k]$. By Lemma 17, there is a derivation π of $\Delta_{t_{i+1},i+1} \vdash u'[x_k \leftarrow s_k] : \infty$, obtained by using only judgments of the form $\Delta_{t_{i+1},i+1} \vdash s_j : \delta_{s_j,l}$ with $l \in S_j$ to type the arguments s_j. To obtain a derivation of

$$\Delta_{t_{i+1},i+1},\, x_1 : \bigwedge_{j\in S_1} \delta_{s_1,j},\, \cdots,\, x_n : \bigwedge_{j\in S_n} \delta_{s_n,j} \vdash u' : \infty$$

we replace each subproof $\Delta_{t_{i+1},i+1} \vdash s_j : \delta_{s_j,l}$ by an Axiom leaf

$$\Delta_{t_{i+1},i+1},\, x_1 : \bigwedge_{j\in S_1} \delta_{s_1,j},\, \cdots,\, x_n : \bigwedge_{j\in S_n} \delta_{s_n,j} \vdash x_j : \delta_{s_j,l}$$

and weaken all the type environments appearing in π accordingly. The application of n instances of the λ rule gives the term $u = \lambda x_1 \cdots x_n.\, u'$ we were looking for, together with a proof that $\Delta_{t_{i+1},i+1} \vdash u : \delta$. So $\vdash \mathcal{G} : \Delta$ and $\vdash (\mathcal{G}, t_i) : \infty$ holds. □

Given a non-deterministic HORS $\mathcal{G}$, we denote by

$$\Delta_{\mathcal{G}} = \bigcup \{\, \Delta \mid \mathcal{G} : \Delta \,\}$$

the greatest set of typings for its non-terminals which is sound with respect to rewriting. We write $\Delta_{\mathcal{G}}(F)$ for the intersection type of F in $\Delta_{\mathcal{G}}$, and write $\delta \in \Delta_{\mathcal{G}}(F)$ if and only if δ appears in the intersection type of F. Theorem 23 can therefore be restated as:

> A non-deterministic higher-order recursion scheme $\mathcal{G}$ has an infinite reduction sequence if and only if $\infty \in \Delta_{\mathcal{G}}(S)$.

8.8.4 On the well-foundedness of $\preccurlyeq$

We shall now prove Lemma 13 by contradiction. Consider a deterministic HORS $\mathcal{G} = \langle \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$ and an infinite rewriting sequence

$$\langle\, 1,\, S^0 \,\rangle = \langle\, 1,\, t_1 \,\rangle \quad \rhd \quad \langle\, 2,\, t_2 \,\rangle \quad \rhd \quad \langle\, 3,\, t_3 \,\rangle \quad \rhd \quad \cdots$$

We suppose that there is no infinite sequence of integers $(i_n)_{n\in\mathbb{N}}$ such that

$$\cdots \prec i_3 \prec i_2 \prec i_1 \prec i_0 = 0$$

that is, that the relation $\prec$ is well-founded. We now define formally the two non-deterministic higher-order recursion schemes $\mathcal{G}'$ and $\mathcal{G}''$ mentioned on p. 178. For convenience, we set $\mathcal{N} = \{\, F_1, \ldots, F_n \,\}$, with $S = F_1$.

The first one is $\mathcal{G}' = \langle \Sigma, \mathcal{N}', \mathcal{R}', S^0 \rangle$, with

$$\begin{cases} \mathcal{N}' &= \left\{ F^i \mid F \in \mathcal{N},\, i \in \mathbb{N} \right\} \\ \mathcal{R}' &= \left\{ F^i \;\to\; \rho_j(t) \mid \mathcal{R}(F) = t \text{ and } j \prec i \right\} \end{cases}$$

where ρ_j renames every non-terminal G occurring in t to G^j. So $\mathcal{G}'$ is essentially $\mathcal{G}$, in which non-terminals are explicitly labeled by the reduction step which creates them. As a consequence, $\mathcal{G}'$ has an infinite reduction sequence for $\rhd$, which implies by Theorem 23 that $\infty \in \Delta_{\mathcal{G}'}(S^0)$.

We define N as the maximal size $\Delta_{\mathcal{G}}$ could possibly have:

$$N \;=\; \Sigma_{F \in \mathcal{N}} \; |\{\, \delta \mid \delta :: \kappa(F) \,\}|$$

and then define the second non-deterministic HORS as

$$\mathcal{G}'' = \langle \Sigma \uplus \{\mathtt{e} : 0\}, \mathcal{N}'', \mathcal{R}'', S^{(N)} \rangle$$

with

$$\begin{cases} \mathcal{N}'' &= \left\{ F^{(i)} \mid F \in \mathcal{N},\, i \in \{0, \ldots, N\} \right\} \\ \mathcal{R}'' &= \left\{ F^{(i)} \;\to\; \rho_{(i-1)}(t) \middle|\; \mathcal{R}(F) = t \text{ and } i \in \{1, \ldots, N\} \right\} \\ & \quad \uplus \left\{ F^{(0)} \;\to\; \lambda\widetilde{x}.\, \mathtt{e} \right\} \end{cases}$$

where $\rho_{(i-1)}$ renames every non-terminal G occurring in t to $G^{(i-1)}$. This recursion scheme unfolds the rules of $\mathcal{G}$ until depth N, but no more: it does not really use recursion, and can be represented as a simply-typed λ-term. As a consequence of the strong normalization of the simply-typed λ-calculus, all its reduction sequences are finite, and thus

$$\infty \notin \Delta_{\mathcal{G}''}(S^{(N)})$$

We shall now prove that the well-foundedness of $\prec$ prevents $\mathcal{G}'$ from having types $\mathcal{G}''$ can not have: in a sense, since N is at least as big as the set of all possible intersection types for non-terminals, $\mathcal{G}''$ can already explore all the types $\mathcal{G}'$ could have, even if it does not have infinite reduction sequences. To this end, we will use the function $\mathcal{F}$ defined on type environments as

$$\mathcal{F}(\Delta) \;=\; \{\, F : \delta :: \kappa(F) \mid \Delta \;\vdash\; \mathcal{R}(F) : \delta :: \kappa(F) \,\}$$

which extends a type environment with all the intersection types the unfoldings of non-terminals admit. This function is monotonic, as weakening is admissible in the type system we consider here, so that it has a least fixpoint computed by the increasing sequence

$$\emptyset \;\subseteq\; \mathcal{F}(\emptyset) \;\subseteq\; \mathcal{F}^2(\emptyset) \;\subseteq\; \cdots \;\subseteq\; \mathcal{F}^N(\emptyset)$$

As the size of $\mathcal{F}^i(\emptyset)$ is at most N by definition of the latter, we have that $\mathcal{F}^N(\emptyset) = \mathcal{F}^{N+1}(\emptyset)$.

Note that $\mathcal{F}^N(\emptyset)$ computes the set of intersection types non-terminals can have after being rewritten N times. Since this is what the labels of $\mathcal{G}''$ count, we obtain

$$\mathcal{F}^N(\emptyset) \;=\; \left\{ F_i : \delta \mid \delta \in \Delta_{\mathcal{G}''}\left(F_i^{(N)}\right) \right\}$$

We now prove, by well-founded induction, that

$$\Delta_{\mathcal{G}'}\left(F_i^j\right) \;\subseteq\; \mathcal{F}^N(\emptyset)(F_i) \;=\; \Delta_{\mathcal{G}''}\left(F_i^{(N)}\right)$$

for $j \in \mathbb{N}$ and $i \in \{1, \ldots, n\}$. Let $S = \{\, j' \mid j' \prec j \,\}$ be the set of labels that can be introduced by the rewriting of $\mathcal{G}'$ when unfolding F^j. We have that

$$\Delta_{\mathcal{G}'}(F_i^j) \;=\; \bigcup_{j' \in S} \left\{ \delta \mid \delta :: \kappa(F_i) \text{ and } F_1^{j'} : \Delta_{\mathcal{G}'}(F_1^{j'}), \ldots, F_n^{j'} : \Delta_{\mathcal{G}'}(F_n^{j'}) \;\vdash\; \rho_{j'}(\mathcal{R}(F_i)) : \delta \right\}$$

By dropping the labels of non-terminals, we obtain:

$$\Delta_{\mathcal{G}'}(F_i^j) \;=\; \bigcup_{j' \in S} \left\{ \delta \mid \delta :: \kappa(F_i) \text{ and } F_1 : \Delta_{\mathcal{G}'}(F_1^{j'}), \ldots, F_n : \Delta_{\mathcal{G}'}(F_n^{j'}) \;\vdash\; \mathcal{R}(F_i) : \delta \right\}$$

Since $j' \prec j$, we apply the induction hypothesis to each $\Delta_{\mathcal{G}'}(F_i^{j'})$, and obtain

$$\Delta_{\mathcal{G}'}(F_i^j) \;\subseteq\; \bigcup_{j' \in S} \left\{ \delta \mid \delta :: \kappa(F_i) \text{ and } F_1 : \mathcal{F}^N(\emptyset)(F_1), \ldots, F_n : \mathcal{F}^N(\emptyset)(F_n) \;\vdash\; \mathcal{R}(F_i) : \delta \right\}$$

that is:

$$\Delta_{\mathcal{G}'}(F_i^j) \;\subseteq\; \bigcup_{j' \in S} \left\{ \delta \mid \delta :: \kappa(F_i) \text{ and } \mathcal{F}^N(\emptyset) \;\vdash\; \mathcal{R}(F_i) : \delta \right\}$$

Since j' no longer appears in the set we consider, the union can be removed, and we get that

$$\Delta_{\mathcal{G}'}(F_i^j) \;\subseteq\; \left\{ \delta \mid \delta :: \kappa(F_i) \text{ and } \mathcal{F}^N(\emptyset) \;\vdash\; \mathcal{R}(F_i) : \delta \right\}$$

This set coincides with $\mathcal{F}^{N+1}(\emptyset) = \mathcal{F}^N(\emptyset)$, so that we can conclude by well-founded induction that

$$\Delta_{\mathcal{G}'}(F_i^j) \;\subseteq\; \mathcal{F}^N(\emptyset)(F_i) \;=\; \Delta_{\mathcal{G}''}(F_i^{(N)})$$

In particular, since $\infty \notin \Delta_{\mathcal{G}''}(S^{(N)})$, we obtain that $\infty \notin \Delta_{\mathcal{G}'}(S^0)$, which contradicts the existence of an infinite reduction sequence for $\mathcal{G}'$ and thus for $\mathcal{G}$. This implies that $\prec$ is not well-founded: there exists an infinite sequence of integers $(i_n)_{n \in \mathbb{N}}$ such that

$$\cdots \prec i_3 \prec i_2 \prec i_1 \prec i_0 = 0$$

This proves Lemma 13, and thus Lemma 11.

Part III

Colored models of linear logic

Chapter 9

An infinitary model of linear logic

In Chapter 5, we investigated the connection between the non-idempotent intersection type system $\mathcal{H}(\mathcal{A})$ accounting for the alternating behavior of a tree automaton $\mathcal{A}$ and a relational semantics of linear logic with an appropriate interpretation of constants, parameterized by this automaton $\mathcal{A}$. This led to Theorem 14, stating that the semantics of a λ-term of ground type t contains the initial state of $\mathcal{A}$ if and only if its normal form $\langle\, t \,\rangle$ is accepted by $\mathcal{A}$.

In this chapter, we start by explaining how this dependency in $\mathcal{A}$ of the type system can be removed, by considering a Church encoding of the term t producing the tree $\langle\, t \,\rangle$ of interest. We then observe that linear logic allows to type such Church encodings, and that the duals of these types are precisely the ones of tree automata over the same signature. Interestingly enough, typing these Church encodings with formulas of linear logic allows to distinguish between non-deterministic tree automata, and alternating tree automata. We explain how this duality between a λ-term t obtained by Church encoding and a tree automaton $\mathcal{A}$ defined over the same signature allows to compute their interaction in the relational semantics. The result of this interaction is the set of states q of the automaton $\mathcal{A}$ from which $\mathcal{A}$ accepts the tree $\langle\, t \,\rangle$ represented by t.

Our purpose is then to *extend* the relational semantics, so as to capture the whole higher-order model-checking problem. As we remarked in §6.5, the addition of a fixpoint operator powerful enough to generate infinite trees leads, in a system of non-idempotent intersection types – or, equivalently, in the relational semantics – to the apparition of *countable* multiplicities. We therefore introduce an infinitary exponential modality $\frac{1}{?}$, and the corresponding infinitary relational semantics.

The next step is to account for the *coloring* that is induced by alternating *parity* automata. We take advantage of the investigations of Chapter 6, in which we proved that higher-order model-checking can be captured using the type system $\Upsilon_{fix}(\mathcal{G}, \mathcal{A})$. Since the coloring operation $\boxdot$ of the type system $\Upsilon_{fix}(\mathcal{G}, \mathcal{A})$ is *modal*, it can be reflected in the infinitary relational semantics by defining a comonad $\Box$. This comonad enriches the denotations of the relational model with color annotations. Moreover, the comonad $\Box$ distributes over the exponential comonad $\frac{1}{?}$, and this distributivity law between $\Box$ and $\frac{1}{?}$ allows us to define an infinitary and colored exponential modality $\lightning$ as the composite $\lightning \;=\; \frac{1}{?} \circ \Box$ of the two modalities. In the Kleisli category of this exponential

modality $\not!$, the simple type $o \to o$ is interpreted as

$$[\![o \to o]\!] \quad = \quad [\![\not! \,\square\, o \multimap o]\!] \quad = \quad \mathcal{M}_{count}(Col \times Q) \times Q$$

A typical element of this denotation is

$$([(0, q_0), (1, q_1)],\ q_0)$$

which corresponds to the colored intersection type

$$(\boxdot_0\ q_0 \ \wedge\ \boxdot_1\ q_1) \to q_0$$

of the colored intersection type system $\gamma(\mathcal{A})$, extending in this way the correspondence between the type system $\mathcal{H}(\mathcal{A})$ and the traditional, uncolored relational semantics of linear logic explored in Chapter 5.

We also extend the various relational semantics we consider with fixpoint operators satisfying appropriate equations: the ones defining Conway operators. We obtain in this way a colored, infinitary relational semantics of the λY-calculus, in which the fixpoint operator $\mathbf{Y}$ is defined as a semantic counterpart to the *fix* rule of the type system $\gamma_{fix}(\mathcal{G}, \mathcal{A})$. We conjecture that these semantics allow to capture fully the higher-order model-checking problem: we believe that the interaction of the semantics of a λ-term with recursion t producing a possibly infinite tree $\langle\, t \,\rangle$ and of these of a dual APT $\mathcal{A}$ computes the set of states from which $\mathcal{A}$ accepts $\langle\, t \,\rangle$.

We conclude the chapter by briefly introducing *colored tensorial logic* in §9.11, a logic which paves the way for an interesting connection between the infinitary, colored relational semantics and an infinitary, colored *game* semantics which would compute interactively the denotations of the relational model.

9.1 Linear logic and the duality between trees and alternating tree automata

Thanks to the seminal works by Girard and Reynolds on polymorphism and parametricity in the 1970s, it has been recognized that every finite tree t on a given signature Σ can be seen alternatively as a simply-typed λ-term of an appropriate type depending on Σ. This correspondence between trees and λ-terms is even bijective if one considers λ-terms up to $\beta\eta$-equivalence, see for instance Girard [GTL89]. Typically, a finite tree t on the signature

$$\Sigma \quad = \quad \{\, a : 2\,,\, b : 1\,,\, c : 0 \,\} \tag{9.1}$$

is the same thing under this Church encoding as a simply-typed λ-term t of type

$$(o \to o \to o) \to (o \to o) \to o \to o \tag{9.2}$$

modulo $\beta\eta$-equivalence. The idea underlying the correspondence is that every constructor a, b, c of the signature Σ should be treated as a variable

$$a \ :\ o \to o \to o \qquad b \ :\ o \to o \qquad c \ :\ o \tag{9.3}$$

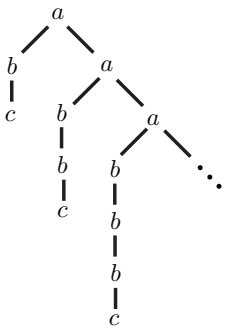
where the number of inputs o in the type $o \to \cdots \to o \to o$ of the variable a, b, c indicates the arity of the combinator. An equally well-known fact is that

this translation extends to infinite trees generated by higher-order recursion schemes on the signature Σ if one extends the simply-typed λ-calculus with a fixpoint operator Y. For example, the higher-order recursion scheme $\mathcal{G}$ on the signature Σ

$$\mathcal{G} \;=\; \left\{ \begin{array}{lcl} S & \mapsto & F\ a\ b\ c \\ F\ x\ y\ z & \mapsto & x\ (y\ z)\ (F\ x\ y\ (y\ z)) \end{array} \right. \tag{9.4}$$

constructs the infinite tree

$$\langle\, \mathcal{G} \,\rangle \quad = \tag{9.5}$$



and can be formulated as a λ-term of the same type (9.3) as previously but defined in the simply-typed λ-calculus extended with the fixpoint operator Y:

$$\lambda abc.\ ((Y\ (\lambda F.\,(\lambda xyz.\,x\,(y\,z)\ (F\,x\,y\,(y\,z)))))\ a\,b\,c) \tag{9.6}$$

A natural temptation is to study the correspondence between higher-order recursion schemes (9.4) and simply-typed λ-terms with fixpoints (9.6) from the resource-aware point of view of linear logic. Recall that the intuitionistic type (9.3) is traditionally translated in linear logic as the formula

$$A \quad = \quad !(\,!o \multimap !o \multimap o\,) \multimap !(\,!o \multimap o\,) \multimap !o \multimap o.$$

As expected, the higher-order recursion scheme $\mathcal{G}$ in (9.4) can be translated as a proof t_A of this formula A in linear logic extended with a fixpoint operator Y. An amusing and slightly puzzling observation is that the scheme $\mathcal{G}$ can be alternatively translated as a proof t_B of the formula B below:

$$B \quad = \quad !(\,o \multimap o \multimap o\,) \multimap !(\,o \multimap o\,) \multimap !o \multimap o$$

with the same underlying simply-typed λ-term with fixpoint operator Y. The difference between the terms t_A and t_B is not syntactic, but type-theoretic: in the case of the term t_A, the type A indicates that each tree-constructor a, b and c of the signature Σ is allowed to call its hypothesis as many times as desired:

$$a \;:\; !o \multimap !o \multimap o \qquad b \;:\; !o \multimap o \qquad c \;:\; o$$

whereas in the case of the term t_B, the type B indicates that each variable a, b, c calls each of its hypothesis exactly once:

$$a \;:\; o \multimap o \multimap o \qquad b \;:\; o \multimap o \qquad c \;:\; o$$

As a matter of fact, it appears that the proof t_B is the image of the proof t_A along a canonical coercion of linear logic

$$\vdash \iota : A \multimap B.$$

The status of this program transformation ι is difficult to understand unless one recalls that linear logic is based on the existence of a perfect duality between the programs of a given type A and their environments or counter-programs which are typed by the linear negation $A^\perp$ of the original type A. Accordingly, since the two terms t_A and $t_B = \iota \circ t_A$ are syntactically equal, their difference should lie in the class of counter-programs of type $A^\perp$ or $B^\perp$ which are allowed to interact with them. This idea takes its full flavor in the context of model-checking, when one realizes that every tree automaton $\mathcal{A}$ on the signature Σ may be seen as a counter-program whose purpose is indeed to interact with t_A or t_B in order to check whether a specific property of interest is satisfied by the infinite tree $\langle\, \mathcal{G} \,\rangle$ generated by the recursion scheme $\mathcal{G}$. This leads to the tentative duality principle:

$$\begin{array}{c}\text{higher-order}\\ \text{recursion schemes } \mathcal{G}\end{array} \qquad \leftrightsquigarrow \qquad \text{tree automata } \mathcal{A}$$

where a tree automaton $\mathcal{A}$ on the signature Σ is thus seen as a counter-program of type $A^\perp$ or $B^\perp$ interacting with the higher-order recursion scheme $\mathcal{G}$ seen as a program of type A or B. An apparent obstruction to this duality principle is that, in contrast to what happens with recursion schemes $\mathcal{G}$, it is in general impossible to translate a tree automaton $\mathcal{A}$ as a proof of linear logic — in particular because linear logic lacks non-determinism. However, one neat way to resolve this matter and to extend linear logic with non-determinism is to embed the logic in its relational semantics, based on the monoidal category *Rel* of sets and relations, which we recalled in §5.2. The relational semantics of linear logic is indeed entitled to be seen as a non-deterministic extension of linear logic where every nondeterministic tree automaton $\mathcal{A} = \langle \Sigma,\, Q,\, \delta, q_0 \rangle$ may be "implemented" by interpreting the base type o as the set Q of states of the automaton, and by interpreting each variable a, b, c as the following relations

$$a \;:\; Q \multimap Q \multimap Q \qquad b \;:\; Q \multimap Q \qquad c \;:\; Q$$

deduced from the transition function δ of the automaton:

$$\begin{array}{c} a = \{(q_1, q_2, q) \in Q \times Q \times Q \mid (1, q_1) \wedge (2, q_2) \in \delta(q, a)\} \\ b = \{(q_1, q) \in Q \times Q \mid (1, q_1) \in \delta(q, b)\} \\ c = \{q \in Q \mid \delta(q, c) = \mathit{true}\} \end{array}$$

The nondeterministic tree automaton $\mathcal{A}$ is then interpreted as the counter-program $\mathcal{A}_B = !\, a \otimes !\, b \otimes !\, c \otimes d$ of type

$$B^\perp \;=\; !\,(Q \multimap Q \multimap Q) \otimes !\,(Q \multimap Q) \otimes !\, Q \;\otimes\; Q^\perp.$$

obtained by tensoring the three relations a, b, c lifted with by the exponential modality $!$ together with the singleton $d = \{q_0\}$ consisting of the initial state of the automaton, and understood as a counter-program of type $Q^\perp$. Note that by composition with the contraposite $\iota^\perp : B^\perp \multimap A^\perp$ of the coercion ι, one gets a counter-program $\mathcal{A}_A = \iota^\perp \circ \mathcal{A}_B$ of type

$$A^\perp \;=\; !\,(!\, Q \multimap !\, Q \multimap Q) \otimes !\,(!\, Q \multimap Q) \otimes !\, Q \;\otimes\; Q^\perp.$$

Note also that when the type o is interpreted as Q in the relational model, the counter-programs of type $B^\perp$ of the form $!\,a \otimes !\,b \otimes !\,c \otimes d$ with $d = \{q_0\}$ correspond exactly to the non-deterministic tree automata on the signature Σ with set of states Q and initial state q_0. The difference between the two types A and B becomes very clear and meaningful at this stage: shifting to the type $A^\perp$ enables one to extend the class of nondeterministic tree automata to nondeterministic *alternating* tree automata $\mathcal{A}$ with typical transitions of the form

$$\delta(q,a) \quad = \quad (1,q_1) \ \wedge \ (1,q_2) \tag{9.7}$$

meaning that the tree automaton $\mathcal{A}$ meeting the tree $a(t_1,t_2)$ at state q explores the left subtree t_1 *twice* with states q_1 and q_2 and does not explore *at all* the right subtree t_2. Such a transition $\delta(q,a)$ is typically represented in the relational semantics of linear logic by the singleton relation

$$a \ = \ \{ \ (\,[q_1,q_2]\,,\,\emptyset\,,\,q\,) \ \} \ : \ !Q \multimap !Q \multimap Q \tag{9.8}$$

where one uses the set $!\,Q$ of finite multisets of Q to encode the transition (9.7) with the finite multiset $[q_1,q_2]$ consisting of the two states $q_1,q_2 \in Q$ and the empty multiset $\emptyset$ of states. It should be stressed that a tree automaton $\mathcal{A}$ admitting such an "alternating" transition $\delta(q,a)$ *cannot* be encoded as a counter-program of type $B^\perp$ because the transitions of the tree automaton $\mathcal{A}$ are *linear* in that type and thus explore *exactly* once each subtree t_1 and t_2 of the tree $a(t_1,t_2)$. Summarizing the current discussion, we are entitled to consider that each linear type $A^\perp$ and $B^\perp$ reflects a specific class of tree automata on the signature Σ:

$$\begin{array}{lcl} B^\perp & \leftrightarrow & \text{non-deterministic tree automata} \\ A^\perp & \leftrightarrow & \text{non-deterministic } \textit{alternating} \text{ tree automata} \end{array}$$

Accordingly, the purpose of the coercion ι from t_A to t_B is to restrict the power of the class of alternating non-deterministic tree automata allowed to explore the infinite tree $\langle\,\mathcal{G}\,\rangle$ generated by the higher-order recursion scheme $\mathcal{G}$ of signature Σ.

Remark 2. Note that the higher-order recursion scheme $\mathcal{G}$ of (9.4) can be *typed* with the *formula*

$$A \quad = \quad !(\ !o \multimap !o \multimap o\ ) \multimap !(\ !o \multimap o\ ) \multimap !o \multimap o.$$

However, when it comes to its relational interpretation, the exponential modality ! is traditionally interpreted as the *finite* multiset construction, and does not allow as such to build infinite trees. As explained in §9.3, this is the reason why we introduce an infinitary exponential, to reflect in the infinitary relational semantics the syntactic duality between higher-order recursion schemes and tree automata.

9.2 Duality and model-checking in the relational semantics

Before we consider extensions of the relational semantics with an infinitary exponential and a coloring modality, let us remark that the Church encoding

of terms generating trees introduced in the previous section allows a nice generalization of Theorem 14, in which the semantics we consider are no longer dependent of an interpretation of constants, but only of the choice of a set of states Q to interpret the base type o.

The Church encoding of a finite ranked tree over the signature $\Sigma = \{f_i : ar_i \mid i \in I\}$ defines a λ-term t of simple type o with free variables f_i of simple type $o^{ar_i} \to o$, translated as the following formula of linear logic:

$$f_i \quad : \quad \underbrace{!\,o \multimap \cdots \multimap !\,o}_{ar_i} \multimap o \qquad \text{for } i \in I.$$

The λ-term t itself is thus typed by the following sequent of linear logic:

$$\cdots \quad , \quad f_i \quad : \quad !\Big(\underbrace{!\,o \multimap \cdots \multimap !\,o}_{a_i} \multimap o \Big) \quad , \quad \cdots \quad \vdash \quad t \quad : \quad o$$

From this follows that its interpretation $[\![t]\!]$ in the relational semantics of linear logic defines a subset of the following set of "higher-order states"

$$[\![t]\!] \subseteq \left[\!\!\left[\bigotimes_{i\in I} !\left(\underbrace{!\,o \multimap \cdots \multimap !\,o}_{ar_i} \multimap o \right) \multimap o \right]\!\!\right]$$

where the return type o is naturally interpreted as the set of states $[\![o]\!] = Q$ of the alternating tree automaton. As explained in the previous section, the transition function δ of the alternating tree automaton $\mathcal{A}$ is itself interpreted as a subset

$$[\![\delta]\!] \quad \subseteq \quad \left[\!\!\left[\bigwith_{i\in I} \left(\underbrace{!\,o \multimap \cdots \multimap !\,o}_{ar_i} \multimap o \right) \right]\!\!\right]$$

which may be "strengthened" in the categorical sense as a subset

$$[\![\delta^\dagger]\!] \quad \subseteq \quad \left[\!\!\left[\bigotimes_{i\in I} ! \left(\underbrace{!\,o \multimap \cdots \multimap !\,o}_{ar_i} \multimap o \right) \right]\!\!\right]$$

where we turn to our advantage the well-known Seely isomorphism of linear logic:

$$!(A \,\&\, B) \quad \cong \quad !A \otimes !B.$$

We may now compose the dual relations $[\![t]\!]$ and $[\![\delta^\dagger]\!]$, and obtain in this way a subset of Q. Following the intuition of Theorem 14, this is the set of states from which the automaton of transition function δ accepts the tree t, as it will be confirmed by the next theorem. Note also that, since the model is denotational, t does not have to be the Church encoding of a tree, but can more generally be any term normalizing to (the Church encoding of) a tree. We obtain in this way a generalization of Theorem 14 in which the interpretation of the term is no longer dependent on the APT of interest:

Definition 35 (Church encoding)**.** Let t be a λ-term of simple type o, defined over the signature of constants $\Sigma = \{f_1 : ar_1, \ldots, f_n : ar_n\}$. We define its Church encoding t^Σ as the simply-typed λ-term

$$t^\Sigma \quad = \quad \lambda f_1. \ldots . \lambda f_n . t$$

of simple type

$$o^{ar_1} \to \cdots \to o^{ar_n} \to o$$

and defined without constants.

Theorem 24. *An alternating tree automaton $\mathcal{A} = \langle \Sigma, Q, \delta, q_0 \rangle$ has an accepting run-tree with initial state q_0 over the tree generated by a λ-term t of base type, defined over the signature of constants Σ, if and only if there exists $u \in [\![\delta^\dagger]\!]$ such that $(u, q_0) \in [\![t^\Sigma]\!]$, where t^Σ is the Church encoding of t and where $[\![\delta^\dagger]\!] = \mathcal{M}_{fin}([\![\delta]\!])$ denotes the set of finite multisets of elements of $[\![\delta]\!]$.*

Another equivalent way to state the theorem is that the set of accepting states q_0 for a finite run-tree of the alternating tree automaton $\mathcal{A}$ is equal to the composition of $[\![t^\Sigma]\!]$ and of $[\![\delta^\dagger]\!]$ in the relational semantics.

Proof of Theorem 24. Let us consider an alternating tree automaton $\mathcal{A} = \langle \Sigma, Q, \delta, q_0 \rangle$, and a closed λ-term $t :: o$ of base type, defined over the set of constants $\Sigma = \{f_1 : ar_1, \ldots, f_n : ar_n\}$. Suppose that $\mathcal{A}$ has an accepting run-tree with initial state q_0 over the tree generated by t. By Theorem 13, there is a proof in $\mathcal{H}(\mathcal{A})$ of the sequent

$$\emptyset \vdash t : q_0 :: o$$

In this proof, we replace every rule δ introducing the *tree constructor* f_i by a rule Axiom introducing the *free variable* f_i. This introduces in the contexts of the proof occurrences of the free variables $f_1, \ldots, f_n$, so that we obtain a new proof of the sequent

$$f_1 : \tau_1, \ldots, f_n : \tau_n \vdash t : q_0$$

By appending n Abstraction rules to the conclusion of this proof, we obtain a proof of the sequent

$$\emptyset \vdash t^\sigma : \tau_1 \to \cdots \to \tau_n q_0$$

Using the connection between the relational model and the non-idempotent intersection type system $\mathcal{H}(\mathcal{A})$ explained in §5.3, we get that $((u_1, \ldots, u_n), q_0) \in [\![t^\sigma]\!]$, where u_i is the element of the relational model corresponding to the intersection type τ_i, and where by construction $(u_1, \ldots, u_n) \in [\![\delta^\dagger]\!]$ – as each type occurring in τ_i was introduced by the rule δ of the type system $\mathcal{H}(\mathcal{A})$.

Let us now prove the converse direction, in a very similar way. Suppose that there exists $u \in [\![\delta^\dagger]\!]$ such that $(u, q_0) \in [\![t^\Sigma]\!]$. Using the connection between the relational model and the non-idempotent intersection type system $\mathcal{H}(\mathcal{A})$ of §5.3 in the converse direction, we obtain a proof of the sequent

$$\emptyset \vdash t^\sigma : \tau_1 \to \cdots \to \tau_n q_0$$

in $\mathcal{H}(\mathcal{A})$, which does not contain the rule δ as t^Σ is defined without constants. In this proof, each type τ_i corresponds to the elements of the multiset u regarding the symbol f_i. We remove the λ-abstractions occurring at the root of the proof, and obtain a proof of the sequent

$$f_1 : \tau_1, \ldots, f_n : \tau_n \vdash t : q_0$$

Now, each Axiom rule introducing a free variable f_i introduces an intersection type whose translation as a relational denotation belongs to the multiset $[\![\delta^\dagger]\!]$. This implies that all the intersection types introduced by these Axiom rules can be introduced by a δ rule as well. So we replace all the Axiom rules introducing a *free variable* f_i by δ rules introducing a *tree constructor* f_i. After this process, there are no longer occurrences of f_i in the contexts of the proof. We therefore obtain a proof of the sequent

$$\emptyset \vdash t : q_0 :: o$$

in the type system $\mathcal{H}(\mathcal{A})$. By Theorem 13, the tree generated by the normalization of t is accepted from q_0 by the alternating tree automaton $\mathcal{A}$.

An extension to the λY-calculus. At this point, it is tempting to extend the result to λY-terms, by considering the inductive fixpoint the relational semantics feature, and which is presented in §9.5. However, all we can obtain in the relational semantics is the following theorem, in which we implicitly assume the productivity of the λY-term of interest:

Theorem 25. *An alternating tree automaton $\mathcal{A} = \langle \Sigma, Q, \delta, q_0 \rangle$ has a* finite *accepting run-tree with initial state q_0 over the possibly infinite tree generated by a λY-term t of base type, defined over the signature of constants Σ, if and only if*

$$q_0 \in [\![t^\Sigma]\!] \circ [\![\delta^\dagger]\!] \subseteq Q$$

Indeed, in order to build *infinite* run-trees, the semantics needs to use *infinitely* some tree constructors, which are treated as free variables in this Church encoding. It therefore appears that the only hurdle towards an extension of this theorem to the alternating tree automata with coinductive (rather than inductive) acceptance condition is the *finiteness* of multiplicities in the traditional relational interpretation of the exponential modality.

9.3 Towards an infinitary model of linear logic

In many respects, denotational semantics started in the late 1960's with Dana Scott's introduction of domains and the fundamental intuition that λ-terms should be interpreted as *continuous* rather than general functions between domains. This seminal insight has been so influential in the history of our discipline that it remains deeply rooted in the foundations of denotational semantics more than forty-five years later. In the case of linear logic, this inclination for continuity means that the interpretation of the exponential modality

$$A \quad \mapsto \quad !A$$

is *finitary* in most denotational semantics of linear logic. This finitary nature of the exponential modality is tightly connected to continuity because this modality regulates the linear decomposition of the intuitionistic implication:

$$A \Rightarrow B \quad = \quad !A \multimap B.$$

Typically, in the qualitative and quantitative coherence space semantics of linear logic, the coherence space $!A$ is either defined as the coherence space $!A$ of *finite* cliques (in the qualitative semantics) or of *finite* multi-cliques (in the quantitative semantics) of the original coherence space A. This finiteness condition on the cliques $\{a_1, \ldots, a_n\}$ or multi-cliques $[a_1, \ldots, a_n]$ of the coherence space $!A$ captures the computational intuition that, in order to reach a given position b of the coherence space B, every proof or program

$$f \quad : \quad !A \multimap B$$

will only explore a *finite* number of copies of the hypothesis A, and reach at the end of the computation a specific position a_i in each copy of the coherence space A. In other words, the finitary nature of the interpretation of $!A$ is just an alternative and very concrete way to express in these traditional models of linear logic the continuity of proofs and programs.

In this chapter, we would like to revisit this well-established semantic tradition and accommodate another equally well-established tradition, coming this time from verification and model-checking. We find especially important to address and to clarify an apparent antagonism between the two traditions. Model-checking is generally interested in infinitary (typically ω-regular) inductive and coinductive behaviors of programs which lie obviously far beyond the scope of Scott continuity. For that reason, we introduce a variant of the relational semantics of linear logic where the exponential modality, noted in this context

$$A \quad \mapsto \quad \lightning A$$

is defined as the set of *finite* or *countable* multisets of the set A. From this follows that a proof or a program

$$A \Rightarrow B \quad = \quad \lightning A \multimap B.$$

is allowed in the resulting infinitary semantics to explore a possibly countable number of copies of his hypothesis A in order to reach a position in B. By relaxing the continuity principle, this mild alteration of the original relational semantics paves the way to a fruitful interaction between linear logic and model-checking. This link between linear logic and model-checking is supported by the somewhat unexpected observation that the binary relation

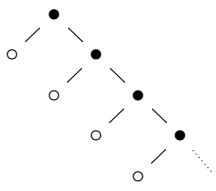
$$Y(f) \quad : \quad !X \quad \longrightarrow \quad A$$

defining the fixpoint $\mathbf{Y}(f)$ associated to a morphism

$$f \quad : \quad !X \otimes !A \quad \longrightarrow \quad A$$

in the familiar (and thus finitary) relational semantics of linear logic is defined by performing a series of explorations of the infinite binary tree

comb =



by an alternating tree automaton $\langle\, \Sigma\,,\, Q\,,\, \delta_f \,\rangle$ on the alphabet $\Sigma = \{\bullet, \circ\}$ defined by the binary relation f. The key idea is to define the set of states of the automaton as $Q = A \uplus X$ and to associate a transition

$$\delta_f(\bullet, a) \quad = \quad (\, x_1 \wedge \cdots \wedge x_k \,,\, a_1 \wedge \cdots \wedge a_n \,)$$

of the automaton to any element $(([x_1, \ldots, x_k], [a_1, \ldots, a_n]), a)$ of the binary relation f, where the x_i's are elements of X and the a_i's are elements of A ; and to let the symbol $\circ$ accept any state $x \in X$. Then, it appears that the traditional definition of the fixpoint operator $\mathbf{Y}(f)$ as a binary relation $!X \to A$ may be derived from the construction of run-trees of the tree-automaton $\langle\, \Sigma\,,\, Q\,,\, \delta_f \,\rangle$ on the infinitary tree **comb**. More precisely, the binary relation $Y(f)$ contains all the elements $([x_1, \ldots, x_k], a)$ such that there exists a finite run-tree (called *witness*) of the tree automaton $\langle\, \Sigma\,,\, Q\,,\, \delta_f \,\rangle$ accepting the state a with the multiset of states $[x_1, \ldots, x_k]$ collected at the leaves $\circ$. As far as we know, this automata-theoretic account of the traditional construction of the fixpoint operator $\mathbf{Y}(f)$ in the relational semantics of linear logic appeared for the first time in [GM15c], and we carefully develop it in §9.5.

Once this healthy bridge between linear logic and tree automata theory identified, it makes sense to study variations of the relational semantics inspired by verification. This is precisely the path we follow here by replacing the finitary interpretation $!A$ of the exponential modality by the finite-or-countable one $\lightning A$. This alteration enables us to define an inductive as well as a coinductive fixpoint operator $\mathbf{Y}$ in the resulting infinitary relational semantics. The two fixpoint operators only differ in the acceptance condition applied to the run-tree *witness*. We carry on in this direction, and introduce a *colored* variant of the relational semantics, designed in such a way that the tree automaton $\langle\, \Sigma\,,\, Q\,,\, \delta_f \,\rangle$ associated to a morphism $f : !X \otimes !A \to A$ defines a parity tree automaton. This leads us to the definition of an inductive-coinductive fixpoint operator $\mathbf{Y}$ tightly connected to our investigations on higher-order model-checking.

9.4 Fixpoint operators in models of linear logic

We want to extend linear logic with a fixpoint rule with parameters:

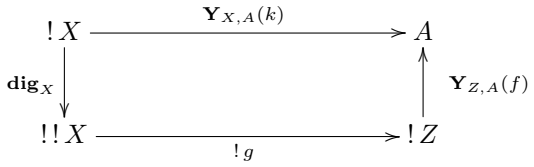
$$\frac{!\,X \otimes !\,A \vdash A}{!\,X \vdash A} \quad \textit{fix}$$

In order to interpret it in a Seely category, we need a parametrized fixpoint operator, defined below as a family of functions

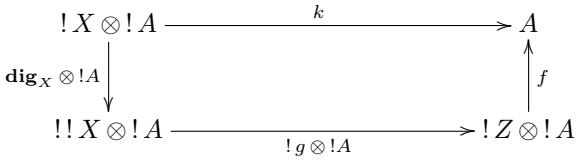
$$\mathbf{Y}_{X,A} \;:\; \mathscr{C}(!\,X \otimes !A\,,\, A\,) \quad \longrightarrow \quad \mathscr{C}(!\,X, A)$$

parametrized by X, A and satisfying two elementary conditions, mentioned for instance by Simpson and Plotkin in [SP00] in the framework of cartesian categories, and which we adapt here to Seely categories.

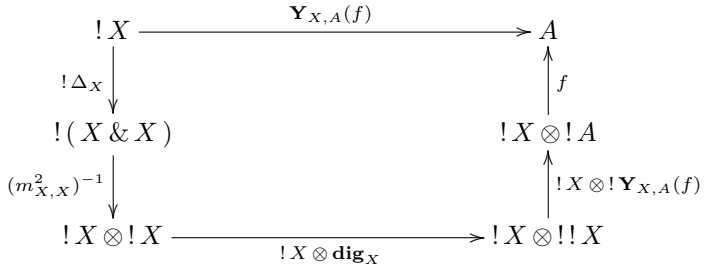
- **Naturality:** for any $g : !\,X \multimap Z$ and $f : !\,Z \otimes !\,A \multimap A$, the diagram:



commutes, where the morphism $k : !X \otimes !A \multimap A$ in the upper part of the diagram is defined as the composite



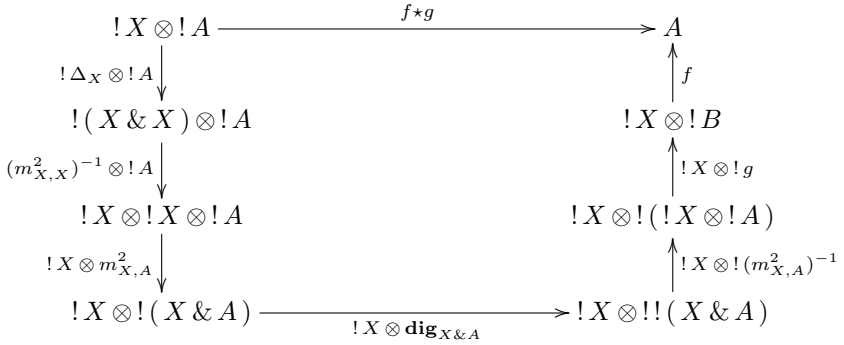
- **Parametrized fixpoint property:** for any $f : !X \otimes !A \multimap A$, the following diagram commutes:



These two equations are fundamental but they do not reflect all the equational properties of the fixpoint operator in domain theory. For that reason, Bloom and Esik introduced the notion of *Conway theory* in their seminal work on iteration theories [BÉ93, BÉ96]. This notion was then rediscovered and adapted to cartesian categories by Hasegawa [Has99], by Hyland and by Simpson and Plotkin [SP00]. Hasegawa and Hyland moreover independently established a nice correspondence between the resulting notion of *Conway fixpoint operator* and the notion of trace operator introduced a few years earlier by Joyal, Street and Verity [JSV96]. Here, we adapt in the most straightforward way this notion of Conway fixpoint operator to the specific setting of Seely categories. Before going any further, we find useful to introduce the following notation: for every pair of morphisms

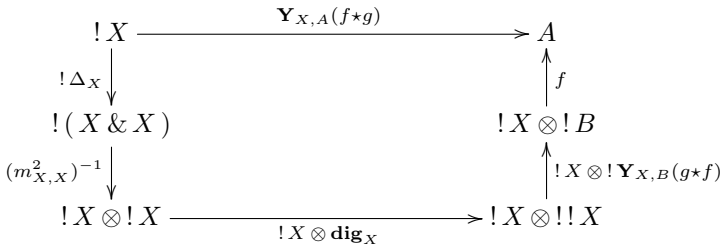
$$f : !X \otimes !B \multimap A \quad \text{and} \quad g : !X \otimes !A \multimap B$$

we write $f \star g : !X \otimes !A \multimap A$ for the composite:



A Conway operator is then defined as a parametrized fixpoint operator satisfying the two additional properties below:

- **Parametrized dinaturality:** for any $f : !X \otimes !B \multimap A$ and $g : !X \otimes !A \multimap B$, the following diagram commutes:



- **Diagonal property:** for every morphism $f : !X \otimes !A \otimes !A \multimap A$,

$$\mathbf{Y}_{X,A}\,(\,(m^2_{X,A})^{-1} \circ \mathbf{Y}_{X\&A,A}\,(\,f \circ (\,(m^2_{X,A})^{-1} \otimes !A\,)\,)\,) \qquad (9.9)$$

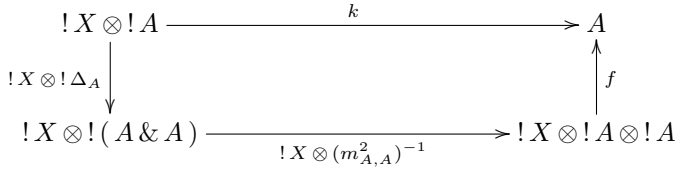
belongs to $!X \multimap A$, since

$$!(\,X \,\&\, A\,) \otimes !A \xrightarrow{(m^2_{X,A})^{-1} \otimes !A} !X \otimes !A \otimes !A \xrightarrow{\quad f \quad} A$$

is sent by $\mathbf{Y}_{X\&A,A}$ to a morphism of $!(\,X \,\&\, A\,) \multimap A$, so that

$$(m^2_{X,A})^{-1} \circ \mathbf{Y}_{X\&A,A}\,(\,f \circ (\,(m^2_{X,A})^{-1} \otimes !A\,)\,) : !X \otimes !A \multimap A$$

to which the fixpoint operator $\mathbf{Y}_{X,A}$ can be applied, giving the morphism (9.9) of $!X \multimap A$. This morphism is required to coincide with the morphism $\mathbf{Y}_{X,A}(k)$, where the morphism $k : !X \otimes !A \to A$ is defined as the composite



Just as expected, we recover in that way the familiar notion of Conway fixpoint operator as formulated in any cartesian category by Hasegawa, Hyland, Simpson and Plotkin:

Proposition 18. *A Conway operator in a Seely category is the same thing as a Conway operator (in the sense of [Has99, SP00]) in the cartesian closed category associated to the exponential modality by the Kleisli construction.*

Proof. In [SP00], Simpson and Plotkin give four equations defining Conway operators in cartesian closed categories. As proved by Seely in [See89], in a Seely category $\mathcal{C}$ (see Definition 32), the coKleisli construction applied to the exponential comonad ! gives birth to a cartesian closed category $\mathcal{C}_!$. This cartesian closed category $\mathcal{C}_!$ is defined in terms of the objects and morphisms of the original Seely category $\mathcal{C}$. The proposition can then be proved simply by writing the equations found in [SP00] using the morphisms of $\mathcal{C}$ used to define $\mathcal{C}_!$. □

9.5 A fixpoint operator in the relational semantics

The relational model of linear logic can be equipped with a natural parameterized fixpoint operator $\mathbf{Y}$ which transports any binary relation

$$f \quad : \quad !X \otimes !A \quad \multimap \quad A$$

to the binary relation

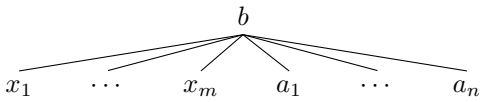
$$\mathbf{Y}_{X,A}(f) \quad : \quad !X \multimap A$$

defined in the following way:

$$\mathbf{Y}_{X,A}(f) \quad = \quad \{\,(w,a)\,|\quad \exists \mathit{witness} \in \mathbf{run\text{-}tree}(f,a) \text{ with } w = \mathbf{leaves}(\mathit{witness}) \text{ and } \mathit{witness} \text{ is accepting}\,\} \tag{9.10}$$

where $\mathbf{run\text{-}tree}(f,a)$ is the set of "run-trees" defined as trees with nodes labeled by elements of the set $X \uplus A$ and such that:

- the root of the tree is labeled by a,
- the inner nodes are labeled by elements of the set A,
- the leaves are labeled by elements of the set $X \uplus A$,
- and for every node labeled by an element $b \in A$:
  - if b is an inner node, and letting $a_1, \ldots, a_n$ denote the labels of its children belonging to A and $x_1, \ldots, x_m$ the labels belonging to X:



    then $(([x_1, \ldots, x_m], [a_1, \ldots, a_n]), b) \in f$
  - if b is a leaf, then $(([\,],[\,]), b) \in f$.

and where **leaves**(*witness*) is the multiset obtained by enumerating the labels of the leaves of the run-tree *witness*. Recall that multisets account for the number of occurrences of an element, so that **leaves**(*witness*) has the same number of elements as there are leaves in the run-tree *witness*. Moreover, **leaves**(*witness*) is independent of the enumeration of the leaves, since multisets can be understood as abelian versions of lists. Finally, we declare that a run-tree is *accepting* when it is a finite tree.

Proposition 19. *The fixpoint operator* $\mathbf{Y}$ *is a Conway operator on Rel.*

Proof. We need to check that $\mathbf{Y}$ satisfies the four diagrams defining Conway operators of §9.4.

Naturality. Let $g : !X \multimap Z$ and $f : !Z \otimes !A \multimap A$ be two morphisms in Rel. We need to check that the diagram:

$$\begin{array}{ccc} !X & \xrightarrow{\mathbf{Y}_{X,A}(k)} & A \\ {\scriptstyle \mathbf{dig}_X}\downarrow & & \uparrow{\scriptstyle \mathbf{Y}_{Z,A}(f)} \\ !!X & \xrightarrow[!g]{} & !Z \end{array}$$

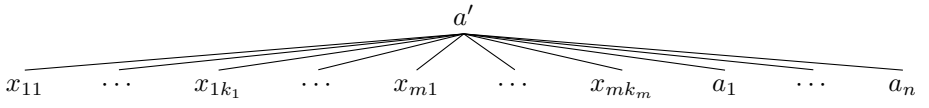
commutes, where the morphism $k : !X \otimes !A \multimap A$ is defined as the composite

$$\begin{array}{ccc} !X \otimes !A & \xrightarrow{k} & A \\ {\scriptstyle \mathbf{dig}_X \otimes !A}\downarrow & & \uparrow{\scriptstyle f} \\ !!X \otimes !A & \xrightarrow[!g \otimes !A]{} & !Z \otimes !A \end{array}$$

We start by expliciting the morphism k. An element $((u, [a_1, \ldots, a_n]), b)$, where $u \in !X$, is in k if and only if

- $u = [x_{11}, \ldots, x_{1k_1}, \ldots, x_{i1}, \ldots, x_{ik_i}, \ldots, x_{m1}, \ldots, x_{mk_m}]$,
- for every $i \in \{1, \ldots, m\}$, $([x_{i1}, \ldots, x_{ik_i}], z_i) \in g$,
- $(([z_1, \ldots, z_m], [a_1, \ldots, a_n]), b) \in f$.

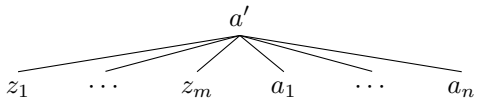
Let us now consider $(u, b) \in \mathbf{Y}_{X,A}(k)$, and prove that it is in $\mathbf{dig}_X \circ !g \circ \mathbf{Y}_{Z,A}(f)$. By definition of $\mathbf{Y}_{X,A}(k)$, u is the multiset of leaves of a finite run-tree $\mathcal{T}_k$ obtained by pasting together trees of the shape:



(9.11)

where $(([x_{11}, \ldots, x_{1k_1}, \ldots, x_{m1}, \ldots, x_{mk_m}], [a_1, \ldots, a_n]), a') \in k$. By definition of k, this element comes from $(([z_1, \ldots, z_m], [a_1, \ldots, a_n]), a') \in f$, and $([x_{i1}, \ldots, x_{ik_i}], z_i) \in g$ for every $i \in \{1, \ldots, m\}$. We say that z_i is obtained

from the multiset $u_{z_i} = [x_{i1}, \ldots, x_{ik_i}]$. Now we replace each finite tree (9.11) in $\mathcal{T}_k$ by the tree



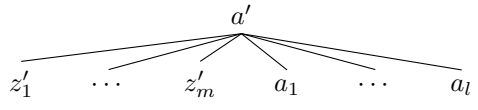
(9.12)

associated to $(([z_1, \ldots, z_m], [a_1, \ldots, a_n]), a') \in f$. Since this only affects the parameters, we can compose the trees (9.12) in the same manner as the trees (9.11) were composed to produce $\mathcal{T}_k$. We obtain in this way a run-tree $\mathcal{T}_f$ over f. We call $v \in !\, Z$ its multiset of leaves. It follows that $(v, b) \in \mathbf{Y}_{Z,A}(f)$. By construction, each element $z \in v$ comes from a multiset $u_z \in !\, X$, and $u = \sum_{z \in v} u_z$. Since $(u_z, z) \in g$, it follows that $([u_z \mid z \in v], v) \in !\, g$. By composition, $(u, b) \in \mathbf{dig}_X \circ !\, g \circ \mathbf{Y}_{Z,A}(f)$.

We prove the converse direction in the same way. Let us consider $(u, b) \in \mathbf{dig}_X \circ !\, g \circ \mathbf{Y}_{Z,A}(f)$, and prove that it is in $\mathbf{Y}_{X,A}(k)$. By construction, $u = u_1 + \cdots + u_n$ and there exists $z_1, \ldots, z_n \in Z$ such that

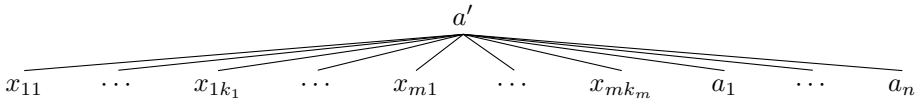
- $\forall i \in \{\, 1, \ldots, n \,\},\ (u_i, z_i) \in g$,
- $([z_1, \ldots, z_n], b) \in \mathbf{Y}_{Z,A}(f)$.

The second point implies that there is a run-tree $\mathcal{T}_f$ over f of root b, built from trees of the shape



(9.13)

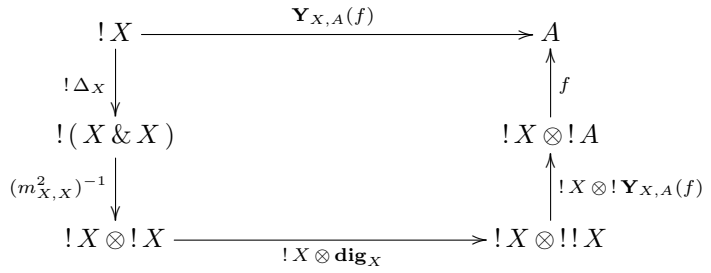
with $[z'_1, \ldots, z'_m] \subseteq [z_1, \ldots, z_n]$. This implies that $(([z'_1, \ldots, z'_m], [a_1, \ldots, a_l]), a') \in f$. We denote by u'_i the multiset allowing to obtain z'_i (via $(u'_i, z'_i) \in g$). By composition of this element of f with $(!\, g \circ \mathbf{dig}_X) \otimes !\, A$ we have that $((u'_1 + \cdots + u'_n, [a_1, \ldots, a_l]), a') \in k$. This allows us to built a run-tree $\mathcal{T}_k$ for k, obtained by replacing each tree (9.13) by the tree



(9.14)

where $u'_i = [x_{i1}, \ldots, x_{ik_i}]$. By construction, the multiset of leaves of the run-tree $\mathcal{T}_k$ is u, so that $(u, b) \in \mathbf{Y}_{X,A}(k)$.

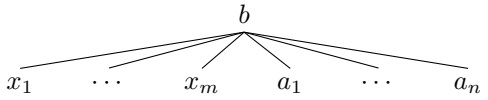
Parameterized fixpoint property. Let us consider $f : !\, X \otimes !\, A \multimap A$, and prove that the following diagram commutes:



We call $k : !X \to A$ the composite of the morphisms of the lower part of the diagram. Suppose that $(u, b) \in k$, we prove that $(u, b) \in \mathbf{Y}_{X,A}(f)$. By definition of k, we have that $u = v_0 + v_1 + \cdots + v_n$, with:

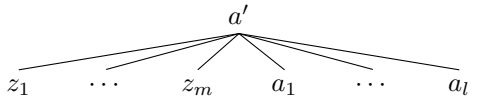
- $((v_0, [a_1, \ldots, a_n]), b) \in f$,
- for every $i \in \{1, \ldots, n\}$, $(v_i, a_i) \in \mathbf{Y}_{X,A}(f)$.

We can therefore obtain, for every $i \in \{1, \ldots, n\}$, a run-tree of root a_i and whose leaves form a multiset v_i. We plug them to



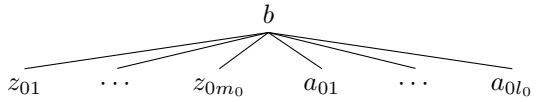
where we denote $v_0 = [x_1, \ldots, x_m]$. We obtain in this way a run-tree over f, with root b and multiset of leaves $u = v_0 + v_1 + \cdots + v_n$. It follows that $(u, b) \in \mathbf{Y}_{X,A}(f)$.

The converse direction is proven similarly. Let $(u, b) \in \mathbf{Y}_{X,A}(f)$. It follows that there is a run-tree of root b, of multiset of leaves u, and built from trees of the shape



(9.15)

where $(([z_1, \ldots, z_m], [a_1, \ldots, a_l]), a') \in f$. The root of the tree is of the following shape:



(9.16)

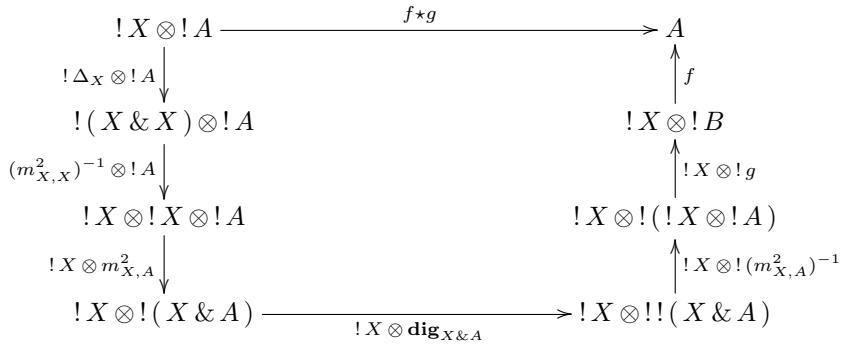
This implies that $(([z_{01}, \ldots, z_{0m_0}], [a_{01}, \ldots, a_{0l_0}]), b) \in f$. Each a_{0i} is itself the root of a tree $\mathcal{T}_i$. Note that there may be no such tree, as the element of f considered at the root may be of the form $(([z_1, \ldots, z_m], [\,]), b)$. Each tree $\mathcal{T}_i$ is a run-tree over f, of root a_i and of multiset of leaves u_i, with

$$u \quad = \quad [z_{01}, \ldots, z_{0m_0}] + u_0 + \cdots + u_{l_0}$$

It follows that, for every $i \in \{\, 0, \ldots, l_0 \,\}$, $(u_i, a_i) \in \mathbf{Y}_{X,A}(f)$. By composition with of all these elements with $(([z_{01}, \ldots, z_{0m_0}], [a_{01}, \ldots, a_{0l_0}]), b) \in f$, we get

that $(u, b) \in k$, where k denotes again the composite of the lower part of the diagram expressing the parameterized fixpoint property.

Parametrized dinaturality. Consider $f : !X \otimes !B \multimap A$ and $g : !X \otimes !A \multimap B$. Let us explicit in a first time the composite $f \star g : !X \otimes !A \multimap A$, defined as:



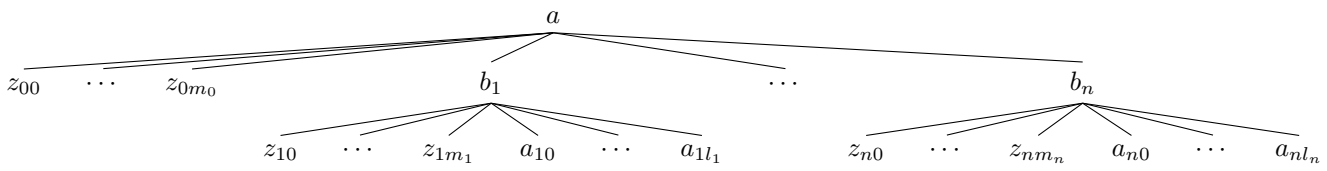
The relation $f \star g$ contains precisely the elements of the shape

$$((u_0 + u_1 + \cdots + u_n,\, v_1 + \cdots + v_n)\,, a)$$

with $u_i \in !X$, $v_j \in !A$, $a \in A$ and such that there exists $b_1, \ldots, b_n \in B$ satisfying:

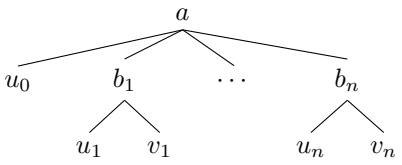
- $\forall i \in \{1, \ldots, n\}\,,\ ((u_i, v_i), b_i) \in g$,
- $(u_0, [b_1, \ldots, b_n]) \in f$.

To ease the remainder of the proof, we represent this composition as the following tree:



(9.17)

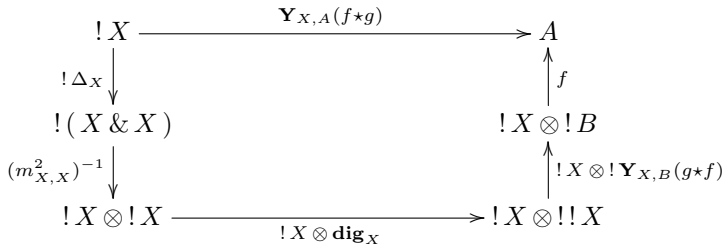
where $u_i = [z_{i0}, \ldots, z_{im_i}]$ and $v_i = [a_{i0}, \ldots, a_{im_i}]$. To ease reading, we also use the more compact representation



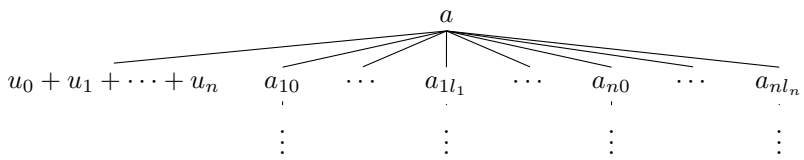
(9.18)

in which we use nodes labeled by multisets to represent multisets of leaves.

We need to check that the following diagram commutes:

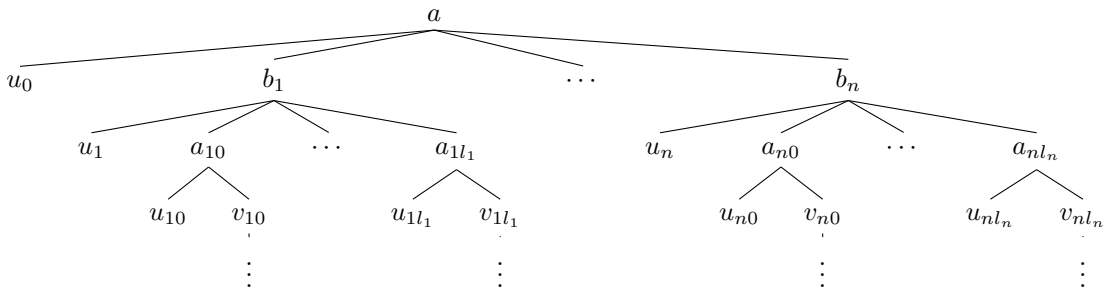


We denote by k the composite of the morphisms on the lower part of the diagram. Let $(u, a) \in \mathbf{Y}_{X,A}(f \star g)$. It follows that there exists a run-tree over $f \star g$ of root a, and of multiset of leaves u, starting as follows:



(9.19)

By definition of $f \star g$, and using the tree representation of compositions given in (9.17) and (9.18), we can expand this run-tree (9.19) as the composition of relations starting as follows:



(9.20)

In 9.20, we see immediately that each tree rooted in b_i ($i \in \{ 1, \ldots, n \}$) is a run-tree over $g \star f$, so that we obtain a family of morphisms $(u'_i, b_i) \in \mathbf{Y}_{X,B}(g \star f)$, where every u'_i is the multiset of leaves of the run-tree over $g \star f$ rooted at b_i. It follows that

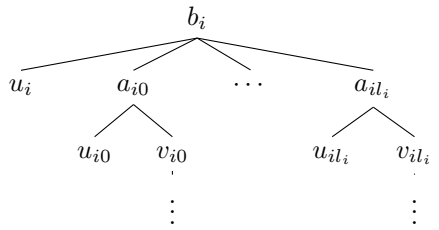
$$u \quad = \quad u_0 + u'_1 + \cdots + u'_n$$

from which we deduce, by composing the appropriate elements, that $(u, a) \in k$.

Conversely, let $(u, a) \in k$. It follows that there exists $b_1, \ldots, b_n \in B$ and a decomposition $u \quad = \quad u_0 + u'_1 + \cdots + u'_n$, such that

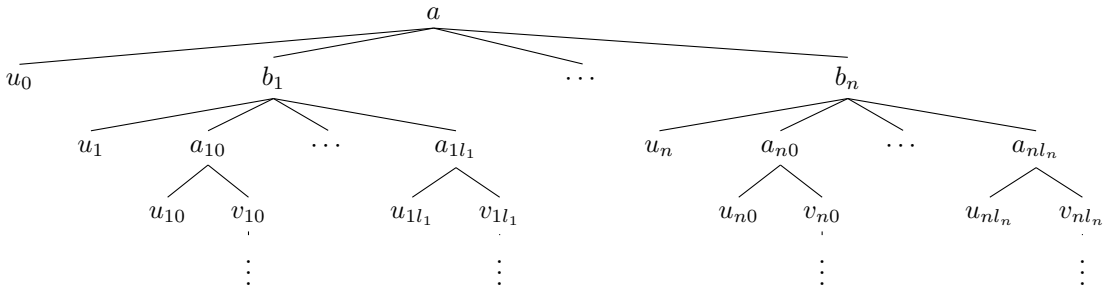
- for every $i \in \{ 1, \ldots, n \}$, $(u'_i, b_i) \in \mathbf{Y}_{X,B}(g \star f)$,
- $(u_0, [b_1, \ldots, b_n]) \in f$.

For every $i \in \{1, \ldots, n\}$, there is a run-tree justifying that $(u'_i, b_i) \in \mathbf{Y}_{X,B}(g \star f)$, and which can be written, by definition of $g \star f$ and using the graphical representation of (9.17) and (9.18), as



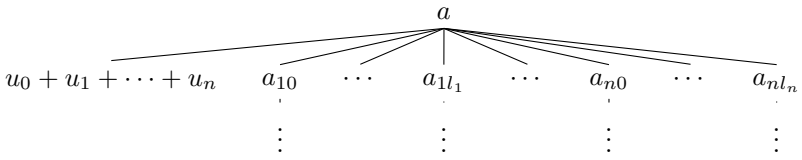
(9.21)

The composition of the elements (u'_i, b_i) with $(u_0, [b_1, \ldots, b_n]) \in f$ is itself represented by the tree



(9.22)

which, as we remarked earlier, represents the run-tree



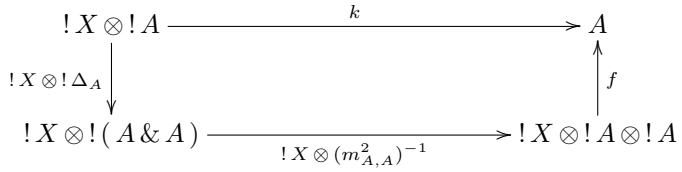
(9.23)

over $f \star g$ justifying that $(u, a) \in \mathbf{Y}_{X,A}(f \star g)$.

Diagonal property. Let $f : !X \otimes !A \otimes !A \multimap A$ be a morphism in *Rel*. We want to check that the morphism

$$\mathbf{Y}_{X,A}\,(\,(m^2_{X,A})^{-1} \circ \mathbf{Y}_{X\&A,A}\,(\,f \circ (\,(m^2_{X,A})^{-1} \otimes !A\,)\,)\quad :\ !X \multimap A \tag{9.24}$$

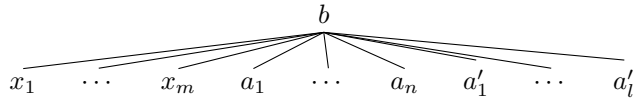
coincides with the morphism $\mathbf{Y}_{X,A}(k)$, where the morphism $k : !X \otimes !A \to A$ is defined as the composite



Let us explain first how the elements of the morphism (9.24) are computed. Each element

$$(([x_1,\ldots,x_m],[a_1,\ldots,a_n],[a'_1,\ldots,a'_l])\,,b) \quad \in \quad f$$

with $[x_1,\ldots,x_m] \in !\,X$, $[a_1,\ldots,a_n] \in !\,A$ and $[a'_1,\ldots,a'_l] \in !\,A$, can be represented as a finite tree



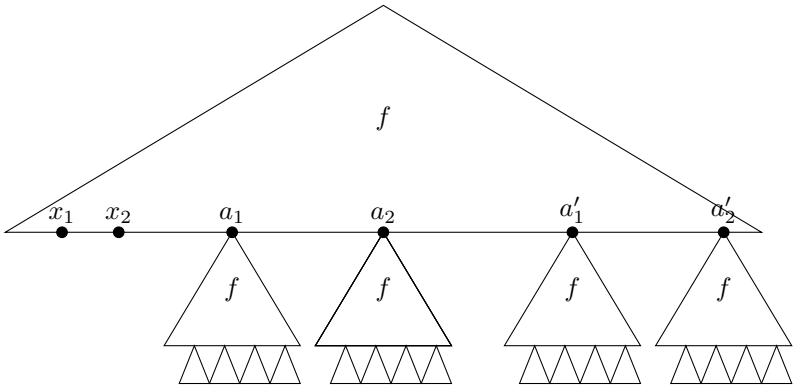
(9.25)

The computation of $\mathbf{Y}_{X\&A,A}\,(\,f \circ (\,(m^2_{X,A})^{-1} \otimes !\,A\,)$ involves the construction of run-trees in which the elements $x_1,\ldots$ but also $a_1,\ldots$ are treated as parameters: the a_i are leaves of run-trees, while the elements a'_j are iteratively pasted to other trees of the form (9.25) and of root a'_j. We denote by $\mathfrak{R}$ the set of these run-trees. The multiset of leaves of a run-tree of $\mathcal{R} \in \mathfrak{R}$ is an element of $!\,(X\,\&\,A) \cong !\,X \otimes !\,A$. The computation of the morphism (9.24) can then be understood as a process composing together run-trees of $\mathfrak{R}$ on their leaves labeled by A – which are no longer treated as parameters in this second step.

On the other hand, the morphism k contains the elements

$$((u,v_1+v_2)\,,a) \quad \text{such that} \quad ((u,v_1,v_2)\,,a) \in f$$

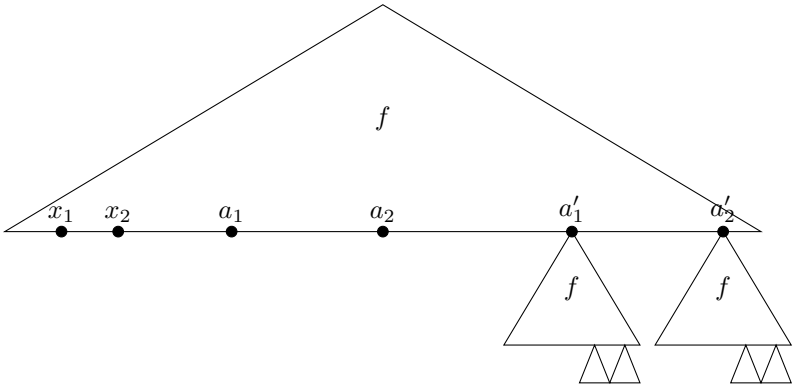
It follows that a run-tree over k is again obtained by composing together trees of the shape (9.25) but this time by considering only the elements $x_k \in X$ as parameters: both the a_i and a'_j are pasted to other trees of the shape (9.25) to obtain a run-tree over k. So, run-trees for (9.24) and for k are the same, but are computed differently: the run-trees for k are computed in one step:



(9.26)

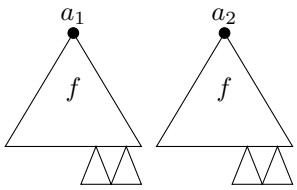
while the run-trees for (9.24) are computed by iterating first on the second

copy of $!A$:



(9.27)

and in a second time by iterating on the other copy of $!A$, obtaining again (9.27) as follows. The fact that (9.26) is a run-tree of k implies that the two trees



are run-trees over $(m^2_{X,A})^{-1} \circ \mathbf{Y}_{X\&A,A}\,(\,f \circ (\,(m^2_{X,A})^{-1} \otimes !\,A\,)$, but also that the remaining « tiny triangles » are. It follows that, when taking the second fixpoint operator, we can compute (9.26) by plugging them altogether. The diagonal property follows.

□

Example 9. Suppose that

$$f \quad = \quad \{([], a)\} \cup \{([a, x], a)\}$$

where $A = \{a\}$ and $X = \{x\}$. Denote by $\mathcal{M}_n$ the finite multiset containing the element x with multiplicity n. Then, for every $n \in \mathbb{N}$, we have that $(\mathcal{M}_n, a) \in \mathbf{Y}_{X,A}(f)$ since $(\mathcal{M}_n, a)$ can be obtained from the $\{a, x\}$-labeled witness run-tree of Figure 9.1, which has $n+1$ internal occurrences of the element a, and n occurrences of the element x at the leaves. The witness tree is finite, so that it is accepted. Now, consider the relation

$$g \quad = \quad \{([a], a)\} \cup \{([a, x], a)\}$$

In that case, $(\mathcal{M}_n, a)$ is not an element of $\mathbf{Y}_{X,A}(g)$ for any $n \in \mathbb{N}$ because all run-trees are necessarily infinite, as depicted in Figure 9.2, and thus none is accepting. As a consequence, $\mathbf{Y}_{X,A}(g)$ is the empty relation.

The terminology which we have chosen for the definition of $\mathbf{Y}$ is obviously automata-theoretic. In fact, as we already mentioned in §9.3, this definition may be formulated as an exploration of the infinitary tree **comb** on the ranked alphabet $\Sigma = \{\bullet : 2, \circ : 0\}$ by an alternating tree automaton associated

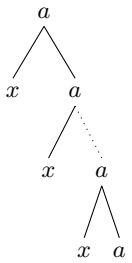


Figure 9.1: An accepting run-tree.

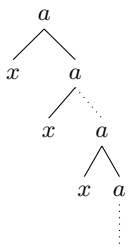


Figure 9.2: A non-accepting run-tree.

to the binary relation $f \ :!X \otimes !A \multimap A$. Indeed, given an element $a \in A$, consider the alternating tree automata $\mathcal{A}_{f,a} = \langle \Sigma,\, X \uplus A,\, \delta,\, a \rangle$ where, for $b \in A$ and $x \in X$:

$$\delta(b,\, \bullet) = \bigvee_{((\![x_1,\cdots,x_n)\!),(\![a_1,\cdots,a_m]\!),b)\in f} \big(\,(1,x_1) \wedge \cdots \wedge (1,x_n) \wedge (2,a_1) \wedge \cdots \wedge (2,a_m)\big)$$

$$\delta(x,\, \bullet) \;=\; \bot \qquad \delta(x,\, \circ) \;=\; \top \qquad \delta(b,\, \circ) \;=\; \begin{cases} \top & \text{if } (\![]\!], b) \in f \\ \bot & \text{else} \end{cases}$$

Note that we allow here the use of an infinite non-deterministic choice operator $\bigvee$ in formulas describing transitions, but only with *finite* alternation. Now, our point is that **run-tree**(f, a) coincides with the set of run-trees of the alternating automaton $\mathcal{A}_{f,a}$ over the infinite tree **comb** depicted in §9.3. Notice that only finite run-trees are accepting: this requires that for some $b \in A$ the transition $\delta(b,\, \bullet)$ contains the alternating choice $\top$, in which the exploration of the infinite branch of **comb** stops and produces an accepting run-tree. This requires in particular the existence of some $b \in A$ such that $(\![]\!], b) \in f$.

9.6 Infinitary exponentials

Now that we established a link with tree automata theory, it is tempting to relax the finiteness acceptance condition on run-trees applied in the previous section. To that purpose, however, we need to relax the usual assumption that the formulas of linear logic are interpreted as finite or countable sets. Suppose indeed that we want to interpret the exponential modality

$$\lightning\, A$$

as the set of finite or countable multisets, where a countable multiset of elements of A is defined as a function

$$A \quad \longrightarrow \quad \overline{\mathbb{N}}$$

with finite or countable support. Quite obviously, the set

$$\lightning\, \mathbb{N}$$

has the cardinality of the reals $2^{\aleph_0}$. We thus need to go beyond the traditionally countable relational interpretations of linear logic. However, we may suppose

that every set A interpreting a formula has a cardinality below or equal $2^{\aleph_0}$. In order to understand why, it is useful to reformulate the elements of $\frac{1}{2}A$ as finite or infinite words of elements of A modulo an appropriate notion of equivalence of finite or infinite words up to permutation of letters. Given a finite word u and a finite or infinite word w, we write

$$u \sqsubseteq w$$

when there exists a finite prefix v of w such that u is a prefix of v modulo permutation of letter. We write

$$w_1 \simeq w_2 \stackrel{def}{\iff} \forall u \in A^*, \quad u \sqsubseteq w_1 \iff u \sqsubseteq w_2$$

where A^* denotes the set of finite words on the alphabet A.

Proposition 20. *There is a one-to-one relationship between the elements of $\frac{1}{2}A$ and the finite or infinite words on the alphabet A modulo the equivalence relation $\simeq$.*

Proof. Let us consider the application $\varphi : ((A^* \cup A^\omega)/\simeq) \longrightarrow \frac{1}{2}A$ defined as follows: for every equivalence class $\widehat{w}$ containing a word $w \in A^* \cup A^\omega$, φ maps $\widehat{w}$ to the multiset $m_w : A \to \overline{\mathbb{N}}$ defined as $m_w(a) = |w|_a$ for every $a \in A$. Note that m_w has a finite or countable support, as w is finite or countable.

φ is properly defined: for every $w, w' \in \widehat{w}$, $m_w = m_{w'}$. Indeed, suppose that it is not the case. This implies that there exists a letter $a \in A$ such that $|w|_a \neq |w'|_a$. Suppose, without loss of generality, that $|w|_a > |w'|_a$. Note that this implies that $|w'|_a$ is a finite number. By hypothesis, $w \simeq w'$, so that $\forall u \in A^*, \quad u \sqsubseteq w \iff u \sqsubseteq w'$. Let us consider the finite word $u = a^{|w'|_a+1}$. Then $u \sqsubseteq w$ but $u \sqsubseteq w'$ does not hold, as there is no finite prefix of w' containing $|w'|_a + 1$ occurrences of the letter a. This contradicts $w \simeq w'$. It follows that φ is properly defined.

Let us prove that φ is injective. Suppose that there exists two words $w, w' \in A^* \cup A^\omega$ such that $\widehat{w} \neq \widehat{w'}$ but $\varphi(\widehat{w}) = \varphi(\widehat{w'})$. This implies that $m_w = m_{w'}$. Since $\widehat{w} \neq \widehat{w'}$, there exists a finite word $u \in A^*$ such that either $u \not\sqsubseteq w$ and $u \sqsubseteq w'$, or that $u \sqsubseteq w$ and $u \not\sqsubseteq w'$. Let us treat the first case, without loss of generality. This implies that there exists a letter $a \in A$ such that $|w|_a < |u|_a \leq |w'|_a$. It follows that $|w|_a \neq |w'|_a$, which contradicts $m_w = m_{w'}$.

To conclude, let us prove that φ is surjective. Let $m \in \frac{1}{2}A$ be a multiset; it is at most countable, as it is a collection of at most countably many elements with multiplicity at most ω. It follows that its cardinality is bounded by $\aleph_0 \times \aleph_0 = \aleph_0$. We can thus enumerate these elements, and build in this way a word $w \in A^* \cup A^\omega$ such that for every $a \in A$, $|w|_a = m(a)$. By construction, $\varphi(\widehat{w}) = m$. □

Proposition 20 means in particular that, for every set A, there is a surjection from the set $A^\infty = A^* \uplus A^\omega$ of finite or infinite words on the alphabet A to the set $\frac{1}{2}A$ of finite or countable multisets. An element of the equivalence class associated to a multiset is called a *representation* of this multiset. Notice that if a set A has cardinality at most $2^{\aleph_0}$, the set A^∞ is itself bounded by $2^{\aleph_0}$, since $(2^{\aleph_0})^{\aleph_0} = 2^{\aleph_0 \times \aleph_0} = 2^{\aleph_0}$. This property leads us to define the following extension of *Rel*:

Definition 36. The category $\underline{Rel}$ has the sets A, B of cardinality at most $2^{\aleph_0}$ as objects, and binary relations $f \subseteq A \times B$ between A and B as morphisms $A \to B$.

Since a binary relation between two sets A and B is a subset of $A \times B$, the cardinality of a binary relation in $\underline{Rel}$ is also bounded by $2^{\aleph_0}$. Note that the hom-set $\underline{Rel}(A, B)$ is in general of higher cardinality than $2^{\aleph_0}$, yet it is bounded by the cardinality of the powerset of the reals. It is immediate to establish that:

Proposition 21. *The category $\underline{Rel}$ is $*$-autonomous and has finite products. As such, it provides a model of multiplicative additive linear logic.*

Proof. It is well-known that the category with all sets as objects and their relations as morphisms is $*$-autonomous and has finite products. $\underline{Rel}$ is a subcategory of this category, contains the dualizing object $\bot = 1$, and is stable under finite tensors and finite products. It follows that it is a $*$-autonomous category with finite products. □

There remains to show that the finite-or-countable multiset construction $\frac{1}{2}$ defines a categorical interpretation of the exponential modality of linear logic. Again, just as in the finitary case, we find convenient to check that $\underline{Rel}$ together with the finite-or-countable multiset interpretation $\frac{1}{2}$ satisfy the axioms of a Seely category. In that specific formulation of a model of linear logic, the first property to check is that:

Proposition 22. *The finite-or-countable multiset construction $\frac{1}{2}$ defines a comonad on the category $\underline{Rel}$.*

The counit of the comonad is defined as the binary relation

$$\mathbf{der}_A \quad : \quad \frac{1}{2}\, A \quad \longrightarrow \quad A$$

which relates $[a]$ to a for every element a of the set A. In order to define its comultiplication, we need first to extend the notion of sum of multisets to the infinitary case, which we do in the obvious way, by extending the binary sum of $\mathbb{N}$ to possibly infinite sums in its completion $\overline{\mathbb{N}}$. In order to unify the notation for finite-or-countable multisets with the one for finite multisets used in Section 5.2, we find convenient to denote by $[a_1, a_2, \cdots]$ the countable multiset admitting the representation $a_1 a_2 \cdots$

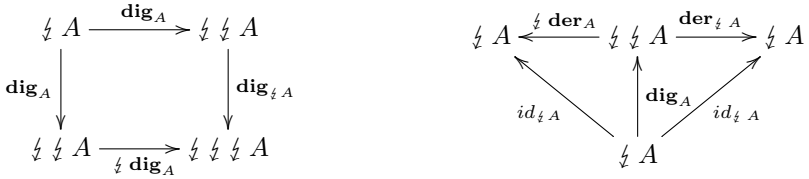
We are now ready to describe the comultiplication

$$\mathbf{dig}_A \quad : \quad \frac{1}{2}\, A \quad \to \quad \frac{1}{2}\,\frac{1}{2}\, A$$

of the comonad $\frac{1}{2}$ as a straightforward generalization of the finite case:

$$\begin{array}{lll} \mathbf{dig}_A & = & \{(w_1 + \cdots + w_k, [w_1, \cdots, w_k]) \mid \forall i \in \{1, \cdots n\}, w_i \in \frac{1}{2}\, A\} \\ & & \cup \{(w_1 + \cdots + w_k + \cdots, [w_1, \cdots, w_k, \cdots]) \mid \forall i \in \mathbb{N}, w_i \in \frac{1}{2}\, A\} \end{array}$$

Proof of Proposition 22. We need to prove that **dig** and **der** are natural transformations making the following diagrams commute for every object $A \in \underline{Rel}$:



The naturality of **dig** and **der** is proved in an immediate way. Let us start by proving that the diagram on the left commutes. It is instructive to exhibit first the elements of the relations $\mathbf{dig}_{\S A}$ and $\S\, \mathbf{dig}_A$. Elements of $\mathbf{dig}_{\S A}$ are of the form

$$([w_{11}, w_{12}, \ldots] + [w_{21}, w_{22}, \ldots] + \cdots, [[w_{11}, w_{12}, \ldots], [w_{21}, w_{22}, \ldots], \ldots])$$

where each enumeration may be either finite or infinite, and where each w_{ij} is an element of $\S A$. Now elements of $\S\, \mathbf{dig}_A$ are of the form

$$([w_{11} + w_{12} + \cdots, w_{21} + w_{22} + \cdots, \ldots], [[w_{11}, w_{12}, \ldots], [w_{21}, w_{22}, \ldots], \ldots])$$

where, again, the w_{ij} are in $\S A$, and the enumerations may be either finite or infinite.

Let us consider an element of $\mathbf{dig}_{\S A} \circ \mathbf{dig}_A$. It is an element of the form

$$(w_{11} + w_{12} + \cdots + w_{21} + w_{22} + \cdots, [[w_{11}, w_{12}, \ldots], [w_{21}, w_{22}, \ldots], \ldots])$$

obtained by composing the element

$$(w_{11} + w_{12} + \cdots + w_{21} + w_{22} + \cdots, [w_{11}, w_{12}, \ldots, w_{21}, w_{22}, \ldots])$$

of $\mathbf{dig}_A$ with the element

$$([w_{11}, w_{12}, \ldots] + [w_{21}, w_{22}, \ldots] + \cdots, [[w_{11}, w_{12}, \ldots], [w_{21}, w_{22}, \ldots], \ldots])$$

of $\mathbf{dig}_{\S A}$, by using the fact that

$$[w_{11}, w_{12}, \ldots, w_{21}, w_{22}, \ldots] \;=\; [w_{11}, w_{12}, \ldots] + [w_{21}, w_{22}, \ldots] + \cdots, [[w_{11}, w_{12}, \ldots]$$

Consider now the element

$$((w_{11} + w_{12} + \cdots) + (w_{21} + w_{22} + \cdots) + \cdots, [w_{11} + w_{12} + \cdots, w_{21} + w_{22} + \cdots, \ldots])$$

of $\mathbf{dig}_A$, and compose it with the element

$$([w_{11} + w_{12} + \cdots, w_{21} + w_{22} + \cdots, \ldots], [[w_{11}, w_{12}, \ldots], [w_{21}, w_{22}, \ldots], \ldots])$$

of $\S\, \mathbf{dig}_A$. We obtain in this way that

$$(w_{11} + w_{12} + \cdots + w_{21} + w_{22} + \cdots, [[w_{11}, w_{12}, \ldots], [w_{21}, w_{22}, \ldots], \ldots])$$

is also in $!\,\mathbf{dig}_A \circ \mathbf{dig}_A$. Therefore $\mathbf{dig}_{!\,A} \circ \mathbf{dig}_A \subseteq !\,\mathbf{dig}_A \circ \mathbf{dig}_A$. The converse inclusion is proved similarly.

We consider now the diagram on the right. The first thing is to prove that $!\,\mathbf{der}_A \circ \mathbf{dig}_A = id_{!\,A}$. Let us consider $w = [a_1, \ldots] \in !\,A$. $(w, w) \in id_A$, and $(w, w) \in !\,\mathbf{der}_A \circ \mathbf{dig}_A$ as well, since we can compose the elements

$$(w, [[a_1], \ldots]) \in \mathbf{dig}_A$$

and

$$([[a_1], \ldots], [a_1, \ldots]) \in !\,\mathbf{der}_A$$

Now if $(u, w) \in !\,\mathbf{der}_A \circ \mathbf{dig}_A$, it is obtained by composing an element $(u, v) \in \mathbf{dig}_A$ with an element $(v, w) \in !\,\mathbf{der}_A$. Since $(v, w) \in !\,\mathbf{der}_A$, v is a multiset of singleton multisets: $v = [[a_1], \ldots]$, and $w = [a_1, \ldots]$. By definition of $\mathbf{dig}_A$, $u = w$ so that $(u, w) \in id_A$.

We now have to prove that $\mathbf{der}_{!\,A} \circ \mathbf{dig}_A = id_{!\,A}$. Consider $w = [a_1, \ldots] \in !\,A$. $(w, w) \in id_A$, and $(w, w) \in \mathbf{der}_{!\,A} \circ \mathbf{dig}_A$ as well, since we can compose the elements

$$(w, [w], \ldots]) \in \mathbf{dig}_A$$

and

$$([w], w) \in \mathbf{der}_{!\,A}$$

Now if $(u, w) \in \mathbf{der}_{!\,A} \circ \mathbf{dig}_A$, it is obtained by composing an element $(u, v) \in \mathbf{dig}_A$ with an element $(v, w) \in \mathbf{der}_{!\,A}$. Since $(v, w) \in \mathbf{der}_{!\,A}$, v is a multiset containing just one multiset of elements of A: $v = [w]$. By definition of $\mathbf{dig}_A$, $u = w$ so that $(u, w) \in id_A$.

Seely isomorphisms. One then defines the isomorphism

$$m^0 \;=\; \{(\star, [])\} \quad : \quad 1 \quad \longrightarrow \quad !\,\top \tag{9.28}$$

and the family of isomorphisms

$$m^2_{A,B} \quad : \quad !\,A \otimes !\,B \quad \longrightarrow \quad !\,(\,A \,\&\, B\,) \tag{9.29}$$

indexed by the objects A, B of the category $\underline{Rel}$ which relates every pair (w_A, w_B) of the set $!\,A \otimes !\,B$ with the finite-or-countable multiset

$$(\{1\} \times w_A) + (\{2\} \times w_B) \quad \in \quad !\,(\,A \,\&\, B\,)$$

where the operation $\{1\} \times w_A$ maps the finite-or-countable multiset $w_A = [a_1, a_2, \ldots]$ of elements of A to the finite-or-countable multiset $[(1, a_1), (1, a_2), \ldots]$ of $!\,(A \& B)$. We define $\{2\} \times w_B$ similarly. Recall that Seely categories are defined in Definition 32, page 111. We check carefully that

Proposition 23. *The comonad $!$ on the category $\underline{Rel}$ together with the isomorphisms (9.28) and (9.29) satisfy the coherence axioms of a Seely category.*

In other words, this comonad $\not!$ over the category $\underline{Rel}$ induces a new and infinitary model of propositional linear logic. The next section is devoted to the definition of two different fixpoint operators living inside this new model.

Proof. The naturality of m^0 and $(m^2_{A,B})_{A,B}$ is immediate. We need to check that the diagrams of Definition 32 commute in $\underline{Rel}$. Let A and B be two objects of $\underline{Rel}$, we start by the diagram

$$\begin{array}{ccc}
\not!A \otimes \not!B & \xrightarrow{m^2_{A,B}} & \not!(A \,\&\, B) \\
\Big\downarrow{\scriptstyle \mathbf{dig}_A \otimes \mathbf{dig}_B} & & \Big\downarrow{\scriptstyle \mathbf{dig}_{A\&B}} \\
 & & \not!\not!(A \,\&\, B) \\
 & & \Big\downarrow{\scriptstyle \not!\langle \not!\pi_1, \not!\pi_2\rangle} \\
\not!\not!A \otimes \not!\not!B & \xrightarrow{m^2_{\not!A,\not!B}} & \not!(\not!A \,\&\, \not!B)
\end{array}$$

The elements of $m^2_{\not!A,\not!B} \circ (\mathbf{dig}_A \otimes \mathbf{dig}_B)$ are obtained by composing an element

$$\begin{array}{l}
((\![a_{11}, a_{12}, \ldots, a_{21}, a_{22}, \ldots], [b_{11}, b_{12}, \ldots, b_{21}, b_{22}, \ldots]\!)\,, \\
\qquad ([\![[a_{11}, a_{12}, \ldots], [a_{21}, a_{22}, \ldots], \ldots], [[b_{11}, b_{12}, \ldots], [b_{21}, b_{22}, \ldots], \ldots]\!]))
\end{array}$$

of $\mathbf{dig}_A \otimes \mathbf{dig}_B$ with the element

$$\begin{array}{l}
((\![[\![a_{11}, a_{12}, \ldots], [a_{21}, a_{22}, \ldots], \ldots], [[b_{11}, b_{12}, \ldots], [b_{21}, b_{22}, \ldots], \ldots]\!)\,, \\
\qquad ([\![(1, [a_{11}, a_{12}, \ldots]), (1, [a_{21}, a_{22}, \ldots]), \ldots], [(2, [b_{11}, b_{12}, \ldots]), (2, [b_{21}, b_{22}, \ldots]), \ldots]\!]))
\end{array}$$

of $m^2_{\not!A,\not!B}$. It follows that the elements of $m^2_{\not!A,\not!B} \circ (\mathbf{dig}_A \otimes \mathbf{dig}_B)$ are precisely these of the shape

$$\begin{array}{l}
((\![a_{11}, a_{12}, \ldots, a_{21}, a_{22}, \ldots], [b_{11}, b_{12}, \ldots, b_{21}, b_{22}, \ldots]\!)\,, \\
\qquad ([\![(1, [a_{11}, a_{12}, \ldots]), (1, [a_{21}, a_{22}, \ldots]), \ldots], [(2, [b_{11}, b_{12}, \ldots]), (2, [b_{21}, b_{22}, \ldots]), \ldots]\!]))
\end{array}$$

We consider such an element, and prove that it is in $\not!\langle \not!\pi_1, \not!\pi_2\rangle \circ \mathbf{dig}_{A\&B} \circ m^2_{A,B}$. We have that

$$\begin{array}{l}
((\![a_{11}, a_{12}, \ldots, a_{21}, a_{22}, \ldots], [b_{11}, b_{12}, \ldots, b_{21}, b_{22}, \ldots]\!)\,, \\
\qquad ([\![(1, a_{11}), (1, a_{12}), \ldots, (1, a_{21}), (1, a_{22}), \ldots, (2, b_{11}), (2, b_{12}), \ldots, (2, b_{21}), (2, b_{22}), \ldots]\!))
\end{array}$$

is in $m^2_{A,B}$. We compose it with the element

$$\begin{array}{l}
((((\![(1, a_{11}), (1, a_{12}), \ldots, (1, a_{21}), (1, a_{22}), \ldots, (2, b_{11}), (2, b_{12}), \ldots, (2, b_{21}), (2, b_{22}), \ldots]\!)\,, \\
\qquad ([\![[\![(1, a_{11}), (1, a_{12}), \ldots], [(1, a_{21}), (1, a_{22}), \ldots], \ldots, [(2, b_{11}), (2, b_{12}), \ldots], [(2, b_{21}), (2, b_{22}), \ldots], \ldots]\!]))
\end{array}$$

of $\mathbf{dig}_{A\&B}$, and we compose the result with the element

$$\begin{array}{l}
((((\![[\![(1, a_{11}), (1, a_{12}), \ldots], [(1, a_{21}), (1, a_{22}), \ldots], \ldots, [(2, b_{11}), (2, b_{12}), \ldots], [(2, b_{21}), (2, b_{22}), \ldots], \ldots]\!)\,, \\
\qquad ((\![(1, [a_{11}, a_{12}, \ldots]), (1, [a_{21}, a_{22}, \ldots]), \ldots, (2, [b_{11}, b_{12}, \ldots]), (2, [b_{21}, b_{22}, \ldots]), \ldots]\!))
\end{array}$$

of $\not!\langle \not!\pi_1, \not!\pi_2\rangle$, obtaining the desired result. The converse direction is proved similarly.

We now focus on the diagram

$$\begin{array}{ccc}
(\not{!}A \otimes \not{!}B) \otimes \not{!}C & \xrightarrow{\quad\alpha\quad} & \not{!}A \otimes (\not{!}B \otimes \not{!}C) \\
\downarrow{\scriptstyle m\otimes \not{!}C} & & \downarrow{\scriptstyle \not{!}A\otimes m} \\
\not{!}(A \,\&\, B) \otimes \not{!}C & & \not{!}A \otimes \not{!}(B \,\&\, C) \\
\downarrow{\scriptstyle m} & & \downarrow{\scriptstyle m} \\
\not{!}((A \,\&\, B) \,\&\, C) & \xrightarrow{\quad\not{!}\alpha\quad} & \not{!}(A \,\&\, (B \,\&\, C))
\end{array}$$

An element of the "down-left" relation $\not{!}\alpha \circ m \circ (m \otimes \not{!}C)$ is obtained by composing an element

$$\begin{array}{l}
((([a_1, a_2, \ldots], [b_1, b_2, \ldots]), [c_1, c_2, \ldots]), \\
\qquad ([(1, a_1), (1, a_2), \ldots, (2, b_1), (2, b_2), \ldots], [c_1, c_2, \ldots]))
\end{array}$$

of $m \otimes \not{!}C$ with an element

$$\begin{array}{l}
(([(1, a_1), (1, a_2), \ldots, (2, b_1), (2, b_2), \ldots], [c_1, c_2, \ldots]), \\
\qquad ([(1, (1, a_1)), (1, (1, a_2)), \ldots, (1, (2, b_1)), (1, (2, b_2)), \ldots, (2, c_1), (2, c_2), \ldots]))
\end{array}$$

of m, and then by composing the result with the element

$$\begin{array}{l}
(([(1, (1, a_1)), (1, (1, a_2)), \ldots, (1, (2, b_1)), (1, (2, b_2)), \ldots, (2, c_1), (2, c_2), \ldots]), \\
\qquad ([(1, a_1)), (1, a_2)), \ldots, (2, (1, b_1)), (2, (1, b_2)), \ldots, (2, (2, c_1)), (2, (2, c_2)), \ldots]))
\end{array}$$

of $\not{!}\alpha$. So the relation $\not{!}\alpha \circ m \circ (m \otimes \not{!}C)$ is precisely made of the elements which can be written as

$$\begin{array}{l}
((([a_1, a_2, \ldots], [b_1, b_2, \ldots]), [c_1, c_2, \ldots]), \\
\qquad ([(1, a_1)), (1, a_2)), \ldots, (2, (1, b_1)), (2, (1, b_2)), \ldots, (2, (2, c_1)), (2, (2, c_2)), \ldots]))
\end{array}$$

with $a_i \in A,\ b_j \in B,\ c_k \in C$. Let us consider such an element, and prove that it belongs to $m \circ (\not{!}\, A \otimes m) \circ \alpha$. We obtain it by composing the element

$$\begin{array}{l}
((([a_1, a_2, \ldots], [b_1, b_2, \ldots]), [c_1, c_2, \ldots]), \\
\qquad ([a_1, a_2, \ldots], ([b_1, b_2, \ldots], [c_1, c_2, \ldots])))
\end{array}$$

of the associativity isomorphism α with the element

$$\begin{array}{l}
(([a_1, a_2, \ldots], ([b_1, b_2, \ldots], [c_1, c_2, \ldots])), \\
\qquad ([a_1, a_2, \ldots], ([(1, b_1), (1, b_2), \ldots, (2, c_1), (2, c_2), \ldots])))
\end{array}$$

of $(\not{!}\, A \otimes m)$. We then compose the result with the element

$$\begin{array}{l}
(([a_1, a_2, \ldots], ([(1, b_1), (1, b_2), \ldots, (2, c_1), (2, c_2), \ldots])), \\
\qquad ([(1, a_1), (1, a_2), \ldots], ([(1, (1, b_1)), (1, (1, b_2)), \ldots, (1, (2, c_1)), (1, (2, c_2)), \ldots])))
\end{array}$$

and we obtain in this way that

$$\not{!}\alpha \circ m \circ (m \otimes \not{!}C) \quad \subseteq \quad m \circ (\not{!}\, A \otimes m) \circ \alpha$$

The converse inclusion is proved in the exact same way.

We now have to consider two diagrams concerning the compatibility of m with the units:

$$\begin{array}{ccc} \not\downarrow A \otimes 1 & \xrightarrow{\rho^{\otimes}} & \not\downarrow A \\ {\scriptstyle \not\downarrow A \otimes m}\downarrow & & \uparrow{\scriptstyle \not\downarrow \rho^{\&}} \\ \not\downarrow A \otimes \not\downarrow \top & \xrightarrow{m} & \not\downarrow (A \,\&\, \top) \end{array} \qquad \begin{array}{ccc} 1 \otimes \not\downarrow B & \xrightarrow{\lambda^{\otimes}} & \not\downarrow B \\ {\scriptstyle m \otimes \not\downarrow B}\downarrow & & \uparrow{\scriptstyle \not\downarrow \lambda^{\&}} \\ \not\downarrow \top \otimes \not\downarrow B & \xrightarrow{m} & \not\downarrow (\top \,\&\, B) \end{array}$$

We restrict our attention to the one on the left, as the proof of the other is exactly the same modulo the commutation isomorphism $\gamma : A \otimes B \to B \otimes A$ of $\underline{Rel}$. We need to prove that $\not\downarrow \rho^{\&}_A \circ m \circ (\not\downarrow A \otimes m) = \rho^{\otimes}_{\not\downarrow A}$, where

$$\rho^{\otimes}_A = \{ ((a, \star), a) \mid a \in A \} : A \otimes 1 \to A$$

and

$$\rho^{\&}_A = \{ ((1, a), a) \mid a \in A \} : A \,\&\, \top \to A$$

are the left unit morphisms for the tensorial and cartesian structure, respectively. Let us consider $(([a_1, a_2, \ldots], []), [a_1, a_2, \ldots]) \in \rho^{\otimes}_{\not\downarrow A}$ and prove that it is in $\not\downarrow \rho^{\&}_A \circ m \circ (\not\downarrow A \otimes m)$. We obtain it by composing the element

$$(([a_1, a_2, \ldots], \star), ([a_1, a_2, \ldots], [])) \quad \in \quad \not\downarrow A \otimes m^0$$

with the element

$$(([a_1, a_2, \ldots], []), [(1, a_1), (1, a_2), \ldots]) \quad \in \quad m^2_{A, \top}$$

and then with the element

$$([(1, a_1), (1, a_2), \ldots], [a_1, a_2, \ldots]) \quad \in \quad \not\downarrow \rho^{\&}_A$$

In this way we get the inclusion $\rho^{\otimes}_{\not\downarrow A} \subseteq \not\downarrow \rho^{\&}_A \circ m \circ (\not\downarrow A \otimes m)$. The converse inclusion is proven in the exact same way.

The commutation of the last diagram:

$$\begin{array}{ccc} \not\downarrow A \otimes \not\downarrow B & \xrightarrow{\gamma} & \not\downarrow B \otimes \not\downarrow A \\ {\scriptstyle m}\downarrow & & \downarrow{\scriptstyle m} \\ \not\downarrow (A \,\&\, B) & \xrightarrow{\not\downarrow \gamma} & \not\downarrow (B \,\&\, A) \end{array}$$

is particularly straightforward to prove. □

9.7 Inductive and coinductive fixpoint operators

In the infinitary relational semantics, a binary relation

$$f : \not\downarrow A \multimap B$$

may require a countable multiset w of elements (or positions) of the input set A in order to reach a position b of the output set B. For that reason, we

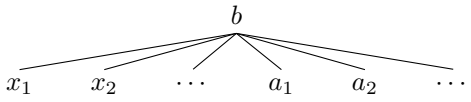
need to generalize the notion of alternating tree automata to *finite-or-countable alternating tree automata*, a variant in which formulas defining transitions use of a possibly countable alternation operator $\bigwedge$ and of a possibly countable non-deterministic choice operator $\bigvee$. The generalization of the family of automata $\mathcal{A}_{f,a}$ of §9.5 leads to a new definition of the set **run-tree**(f, a), in which witness trees may have internal nodes of countable arity. A first important observation is the following result:

Proposition 24. *Given* $f : \not{!}\, A \otimes \not{!}\, X \multimap A$, $a \in A$, *and* $witness \in$ ***run-tree***(f, a), *the multiset* ***leaves***$(witness)$ *is finite or countable.*

Proof. Let us consider the worst case, in which *witness* is a tree of countable depth, and build only from elements

$$([(1, x_1), (1, x_2), \ldots, (2, a_1), (2, a_2), \ldots], b) \ \in \ f$$

where the enumerations are countable, so that it looks locally as



where the enumerations are, again, countable. It follows that the cardinal of the set of leaves of *witness* is

$$\sum_{n \in \mathbb{N}\setminus\{0\}} \aleph_0^n \ = \ \sum_{n \in \mathbb{N}\setminus\{0\}} \aleph_0 \ = \ \aleph_0 \times \aleph_0 \ = \ \aleph_0$$

so that **leaves**$(witness)$ is countable.

□

An important consequence of this observation is that the definition of the Conway operator $\mathbf{Y}$ given in Equation (9.10) can be very simply adapted to the finite-or-countable interpretation of the exponential modality $\not{!}$ in the Seely category $\underline{Rel}$. Moreover, in this infinitary model of linear logic, we can give more elaborate acceptation conditions, among which two are canonical:

- considering that any run-tree is accepting, one defines the *coinductive* fixpoint on the model, which is the greatest fixpoint over $\underline{Rel}$.
- on the other hand, by accepting only trees without infinite branches, we obtain the *inductive* interpretation of the fixpoint, which is the least fixpoint operator over $\underline{Rel}$.

It is easy to see that the two fixpoint operators are different: recall Example 9, and observe that the binary relation g is also a relation in the infinitary semantics. It turns out that its inductive fixpoint is the empty relation, while its coinductive fixpoint coincides with the relation

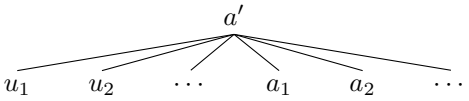
$$\{(\mathcal{M}_n, a) \mid n \in \mathbb{N}\} \ \cup \ \{([x, x, \cdots], a)\}$$

In this coinductive interpretation, the run-tree obtained by using infinitely $([x, a], a)$ and never $([a], a)$ is accepting and is the witness tree generating $\{([x, x, \cdots], a)\}$.

Proposition 25. *The inductive and coinductive fixpoint operators over the infinitary relational model of linear logic are Conway operators on this Seely category.*

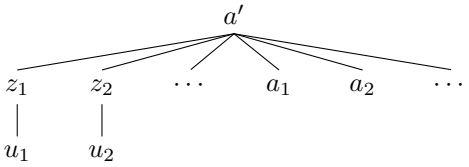
Proof. This proposition can be proved by adapting the proof of Proposition 19. The arguments are the same, except that we need to consider finite-or-countable multisets and therefore trees with countable branching. In the coinductive case, we also need to consider trees of infinite depth. Let us briefly explain why the property holds.

- **Naturality.** The core argument of the proof of naturality is the fact that every tree



(9.30)

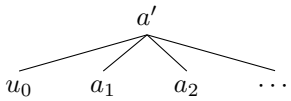
  representing the fact that $((u_1 + u_2 + \cdots, [a_1, a_2, \ldots]), a') \in k$ can be decomposed as before as the tree



(9.31)

  provided that $\forall i$, $(u_i, z_i) \in f$, so that any run-tree over k justifying that $(u, a) \in \mathbf{Y}_{X,A}(k)$ can be turned into a run-tree over f differing only by its arguments, and justifying that $(v, a) \in \mathbf{Y}_{Z,A}(f)$. Composing (v, a) with $(\mathbf{dig}_X \otimes !\, A) \circ (\lightning\, g \otimes !\, A)$, we obtain as earlier that $(u, a) \in (\mathbf{dig}_X \otimes !\, A) \circ (\lightning\, g \otimes !\, A) \circ \mathbf{Y}_{Z,A}(f)$. The converse inclusion is proved in the same way.

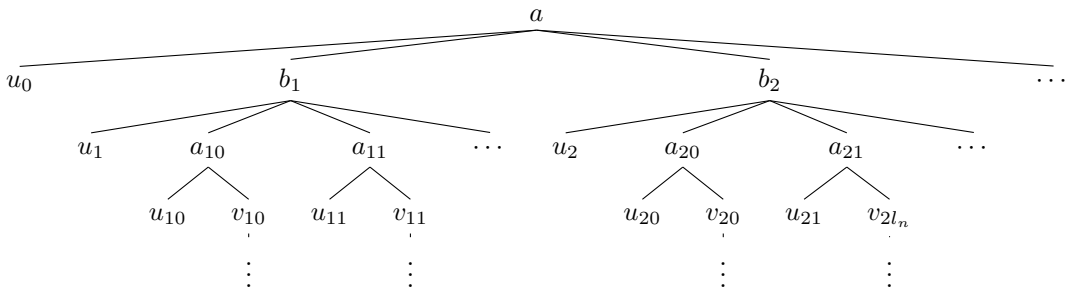
- **Parametrized fixpoint property.** This is again a straightforward extension of the proof of Proposition 19 to the case where trees may be of countable width. The key property here is that any run-tree whose root is of the shape



(9.32)

  gives a family of run-trees of root a_i, each justifying that $(u_i, a_i) \in \mathbf{Y}_{X,A}(f)$, with $((u_0, [a_1, a_2, \ldots]), a') \in f$. It follows that the relation $\mathbf{Y}_{X,A}(f)$ is contained in the one defined by the lower part of the diagram describing the property. The converse inclusion is obtained similarly, as the composition of the morphism f with elements of $\mathbf{Y}_{X,A}(f)$ gives a run-tree over f whose root is of the shape (9.31) and where a run-tree of root a_i is pasted to the leaf labeled a_i.

- **Parametrized dinaturality.** The key point here is that the composition diagram (9.20) can be adapted to our infinitary setting by considering its extension to trees of countable width:



(9.33)

As earlier, we obtain that the trees rooted in b_i are run-trees over $g \star f$, and that we can compose their denotations with f to obtain precisely the elements of $\mathbf{Y}_{X,A}(f \star g)$.

- **Diagonal property.** This property holds as well in this infinitary case: it suffices to extend the figures (9.26) and (9.27) to trees of countable width and depth to obtain the result.

□

9.8 The colored exponential modality

The purpose of Chapter 6 was to carefully study the Kobayashi-Ong type system, and to excavate from its reformulation using the type system $\mathbf{\Upsilon}(\mathcal{A})$ the very *modal* nature of the coloring operation. As a modality, coloring can be interpreted in the relational semantics as a comonad defined as the functor

$$\Box \quad : \quad A \quad \mapsto \quad Col \times A \quad : \quad Rel \quad \to \quad Rel$$

equipped with the coercion maps

$$\begin{array}{rcrcl}
\{(((m,a),(m,b)),(m,(a,b))) \mid a \in A,\, b \in B,\, m \in Col\} & : & \Box A \otimes \Box B & \to & \Box(A \otimes B) \\
\{(\star,(m,\star)) \mid m \in Col\} & : & 1 & \to & \Box 1 \\
\{((\max(m_1,m_2),a),(m_1,(m_2,a))) \mid a \in A\} & : & \Box A & \to & \Box\Box A \\
\{((\varepsilon,a),a) \mid a \in A\} & : & \Box A & \to & A
\end{array}$$

We denote the two latest morphisms as $\mathbf{dig}^{\Box}_A$ and $\mathbf{der}^{\Box}_A$, respectively. To distinguish clearly these morphisms from the comultiplication and counity of the countable exponential functor $\not!$, we will now denote these morphisms as $\mathbf{dig}^{\not!}_A$ and $\mathbf{der}^{\not!}_A$. We may however, when it is clear from context, forget the explicit annotation on **dig** and **der**.

The four morphisms we just defined induce a lax monoidal comonad $\Box : Rel \to Rel$ on the category Rel of sets and relations. As earlier in Chapter 6,

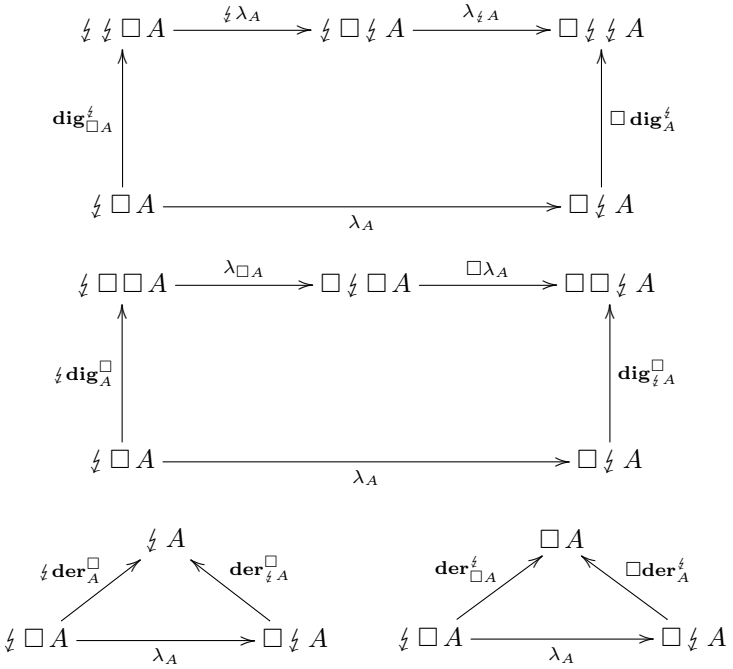
the set *Col* is a finite set of integers called *colors*, together with an additional *neutral* color ε which should be understood as the *absence* of color. The colors are introduced in order to regulate the fixpoint discipline: following the idea behind the parity condition, in the immediate scope of an even color, fixpoints should be interpreted coinductively, and inductively in the immediate scope of an odd color.

Moreover, the comonad distributes (or better: commutes) with the exponential modality $\frac{1}{2}$: there exists a distributive law [Bec69] between them.

Definition 37. A *distributive law* between two comonads $(\frac{1}{2}, \mathbf{dig}^{\frac{1}{2}}, \mathbf{der}^{\frac{1}{2}})$ and $(\Box, \mathbf{dig}^{\Box}, \mathbf{der}^{\Box})$ on a category $\mathcal{C}$ is a natural transformation

$$(\lambda_A \,:\, \tfrac{1}{2}\,\Box\, A \to \Box\,\tfrac{1}{2}\, A)_A$$

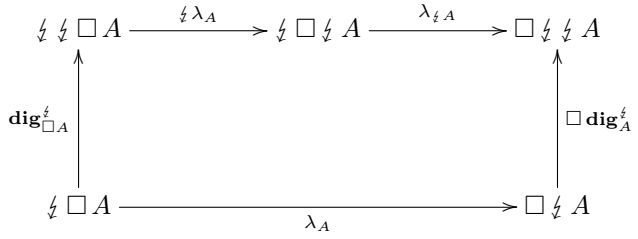
making the four following diagrams commute for every object $A \in \mathcal{C}$:



Proposition 26. *There exists a distributive law* $\lambda : \frac{1}{2}\,\Box \to \Box\,\frac{1}{2}$ *between the comonads* $\frac{1}{2}$ *and* $\Box$*. This natural transformation is defined on the object* A *as*

$$\lambda_A \;=\; \{([(c, a_1),\, (c, a_2), \ldots], (c,\, [a_1,\, a_2, \ldots])) \mid a_i \in A,\, c \in \mathit{Col}\} \;:\; \tfrac{1}{2}\,\Box\, A \to \Box\,\tfrac{1}{2}\, A$$

Proof. The naturality of λ is trivial. It remains to check that the four diagrams of Definition 37 commute. We start by the diagram:



An element of $\Box\,\mathbf{dig}^{\natural}_{A} \circ \lambda_A$ is obtained by composing an element

$$([(c, a_{11}),\, (c, a_{12}), \ldots, (c, a_{21}),\, (c, a_{22}), \ldots], (c,\, [a_{11},\, a_{12}, \ldots, a_{21},\, a_{22}, \ldots]))$$

of λ_A with an element

$$((c,\, [a_{11},\, a_{12}, \ldots, a_{21},\, a_{22}, \ldots])\,, (c,\, [[a_{11},\, a_{12}, \ldots], [a_{21},\, a_{22}, \ldots], \ldots]))$$

of $\Box\,\mathbf{dig}^{\natural}_{A}$. It follows that the elements of $\Box\,\mathbf{dig}^{\natural}_{A} \circ \lambda_A$ are precisely these of the shape

$$([(c, a_{11}),\, (c, a_{12}), \ldots, (c, a_{21}),\, (c, a_{22}), \ldots], (c,\, [[a_{11},\, a_{12}, \ldots], [a_{21},\, a_{22}, \ldots], \ldots]))$$

We consider such an element, and prove that it belongs to $\lambda_{\natural\, A} \circ \natural\ \lambda_A \circ \mathbf{dig}^{\natural}_{\Box A}$. We obtain it by composing the element

$$([(c, a_{11}),\, (c, a_{12}), \ldots, (c, a_{21}),\, (c, a_{22}), \ldots], [[(c, a_{11}),\, (c, a_{12}), \ldots], [(c, a_{21}),\, (c, a_{22}), \ldots], \ldots])$$

of $\mathbf{dig}^{\natural}_{\Box A}$ with the element

$$([[(c, a_{11}),\, (c, a_{12}), \ldots], [(c, a_{21}),\, (c, a_{22}), \ldots], \ldots], [(c, [a_{11},\, a_{12}, \ldots]), (c, [a_{21},\, a_{22}, \ldots]), \ldots])$$

of $\natural\ \lambda_A$, and we compose the result with the element

$$([(c, [a_{11},\, a_{12}, \ldots]), (c, [a_{21},\, a_{22}, \ldots]), \ldots], (c,\, [[a_{11},\, a_{12}, \ldots], [a_{21},\, a_{22}, \ldots], \ldots]))$$

of $\lambda_{\natural\, A}$. So $\Box\,\mathbf{dig}^{\natural}_{A} \circ \lambda_A \;\subseteq\; \lambda_{\natural\, A} \circ \natural\ \lambda_A \circ \mathbf{dig}^{\natural}_{\Box A}$. The converse inclusion is proved in the very same way.

We now consider the diagram

$$\begin{array}{ccccc}
\natural\,\Box\,\Box\, A & \xrightarrow{\lambda_{\Box A}} & \Box\,\natural\,\Box\, A & \xrightarrow{\Box\lambda_A} & \Box\,\Box\,\natural\, A \\
\uparrow{\scriptstyle \natural\,\mathbf{dig}^{\Box}_{A}} & & & & \uparrow{\scriptstyle \mathbf{dig}^{\Box}_{\natural A}} \\
\natural\,\Box\, A & & \xrightarrow[\lambda_A]{} & & \Box\,\natural\, A
\end{array}$$

The elements of $\mathbf{dig}^{\Box}_{\natural A} \circ \lambda_A$ are precisely these of the shape

$$([(c, a_1),\, (c, a_2), \ldots], (c_1, (c_2, [a_1,\, a_2, \ldots])))$$

with $\max(c_1, c_2) = c$. Let us consider such an element and prove that it belongs to $\Box\lambda_A \circ \lambda_{\Box A} \circ \natural\,\mathbf{dig}^{\Box}_{A}$. We obtain it by composing the element

$$([(c, a_1),\, (c, a_2), \ldots], [(c_1, (c_2, a_1)),\, (c_1, (c_2, a_2)), \ldots])$$

of $\natural\,\mathbf{dig}^{\Box}_{A}$ with the element

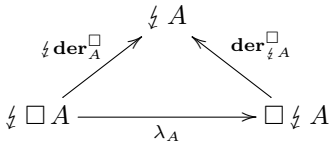
$$([(c_1, (c_2, a_1)),\, (c_1, (c_2, a_2)), \ldots], (c_1, [(c_2, a_1),\, (c_2, a_2), \ldots]))$$

of $\lambda_{\Box A}$, and we then compose the result with the element

$$((c_1, [(c_2, a_1),\, (c_2, a_2), \ldots]), (c_1, (c_2, [a_1,\, a_2, \ldots])))$$

of $\Box\lambda_A$. We deduce that $\mathbf{dig}^{\Box}_{\lightning A} \circ \lambda_A \subseteq \Box\lambda_A \circ \lambda_{\Box A} \circ \lightning \mathbf{dig}^{\Box}_A$. The converse direction is proved in the very same way.

Let us now consider the diagram



The relation $\lightning \mathbf{der}^{\Box}_A$ is precisely composed of the elements

$$([(\varepsilon, a_1), (\varepsilon, a_2), \ldots], [a_1, a_2, \ldots])$$

On the other hand, $\mathbf{der}^{\Box}_{\lightning A}$ contains the elements

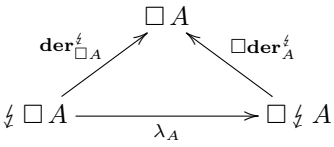
$$((\varepsilon, [a_1, a_2, \ldots]), [a_1, a_2, \ldots])$$

and precomposing this relation with λ_A precisely gives the set of elements

$$([(\varepsilon, a_1), (\varepsilon, a_2), \ldots], [a_1, a_2, \ldots])$$

so that $\lightning \mathbf{der}^{\Box}_A = \mathbf{der}^{\Box}_{\lightning A} \circ \lambda_A$.

Let us consider the last diagram:



The relation $\mathbf{der}^{\lightning}_{\Box A}$ is the set of all elements $([(c, a)], (c, a))$. On the other hand, $\Box\mathbf{der}^{\lightning}_A$ is the set of all elements $((c, [a]), (c, a))$. By precomposition with λ_A, $\mathbf{der}^{\lightning}_{\Box A} = \Box\mathbf{der}^{\lightning}_A \circ \lambda_A$.

□

A fundamental consequence of Proposition 26 is that the two comonads can be composed into a single comonad $\lightning$ defined as follows:

$$\lightning = \lightning \circ \Box$$

The resulting *infinitary* and *colored* relational semantics of linear logic is obtained from the category $\underline{Rel}$ equipped with the composite comonad $\lightning$.

Theorem 26. *The category $\underline{Rel}$ together with the comonad $\lightning$ defines a Seely category and thus a model of propositional linear logic.*

A consequence of Theorem 26 is that the Kleisli category $\underline{Rel}_{\lightning}$ is a model of the λ-calculus. The next section extends it with recursion, allowing the interpretation of the λY-calculus.

To prove Theorem 26, we may check that the diagrams defining a Seely category are indeed commutative. An other way is to observe that the canonical morphism

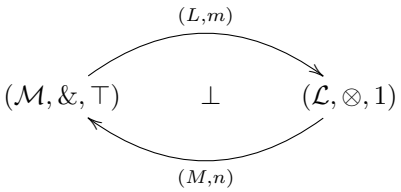
$$\begin{array}{rcl} d^{\Box}_{A,B} & : & \Box\,(A\&B) \to \Box\,A \,\&\, \Box\,B \\ & = & \{\,((c,(1,a)),(1,(c,a)))\ \mid a \in A,\, c \in Col\,\} \\ & & \cup\ \{\,((c,(2,b)),(2,(c,b)))\ \mid b \in B,\, c \in Col\,\} \end{array}$$

is obviously an isomorphism in $\underline{Rel}$, so that the theorem is a direct consequence of the following proposition:

Proposition 27. *Let $\mathcal{C}$ be a category endowed with a comonad $\not!$ turning it into a Seely category. Let $\Box$ be a comonad on $\mathcal{C}$ distributing over $\not!$ by means of a distributive law $\lambda : \not!\,\Box \to \Box\,\not!$. Suppose moreover that the canonical morphism $\Box\,(A\&B) \to \Box\,A \,\&\, \Box\,B$ is an isomorphism in the category $\mathcal{C}$. Then the composit comonad $\not! \circ \Box$ defines a Seely category over $\mathcal{C}$.*

To prove it, we use some propositions appearing in [Mel09], and relying on the notion of linear-non-linear adjunction:

Definition 38. A *linear-non-linear* adjunction is a symmetric monoidal adjunction between lax symmetric monoidal functors



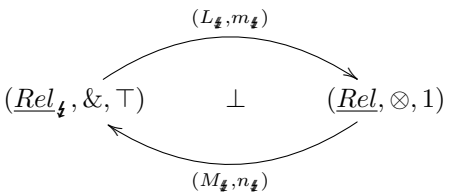
in which the category $\mathcal{M}$ is equipped with a cartesian product & and a terminal object $\top$.

The notions of symmetric monoidal adjunction and of lax symmetric monoidal functors appear in [Mel09].

Proposition 28 ([Mel09, Proposition 24])**.** *Every Seely category defines a linear-non-linear adjunction, and thus a model of intuitionistic linear logic with additives.*

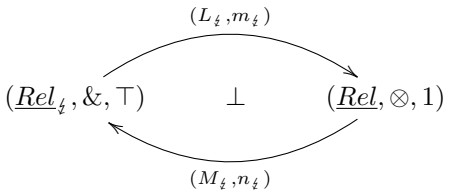
Proposition 29 ([Mel09, Proposition 25])**.** *Suppose that $\mathcal{L}$ is a cartesian and symmetric monoidal closed category, involved in a linear-non-linear adjunction. Suppose moreover that the functor $M : \mathcal{L} \to \mathcal{M}$ is bijective on objects. Then, the category $\mathcal{L}$ and the comonad $L \circ M$ define a Seely category, whose Kleisli category $\mathcal{L}_!$ is isomorphic to the category $\mathcal{M}$.*

Proof of Proposition 27. By Proposition 29, it suffices to show that the adjunction



is symmetric monoidal to obtain that $\underline{Rel}$ together with the composite comonad $\lightning = \not{\iota} \circ \Box$ is a Seely category. For this purpose, we just have to prove that $(L_{\lightning}, m_{\lightning}) : \underline{Rel}_{\lightning} \to \underline{Rel}$ is strongly monoidal.

By Proposition 23, we have that $\underline{Rel}$ together with the comonad $\not{\iota}$ is a Seely category. Proposition 28 therefore gives us a linear-non-linear adjunction



Moreover, the functor $L_{\lightning}$ factors as $L_{\not{\iota}} \circ B$, where

- $L_{\not{\iota}}$ is the strongly monoidal functor from $\underline{Rel}_{\not{\iota}}$ to $\underline{Rel}$,
- and B is the functor from $\underline{Rel}_{\lightning}$ to $\underline{Rel}_{\not{\iota}}$ which acts on morphisms as follows: it transports a morphism $g : \not{\iota}\,\Box A \to B$ to the morphism $Bg : \not{\iota}\,\Box A \to \Box B$ defined as

$$Bg \quad : \quad \not{\iota}\,\Box A \xrightarrow{\not{\iota}\,\mathbf{dig}^{\Box}_{A}} \not{\iota}\,\Box\Box A \xrightarrow{\lambda_{\Box A}} \Box\,\not{\iota}\,\Box A \xrightarrow{\Box g} \Box B$$

To prove that the adjunction between $L_{\lightning}$ and $M_{\lightning}$ is strongly monoidal, it simply remains to show that B preserves finite products. But this is the case, as by hypothesis the canonical morphism $\Box(A \& B) \to \Box A \,\&\, \Box B$ is an isomorphism.

9.9 The inductive-coinductive fixpoint operator Y

We combine the results of the previous sections in order to define a fixpoint operator $\mathbf{Y}$ over the infinitary colored relational model, which generalizes both the inductive and the coinductive fixpoint operators. Note that in this infinitary and colored framework, we wish to define a fixpoint operator $\mathbf{Y}$ which transports a binary relation

$$f \quad : \quad \lightning X \otimes \lightning A \;\multimap\; A$$

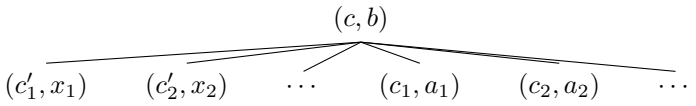
into a binary relation

$$\mathbf{Y}_{X,A}(f) \quad : \quad \lightning X \;\multimap\; A.$$

To that purpose, we adapt the definition given in §9.5 of the set **run-tree**(f, a) as follows: run-trees are defined as trees with nodes labeled by elements of the set $(Col \times X) \uplus (Col \times A)$ and such that:

- the root of the tree is labeled by (ε, a),
- the inner nodes are labeled by elements of the set $Col \times A$,
- the leaves are labeled by elements of the set $(Col \times X) \uplus (Col \times A)$,
- and for every node labeled by an element $(c, b) \in Col \times A$:

– if (c, b) labels an inner node, and letting (c_1, a_1), $(c_2, a_2), \ldots$ denote the labels of the children of this node belonging to A and (c'_1, x_1), $(c'_2, x_2), \ldots$ denote the labels belonging to X:



then $(([[(c'_1, x_1), (c'_2, x_2), \ldots], [(c_1, a_1), (c_2, a_2), \ldots]), b) \in f$

– if (c, b) labels a leaf, then $(([\,], [\,]), b) \in f$.

Note that the color of an inner node (c, b) is either ε if it is the root of the tree *witness*, or the color c it receives from its parent node.

We call the first element of the label of a node its color. We define the colored, finite-or-countable multiset of leaves $\mathbf{leaves}(\mathit{witness}) \in \lightning X$ as follows. We consider an enumeration of the leaves, which are countable by Proposition 24. We start from the empty multiset $[\,] \in \lightning X$ and, for each leaf of *witness* labeled with $(c, x) \in \mathit{Col} \times X$, we add (c', x) to $\mathbf{leaves}(\mathit{witness})$, where c' is the maximal color seen on the finite path leading from the leaf we consider to the root of *witness*.

It remains to explain the acceptance condition of this inductive-coinductive fixpoint operator. As earlier, accepting all run-trees would lead to the coinductive fixpoint, while accepting only run-trees whose branches are finite would lead to the inductive fixpoint. We define our inductive-coinductive acceptance condition for run-trees in the expected way, directly inspired by the notion of alternating parity tree automaton:

- a finite branch is accepting,
- an infinite branch is accepting precisely when the greatest color appearing infinitely often in the labels of its nodes is even.
- a run-tree is accepting precisely when all its branches are accepting.

Note that a run-tree whose nodes are all of even color will be accepted independently of its depth, as in the coinductive interpretation, while a run-tree labeled only with odd colors will be accepted precisely when it is finite, just as in the inductive interpretation.

After these modifications, we define as earlier the fixpoint operator $\mathbf{Y}$ by the equation:

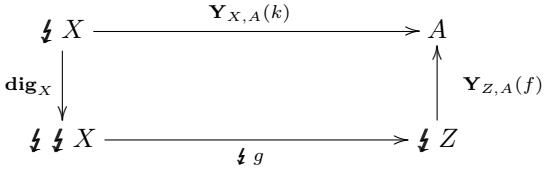
$$\mathbf{Y}_{X,A}(f) \quad = \quad \{\,(w, a)\,|\quad \exists \mathit{witness} \in \mathbf{run\text{-}tree}(f, a) \text{ with } w = \mathbf{leaves}(\mathit{witness}) \text{ and } \mathit{witness} \text{ is accepting}\,\} \tag{9.34}$$

Theorem 27. *The inductive-coinductive fixpoint operator $\mathbf{Y}$ defined over the infinitary colored relational semantics of linear logic is a Conway operator.*

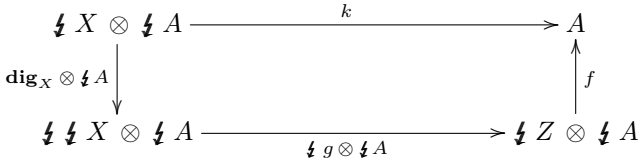
The cartesian closed category $\underline{\mathit{Rel}}_{\lightning}$ is therefore a model of the λY-calculus.

Proof. We explain how the proof of Proposition 19, extended to the infinitary semantics in the proof of Proposition 25, can be accommodated to the colored case.

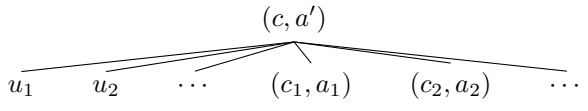
- **Naturality.** We want to prove that the diagram



commutes, where the morphism $k : !\,X \otimes !\,A \multimap A$ in the upper part of the diagram is defined as the composite



Suppose that $(u, b) \in \mathbf{Y}_{X,A}(k)$. It follows that there exists a winning run-tree $\mathcal{T}_k$ over k of root b, and of colored, finite-or-countable multiset of leaves u. This winning run-tree $\mathcal{T}_k$ is built by pasting together trees of the shape

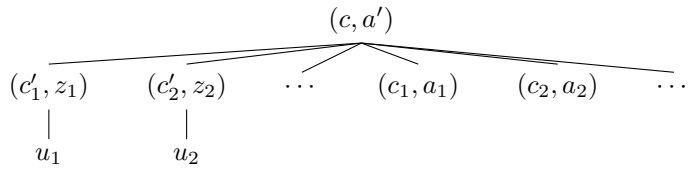


(9.35)

where $u_i \in !\,X$, $a_j \in A$ and

$$((u_1 + u_2 + \cdots, [(c_1, a_1), (c_2, a_2), \ldots]), a') \quad \in \quad k$$

Note that the multisets u_i are colored. By definition of k, each tree (9.35) can be decomposed as

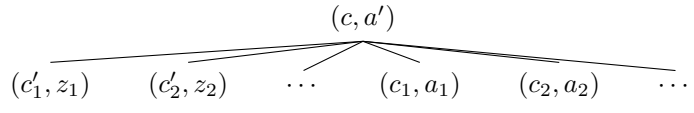


(9.36)

where

- $(([(c'_1, z_1), (c'_2, z_2), \ldots], [(c_1, a_1), (c_2, a_2), \ldots]), a') \quad \in \quad f$,
- $\forall i,\ (u_i, z_i) \in g$.

As earlier, we construct from the run-tree $\mathcal{T}_k$ over k the run-tree $\mathcal{T}_f$ over f by replacing all the trees (9.36) in $\mathcal{T}_k$ by:



(9.37)

As the trees (9.36) and (9.37) differ only on their parameters, the trees $\mathcal{T}_k$ and $\mathcal{T}_f$ have the same infinite branches: it follows that $\mathcal{T}_f$ is a winning run-tree. Let us compare the multisets $u = \mathbf{leaves}(\mathcal{T}_k)$ and $v = \mathbf{leaves}(\mathcal{T}_f)$:

- the elements of $\mathbf{leaves}(\mathcal{T}_f)$ are of the shape (d_i, z_i), where z_i occurs in $\mathcal{T}_f$ as in (9.37), and where $d_i = \max(c'_i, c, \ldots)$ is the maximal color seen on the finite path from the leaf (c'_i, z_i) to the root of $\mathcal{T}_f$,
- the elements of $\mathbf{leaves}(\mathcal{T}_k)$ are of the shape (d'_j, x_j), with $(c''_j, x_j) \in u_i$ for some u_i occurring in $\mathcal{T}_k$ as in (9.36). The color

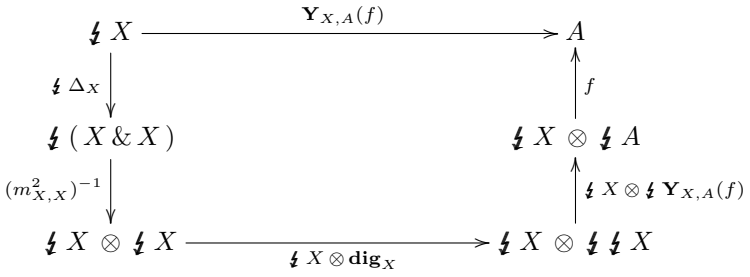
$$d'_j \quad = \quad \max(c''_j, c'_i, c, \ldots) \quad = \quad \max(c''_j, d_i)$$

  is the maximal color seen on the finite path from the leaf (c'_j, x) to the root of $\mathcal{T}_k$. It therefore factors as the maximum of the color c''_j of (c''_j, x_j) in u_i, and of the color c'_i which is the maximal color seen on the finite path from the leaf (c'_i, z_i) to the root of $\mathcal{T}_k$ (as the paths in $\mathcal{T}_k$ and in $\mathcal{T}_f$ only differ on parameter nodes).

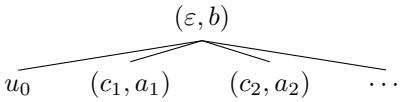
It follows from the second point that, when we compose the element (v, b) of $\mathbf{Y}_{Z,A}(f)$, justified by $\mathcal{T}_f$, with the morphism $!\,g \circ \mathbf{dig}_X$, we obtain (u, b). Indeed, the composition with $!\,g$ updates the color c''_j labeling the element (c''_j, x_j) in u_i by computing its maximum with the color d_i labeling (d_i, z_i) in v. It follows that the element x_j receives the color $d'_j = \max(c''_j, d_i)$.

The converse direction proceeds in the very same way.

- **Parametrized fixpoint property.** Let $f : \lightning\, X \otimes \lightning\, A \multimap A$. We shall prove that the following diagram commutes:



We call $k : \lightning X \to A$ the composite of the morphisms of the lower part of the diagram. Let $(u, b) \in \mathbf{Y}_{X,A}(f)$, and $\mathcal{T}$ be a winning run-tree over f of root b and such that $\mathbf{leaves}(\mathcal{T}) = u$. The run-tree $\mathcal{T}$ starts as follows:



(9.38)

where $((u_0, [(c_1, a_1), (c_2, a_2), \ldots]), b) \in f$. Each tree rooted at (c_i, a_i) induces a winning run-tree $\mathcal{T}_i$ over f, obtained simply by changing the color c_i to be ε. The run-tree $\mathcal{T}_i$ justifies that $(v_i, a_i) \in \mathbf{Y}_{X,A}(f)$. We obtain in this way the element

$$([\Box_{c_1} v_1, \Box_{c_2} v_2, \ldots], [(c_1, a_1), (c_2, a_2), \ldots]) \quad \in \lightning \mathbf{Y}_{X,A}(f)$$

where, given a multiset $v_i = [(c'_{i1}, x_{i1}), (c'_{i2}, x_{i2}), \ldots]$, the multiset $\Box_{c_i} v_i$ is defined as

$$\Box_{c_i} v_i \quad = \quad [(\max(c_i, c'_{i1}), x_{i1}), (\max(c_i, c'_{i2}), x_{i2}), \ldots]$$

It follows that

$$(u_0 + \Box_{c_1} v_1 + \Box_{c_2} v_2 + \cdots, b) \quad \in \quad k$$

Now, remark that the color c''_i of an element $(c''_i, x_i) \in v_i$ is the maximal color in $\mathcal{T}_i$ from the corresponding parameter leave to the root, so that it is also the maximal color from this leaf to the node (c_i, a_i) in $\mathcal{T}$. It follows that the element of $u = \mathbf{leaves}(\mathcal{T})$ corresponding to this leaf in $\mathcal{T}$ has color $\max(c_i, c''_i)$. This argument shows that

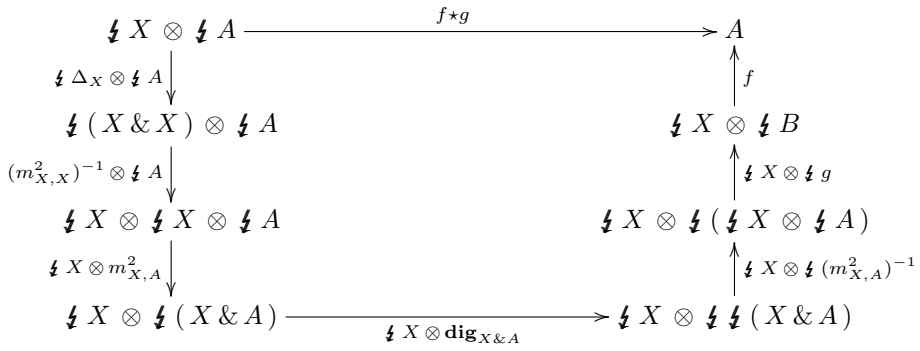
$$u \quad = \quad u_0 + \Box_{c_1} v_1 + \Box_{c_2} v_2 + \cdots$$

so that $(u, b) \in k$.

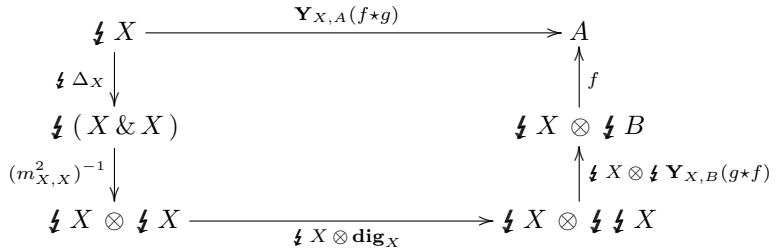
- **Parametrized dinaturality.** Again, considering

$$f : \lightning X \otimes \lightning B \multimap A \quad \text{and} \quad g : \lightning X \otimes \lightning A \multimap B$$

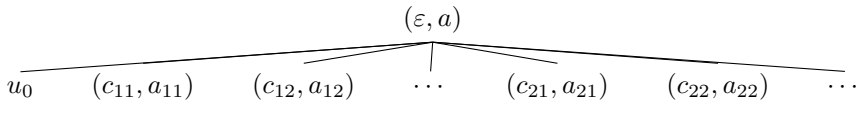
we write $f \star g : \lightning X \otimes \lightning A \multimap A$ for the composite:



and we need to prove that for any $f : \lightning X \otimes \lightning B \multimap A$ and $g : \lightning X \otimes \lightning A \multimap B$, the following diagram commutes:

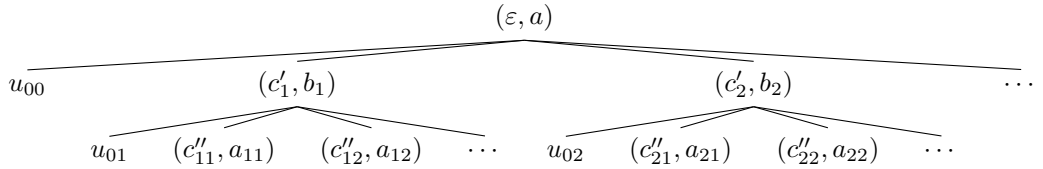


We call $k : \lightning X \to A$ the morphism obtained by composing the morphisms on the lower part of the diagram. As before, the point is to decompose a winning run-tree for $f \star g$ into a series of winning run-trees for $g \star f$, composed to the function f, and to show that the colors of parameters we obtain are the same in both cases. Let $(u, a) \in \mathbf{Y}_{X,A}(f \star g)$ and $\mathcal{T}$ be a winning run-tree of root a and such that $\mathbf{leaves}(\mathcal{T}) = u$. The tree $\mathcal{T}$ starts as:



(9.39)

with $((u_0, [(c_{11}, a_{11}), (c_{12}, a_{12}), \ldots, (c_{21}, a_{21}), (c_{22}, a_{22}), \ldots]), a) \in f \star g$. By definition of $f \star g$, we can decompose this tree (9.39) as



(9.40)

where, by definition of $f \star g$:

- $c_{ij} = \max(c'_i, c''_{ij})$,
- $u_0 = u_{00} + \Box_{c'_1} u_{01} + \Box_{c'_2} u_{02} + \cdots$,
- $\forall i,\ ((u_{0i}, [(c''_{i1}, a_{i1}), (c''_{i2}, a_{i2}), \ldots]), b_i) \in g$,
- $((u_{00}, [(c'_1, b_1), (c'_2, b_2), \ldots]), a) \in f$.

From the decomposition (9.40), we obtain a family of trees $\mathcal{T}_i$ by defining $\mathcal{T}_i$ to be the subtree rooted at (c'_i, b_i) and in which we replace the color c'_i of the root by ε, leaving all other colors unchanged. Each tree $\mathcal{T}_i$ is a winning run-tree over $g \star f$, justifying that $(v_i, b_i) \in \mathbf{Y}_{X,B}(g \star f)$. By composition with f, we get that

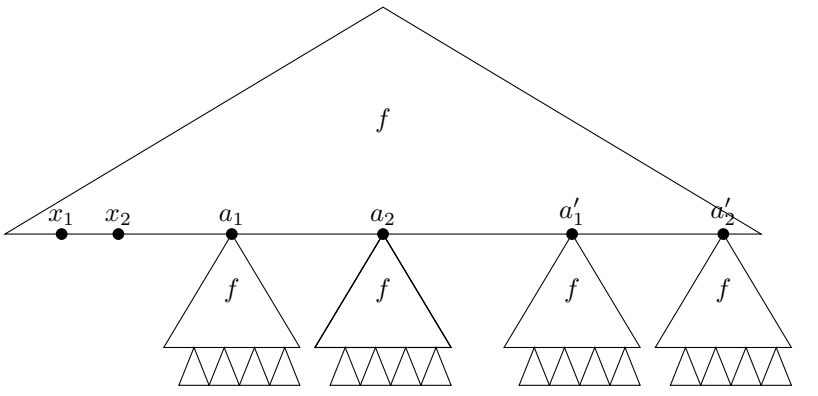
$$\left(u_{00} + \Box_{c'_1} v_1 + \Box_{c'_2} v_2 + \cdots, a\right) \in k$$

It remains to prove that $u = u_{00} + \Box_{c'_1} v_1 + \Box_{c'_2} v_2$. The argument is exactly as at the end of the proof of the parametrized fixpoint property. The idea is essentially that the color of a parameter x in $\mathbf{leaves}(\mathcal{T})$ is computed by considering the path from the leaf of interest to the root of (9.40), while in $\mathbf{leaves}(\mathcal{T}_i)$ it is computed by considering the path from the leaf to the node labeled with (c'_i, b_i) of (9.40). But the application of f in the second case updates the color labeling x by computing its maximum with c'_i, so that we obtain the same colors for parameters in k and in $\mathbf{Y}_{X,A}(f \star g)$.

The converse direction is proved in the same way.

- **Diagonal property.** Let $f : \lightning X \otimes \lightning A \otimes \lightning A \multimap A$ be a morphism in $\underline{Rel}$. As explained on p.209, the diagonal property expresses the fact that every winning run-tree obtained by iterating on both copies of $\lightning A$ at the same time can be obtained by plugging together run-trees in which only the second copy is iterated, and conversely.

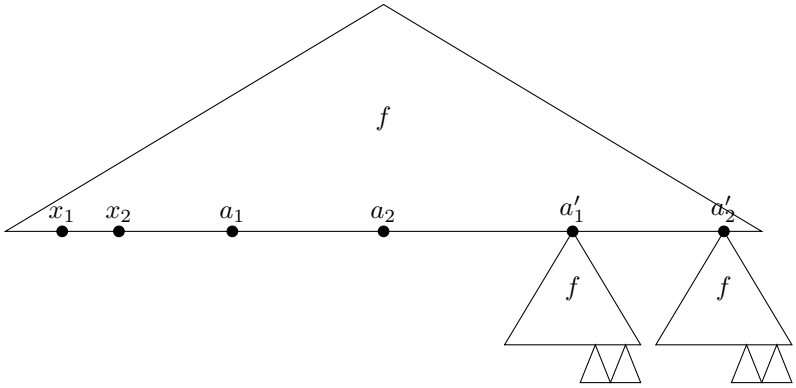
Consider a run-tree computed "in only one step", by composing f with the diagonal morphism on its component $\lightning A \otimes \lightning A$:



(9.41)

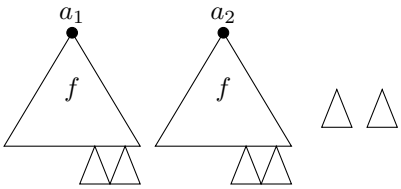
We can decompose it as run-trees where the fixpoint is only taken on the second copy of $\lightning A$, which we then plug together using the fixpoint

operator another time. In a first step, we obtain trees as



(9.42)

and as



These run-trees are all winning, as their infinite branches are included in the ones of (9.41). Moreover, on each finite branch leading to an element $a_i \in A$ treated as a parameter in this first step, the construction collects the maximal color seen on the finite path. So, the second execution step of the fixpoint, which iterates the construction of the run-trees on the copy of $\lightning A$ which was treated as a parameter by the first execution of the fixpoint, builds infinite branches coming from infinite branches of (9.41) and having the same color (in the infinitary sense given earlier in this document: the color of an infinite branch is the maximal color seen infinitely often along it). It follows that every winning run-tree (9.41) induces a winning run-tree with the same root and the same colored multiset of leaves, and obtained by this two-step construction. Conversely, every run-tree obtained by the two-step construction can be built in one step, and this converse construction builds a winning run-tree from a winning run-tree. The diagonal property follows.

□

9.10 Relational semantics of linear logic and higher-model model-checking

In §5.3, we briefly explained how the relational semantics of linear logic could be related to a system of non-idempotent intersection types using an *indexed* variant of linear logic due to Bucciarelli and Ehrhard [BE00, BE01]. The idea, as we discuss in the next section, is to extend this theoretic bridge to connect

a non-idempotent variant of $\mathbf{\gamma}_{fix}(\mathcal{G},\mathcal{A})$, where intersection types have finite-or-countable multiplicities, with the cartesian closed category $\underline{Rel}_{\lightning}$. We denote $[\![t]\!]_{\lightning}$ the relational interpretation of a λY-term t in this infinitary, colored model. The interpretation $[\![\delta]\!]_{\lightning}$ of the transition function δ is defined similarly as we did in the relational semantics for $[\![\delta]\!]$ in §9.2, as a subset

$$[\![\delta]\!] \quad \subseteq \quad \left[\!\!\left[\mathop{\&}_{i\in I} \left(\underbrace{\lightning\, o \multimap \cdots \multimap \lightning\, o}_{ar_i} \multimap o \right) \right]\!\!\right]$$

which may be "strengthened" in the categorical sense as a subset

$$[\![\delta^\dagger]\!] \quad \subseteq \quad \left[\!\!\left[\bigotimes_{i\in I} \lightning \left(\underbrace{\lightning\, o \multimap \cdots \multimap \lightning\, o}_{ar_i} \multimap o \right) \right]\!\!\right]$$

Note that the color information is incorporated in the semantics following the comonadic principles disclosed in the shift from the type system $KO(\mathcal{A})$ to $\mathbf{\gamma}(\mathcal{A})$: typically, the transition

$$\delta(q_0,\, \texttt{if}) \quad = \quad (2,q_0) \wedge (2,q_1)$$

is interpreted in the colored relational semantics as

$$(d, ([\,], ([[(\Omega(q_0),q_0),(\Omega(q_1),q_1)],q_0))) \quad \in \quad [\![\delta]\!]_{\lightning}$$

where d selects the component of the cartesian product corresponding to `if` in δ.

As explained in §6.5, we conjecture that the proof of soundness and completeness of Theorem 19 can be smoothly adapted to a non-idempotent intersection type system, in which countable multiplicities may appear. We also conjecture, as explained in the next section, that this type system can be related – using an indexed, colored and infinitary variant of tensorial logic – to the infinitary, colored model of the λY-calculus $\underline{Rel}_{\lightning}$.

This leads us to the following conjecture, which underlies the developments of the previous sections, and would establish a clean correspondence between the relational semantics of a higher-order recursion scheme $\mathcal{G}$ (seen below as a λY-term $t_{\mathcal{G}}$, thanks to the correspondence provided by Proposition 5) and the exploration of the associated ranked tree $\langle\, \mathcal{G} \,\rangle$ by an alternating *parity* automaton $\mathcal{A}$:

Conjecture 1. *An alternating* parity *tree automaton $\mathcal{A}$ with a set of states Q has a* winning *run-tree with initial state q_0 over the ranked tree $\langle\, \mathcal{G} \,\rangle$ generated by the λY-term $t_{\mathcal{G}}$ if and only if there exists $u \in [\![\delta^\dagger]\!]_{\lightning}$ such that $(u,q_0) \in [\![t_{\mathcal{G}}]\!]_{\lightning}$, where $[\![\delta^\dagger]\!]_{\lightning} = \mathcal{M}_{count}(Col \times [\![\delta]\!]_{\lightning})$ denotes the set of finite-or-countable colored multisets of elements of $[\![\delta]\!]_{\lightning}$.*

9.11 An indexed tensorial logic with colors

In this section, we introduce a colored and indexed extension of tensorial logic [MT10], with the aim that it can serve as a bridge between the infinitary,

colored relational semantics defined in this chapter, and a non-idempotent variant of the type system $\curlyvee_{fix}(\mathcal{G}, \mathcal{A})$.

Note that the interest of colored tensorial logic ranges far beyond this utilitarian perspective, as it opens natural connections to *game models*. Tensorial logic is indeed deeply rooted in Melliès' analysis of asynchronous games [Mel06a, Mel05], which started from the will to find a fully complete model of propositional linear logic. Melliès realized that the apparent schism between linear logic and game semantics could be addressed by identifying strategies which reach the same positions of the game, but using different *sequentializations* of *parallel* events. For instance, game models distinguish two strategies for computing the addition $(x, y) \mapsto x + y$, one getting the value of x first while the other asks for y before. On the other hand, describing addition as a proof of linear logic gives a unique object in which the queries for x and y are parallelized. This parallelization is implemented in asynchronous games, leading to a fully complete model of linear logic [Mel05].

Tensorial logic [MT10] comes from the will to unify game semantics, linear logic, and continuations, with the aim to provide a convenient logical framework for the study of coeffects – an example of such coeffect being our coloring modality. The idea is to follow at the logical level the converse of the path that lead from "traditional" game semantics to asynchronous games, and to introduce sequentialization in the logic. Tensorial logic is therefore obtained from linear logic by relaxing the hypothesis that negation should be *involutive*; negation plays the rôle of the exchange of players in game semantics, so that a double negation $\neg\neg A$ of the formula A is understood as the succession of a player and an opponent move on the arena corresponding to the formula A. In this perspective, $\neg\neg A$ should not be collapsed to A by the traditional involution hypothesis. Tensorial logic notably distinguishes the left and right implementations of addition, just as game models do, and allows more generally to reconcile logic with game semantics [Mel12].

A very interesting point in relation to our approach is that the relational model of linear logic is a natural model of tensorial logic, in spite of its degeneracy – in the sense that it identifies A and $\neg\neg A$. The derivations of tensorial logic can be at the same time understood as strategies over the arena defined by the formula of interest, and as the computation of denotations of this formula in the relational model. Studying *colored* tensorial logic is certainly the first step towards such a bridge between our colored, infinitary relational semantics, and a game model with infinite colored interactions, discriminated by the parity condition. This connection, unfortunately, is beyond the scope of this thesis.

Before we define the colored extension of tensorial logic, let us stress once again that the notation $\boxdot_m \theta$ is used in our intersection type system $\curlyvee_{fix}(\mathcal{G}, \mathcal{A})$ as a way to stress the *modal* nature of colors, and that it replaces for the better the notation (θ, m) used by Kobayashi and Ong in [KO09]. As we explained in §6.5, the discovery of the modal nature of colors is fundamental, and is not just a matter of using the appropriate notation. In particular, it enables us to simplify both technically and conceptually the original intersection type system in [KO09]. By way of illustration, the original intersection type (6.1) of the

Kobayashi-Ong intersection type system:

$$a \quad : \quad (q_1, m_1) \to (q_2, m_2) \to q$$

where $m_i = \max(\Omega(q), \Omega(q_i))$, is replaced by the simpler intersection type:

$$a \quad : \quad \boxdot_{n_1} q_1 \to \boxdot_{n_2} q_2 \to q \tag{9.43}$$

where $n_1 = \Omega(q_1)$ and $n_2 = \Omega(q_2)$. Interestingly, the color of the state q is not mentioned in the type anymore. The reason is that this alternative account of colors achieved in our type system is not just "simpler" than the original one: it also reveals a deep and somewhat unexpected connection with linear logic, since as we will see, this "disparition" of the color $\Omega(q)$ in (9.43) is related to the well-known linear decomposition $A \Rightarrow B = !A \multimap B$ of the intuitionistic implication in linear logic. One essential difference however is that the exponential modality « ! » of linear logic is replaced by a family of modal boxes $\Omega(m)$ which formally defines what Melliès calls a *parametric comonad* in [Mel14b] [Mel06b].

This key observation enables us to look for a translation of the intersection type system $\gamma_{fix}(\mathcal{G}, \mathcal{A})$ into an infinitary variant of linear logic equipped with a family of color modalities noted $\square_m$ for $m \in \mathbb{N}$. A nice feature of the translation we sketch is that it transports the intersection type system $\gamma_{fix}(\mathcal{G}, \mathcal{A})$ which depends on $\mathcal{G}$ and $\mathcal{A}$ into an intersection type system which does not depend on them anymore — although it still depends on the set Q of states of the automaton. In fact, the translation performs an on-the-fly Church encoding, in the spirit of §9.1. The infinitary variant of linear logic which we consider for defining such a translation is

- *indexed* in the sense of Bucciarelli and Ehrhard [BE00, BE01]. In particular, the finite or countable intersection types $\bigwedge_{i \in I} \theta_i$ of $\gamma_{fix}(\mathcal{G}, \mathcal{A})$ are translated as finite or countable indexed families $[\theta_i \mid i \in I]$ of formulas of the logic,
- *tensorial* in the sense of Melliès [MT10, Mel12, Mel14b]. In this specific case, every negated formula of the logic is negated with respect to a specific state $q \in Q$ of the automaton, and is thus of the form $\sigma \multimap q$, which may be alternatively written as $\neg_q \sigma$ or even as $\overset{q}{\neg} \sigma$.

In this way, one obtains an indexed and colored variant of tensorial logic, called $\mathrm{LT}(Q)$ in the sequel, and whose formulas are inductively generated by the following grammar:

$$A, B \quad ::= \quad 1 \quad | \quad A \otimes B \quad | \quad \neg_q A \quad | \quad \square_m A \quad | \quad [\, A_j \mid j \in J \,] \qquad (m \in \mathit{Col},\ q \in Q)$$

As already mentioned, following the philosophy in [BE01], the finite or countable indexed set $[\sigma_j \mid j \in J]$ internalizes the intersection operator of $\gamma_{fix}(\mathcal{G}, \mathcal{A})$ in our indexed tensorial logic, see [GM15b] for details in the finite case. Importantly, the resulting indexed logic $TL(Q)$ can be used as an intersection type system refining the simply-typed λ-calculus in just the same way as $\gamma_{fix}(\mathcal{G}, \mathcal{A})$. The main logical rules of the system are formulated below in Figure 9.3. In order to simplify their presentation, we choose to write

$$\neg_q (A_1, \cdots, A_n)$$

$$\text{Axiom}\quad \frac{q \in Q}{x : q :: o \vdash x : q :: o}$$

$$\text{Left } \square\quad \frac{\Gamma, x : A :: \kappa \vdash M : B :: \kappa'}{\Gamma, x : \square_\varepsilon A :: \kappa \vdash M : B :: \kappa'} \qquad \frac{\Gamma \vdash M : A :: \kappa}{\square_m \Gamma \vdash M : \square_m A :: \kappa}\quad \text{Right } \square_m$$

$$\text{Dereliction}\quad \frac{\Gamma, x : A :: \kappa \vdash M : B :: \kappa'}{\Gamma, x : [A] :: \kappa \vdash M : B :: \kappa'} \qquad \frac{\Gamma_j \vdash M : A_j :: \kappa \quad (\forall j \in J)}{\sum_{j\in J} \Gamma_j \vdash M : [A_j \,|\, j \in J] :: \kappa}\quad \text{Promotion}$$

$$\text{Left negation}\quad \frac{\Gamma_1 \vdash N_1 : A_1 :: \kappa_1 \quad \cdots \quad \Gamma_n \vdash N_n : A_n :: \kappa_n \qquad q \in Q}{(\sum_{i=1}^n \Gamma_i), f : \neg_q (A_1, \cdots, A_n) :: \neg (\kappa_1, \ldots, \kappa_n) \vdash f\ N_1 \cdots N_n : q :: o}$$

$$\text{Right negation}\quad \frac{\Gamma, x_1 : A_1 :: \kappa_1, \ldots, x_n : A_n :: \kappa_n \vdash M : q :: o \qquad q \in Q}{\Gamma \vdash \lambda x_1 \cdots \lambda x_n . M : \neg_q (A_1, \ldots, A_n) :: \neg (\kappa_1, \ldots, \kappa_n)}$$

Figure 9.3: Extension of tensorial logic with intersection types and color modalities (main rules)

for the formula of indexed tensorial logic :

$$\neg_q (A_1 \otimes \cdots \otimes A_n) \quad = \quad A_1 \multimap \cdots \multimap A_n \multimap q.$$

Similarly, and for the sake of uniformity, we choose to write

$$\neg (\kappa_1, \cdots, \kappa_n)$$

for the type (or kind) of simply-typed λ-calculus:

$$\kappa_1 \to \cdots \to \kappa_n \to o.$$

Note that, for the sake of simplicity, we prefer to keep implicit the index I, J or K appearing on the side of each sequent of the logical rules below. The reader interested in the precise treatment of such indexes will find the detailed treatment in the work by Bucciarelli and Ehrhard [BE00, BE01] as well as in our paper with Melliès [GM15b].

In particular, as we pointed out in §6.5, the fact that $\square$ defines a parametric monoidal comonad in the logic means that the sequents

$$\begin{array}{ccc} \square_\varepsilon A & \vdash & A \\ \square_{\max(m_1, m_2)} A & \vdash & \square_{m_1} \square_{m_2} A \\ \square_m A \otimes \square_m B & \vdash & \square_m (A \otimes B) \end{array}$$

are provable for all colors $m, m_1, m_2 \in \mathbb{N}$, and all formulas A, B. In order to deal with recursion schemes, we admit derivation trees with finite or countable depth in the logical system $TL(Q)$. The nodes of the derivation trees of $TL(Q)$ are then colored in the following way:

- every node $\Gamma \vdash M \,:\, A \,::\, \kappa$ in a Right introduction of the modality $\Box_m$:

$$\frac{\Gamma \vdash M \,:\, A \,::\, \kappa}{\Box_m \Gamma \vdash M \,:\, \Box_m A \,::\, \kappa} \quad \text{Right } \Box_m$$

  is assigned the color m of the modality,

- all the other nodes of the derivation tree are assigned the neutral color ε.

The winning condition on an infinite derivation tree of $TL(Q)$ is then directly adapted from the similar condition in $\gamma_{fix}(\mathcal{G}, \mathcal{A})$. Thanks to this condition, we are ready to state a useful correspondence theorem between $\gamma_{fix}(\mathcal{G}, \mathcal{A})$ and $TL(Q)$ for any (productive) recursion scheme $\mathcal{G}$. We believe that this correspondence would be of help to prove Conjecture 1: as the relational model is a model of tensorial logic, we believe that the infinitary, colored relational model defined in the previous sections is a model of colored tensorial logic, so that a correspondence between $\gamma_{fix}(\mathcal{G}, \mathcal{A})$ and $TL(Q)$ could be an important step towards the relational reformulation of Theorem 19 that Conjecture 1 is.

We now explain the translation. Suppose that for each $F \in \mathcal{N}$ of kind $\kappa(F)$ of the recursion scheme $\mathcal{G}$, we introduce a new free variable $freeze(F)$ of kind $\kappa(F) \to \kappa(F)$; that we replace each λ-term $\mathcal{R}(F)$ by its $\beta\eta$-long normal form ; and finally, that we substitute each occurrence of F appearing in any $\beta\eta$-long normal form $\mathcal{R}(G)$ of the recursion scheme $\mathcal{G}$ with the λ-term $freeze(F)\ F$ of the same kind $\kappa(F)$. This transformation induces a context-free grammar of « blocks» consisting of the $\beta\eta$-long $\mathcal{R}(G)$'s, which generates an infinite λ-term in $\beta\eta$-long normal form, noted $term(\mathcal{G})$, with free variables of the form $freeze(F)$. Moreover, this infinite λ-term $term(\mathcal{G})$ is coinductively typed in the simply-typed λ-calculus by the typing judgment:

$$\ldots \quad , \quad freeze(F) : \kappa(F) \to \kappa(F) \quad , \quad \ldots \quad \vdash \quad term(\mathcal{G}) \quad : \quad o \qquad (9.44)$$

where F runs over all the non-terminals $F \in \mathcal{N}$ of the higher-order recursion scheme $\mathcal{G}$. Notice that the reason why we need to consider terms in β-normal form is that $TL(Q)$ does not contain a Cut rule, and we need η-long forms as the Axiom rule is only defined for variables of ground type. At this point, we are ready to recast our Theorem 19 in the proof-theoretic language of indexed tensorial logic:

Proposition 30. *There exists a winning derivation tree in $\gamma_{fix}(\mathcal{G}, \mathcal{A})$ of the sequent*

$$S \,:\, \boxdot_\varepsilon\ q_0 \,::\, o \ \vdash\ S \,:\, q_0 \,::\, o \qquad (9.45)$$

if and only if there exists a winning derivation tree in $TL(Q)$ of a sequent

$$\Gamma \ \vdash \ \ term(\mathcal{G}) \,:\, q_0 \,::\, o \qquad (9.46)$$

refining the typing judgment (9.44).

In order to prove Conjecture 1, it would remain to check that the colored, infinitary relational semantics of linear logic we defined are precisely capture by colored tensorial logic: one should check that

$$\Gamma \ \vdash \ \ term(\mathcal{G}) \,:\, q_0 \,::\, o$$

has a winning derivation tree in $TL(Q)$ if and only if the interpretation of the sequent $\Gamma \vdash term(\mathcal{G}) : q_0 :: o$ in the infinitary, colored relational model contains q_0:

$$q_0 \in [\![\Gamma \vdash term(\mathcal{G}) :: o]\!]_{\not\downarrow}$$

We prove a similar connection for a finitary colored model of linear logic and an associated intersection type system in §10.2.

Chapter 10

Finitary semantics and decidability of higher-order model-checking

In this chapter, we explain how the connection between higher-order model-checking and linear logic exhibited in the previous chapters leads to a new and conceptually enlightening proof of the decidability of the local higher-order model-checking problem, and of the selection problem originally established by Carayol and Serre using collapsible pushdown automata.

The main idea is to start from the infinitary and colored relational semantics of the λY-calculus we formulated in Chapter 9, and to replace it by its finitary counterpart based on finite prime-algebraic lattices. As explained in Chapter 5, this shift to the finitary Scott semantics is strongly suggested by a theorem of Ehrhard proving that it is the extensional collapse of the relational model [Ehr12b].

We therefore extend this finitary model with a coloring modality and a parity fixpoint operator, both adapted from the analogous constructions of Chapter 9. We prove that the denotations in this model can be computed type-theoretically, and that the resulting type system can be related to $\boldsymbol{\gamma}^{st}_{fix}(\mathcal{G}, \mathcal{A})$, allowing us to prove a semantic version of the soundness-and-completeness theorem of Chapter 6.

As a consequence, the interpretation $[\![\mathcal{G}]\!]_{\mathcal{A}}$ of a higher-order recursion scheme $\mathcal{G}$ with respect to an alternating parity automaton $\mathcal{A}$ in this finitary semantics is precisely the set of states from which $\mathcal{A}$ accepts $\langle\, \mathcal{G} \,\rangle$. The decidability of the local model-checking problem then follows from the finiteness of the semantics, which also allows us to solve the selection problem.

10.1 The colored Scott semantics of linear logic

As we have shown in the previous chapters, the treatment of colors by alternating parity automata follows essentially the same comonadic principles as the treatment of copies in linear logic. This connection between higher-order model checking and linear logic leads to a coloring monoidal comonad $\Box$ on the relational semantics of linear logic, which we adapt here to the qualitative Scott semantics introduced in §5.4. To that purpose, we fix a finite set of colors Col containing a neutral element ε, and consider the coloring function $Q \to Col$ which associates a color to every state of a parity tree automaton $\mathcal{A}$.

The modality $\Box$ is then defined in the following way for an ordered set $(A, \leq_A)$ and a morphism $R : (A, \leq_A) \to (B, \leq_B)$:

$$\begin{array}{rcll} \Box\,(A,\leq_A) & = & (A,\leq_A)\,\&\,\cdots\,\&\,(A,\leq_A) & \\ & \cong & (\{(c,a) \mid c \in Col,\, a \in A\},\leq_{\Box A}) & \\ (c_1,a)\ \Box R\ (c_2,b) & \text{iff} & c_1 = c_2 \text{ and } a\,R\,b & \end{array}$$

where $(c_1, a) \leq_{\Box A} (c_2, a')$ iff $c_1 = c_2$ and $a \leq_A a'$. The comonadic structure of $\Box$ is provided by the following structural morphisms

$$\begin{array}{rcll} \mathbf{dig}^{\Box}_A & = & \{((\max(c_1,c_2),a),(c_1,(c_2,a'))) \mid a' \leq_A a\} & : \quad \Box A \to \Box\Box A \\ \mathbf{der}^{\Box}_A & = & \{((\varepsilon,a),a') \mid a' \leq_A a\} & : \quad \Box A \to A \\ m^{\Box}_{A,B} & = & \{(((c,a),(c,b)),((c,(a',b')))) \mid a' \leq_A a,\, b' \leq_B b\} & : \quad \Box A \otimes \Box B \to \Box(A \otimes B) \\ m^{\Box}_1 & = & \{\,(\star,(c,\star)) \mid c \in Col\,\} & : \quad 1 \to \Box 1 \end{array}$$

As we did in the case of the relational semantics in §9.8, we define a distributive law $\lambda : ! \circ \Box \Rightarrow \Box \circ !$ between the comonads ! and $\Box$ defined as the natural transformation:

$$\lambda_A \;=\; \left\{\left(\left\{\left(c_j,\, a'_j\right)\right\},(c,\{a_i\})\right) \mid \forall i\, \exists j \;\; c = c_j \text{ and } a_i \leq_A a'_j \right\} \;: !\Box A \to \Box ! A$$

The existence of such a distributive law λ enables us to equip the composite functor $\lightning = ! \circ \Box$ with a comonadic structure. Unless the comonad we consider is clear from context, we denote $\mathbf{dig}^{!}_A$, $\mathbf{der}^{!}_A$,... the structural morphisms of the comonad !, and $\mathbf{dig}^{\lightning}_A$, $\mathbf{der}^{\lightning}_A$,... the ones of $\lightning$.

Proposition 31. *The colored exponential functor $\lightning$ satisfies the axioms of a Seely category, and thus defines a model of full propositional linear logic.*

We denote by $\mathbf{ScottL}_{\lightning}$ its Kleisli category. To ease the description of the structural morphisms of the comonad $\lightning$, we introduce the operation $\Box_c\, v$ which maps a colored set $v = \{(c_i, \alpha_i) \mid i \in I\} \in \lightning A$ to the colored set

$$\Box_c\, v \;=\; \{(\max(c, c_i), \alpha_i) \mid i \in I\} \in \lightning A$$

Lemma 19 (Structural morphisms of $\lightning$)**.** *The counit $\mathbf{der}^{\lightning}$ of $\lightning$ is given, on an object A, by the downward-closed relation*

$$\{(\{(c_i,a_i) \mid i \in I\},a') \mid \exists i \in I \text{ s.t. } c_i = \varepsilon \text{ and } a' \leq_A a_i\} \;:\; !\Box A \to A$$

Its comultiplication $\mathbf{dig}^{\lightning}_A$ on A is the downward-closed relation $!\Box A \to !\Box!\Box A$ which relates every set

$$\{(c_i, a_i) \mid i \in I\} \;\in\; !\Box A$$

with every set

$$\{(d_j, \{(e_{kj}, b_{kj}) \mid k \in K_j\}) \mid j \in J\} \;\in\; !\Box!\Box A$$

such that for every $j \in J$ and $k \in K_j$ there exists $i \in I$ with

$$b_{kj} \leq a_i \quad \text{and} \quad c_i \;=\; \max(d_j, e_{kj})$$

In other terms,

$$\mathbf{dig}^{\lightning}_A \;=\; \{(u, \{(c_1,v_1),\ldots,(c_n,v_n)\}) \mid \Box_{c_1} v_1 \cup \cdots \cup \Box_{c_n} v_n \leq_{\lightning A} u\}$$

Proof. The counit is obtained simply by composition of the counits of ! and □. The comultiplication is obtained using the distributive law, as the composite:

$$\lightning A \;=\; !\Box A \xrightarrow{\mathbf{dig}^{!}_{\Box A}} !!\Box A \xrightarrow{!!\mathbf{dig}^{\Box}_{A}} !!\Box\Box A \xrightarrow{!\lambda_{\Box A}} !\Box!\Box A \;=\; \lightning\lightning A$$

□

10.2 A finitary interpretation of the simply-typed λ-calculus

In order to simplify the discussion, we suppose given an alternating parity tree $\mathcal{A}$ over a signature Σ, with set of states Q and with transition function δ. As a Kleisli category associated to a model of linear logic, the category $\mathbf{ScottL}_{\lightning}$ is cartesian closed and thus a model of the simply-typed λ-calculus. The simple types are interpreted inductively as

$$[\![\sigma \to \tau]\!]_{\lightning} \;=\; \lightning [\![\sigma]\!]_{\lightning} \multimap [\![\tau]\!]_{\lightning} \quad \text{and} \quad [\![o]\!]_{\lightning} \;=\; \perp\!\!\!\perp \;=\; (Q, =)$$

The interpretation of the simply-typed λ-terms is standard, except for the interpretation of the elements of the ranked alphabet Σ, seen here as constants of the simply-typed λ-calculus. The idea is similar to the case of the relational semantics, except that we want the relation interpreting a constant $a \in \Sigma$ to be downward-closed. In this goal, we introduce the following definition:

Definition 39. Given a state $q \in Q$ and an n-ary constructor $a \in \Sigma$, we say that an n-tuple $\alpha \in (\mathcal{P}_{fin}(Col \times Q))^n$ *validates* the formula $\delta(q, a)$ when α is of the form

$$\alpha \;=\; (\,\{\,(c_{1i_1}, q_{1i_1}) \mid i_1 \in I_1\,\}\;,\;\ldots\;,\;\{\,(c_{ni_n}, q_{ni_n}) \mid i_n \in I_n\,\}\,)$$

and there exists an n-tuple of subsets $J_1 \subseteq I_1, \ldots, J_n \subseteq I_n$ such that

$$\bigwedge_{k=1}^{n} \bigvee_{j_k \in J_k} (k, q_{kj_k}) \tag{10.1}$$

defines a conjunctive clause of the formula $\delta(q, a)$, and such that moreover

$$\forall k \in \{1, \ldots, n\} \;\; \forall j \in J_k \;\; c_{kj} \;=\; \Omega(q_{kj}).$$

When $J_1 = I_1, \ldots, J_n = I_n$, we say that α *satisfies* $\delta(q, a)$, as this corresponds to a minor alteration of the notion of satisfaction of a boolean formula and of a transition function already given for alternating parity automata on p 47.

In other words, α is a n-tuple of sets $\{(c_{1i_k}, q_{1i_k}) \mid i_k \in I_k\}$ of states annotated with colors, each of them corresponding to one of the n subtrees below the symbol a. Moreover, each such set should contain a subset $\{(\Omega(q_{ki_k}), q_{ki_k}) \mid i_k \in J_k\}$ of appropriately colored states, such that (10.1) defines a conjunctive clause of the formula $\delta(q, a)$. The general idea is that the n-tuple is allowed to contain more colored states than what is strictly required for the transition $\delta(q, a)$ to be performed by the alternating parity automaton $\mathcal{A}$.

The constants of the signature Σ are then interpreted as follows:

$$[\![a]\!]_{\mathcal{A}} \quad = \quad \{ (\alpha, q) \mid q \in Q \text{ and } \alpha \text{ validates the formula } \delta(q, a) \}$$

As explained in §9.1 in the case of the quantitative relational semantics of linear logic, this interpretation of the elements of Σ corresponds to a Church encoding of the alternating parity automaton $\mathcal{A}$, encoded in the present case in the qualitative Scott semantics of linear logic.

Example 10. Consider the two transitions:

$$\delta(q_0, \texttt{if}) = (2, q_0) \wedge (2, q_1) \qquad\qquad \delta(q_1, \texttt{if}) = (1, q_1) \wedge (2, q_0)$$

Setting $c_i = \Omega(q_i)$, these transitions imply that

$$(u_1, u_2, q_0) \in [\![if]\!]_{\mathcal{A}} \quad \text{ and } \quad (v_1, v_2, q_1) \in [\![if]\!]_{\mathcal{A}}$$

for all finite sets $u_1, u_2, v_1, v_2 \in \mathbb{1} \!\!\perp \!\!\perp = \mathcal{P}_{fin}(Col \times Q)$ satisfying moreover that $\{(c_0, q_0), (c_1, q_1)\} \subseteq u_2$, that $(c_1, q_1) \in v_1$ and that $(c_0, q_0) \in v_2$.

Using these interpretations in **ScottL** of the elements of the ranked alphabet Σ, we construct the interpretation

$$[\![\Gamma \vdash t :: \tau]\!]_{\mathcal{A}} \quad \subseteq \quad (\mathbb{1} [\![\sigma_1]\!]_{\mathbb{1}} \otimes \cdots \otimes \mathbb{1} [\![\sigma_n]\!]_{\mathbb{1}}) \multimap [\![\tau]\!]_{\mathbb{1}}$$

of any λ-term t of type τ in a context of typed variables Γ, with constants in the ranked alphabet Σ. An alternative way to describe this interpretation is to express it as an intersection type system with subtyping, in the style of Coppo, Dezani, Honsell and Longo [CDHL84] and more recently Terui [Ter12] and Ehrhard [Ehr12a] in the framework of linear logic. Note that Terui and Ehrhard present their type system with additively managed contexts, while the incorporation of the coloring modality in our system requires a multiplicative management, as it will be clear from the Application rule. In this type-theoretic formulation, sequents are of the following form:

$$\Gamma \quad = \quad x_1 : u_1 :: \sigma_1, \ldots, x_n : u_n :: \sigma_n \quad \vdash \quad t : \alpha :: \tau$$

where $u_i \in \mathbb{1} [\![\sigma_i]\!]_{\mathbb{1}}$ and $\alpha \in [\![\tau]\!]_{\mathbb{1}}$. The typing rules are presented in Figure 10.2, with the subtyping relation $\leq_A$ defined inductively in Figure 10.1. Note that the coloring $\Box_c \, \Gamma$ of a context is defined inductively as

$$\begin{array}{rcl} \Box_c \, (x : u :: \sigma, \Gamma) & = & x : \Box_c \, u :: \sigma, \, \Box_c \, \Gamma \\ \Box_c \, \{ (c_i, \alpha_i) \} & = & \{ (\max(c, c_i), \alpha_i) \} \end{array}$$

It is a syntactic reflection of semantic properties of the comonad $\Box$ and of the distributive law λ: the monoidality justifies the first equality, while the second one proceeds from the distributive law, followed by the comultiplication of $\Box$.

Proposition 32. *The sequent*

$$\Gamma \; = \; x_1 : u_1 :: \sigma_1, \ldots, x_n : u_n :: \sigma_n \; \vdash \; t : \alpha :: \tau$$

is provable in this intersection type system if and only if

$$(u_1, \ldots, u_n, \alpha) \; \in \; [\![\Gamma \vdash t :: \tau]\!]_{\mathcal{A}} \; \subseteq \; (\mathbb{1} [\![\sigma_1]\!]_{\mathbb{1}} \otimes \cdots \otimes \mathbb{1} [\![\sigma_n]\!]_{\mathbb{1}}) \multimap [\![\tau]\!]_{\mathbb{1}}$$

$$\frac{}{q \leq_{\perp\!\!\!\perp} q} \qquad \frac{\forall\,(c,\,\alpha) \in u \;\; \exists\,(c,\,\beta) \in v \;\; \alpha \leq_A \beta}{u \leq_{\not\iota A} v} \qquad \frac{v \leq_{\not\iota A} u \qquad \alpha \leq_B \beta}{u \to \alpha \leq_{\not\iota A \multimap B} v \to \beta}$$

Figure 10.1: Inference rules for the preorders associated with simple types.

$$\text{Ax}\quad \frac{\exists\,(\varepsilon,\,\alpha') \in u \quad \alpha \leq_{[\![\sigma]\!]_{\not\iota}} \alpha'}{x : u :: \sigma \vdash x : \alpha :: \sigma} \qquad \frac{\Gamma,\, x : u :: \sigma \vdash M : \alpha :: \tau}{\Gamma \vdash \lambda x.\, M : u \to \alpha :: \sigma \to \tau}\ \lambda$$

$$\text{App}\quad \frac{\Gamma_0 \vdash M : \{(c_1, \beta_1), \ldots, (c_n,\, \beta_n)\} \to \alpha :: \sigma \to \tau \qquad \Gamma_i \vdash N : \beta_i :: \sigma \;\; (\forall i)}{\Gamma_0 \cup \Box_{c_1} \Gamma_1 \cup \cdots \cup \Box_{c_n} \Gamma_n \vdash M\,N : \alpha :: \tau}$$

$$\text{Weak}\quad \frac{\Gamma \vdash t : \alpha :: \tau \qquad u \in \not\iota\,[\![\sigma]\!]_{\not\iota}}{\Gamma,\, x : u :: \sigma \vdash t : \alpha :: \tau} \qquad \frac{q \in Q \text{ and } \alpha \text{ validates } \delta(q, a)}{\emptyset \vdash a : \alpha \to q :: o^{\mathrm{ar}(a)} \to o}\ \delta$$

Figure 10.2: The type system $\mathcal{S}(\mathcal{A})$ computing denotations in $\mathbf{ScottL}_{\not\iota}$

For a closed term t of simple type σ, we set $[\![t]\!]_{\mathcal{A}} = [\![\emptyset \vdash t :: \sigma]\!]_{\mathcal{A}}$. A consequence of Proposition 32 is that the typing derivations describe the computations of the elements of the denotation of a term. Another important consequence of this proposition is the following corollary, which is a type-theoretic counterpart to the fact that $\mathbf{ScottL}_{\not\iota}$ is a cartesian closed category and therefore allows an interpretation of terms which is invariant under the rules β and η:

Corollary 4. *Suppose that* $\Gamma \vdash t : \alpha :: \sigma$ *in* $\mathcal{S}(\mathcal{A})$*. Then, given a term* t'*,*

- *if* $t \to_\beta t'$*, then* $\Gamma \vdash t' : \alpha :: \sigma$*,*
- *if* $t' \to_\beta t$*, then* $\Gamma \vdash t' : \alpha :: \sigma$*,*
- *if* $t \to_\eta t'$*, then* $\Gamma \vdash t' : \alpha :: \sigma$*,*
- *if* $t' \to_\eta t$*, then* $\Gamma \vdash t' : \alpha :: \sigma$*.*

Note that, as remarked for instance in [Sal10], the stability under η-reduction does not hold in a qualitative intersection type system (that is, idempotent) in the absence of subtyping. This property of subtyping is therefore crucial to obtain a model in which the interpretation of a term is invariant not only under the rule β, but also under η.

The remaining of this section is devoted to the proof of the correspondence between the denotations in $\mathbf{ScottL}_{\not\iota}$ and the typing derivations in $\mathcal{S}(\mathcal{A})$.

Proof of Proposition 32. We prove the proposition by induction on the derivation tree justifying

$$\Gamma = x_1 : u_1 :: \sigma_1, \ldots, x_n : u_n :: \sigma_n \vdash t : \alpha :: \tau$$

by applying the following sequence of lemmas (Lemmas 20-24) exhibiting the structure of the interpretation of terms in $\mathbf{ScottL}_{\not!}$. Note that all these interpretations are morphisms of the model, and are therefore downward-closed relations.

The first lemma justifies the Axiom rule:

Lemma 20. *The interpretation of* $x :: \sigma \vdash x :: \sigma$ *is*

$$[\![x :: \sigma \vdash x :: \sigma]\!]_{\mathcal{A}} = \{(u, \alpha) \mid u \in \not![\![\sigma]\!]_{\not!} \text{ and } \exists(\varepsilon, \alpha') \in u \;\; \alpha \leq_{[\![\sigma]\!]_{\not!}} \alpha'\}$$

Proof. This interpretation is given, in the cartesian closed category $\mathbf{ScottL}_{\not!}$, by the counit of the comonad $\not!$, considered in **ScottL**. It is itself obtained as the composite the counit of ! with the one of $\square$, which is on the object $[\![\sigma]\!]_{\not!}$

$$\mathbf{der}_{[\![\sigma]\!]_{\not!}} = \{((\varepsilon, \alpha''), \alpha) \mid \alpha \leq_{[\![\sigma]\!]_{\not!}} \alpha''\} : \square\,[\![\sigma]\!]_{\not!} \to [\![\sigma]\!]_{\not!}$$

The counit of ! on the object $\square\,[\![\sigma]\!]_{\not!}$ relates the set $u \in \not![\![\sigma]\!]_{\not!}$ of pairs of colors with elements of $[\![\sigma]\!]_{\not!}$ to the elements (c, α'') such that there exists $(c, \alpha') \in u$ with $\alpha'' \leq_{[\![\sigma]\!]_{\not!}} \alpha'$. The conclusion follows from the composition of these two relations, which notably composes only elements such that $c = \varepsilon$. □

The second lemma justifies the rule δ for the typing of constants:

Lemma 21. *The interpretation of* $\emptyset \vdash a :: o^n \to o$ *is*

$$\begin{array}{rcl} [\![\emptyset \vdash a :: o^n \to o]\!]_{\mathcal{A}} & = & [\![\, a \,]\!]_{\mathcal{A}} \\ & = & \{\, (\alpha, q) \mid q \in Q \text{ and } \alpha \text{ validates the formula } \delta(q, a) \,\} \\ & = & \{(v_1, \ldots, v_n, q) \mid \exists u_1 \cdots u_n \;\forall i \;\; u_i \leq_{\not!\perp\!\!\!\perp} v_i \\ & & \qquad \text{and } \bigcup_i \{i\} \times u_i \;\; \text{satisfies } \delta(q, a)\} \end{array}$$

Proof. The proof is immediate from the interpretation we gave of constants; we just point out that this interpretation is indeed downward-closed, as the notion of *validation* of a formula we introduced is precisely the closure under subtyping of the notion of *satisfaction* introduced earlier for positive boolean formulas and therefore for alternating tree automata. □

The next lemma justifies the Weakening rule:

Lemma 22. *Given a simple type* σ *and a morphism* $f \in (\not![\![\sigma_1]\!]_{\not!} \otimes \cdots \otimes \not![\![\sigma_n]\!]_{\not!}) \to [\![\tau]\!]_{\not!}$, *the interpretation of the weakening of* f *by* σ *is the morphism of* ***ScottL***

$$Weak_\sigma(f) \in (\not![\![\sigma]\!]_{\not!} \otimes \not![\![\sigma_1]\!]_{\not!} \otimes \cdots \otimes \not![\![\sigma_n]\!]_{\not!}) \to [\![\tau]\!]_{\not!}$$

whose elements are the tuples $(u, u_1, \ldots, u_n, \alpha)$ *such that* $(u_1, \ldots, u_n, \alpha) \in f$ *and* $u \in \not![\![\sigma]\!]_{\not!}$.

Proof. We consider the general case where $f : \not!C \multimap A$, and set $B = [\![\sigma]\!]_{\not!}$. We say that this case is general as, for $C = [\![\sigma_1]\!]_{\not!} \,\&\, \cdots \,\&\, [\![\sigma_n]\!]_{\not!}$, the Seely isomorphisms give

$$C \cong \not![\![\sigma_1]\!]_{\not!} \otimes \cdots \otimes \not![\![\sigma_n]\!]_{\not!}$$

This allows us to treat the general case by conveniently considering only one object in the domain of f. Following the usual interpretation of the λ-calculus in a cartesian closed category, the morphism $Weak_\sigma(f)$ is then obtained, modulo Seely isomorphism, as the composite in **ScottL**

$$!B \otimes !C \xrightarrow{\mathbf{dig}^!_B \otimes \mathbf{dig}^!_C} !!B \otimes !!C \xrightarrow{! \top_{!B} \otimes \mathbf{der}^!_{!C}} !\top \otimes !C \cong 1 \otimes !C \cong !C \xrightarrow{f} A$$

where the morphism $\top_{!B}$ is the unique morphism from $!B$ to the terminal object $\top = (\emptyset, \emptyset)$. It is therefore the empty relation, whose promotion $! \top_{!B}$ is the relation from $!!B$ to $!\top = 1 = (\{\star\}, =)$ such that

$$! \top_{!B} = \{ (u, \{\star\}) \mid u \in !!B \}$$

Moreover, since $!$ is a comonad, we have the equality $\mathbf{der}^!_{!C} \circ \mathbf{dig}^!_C = id_{!C}$, so that $Weak_\sigma(f)$ rewrites as

$$!B \otimes !C \xrightarrow{\mathbf{dig}^!_B \otimes id_{!C}} !!B \otimes !C \xrightarrow{! \top_B \otimes id_{!C}} !\top \otimes !C \cong !C \xrightarrow{f} A$$

This composite relates every pair $(u, v) \in !B \otimes !C$ with a, for $(v, a) \in f$ and $u \in !B$. This proves the lemma. □

The next lemma justifies the rule λ:

Lemma 23. *Given a morphism $f \in (\, !\llbracket\sigma_1\rrbracket_! \otimes \cdots \otimes !\llbracket\sigma_n\rrbracket_! \otimes !\llbracket\sigma\rrbracket_! \,) \to \llbracket\tau\rrbracket_!$ in **ScottL**, the interpretation of the abstraction of σ in f is the morphism*

$$\Lambda_\sigma(f) \in (\, !\llbracket\sigma_1\rrbracket_! \otimes \cdots \otimes !\llbracket\sigma_n\rrbracket_! \,) \to (!\llbracket\sigma\rrbracket_! \multimap \llbracket\tau\rrbracket_!)$$

whose elements are the tuples $(u_1, \ldots, u_n, (u, \alpha))$ such that $(u_1, \ldots, u_n, u, \alpha) \in f$.

Proof. As in the previous lemma, we consider a morphism $f : !C \otimes !B \to A$, with $C = \llbracket\sigma_1\rrbracket_! \& \cdots \& \llbracket\sigma_n\rrbracket_!$, $B = \llbracket\sigma\rrbracket_!$ and $A = \llbracket\tau\rrbracket_!$. Its interpretation is given directly by the cartesian closure of $\mathbf{ScottL}_!$: it is:

$$C \xrightarrow{\Lambda_{C,B,A}(f)} (B \Rightarrow A) \cong !B \multimap A$$

and the definition of $\Lambda_{C,B,A}(f)$ induces the result of the lemma. □

The last lemma justifies semantically the Application rule.

Lemma 24. *Let $f : !C \to (!A \multimap B)$ and $g : !C \to A$ be two morphisms. The morphism*

$$App(f, g) : !C \to B$$

is defined as the set of couples

$$(v_0 \cup \Box_{c_1} v_1 \cup \cdots \cup \Box_{c_n} v_n, \beta) \in !C \times B$$

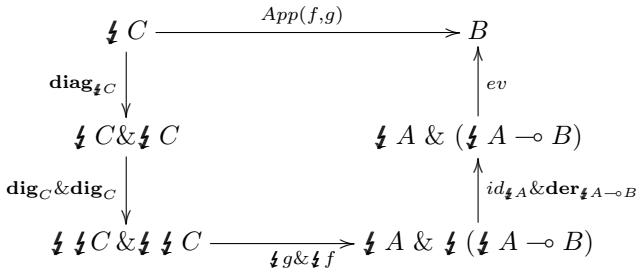
such that there exists

$$(v_0, (\{(c_1, \alpha_1), \ldots, (c_n, \alpha_n)\}, \beta)) \in f$$

and that, for $i \in \{1, \ldots, n\}$,

$$(v_i, \alpha_i) \in g.$$

Proof. In a cartesian closed category obtained from a model of linear logic, the morphism $App(f,g)$ is traditionally obtained – modulo Seely isomorphism – as the composite



where the depicted morphisms are in the model of linear logic we consider – here, **ScottL** – and not in the Kleisli category itself. In this diagram, the morphism **diag** is the diagonal morphism associated with the cartesian product $\&$, and the evaluation ev is just the relational composition. Let us first simplify this diagram by proving that

$$\mathbf{der}_{!A \multimap B} \circ !f \circ \mathbf{dig}_C \quad = \quad f$$

Consider $(v,(u,\beta)) \in f$. We have:

- $(v, \{(\varepsilon, v)\}) \in \mathbf{dig}_C$,
- $(\{(\varepsilon, v)\}, \{(\varepsilon, (u,\beta))\} \in !f$,
- and $(\{(\varepsilon,(u,\beta))\},(u,\beta)) \in \mathbf{der}_{!A \multimap B}$

so that $(v,(u,\beta)) \in \mathbf{der}_{!A \multimap B} \circ !f \circ \mathbf{dig}_C$.

Conversely, consider $(v,(u,\beta)) \in \mathbf{der}_{!A \multimap B} \circ !f \circ \mathbf{dig}_C$. This element is necessarily obtained from the composition of elements of the form:

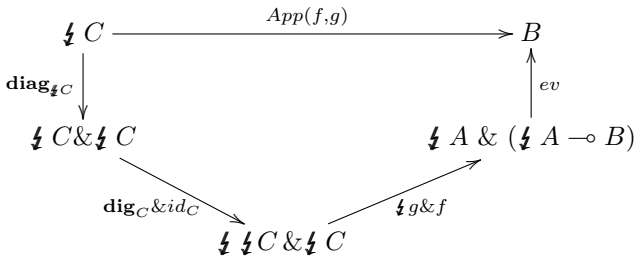
- $(v, \{(\varepsilon, v'_0), (c_1, v'_1), \ldots, (c_n, v'_n)\}) \in \mathbf{dig}_C$,
- $(\{(\varepsilon, v'_0), (c_1, v'_1), \ldots, (c_n, v'_n)\}, \{(\varepsilon, (u',\beta'))\} \in !f$,
- and $(\{(\varepsilon,(u',\beta'))\},(u,\beta)) \in \mathbf{der}_{!A \multimap B}$.

where

- $\Box_\varepsilon v'_0 \cup \Box_{c_1} v'_1 \cup \cdots \cup \Box_{c_n} v'_n \leq_{!C} v$, which implies $v'_0 \leq_{!C} v$, by definition of **dig**,
- $(v'_0, (u',\beta')) \in f$, by definition of $!f$,
- and $u' \leq_{!A} u$ and $\beta \leq_B \beta'$, by definition of **der**.

Since f is downward-closed, it contains $(v,(u,\beta))$. This concludes the proof of equality.

We can therefore simplify the definition of $App(f,g)$ as



From this diagram follows that the elements $(v,\beta) \in App(f,g)$ are the ones such that

- $\exists v_0, \ldots, v_n$ such that $v_0 \cup \Box_{c_1} v_1 \cup \cdots \cup \Box_{c_n} v_n \leq v$,
- for every $i \in \{1,\ldots,n\}$, $(v_i, \alpha_i) \in g$,
- $(\{(c_1,\alpha_1),\ldots,(c_n,\alpha_n))\},\beta) \in f$.

We can moreover assume that $v_0 \cup \Box_{c_1} v_1 \cup \cdots \cup \Box_{c_n} v_n = v$, since g is downward-closed. This concludes the lemma. □

To relate this last lemma with the Application rule, we use a single object C to describe a context, as in the proofs of previous lemmas, taking to our advantage the Seely isomorphism. Moreover, the morphism g we use here implicitly uses a same context domain for every sequent

$$\Gamma_i \vdash N : \beta_i :: \sigma$$

We claim that this is not a restriction, as we can use weakening to introduce an empty refined type to variables missing in one of these contexts to ensure uniformity.

From these lemmas follows Proposition 32.

10.3 A first connection with higher-order model-checking

The purpose of this chapter is to connect $\mathbf{ScottL}_{\text{ϟ}}$ with higher-order model-checking, in order to obtain in this model of the λ-calculus a finitary counterpart to Conjecture 1. In a first time, we aim at relating the type system $\mathcal{S}(\mathcal{A})$ describing the computation of denotations in this model with the intersection type system $\gamma^{st}(\mathcal{A})$ introduced earlier for model-checking. The major hurdle to this connection is the lack of subtyping in $\gamma^{st}(\mathcal{A})$. However, it turns out that the subtyping appearing in $\mathcal{S}(\mathcal{A})$ can be eliminated on η-long forms of λ-terms, as we prove now adapting [Sal10] to $\mathcal{S}(\mathcal{A})$. In [Sal10], Salvati considers an intersection type system which is equivalent to the intersection type systems introduced by Ehrhard [Ehr12a] and Terui [Ter12] for the computation of denotations in $\mathbf{ScottL}_!$. The main differences with our framework are that we consider a type system enriched with a coloring modality, and where contexts are managed multiplicatively, while their system is additive and does not

feature modalities. This multiplicative management of contexts is necessary to design the Application rule, as the modality acts differently on each argument. This makes the proof a little more complicated than in [Sal10], even though the core idea is the same.

η-long forms. First, let us recall the notion of *η-long form* of a simply-typed term, introduced in [Hue76] where the properties we are to formulate are proved. A term t is in η-long form when, for every decomposition $t = C[u]$, we have

- either $u = \lambda x.\, u'$,
- or $C = C'[[\,]\ u']$.

The set of terms in long form is closed under β-reduction. For every term t, there are terms t' in long form such that $t' \to_\eta t$ (where $\to_\eta$ is the η-reduction defined on p.57), which can be obtained by η-expansion. Moreover, when t is in long form and that $t' \to_\eta t$, we also have that $t' \to_\beta t$ – recall that this implication does not hold in general. This notably implies that every term has a "minimal" long form, obtained by iterating β reduction over one of its long forms. The result is its *β-normal η-long form*, and is unique. The set of such forms over a set of variables $\mathcal{V}$ and a signature Σ is described by the grammar

$$t \quad ::= \quad x\ t_1 \cdots t_n \mid a\ t_1 \cdots t_n \mid \lambda x.\, t$$

where $x \in \mathcal{V}$, $a \in \Sigma$, and n is required to be the arity of the simple type of x or a, and may thus be 0.

η-long forms and subtyping. We say a derivation is *without subtyping* when its Axiom leaves are all of the form

$$\mathrm{Ax} \quad \frac{(\varepsilon,\, \alpha) \in u}{x \,:\, u \,::\, \sigma \ \vdash\ x \,:\, \alpha \,::\, \sigma}$$

and all its δ leaves are of the form

$$\frac{q \in Q \text{ and } \alpha \text{ satisfies } \delta(q,a)}{\emptyset \ \vdash\ a \,:\, \alpha \to q \,::\, o^{\mathrm{ar}(a)} \to o}\ \delta$$

where we replaced the fact that α should *validate* $\delta(q,a)$ by the fact that it should *satisfy* it – in other words, we do not allow α to contain more information than strictly needed for the transition $\delta(q,a)$. Notice that for both these rules, these restrictions amount not to use subtyping to introduce the variables and constants.

Lemma 25. *Suppose that $\Gamma \ \vdash\ t \,:\, \alpha \,::\, \sigma$ is derivable in $\mathcal{S}(\mathcal{A})$ without subtyping, and that $\alpha' \leq_{[\![\sigma]\!]_\sharp} \alpha$. There there exists a "$\eta$-longer" term t' such that $t' \to_\eta^* t$ and that $\Gamma \ \vdash\ t' \,:\, \alpha' \,::\, \sigma$ is derivable in $\mathcal{S}(\mathcal{A})$ without subtyping.*

Proof. We prove the lemma by induction on the simple type σ. If $\sigma = o$ is the ground type, then there is a state q of the automaton such that $q = \alpha = \alpha'$, and it suffices to take $t' = t$. Suppose now that

$$\sigma \quad = \quad \sigma_1 \to \cdots \to \sigma_n \to o$$

This implies that α and α' decompose as

$$\alpha \quad = \quad S_1 \to \cdots \to S_n \to q$$

and

$$\alpha' \quad = \quad S'_1 \to \cdots \to S'_n \to q'$$

Since $\alpha' \leq_{[\![\sigma]\!]_\sharp} \alpha$, we have that $q = q'$, and that for every $i \in \{1, \ldots, n\}$ and every $(c, \beta) \in S_i$ there exists (c, β') in S'_i such that $\beta \leq_{[\![\sigma_i]\!]_\sharp} \beta'$.

We introduce n fresh variables $y_1, \ldots, y_n$. By the Axiom rule, we obtain that $y_i : (\varepsilon, \beta') :: \sigma_i \vdash y_i : \beta' :: \sigma_i$ is provable, and the induction hypothesis implies that there exists a term t_i^β such that $t_i^\beta \to_\eta^* y_i$ and that $y_i : (\varepsilon, \beta') :: \sigma_i \vdash t_i^\beta : \beta :: \sigma_i$ is provable. Considering the set of elements $(c, \beta) \in S_i$, we obtain in this way a family of terms $(t_i^\beta)_\beta$ which can all be β-reduced to y_i. But η-expansion is confluent, so that there exists a term t_i such that, for every β, $t_i \to_\eta^* t_i^\beta$. Moreover, η-expansion preserves typing in $\mathcal{S}(\mathcal{A})$ by Corollary 4: for every $(c, \beta) \in S_i$, there exists (c, β') in S'_i such that $\beta \leq_{[\![\sigma_i]\!]_\sharp} \beta'$, and that

$$y_i : (\varepsilon, \beta') :: \sigma_i \vdash t_i : \beta :: \sigma_i \text{ is provable in } \mathcal{S}(\mathcal{A}) \tag{10.2}$$

So, for every $i \in \{1, \ldots, n\}$ and every $(c, \beta) \in S_i$, we have a proof of this sequent. We consider now a proof of $\Gamma \vdash t : \alpha :: \sigma$ and, using weakening, we obtain one of

$$\Gamma, y_1 : S'_1 :: \sigma_1, \ldots, y_n : S'_n :: \sigma_n \vdash t : S_1 \to \cdots \to S_n \to q :: \sigma$$

Using the Application rule n times, we obtain a proof of

$$\Gamma, y_1 : S'_1 :: \sigma_1, \ldots, y_n : S'_n :: \sigma_n \vdash t\ t_1 \cdots t_n : q :: \sigma \tag{10.3}$$

from all the proofs we previously considered. Let us explain why the context does only contain the set S'_i for each variable y_i: the proof of this sequent is obtained by providing, for every i and $(c, \beta) \in S_i$, a proof of the sequent (10.2) in which the context contains only the type (ε, β') for y_i. In the Application building the proof of (10.3), this context is affected by the coloring $\square_c$ which updates it to (c, β'), which belongs to S'_i. The result is now obtained by applying the abstraction rule n times to (10.3), and by remarking that

$$\begin{array}{lll} \lambda x_1. \ldots \lambda x_n. t\ t_1 \cdots t_n & \to_\eta^* & \lambda x_1. \ldots \lambda x_n. t\ y_1\ t_2 \cdots t_n \\ & \to_\eta^* & \lambda x_1. \ldots \lambda x_n. t\ y_1 \cdots y_n \\ & \to_\eta^* & t \end{array}$$

□

Corollary 5. *Suppose that $\Gamma \vdash t : \alpha :: \sigma$ is derivable in $\mathcal{S}(\mathcal{A})$ without subtyping, and that $\alpha' \leq_{[\![\sigma]\!]_\sharp} \alpha$. If t is in η-long form, then $\Gamma \vdash t : \alpha' :: \sigma$ is derivable in $\mathcal{S}(\mathcal{A})$ without subtyping.*

Proof. By Lemma 25, there exists t' such that $t' \to_\eta^* t$ and that $\Gamma \vdash t' : \alpha' :: \sigma$ is derivable in $\mathcal{S}(\mathcal{A})$ without subtyping. But the fact that t is η-long implies that $t' \to_\beta^* t$, and Corollary 4 allows to conclude that $\Gamma \vdash t : \alpha' :: \sigma$ is derivable in $\mathcal{S}(\mathcal{A})$. The fact that it does not require subtyping can be deduced

from the subject reduction procedure, which only manipulates subproofs and does modify them[1]. □

A consequence of this corollary and of the stability of typability in $\mathcal{S}(\mathcal{A})$ expressed in Corollary 4 is the following theorem explaining that subtyping can be eliminated on η-long forms:

Theorem 28. *Let t be a term and $t^{\beta\eta}$ its β-normal η-long form. Then $\Gamma \vdash t : \alpha :: \sigma$ is derivable in $\mathcal{S}(\mathcal{A})$ if and only if $\Gamma \vdash t^{\beta\eta} : \alpha :: \sigma$ is derivable without subtyping in $\mathcal{S}(\mathcal{A})$.*

Proof. Suppose that $\Gamma \vdash t : \alpha :: \sigma$ is derivable in $\mathcal{S}(\mathcal{A})$, and consider a proof π of this sequent. For every Axiom leaf using subtyping, that is, of the form

$$\text{Ax}\quad \frac{\exists\, (\varepsilon,\, \alpha') \in u \quad \alpha \;\leq_{[\![\sigma]\!]_{\not z}}\; \alpha'}{x \,:\, u :: \sigma \;\vdash\; x \,:\, \alpha :: \sigma}$$

where $\alpha \;\leq_{[\![\sigma]\!]_{\not z}}\; \alpha'$ and $\alpha \neq \alpha'$, we have that $x : u :: \sigma \vdash x : \alpha' :: \sigma$ is provable without subtyping, and this implies that the β-normal η-long form t_x of x is such that $x : u :: \sigma \vdash t_x : \alpha' :: \sigma$ is provable without subtyping. By Corollary 5, $x : u :: \sigma \vdash t_x : \alpha :: \sigma$ is provable without subtyping. We replace the Axiom leaf we considered in π by this proof without subtyping for t_x. We proceed similarly for the δ leaves using subtyping, and obtain a proof π' without subtyping of a sequent $\Gamma \vdash t' : \alpha :: \sigma$, such that $t^{\beta\eta} \to_\eta^* t'$. By η-expansion, we obtain that $\Gamma \vdash t^{\beta\eta} : \alpha :: \sigma$ is derivable without subtyping in $\mathcal{S}(\mathcal{A})$.

Suppose now that $\Gamma \vdash t^{\beta\eta} : \alpha :: \sigma$ is derivable without subtyping in $\mathcal{S}(\mathcal{A})$. Then, in particular, $\Gamma \vdash t^{\beta\eta} : \alpha :: \sigma$ is derivable in $\mathcal{S}(\mathcal{A})$ – nothing being said about subtyping anymore. By Corollary 4, since $t^{\beta\eta} \to_\eta^* t$, $\Gamma \vdash t : \alpha :: \sigma$ is derivable in $\mathcal{S}(\mathcal{A})$. □

This is an adaptation to the colored, multiplicative type system $\mathcal{S}(\mathcal{A})$ of the Theorem 4 of [Sal10]. Intuitively, this theorem expresses the fact that, on ground type, weakening is enough to simulate subtyping: an Axiom rule introducing a variable of ground type is of the shape

$$\text{Ax}\quad \frac{(\varepsilon,\, q) \in u}{x \,:\, u :: o \;\vdash\; x \,:\, q :: o}$$

and is therefore without subtyping. It embeds weakening, as we allow to introduce in the context a set of types u for x which is greater than $\{(\varepsilon, q)\}$. When a term is in η-long form, all its Axiom leaves introduce variables of ground type, so that they are without subtyping.

Relating type systems. The type system $\mathcal{S}(\mathcal{A})$ describes the computation of denotations in $\mathbf{ScottL}_{\not z}$, so that the typing of a variable in a context

$$x \,:\, \{\, (c_i,\, \alpha_i) \mid i \in I \,\} :: \sigma \tag{10.4}$$

[1] Only η-reduction actually requires subtyping to preserve typability, see [Sal10] for instance.

describes a set of elements belonging to the interpretation $\lightning [\![\sigma]\!]_\lightning$ of the simple type σ of the variable x. As already pointed out by Ehrhard [Ehr12a] and Terui [Ter12] in the type system they give for describing denotations in the uncolored model $\mathbf{ScottL}_!$, this notation corresponds to an idempotent intersection type. In our situation, the typing (10.4) may be written as the idempotent intersection type

$$x : \bigwedge_{i\in I} \boxdot_{c_i} \theta_i :: \sigma$$

We denote by $(\!|\cdot|\!)$ the translation of sets into intersection types, and by $|\!)\cdot(\!|$ its converse. Once this identification performed, only three differences remain between the type systems $\mathcal{S}(\mathcal{A})$ and $\curlyvee^{st}(\mathcal{A})$:

1. in $\curlyvee^{st}(\mathcal{A})$, there is no subtyping,
2. in $\mathcal{S}(\mathcal{A})$, the Axiom rule allows to introduce a set of elements u to type the variable x in the context, while in $\curlyvee^{st}(\mathcal{A})$ only one type may be introduced,
3. and the weakening rules are not exactly the same in both systems.

We shall now explain why $\curlyvee^{st}(\mathcal{A})$ and the subtyping-free fragment of $\mathcal{S}(\mathcal{A})$ are equivalent on η-long forms:

Theorem 29. *Let t be a term such that*

$$\Gamma \vdash t : \theta :: \kappa$$

is provable in $\curlyvee^{st}(\mathcal{A})$. Then

$$|\!)\,\Gamma\,(\!| \vdash t : |\!)\,\theta\,(\!| :: \kappa$$

is provable without subtyping in $\mathcal{S}(\mathcal{A})$.

Let t be a term such that

$$\Gamma \vdash t : \alpha :: \sigma$$

is provable in $\mathcal{S}(\mathcal{A})$. By Theorem 28, its β-normal η-long form can be given the same type in the same context, without using subtyping, and this proof without subtyping can be translated into a proof of

$$(\!|\,\Gamma\,|\!) \vdash t^{\beta\eta} : (\!|\,\alpha\,|\!) :: \sigma$$

in $\curlyvee^{st}(\mathcal{A})$.

We identify sets and intersection types to ease the statement of this corollary:

Corollary 6. *Let t be a term in β-normal η-long form. Then*

$$\Gamma \vdash t : \alpha :: \sigma$$

is provable in $\curlyvee^{st}(\mathcal{A})$ if and only if it is provable in $\mathcal{S}(\mathcal{A})$.

Let us prove this theorem just by explaining how to deal with the three differences we pointed out between the subtyping-free fragment of $\mathcal{S}(\mathcal{A})$ and $\curlyvee^{st}(\mathcal{A})$.

1. The lack of subtyping is only an obstruction to translate proofs of $\mathcal{S}(\mathcal{A})$ into $\boldsymbol{\gamma}^{st}(\mathcal{A})$. We bypass this problem by considering the β-normal η-long form of the term, which is always provable without subtyping by Theorem 28. Note that we may have used any term obtained by "enough" η-expansions of t to eliminate subtyping. However, as we will not need this accuracy in the sequel, we choose to stick to the most canonical η-long form of t, that is $t^{\beta\eta}$.

2. In $\boldsymbol{\gamma}^{st}(\mathcal{A})$, only one type may be introduced by the Axiom rule, but we can use immediately after the rule $Weak_c$ to add more types to the context and simulate the Axiom rule of $\mathcal{S}(\mathcal{A})$.

3. The weakening rule of $\mathcal{S}(\mathcal{A})$ can be translated using a $Weak$ rule followed by a $Weak_c$ rule. The translation of the weakenings from $\boldsymbol{\gamma}^{st}(\mathcal{A})$ to $\mathcal{S}(\mathcal{A})$ is more subtle, and involves some prior proof rewriting which we only sketch here. We use commutations of rules as in §6.6, but this time to make the rules $Weak$ and $Weak_c$ move upper in the proof. On an Application typing $t\ u$, several choices arise. We always choose to commute the $Weak$ or $Weak_c$ rule with the Application one so that it goes to the (unique) derivation for t occurring as a premise of the Application rule. There are two reasons for this choice: it is not affected by the coloring modality, and this derivation always exists and is unique, while there may be no or multiple derivations for u. We iterate these commutations as much as possible. When two $Weak_c$ rules for the same variable are in a row, we contract them to only one. Once this process done, two situations arise given a $Weak_c$ rule:

   - if there is an Axiom leaf introducing the variable the $Weak_c$ rule weakens above it, the translation combines the effect of both rules by using an Axiom rule in $\mathcal{S}(\mathcal{A})$ which introduces directly in the context the one built by the $Weak_c$ rule,
   - and if there is no such Axiom leaf, then the variable is introduced by a $Weak$ rule. We can use commutations to make this $Weak$ rule and the $Weak_c$ rule we consider occur in a row, and replace them both by a Weakening in $\mathcal{S}(\mathcal{A})$ having the same effect.

These elements sketch how we can mutually translate subtyping-free proofs of $\mathcal{S}(\mathcal{A})$ into proofs of $\boldsymbol{\gamma}^{st}(\mathcal{A})$, and conversely.

Colored Scott semantics and higher-order model-checking. A corollary of Proposition 32 and of Theorem 29 is the following theorem:

Theorem 30. *Let t be a term of simple type τ whose free variables define a context*

$$\Gamma \ = \ x_1 :: \sigma_1, \ldots, x_n :: \sigma_n$$

and let $t^{\beta\eta}$ be its β-normal η-long form. Then

$$[\![\Gamma \ \vdash \ t :: \tau]\!]_{\mathcal{A}} \ \cong \ \{ \, (\!| \theta |\!) \mid (\!| \Gamma |\!) \ \vdash \ t^{\beta\eta} : \theta :: \tau \ \textit{in} \ \boldsymbol{\gamma}^{st}(\mathcal{A}) \, \}$$

Now that we incorporated the coloring modality into the Scott model of linear logic, and connected $\mathbf{ScottL}_{\boldsymbol{\ell}}$ with the type system $\boldsymbol{\gamma}^{st}(\mathcal{A})$ we use for

higher-order model-checking in §6.6, it remains to deal with recursion. The idea is to extend the model $\mathbf{ScottL}_{\lightning}$ of the λ-calculus with a fixpoint operator, and to obtain in this way a model of the λY-calculus in which we will be able to interpret higher-order recursion schemes thanks to their equivalence with λY-terms (Proposition 5). We proceed similarly as we did for the relational model in Chapter 9.

The fixpoint operator will act as the *fix* rule does in $\gamma^{st}_{fix}(\mathcal{G}, \mathcal{A})$: by composing denotations of terms in $\mathbf{ScottL}_{\lightning}$, and checking that the infinite branches of the composition tree satisfy the parity condition.

10.4 The recursion operator Y

At this stage, we are ready to shift from the colored semantics of the simply-typed λ-calculus formulated in §10.2 to a colored semantics of the simply-typed λY-calculus. To that purpose, we construct a Conway operator $\mathbf{Y}$ in the category **FinScottL** defined as the full subcategory of **ScottL** consisting of the *finite* ordered sets. Note that **FinScottL** defines a Seely category, and thus a model of full propositional linear logic. We will see in §10.6 and in §10.7 that the finitary nature of the model will enable us to establish the *decidability* of the local higher-order model-checking problem (Corollary 8) and of the selection problem (Theorem 33). On the other hand, shifting from **ScottL** to **FinScottL** means that we cannot interpret any more infinitary types like the type of natural numbers. The Conway operator $\mathbf{Y}$ is defined a family of operations $\mathbf{Y}_{X,A}$ transporting a binary downward-closed relation

$$R \quad : \quad \lightning X \otimes \lightning A \multimap A$$

into a binary downward-closed relation

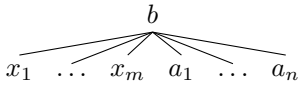
$$\mathbf{Y}_{X,A}(R) \quad : \quad \lightning X \multimap A$$

and satisfying a series of conditions given in §9.4. Again, such a Conway operator on **FinScottL** defines a Conway operator in the sense of [BÉ96] in the cartesian-closed category $\mathbf{FinScottL}_{\lightning}$. Just as in the case of the relational semantics, the important point here is that the colors added to the original Scott semantics will enable us to alternate least and greatest fixpoints (and thus inductive and coinductive reasoning) in the definition of the fixpoint operator $\mathbf{Y}$, using the appropriate parity condition.

Semantic run-trees. Given a relation $R : \lightning X \otimes \lightning A \multimap A$ and $a \in A$, we define the set $\mathbf{run\text{-}tree}(R, a)$ of *semantic run-trees* of R producing $a \in A$ as the set of possibly infinite $(X \uplus A)$-labeled trees, with nodes colored by elements of Col, and such that the four conditions below are satisfied:

1. the root of the tree is labeled by a, and has neutral color ε,
2. the inner nodes of the tree are labeled by elements of the set A,
3. the leaves are labeled by elements of the set $X \uplus A$,
4. for every node labeled by an element $b \in A$:

- if b is an inner node, letting $a_1, \ldots, a_n$ denote the labels of its children belonging to A and $x_1, \ldots, x_m$ the labels belonging to X:



  and letting c_i (resp. d_j) be the color of the node labeled x_i (resp. a_j),

$$(\{(c_1, x_1), \cdots, (c_m, x_m)\}, \{(d_1, a_1), \cdots, (d_n, a_n)\}, b) \in R$$

- if b is a leaf, then $(\emptyset, \emptyset, b) \in R$.

At this point, we adapt to semantic run-trees the usual acceptance condition on the run-trees of an alternating parity automata: an infinite branch of the semantic run-tree is winning if and only if an element of $Col \setminus \{\varepsilon\}$ occurs infinitely often along it, and if the maximal such element is even. A semantic run-tree is declared winning if and only if all its infinite branches are.

Given a semantic run-tree *witness*, we define the set $\mathbf{leaves}(witness) \subseteq \strut\!\!\not!\, X$ as the set of elements (c, x) where (c', x) is a leaf of *witness* labeled with $x \in X$, and c is the maximal color encountered on the path from the leaf to the root of *witness*.

Fixpoint operator. We now define the fixpoint of a binary relation

$$R : \strut\!\!\not!\, X \otimes \strut\!\!\not!\, A \multimap A$$

as the downward-closed binary relation

$$\mathbf{Y}_{X,A}(R) \quad = \quad \{(u, a) \mid \exists witness \in \mathbf{run\text{-}tree}(R, a) \text{ with } u = \mathbf{leaves}(witness) \text{ and } witness \text{ is a winning semantic run-tree.}\} \tag{10.5}$$

Proposition 33. *The fixpoint operator* $\mathbf{Y}$ *is a Conway operator over* ***FinScottL****. The Kleisli category* $\mathbf{FinScottL}_{\strut\!\!\not!}$ *of* $\strut\!\!\not!$ *is therefore a model of the* λY*-calculus.*

As in §10.2, we find useful and even illuminating to formulate a type-theoretic counterpart to our definition of the Conway operator $\mathbf{Y}_{X,A}$ provided by the following typing rule Y_σ which should be added to the type system of Figure 10.2:

$$Y_\sigma \quad \frac{\Gamma_0 \vdash M : \{(c_1, \beta_1), \ldots, (c_n, \beta_n)\} \to \alpha :: \sigma \to \sigma \qquad \Gamma_i \vdash Y_\sigma M : \beta_i :: \sigma}{\Gamma_0 \cup \Box_{c_1} \Gamma_1 \cup \cdots \cup \Box_{c_n} \Gamma_n \vdash Y_\sigma M : \alpha :: \sigma}$$

In the resulting intersection type system, derivations of infinite depth are allowed, and have colored nodes, defined as follows:

- for every occurrence of the rule Y_σ, we assign color c_i to the node $\Gamma_i \vdash Y_\sigma M : \beta_i :: \sigma$.

- all the other nodes are assigned the neutral color ε.

An infinite derivation tree is then accepted when all its branches are winning for the parity condition, just as for the branches of a semantic run-tree. It is easy to see that the typing rule Y_σ is the semantic counterpart of the fixpoint operator $\mathbf{Y}$:

Theorem 31. *Given a λY-term t, the sequent*

$$\Gamma \;=\; x_1 : u_1 :: \sigma_1, \ldots, x_n : u_n :: \sigma_n \;\vdash\; t : \alpha :: \tau$$

has a winning derivation tree in the type system with fixpoints iff

$$(u_1, \ldots, u_n, \alpha) \;\in\; [\![\Gamma \;\vdash\; t :: \tau]\!]_{\mathcal{A}} \;\subseteq\; (\, \lightning\, [\![\sigma_1]\!]_{\lightning} \otimes \cdots \otimes \lightning\, [\![\sigma_n]\!]_{\lightning}\,) \multimap [\![\tau]\!]_{\lightning}$$

10.5 Interpretation of higher-order recursion schemes

Once this fixpoint operator defined in the model, and in order to investigate the decidability of the higher-order model-checking problem using the Scott semantics, it remains to explain how recursion schemes can be interpreted in the model. Our aim is to find a rule analogous to Y_σ, but for non-terminals, and which could then be related to the rule *fix* of $\boldsymbol{\gamma}^{st}_{fix}(\mathcal{G}, \mathcal{A})$. This would connect our reformulation of the result of Kobayashi and Ong with our colored Scott semantics of linear logic, and allow us to solve the higher-order model-checking problem.

Recall that Proposition 5 states that HORS can be translated into closed λY-terms over the same signature, and conversely. The translation, described in [SW12], maps a recursion scheme over the set of non-terminals $\{F_1 = S, \ldots, F_n\}$ to the term T_1 obtained as follows:

$$\begin{array}{rcl} T_n & = & Y\ (\,\lambda F_n.\, \mathcal{R}(F_n)\,) \\ T_{n-1} & = & Y\ (\,\lambda F_{n-1}.\, \mathcal{R}(F_{n-1})[F_n \leftarrow T_n]\,) \\ & \vdots & \\ T_1 & = & Y\ (\,\lambda F_1.\, \mathcal{R}(F_1)[F_2 \leftarrow T_2, \ldots, F_n \leftarrow T_n]\,) \end{array}$$

Adapting the rule Y_σ to this encoding seems particularly tedious, as it features several nested fixpoint calls. Another solution is to consider instead the usual semantic interpretation of the solution of a system of equations in a cartesian closed category with a fixpoint operator, and to set directly

$$\begin{array}{rcl} [\![\mathcal{G}]\!]_{\mathcal{A}} & = & [\![\pi_1\ (Y\ (\,\lambda F.\, \langle\, t_1, \ldots, t_n \,\rangle\,))]\!]_{\mathcal{A}} \\ & = & \pi_1\ (\,\mathbf{Y}\ (\,\Lambda\, \langle\, [\![t_1]\!]_{\mathcal{A}}, \ldots, [\![t_n]\!]_{\mathcal{A}} \,\rangle\,)\,) \end{array}$$

where we consider an extension of λ-terms (and thus of simple types) with cartesian products and projections, and where each term t_i is equal to the term obtained from $\mathcal{R}(F_i)$ by replacing every non-terminal F_j by the projection $\pi_j\, F$. The use of the morphism Λ is a semantic counterpart to the syntactic abstraction λF. Semantically, this object is well-defined, as the morphism

$$\Lambda\, \langle\, [\![t_1]\!]_{\mathcal{A}}, \ldots, [\![t_n]\!]_{\mathcal{A}} \,\rangle$$

is an element of

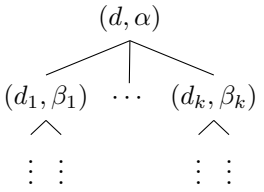
$$!\left(\bigwedge_{i=1}^{n} [\![\kappa(F_i)]\!]_!\right) \multimap \bigwedge_{i=1}^{n} [\![\kappa(F_i)]\!]_!$$

and we can apply the fixpoint operator to the objects $A = \bigwedge_{i=1}^{n} [\![\kappa(F_i)]\!]_!$ and $X = \top$. Note that the set X of parameters is empty, as the terms t_i only contain F as a free variable, which is captured by the abstraction λF. If we did not treat the elements of Σ as constants, but as free variables as in the previous chapter, there would be parameters to take care of.

Since A is an object of **ScottL**, the fixpoint operator is defined, even if A is obtained as a cartesian product: the elements of

$$\mathbf{Y}\ (\ \Lambda\,\langle\, [\![t_1]\!]_{\mathcal{A}}, \ldots, [\![t_n]\!]_{\mathcal{A}}\,\rangle\ )$$

are simply the couples (d, α) consisting of an integer $d \in \{1, \ldots, n\}$ and of an element $\beta \in [\![\kappa(F_d)]\!]_!$, such that there exists a semantic run-tree, winning for the parity condition, of the shape



where the node (d_i, β_i) has color c_i, and

$$(\{\,(c_i, (d_i, \beta_i)) \mid i \in \{1, \ldots, k\}\,\},\ (d,\ \alpha)) \quad \in \quad [\![\lambda F.\,\langle\, t_1, \ldots,\ t_n\,\rangle]\!]_{\mathcal{A}}$$

Similar conditions show that every (d_i, β_i) can be produced by the relation $[\![\lambda F.\,\langle\, t_1, \ldots,\ t_n\,\rangle]\!]_{\mathcal{A}}$, and so on.

To extend the rule Y_σ to this more general framework in which we use cartesian products, a natural temptation would be to extend the type system $\mathcal{S}(\mathcal{A})$ we defined for the λ-calculus with a cartesian product and projections, in order to reflect the cartesian structure of the semantics directly in the refined types of the system, and then to accommodate the fixpoint rule to this extended setting. We find easier, however, to simply introduce a family of operators

$$\mu_i(x_1, \ldots,\ x_n).\,\langle\, t_1, \ldots,\ t_n\,\rangle$$

for terms $t_1, \ldots, t_n$ whose free variables are included in $\{x_1, \ldots,\ x_n\}$ and which would be defined, in this extension of the calculus with products, as

$$\mu_i(x_1, \ldots,\ x_n).\,\langle\, t_1, \ldots,\ t_n\,\rangle \ = \ \pi_i\ (\,Y\ (\,\lambda F.\,\langle\, t_1[x_1 \leftarrow \pi_1\,F, \ldots, x_n \leftarrow \pi_n\,F], \ldots, t_n[x_1 \leftarrow \pi_1\,F, \ldots, x_n \leftarrow \pi_n\,F]\,\rangle\,)\,)$$

In the sequel, we often write $[x_j \leftarrow \pi_j\,F]$ to denote the substitution $[x_1 \leftarrow \pi_1\,F, \ldots, x_n \leftarrow \pi_n\,F]$. These operators unfold using the rule

$$\mu_i(x_1, \ldots,\ x_n).\,\langle\, t_1, \ldots,\ t_n\,\rangle \quad \to \quad t_i[x_j \leftarrow \mu_j(x_1, \ldots,\ x_n).\,\langle\, t_1, \ldots,\ t_n\,\rangle] \quad (10.6)$$

in which, again, the substitution is performed for every $j \in \{1, \ldots, n\}$. This rule is supported by the fact that, in the extension of the calculus with cartesian products, we would have

$$
\begin{aligned}
&\mu_i(x_1,\ldots,x_n).\langle t_1,\ldots,t_n\rangle\\
&= \pi_i\ (Y\ (\lambda F.\langle t_1[x_j \leftarrow \pi_j F],\ldots,t_n[x_j \leftarrow \pi_j F]\rangle))\\
&\to \pi_i(\ \langle (t_1[x_j \leftarrow \pi_j F])\,[F \leftarrow Y\ (\lambda F.\langle t_1[x_j \leftarrow \pi_j F],\ldots,t_n[x_j \leftarrow \pi_j F]\rangle)]\\
&\qquad\qquad \vdots\\
&\qquad (t_n[x_j \leftarrow \pi_j F])\,[F \leftarrow Y\ (\lambda F.\langle t_1[x_j \leftarrow \pi_j F],\ldots,t_n[x_j \leftarrow \pi_j F]\rangle)]\rangle)\\
&= (t_i[x_j \leftarrow \pi_j F])\,[F \leftarrow Y\ (\lambda F.\langle t_1[x_j \leftarrow \pi_j F],\ldots,t_n[x_j \leftarrow \pi_j F]\rangle)]\\
&= t_i[x_j \leftarrow \pi_j\, Y\ (\lambda F.\langle t_1[x_j \leftarrow \pi_j F],\ldots,t_n[x_j \leftarrow \pi_j F]\rangle)]\\
&= t_i[x_j \leftarrow \mu_j(x_1,\ldots,x_n).\langle t_1,\ldots,t_n\rangle]
\end{aligned}
$$

The precise description of the extension of the calculus with a cartesian product is unnecessarily complex, so that we just consider its extension with the family of operators (μ_i), regulated by the family of unfolding rules (10.6). To type terms extended with these constructors, we introduce the family of fixpoint rules[2]

$$
Y_\mu\quad \frac{\begin{array}{c}\emptyset \vdash \mu_k(x_1,\ldots,x_n).\langle t_1,\ldots,t_n\rangle : \beta_{j_k} :: \sigma_k \quad \text{for every } 1\le k\le n \text{ and } j_k\in J_k\\ x_1 : \{(c_{j_1},\beta_{j_1}) \mid j_1\in J_1\} :: \sigma_1,\ldots,x_n : \{(c_{j_n},\beta_{j_n}) :: \sigma_n \mid j_n\in J_n\} \vdash t_i : \alpha :: \sigma_i\end{array}}{\emptyset \vdash \mu_i(x_1,\ldots,x_n).\langle t_1,\ldots,t_n\rangle : \alpha :: \sigma_i}
$$

where, for every $1\le j\le n$, σ_j is the simple type of t_j. We call $\mathcal{S}_{fix}(\mathcal{A})$ the resulting extension of the type system $\mathcal{S}(\mathcal{A})$. In this intersection type system, derivations of infinite depth are allowed, and have colored nodes, defined as follows:

- for every occurrence of the rule Y_μ, we assign color c_{j_k} to the node

$$\emptyset \vdash \mu_k(x_1,\ldots,x_n).\langle t_1,\ldots,t_n\rangle : \beta_{j_k} :: \sigma_k$$

- all the other nodes are assigned the neutral color ε.

An infinite derivation tree is then accepted when all its branches are winning for the parity condition, just as we did for semantic run-trees and for derivation trees with the rule Y_σ.

Note that we restrict our attention to empty contexts and thus to closed terms, as our aim is to interpret the rules of higher-order recursion schemes, which have this property. It would be more subtle to reflect in a typing rule the semantics of the fixpoint operator Y on objects obtained as cartesian products in the presence of parameters, as the model could provide contexts for several

[2] As the rule is very large, the hypothesis are superposed, and not written side-by-side as usually.

components of the product at the same time.

The semantics of $\mu_i(x_1, \ldots, x_n).\langle t_1, \ldots, t_n \rangle$ is naturally defined as

$$[\![\mu_i(x_1, \ldots, x_n).\langle t_1, \ldots, t_n \rangle]\!]_{\mathcal{A}} \quad = \quad \pi_i \; (\, Y \; ([\![\lambda F.\langle u_1, \ldots, u_n \rangle]\!]_{\mathcal{A}}) \,)$$

where $u_i = t_i[x_j \leftarrow \pi_j \, F]$.

We devote the remaining of this section to the proof of the following proposition, which is the keystone of the bridge between the semantics of higher-order recursion schemes in the colored Scott model and their typings in $\curlyvee^{st}_{fix}(\mathcal{G}, \mathcal{A})$, as we explain in the next section:

Proposition 34. *Let $t_1, \ldots, t_n$ be λ-terms whose free variables are included in the set $\{x_1, \ldots, x_n\}$. Then we have that*

$$\alpha \; \in \; [\![\mu_i(x_1, \ldots, x_n).\langle t_1, \ldots, t_n \rangle]\!]_{\mathcal{A}}$$

if and only if, denoting σ_i the simple type of t_i,

$$\emptyset \; \vdash \; \mu_i(x_1, \ldots, x_n).\langle t_1, \ldots, t_n \rangle \; : \; \alpha :: \sigma_i$$

is typable by a winning derivation of $\mathcal{S}_{fix}(\mathcal{A})$.

Note that the semantic interpretation in the colored model $\mathbf{ScottL}_{\lambda}$ of the λY-calculus of a higher-order recursion scheme $\mathcal{G}$ over the set of non-terminals $\{F_1 = S, \ldots, F_n\}$ is simply

$$[\![\mathcal{G}]\!]_{\mathcal{A}} \quad = \quad [\![\mu_1(F_1, \ldots, F_n).\langle \mathcal{R}(F_1), \ldots, \mathcal{R}(F_n) \rangle]\!]_{\mathcal{A}}$$

Definition 40. Given a higher-order recursion scheme $\mathcal{G}$ over the set of non-terminals $\{F_1 = S, \ldots, F_n\}$, we set

$$\mu F_i \quad = \quad \mu_i(F_1, \ldots, F_n).\langle \mathcal{R}(F_1), \ldots, \mathcal{R}(F_n) \rangle$$

so that in particular $[\![\mathcal{G}]\!]_{\mathcal{A}} = [\![\mu S]\!]_{\mathcal{A}}$.

A corollary of this proposition is therefore

Corollary 7. *Let $\mathcal{A}$ be an alternating parity tree automaton of set of states Q, and $\mathcal{G}$ be a higher-order recursion scheme $\mathcal{G}$ over the set of non-terminals $\{F_1 = S, \ldots, F_n\}$. Given a state $q \in Q$, we have that*

$$q \; \in \; [\![\mathcal{G}]\!]_{\mathcal{A}}$$

if and only if there is a winning derivation of the sequent

$$\emptyset \; \vdash \; \mu S \; : \; q :: o$$

in $\mathcal{S}_{fix}(\mathcal{A})$.

In order to prove Proposition 34, we first prove a preparatory lemma.

Lemma 26. *Let $t_1, \ldots, t_n$ be terms whose free variables are contained in $\{x_1, \ldots, x_n\}$, and let $i \in \{1, \ldots, n\}$. The sequent*

$$x_1 : \{(c_{j_1}, \beta_{j_1}) \mid j_1 \in J_1\} :: \sigma_1, \ldots, x_n : \{(c_{j_n}, \beta_{j_n}) \mid j_n \in J_n\} :: \sigma_n \vdash t_i : \alpha :: \sigma_i$$

is provable in $\mathcal{S}(\mathcal{A})$ if and only if

$$(\{(c_{j_k}, (k, \beta_{j_k})) \mid 1 \leq k \leq n,\, j_k \in J_k\},\, (i, \alpha)) \;\in\; [\![\lambda F.\, \langle\, t_1[x_j \leftarrow \pi_j\, F], \ldots, t_n[x_j \leftarrow \pi_j\, F]\,\rangle]\!]_{\mathcal{A}}$$

Proof. We have that

$$x_1 : \{(c_{j_1}, \beta_{j_1}) \mid j_1 \in J_1\} :: \sigma_1, \ldots, x_n : \{(c_{j_n}, \beta_{j_n}) \mid j_n \in J_n\} :: \sigma_n \vdash t_i : \alpha :: \sigma_i$$

if and only if

$$\emptyset \vdash \lambda x_1 \ldots, x_n.\, t_i : \{(c_{j_1}, \beta_{j_1})\} \to \cdots \to \{(c_{j_n}, \beta_{j_n})\} \to \alpha :: \sigma_1 \to \cdots \to \sigma_n \to \sigma_i$$

if and only if, by Proposition 32,

$$(\{(c_{j_1}, \beta_{j_1})\}, \ldots, \{(c_{j_n}, \beta_{j_n})\}, \alpha) \;\in\; [\![\lambda x_1 \ldots\, x_n.\, t_i]\!]_{\mathcal{A}} \;\subseteq\; [\![\sigma_1 \to \cdots \to \sigma_n \to \sigma_i]\!]_{!} \tag{10.7}$$

In the semantics, using the usual isomorphisms of linear logic and notably the Seely isomorphisms, we have that

$$\begin{array}{lcl} [\![\sigma_1 \to \cdots \to \sigma_n \to \sigma_i]\!]_{!} & = & !\,[\![\sigma_1]\!]_{!} \multimap \cdots \multimap !\,[\![\sigma_n]\!]_{!} \multimap [\![\sigma_i]\!]_{!} \\ & \cong & (!\,[\![\sigma_1]\!]_{!} \otimes \cdots \otimes !\,[\![\sigma_n]\!]_{!}) \multimap [\![\sigma_i]\!]_{!} \\ & \cong & !\,([\![\sigma_1]\!]_{!} \,\&\, \cdots \,\&\, [\![\sigma_n]\!]_{!}) \multimap [\![\sigma_i]\!]_{!} \end{array}$$

This sequence of isomorphisms maps the element

$$(\{(c_{j_1}, \beta_{j_1})\}, \ldots, \{(c_{j_n}, \beta_{j_n})\}, \alpha)$$

of the semantics to the element

$$(\{(c_{j_k}, (k, \beta_{j_k})) \mid 1 \leq k \leq n,\; j \in J_k\}, \alpha)$$

and conversely. In the syntax, this amounts to "re-encoding" the term $\lambda x_1 \ldots, x_n.\, t_i$ as the term with projections $\lambda F.\, t_i[x_j \leftarrow \pi_j\, F]$, so that (10.7) is equivalent to

$$(\{(c_{j_k}, (k, \beta_{j_k})) \mid 1 \leq k \leq n,\; j \in J_k\}, \alpha) \;\in\; [\![\lambda F.\, t_i[x_j \leftarrow \pi_j\, F]]\!]_{\mathcal{A}} \;\subseteq\; !\left(\bigwith_{j=1}^{n} [\![\sigma_j]\!]_{!}\right) \multimap [\![\sigma_i]\!]_{!}$$

It is then immediate to check that this is equivalent to the fact that

$$(\{(c_{j_k}, (k, \beta_{j_k})) \mid 1 \leq k \leq n,\; j \in J_k\}, (i, \alpha))$$

belongs to

$$[\![\lambda F.\, \langle\, t_1[x_j \leftarrow \pi_j\, F], \ldots, t_n[x_j \leftarrow \pi_j\, F]\,\rangle]\!]_{\mathcal{A}} \;\subseteq\; !\left(\bigwith_{j=1}^{n} [\![\sigma_j]\!]_{!}\right) \multimap \bigwith_{j=1}^{n} [\![\sigma_j]\!]_{!}$$

□

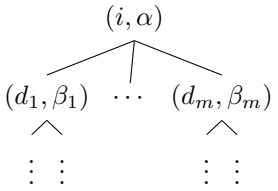
Proof of Proposition 34. We start with the direct implication. Suppose that

$$\begin{array}{rcl} \alpha & \in & [\![\mu_i(x_1,\ldots,x_n).\langle t_1,\ldots,t_n\rangle]\!]_{\mathcal{A}} \\ & = & \pi_i\,(\,Y\,([\![\lambda F.\langle t_1[x_j \leftarrow \pi_j\,F],\ldots,t_n[x_j \leftarrow \pi_j\,F]\rangle]\!]_{\mathcal{A}}\,)\,) \end{array}$$

By definition, this means that

$$(i,\,\alpha)\;\in\;Y\,([\![\lambda F.\langle t_1[x_j \leftarrow \pi_j\,F],\ldots,t_n[x_j \leftarrow \pi_j\,F]\rangle]\!]_{\mathcal{A}}\,)$$

which in turn implies that there exists a winning semantic run-tree of the shape



where the color of the node (d_l,β_l) is c_l, and such that

$$\{\,(c_l,(d_l,\beta_l))\mid 1\leq l\leq m\,\}\;\in\;[\![\lambda F.\langle t_1[x_j \leftarrow \pi_j\,F],\ldots,t_n[x_j \leftarrow \pi_j\,F]\rangle]\!]_{\mathcal{A}}$$

By Lemma 26, we have that the sequent

$$\ldots,\,x_k\,:\,\{(c_l,\beta_l)\mid d_l\,=\,k\}\,::\,\sigma_k,\ldots\;\vdash\;t_i\,:\,\alpha\,::\,\sigma_i$$

is provable in the system $\mathcal{S}(\mathcal{A})$. We obtain the beginning of a derivation tree:

$$\dfrac{\emptyset\;\vdash\;\mu_{d_l}(x_1,\ldots,x_n).\langle t_1,\ldots,t_n\rangle\,:\,\beta_l\,::\,\sigma_{d_l}\quad\text{for every } 1\leq l\leq m \qquad \ldots,\,x_k\,:\,\{(c_l,\beta_l)\mid d_l\,=\,k\}\,::\,\sigma_k,\ldots\;\vdash\;t_i\,:\,\alpha\,::\,\sigma_i}{\emptyset\;\vdash\;\mu_i(x_1,\ldots,x_n).\langle t_1,\ldots,t_n\rangle\,:\,\alpha\,::\,\sigma_i}$$

and we iterate this construction on every leaf

$$\emptyset\;\vdash\;\mu_{d_l}(x_1,\ldots,x_n).\langle t_1,\ldots,t_n\rangle\,:\,\beta_l\,::\,\sigma_{d_l}$$

This builds a potentially infinite derivation tree of $\mathcal{S}_{fix}(\mathcal{A})$, which is winning since every of its infinite branches corresponds to an infinite branch of the winning semantic run-tree we considered, and since precisely the same sequence of colors can be read along it.

Conversely, consider a winning derivation of the sequent

$$\emptyset\;\vdash\;\mu_i(x_1,\ldots,x_n).\langle t_1,\ldots,t_n\rangle\,:\,\alpha\,::\,\sigma_i$$

in $\mathcal{S}_{fix}(\mathcal{A})$. It necessarily starts as

$$\dfrac{\emptyset\;\vdash\;\mu_k(x_1,\ldots,x_n).\langle t_1,\ldots,t_n\rangle\,:\,\beta_{j_k}\,::\,\sigma_k\quad\text{for every } 1\leq k\leq n \text{ and } j_k\in J_k \qquad x_1\,:\,\{(c_{j_1},\beta_{j_1})\mid j_1\in J_1\}\,::\,\sigma_1,\ldots,x_n\,:\,\{(c_{j_n},\beta_{j_n})\,::\,\sigma_n\mid j_n\in J_n\}\;\vdash\;t_i\,:\,\alpha\,::\,\sigma_i}{\emptyset\;\vdash\;\mu_i(x_1,\ldots,x_n).\langle t_1,\ldots,t_n\rangle\,:\,\alpha\,::\,\sigma_i}$$

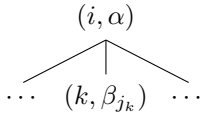
and we can apply Lemma 26 to the sequent

$$x_1 : \{(c_{j_1}, \beta_{j_1}) \mid j_1 \in J_1\} :: \sigma_1, \ldots, x_n : \{(c_{j_n}, \beta_{j_n}) :: \sigma_n \mid j_n \in J_n\} \vdash t_i : \alpha :: \sigma_i$$

from which we deduce that

$$(\, \{(c_{j_k}, (k, \beta_{j_k})) \mid 1 \leq k \leq n,\, j_k \in J_k\}\,,\, (i, \alpha)\,) \;\in\; [\![\lambda F.\, \langle\, t_1[x_j \leftarrow \pi_j\, F], \ldots, t_n[x_j \leftarrow \pi_j\, F]\,\rangle]\!]_{\mathcal{A}}$$

We use this information to start building a semantic run-tree of $[\![\lambda F.\, \langle\, t_1[x_j \leftarrow \pi_j\, F], \ldots, t_n[x_j \leftarrow \pi_j\, F]\,\rangle]\!]_{\mathcal{A}}$:



where we attribute the color c_{j_k} to the node (k, β_{j_k}). We iterate this construction process, and create in this way a semantic run-tree of $[\![\lambda F.\, \langle\, t_1[x_j \leftarrow \pi_j\, F], \ldots, t_n[x_j \leftarrow \pi_j\, F]\,\rangle]\!]_{\mathcal{A}}$. This run-tree is winning, as the colorings of its infinite branches are the same as the ones of the derivation tree we consider. We then obtain that

$$\alpha \;\in\; [\![\mu_i(x_1, \ldots, x_n).\, \langle\, t_1, \ldots, t_n\,\rangle]\!]_{\mathcal{A}}$$

simply by applying the projection π_i.

10.6 Decidability of the local higher-order model-checking problem

In this section, we prove the decidability of two of the higher-order model-checking problems stated in §3.4: the local higher-order model-checking problem, and the selection problem, both in their automata-theoretic formulation. Recall that the third problem, namely global higher-order model-checking, is a consequence of the selection problem. These two decidability results are obtained as a consequence of a semantic soundness-and-completeness theorem, analogous to Theorem 19, and of the finiteness of the colored Scott semantics.

We start by extending the connection between derivations of terms in β-normal η-long forms in the type systems $\boldsymbol{\gamma}^{st}(\mathcal{A})$ and $\mathcal{S}(\mathcal{A})$ studied in §10.3 to the type systems $\boldsymbol{\gamma}^{st}_{fix}(\mathcal{G}, \mathcal{A})$ and $\mathcal{S}_{fix}(\mathcal{A})$. We extend the notion of derivation *without subtyping* to the system $\mathcal{S}_{fix}(\mathcal{A})$ by formulating exactly the same restrictions on the rules Axiom and δ. The notion of β-normal η-long form is adapted to recursion schemes as follows:

Definition 41 (β-normal η-long form of a HORS)**.** Let $\mathcal{G} = \langle \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$ be a recursion scheme. We define its β-normal η-long form as the higher-order recursion scheme $\mathcal{G}^{\beta\eta} = \langle \Sigma, \mathcal{N}, \mathcal{R}^{\beta\eta}, S \rangle$ which only differs of $\mathcal{G}$ by the fact that $\mathcal{R}^{\beta\eta}$ maps any non-terminal $F \in \mathcal{N}$ to the β-normal η-long form of the term $\mathcal{R}(F)$.

We say that $\mathcal{G}$ is in β-normal η-long form precisely when $\mathcal{G} = \mathcal{G}^{\beta\eta}$.

$\mathcal{G}$ and $\mathcal{G}^{\beta\eta}$ generate the same tree, so that we can harmlessly consider $\mathcal{G}^{\beta\eta}$ instead of $\mathcal{G}$ in the sequel:

Proposition 35. $\langle\, \mathcal{G} \,\rangle \;=\; \langle\, \mathcal{G}^{\beta\eta} \,\rangle$.

Theorem 28 adapts in this case to:

Proposition 36. *Let $\mathcal{G} = \langle \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$ be a higher-order recursion scheme and $\mathcal{G}^{\beta\eta} = \langle \Sigma, \mathcal{N}, \mathcal{R}^{\beta\eta}, S \rangle$ be its β-normal η-long form. Suppose that $\mathcal{N} = \{F_1, \ldots, F_n\}$. Then*

$$\emptyset \;\vdash\; \mu_i(F_1, \ldots, F_n).\, \langle\, \mathcal{R}(F_1), \ldots, \mathcal{R}(F_n) \,\rangle \;:\; \alpha \;::\; \kappa(F_i)$$

is provable in $\mathcal{S}_{fix}(\mathcal{A})$ if and only if

$$\emptyset \;\vdash\; \mu_i(F_1, \ldots, F_n).\, \langle\, \mathcal{R}^{\beta\eta}(F_1), \ldots, \mathcal{R}^{\beta\eta}(F_n) \,\rangle \;:\; \alpha \;::\; \kappa(F_i)$$

is provable without subtyping in $\mathcal{S}_{fix}(\mathcal{A})$. Moreover, there exists a winning derivation of one of these sequents if and only if there exists one of the other.

Proof. A derivation of

$$\emptyset \;\vdash\; \mu_i(F_1, \ldots, F_n).\, \langle\, \mathcal{R}(F_1), \ldots, \mathcal{R}(F_n) \,\rangle \;:\; \alpha \;::\; \kappa(F_i) \tag{10.8}$$

is of the shape

$$\frac{\begin{array}{c}\emptyset \;\vdash\; \mu_k(F_1, \ldots, F_n).\, \langle\, \mathcal{R}(F_1), \ldots, \mathcal{R}(F_n) \,\rangle \;:\; \beta_{j_k} \;::\; \kappa(F_k) \quad \text{for every } 1 \leq k \leq n \text{ and } j_k \in J_k \\ F_1 \;:\; \{(c_{j_1}, \beta_{j_1}) \mid j_1 \in J_1\} \;::\; \kappa(F_1), \ldots, F_n \;:\; \{(c_{j_n}, \beta_{j_n}) \;::\; \kappa(F_n) \mid j_n \in J_n\} \;\vdash\; \mathcal{R}(F_i) \;:\; \alpha \;::\; \kappa(F_i)\end{array}}{\emptyset \;\vdash\; \mu_i(F_1, \ldots, F_n).\, \langle\, \mathcal{R}(F_1), \ldots, \mathcal{R}(F_n) \,\rangle \;:\; \alpha \;::\; \kappa(F_i)}$$

By Theorem 28, the sequent

$$F_1 \;:\; \{(c_{j_1}, \beta_{j_1}) \mid j_1 \in J_1\} \;::\; \sigma_1, \ldots, F_n \;:\; \{(c_{j_n}, \beta_{j_n}) \;::\; \sigma_n \mid j_n \in J_n\} \;\vdash\; \mathcal{R}(F_i) \;:\; \alpha \;::\; \sigma_i$$

can be translated into a proof without subtyping in $\mathcal{S}(\mathcal{A})$ of the sequent

$$F_1 \;:\; \{(c_{j_1}, \beta_{j_1}) \mid j_1 \in J_1\} \;::\; \sigma_1, \ldots, F_n \;:\; \{(c_{j_n}, \beta_{j_n}) \;::\; \sigma_n \mid j_n \in J_n\} \;\vdash\; \mathcal{R}^{\beta\eta}(F_i) \;:\; \alpha \;::\; \sigma_i$$

We replace in the proof of (10.8) the first derivation by the second, and iterate the process on the nodes

$$\emptyset \;\vdash\; \mu_k(F_1, \ldots, F_n).\, \langle\, \mathcal{R}(F_1), \ldots, \mathcal{R}(F_n) \,\rangle \;:\; \beta_{j_k} \;::\; \kappa(F_k)$$

We obtain in this way a proof without subtyping of

$$\emptyset \;\vdash\; \mu_i(F_1, \ldots, F_n).\, \langle\, \mathcal{R}^{\beta\eta}(F_1), \ldots, \mathcal{R}^{\beta\eta}(F_n) \,\rangle \;:\; \alpha \;::\; \kappa(F_i)$$

The converse direction proceeds similarly. Note that the translation does not affect the coloring of nodes.

□

Recall the translations $(\!|\, \cdot \,|\!)$ and $|\!(\, \cdot \,)\!|$ between colored sets and colored intersection types of §10.3. We may now adapt Corollary 6 to higher-order recursion schemes:

Proposition 37. *Let $\mathcal{G}$ be a higher-order recursion scheme in β-normal η-long form, whose set of non-terminals is $\{F_1, \ldots, F_n\}$. Then the sequent*

$$\emptyset \ \vdash \ \mu F_i \ : \ \alpha \ :: \ \kappa(F_i)$$

is provable in $\mathcal{S}_{fix}(\mathcal{A})$ if and only if the sequent

$$F_i \ : \ \bigwedge_{\{1\}} \ \boxdot_\varepsilon \ (\!|\, \alpha \,|\!) \ :: \ \kappa(F_i) \ \vdash \ F_i \ : \ (\!|\, \alpha \,|\!) \ :: \ \kappa(F_i)$$

is provable in $\boldsymbol{\gamma}^{st}_{fix}(\mathcal{G}, \mathcal{A})$. Moreover, there is a winning derivation of the first sequent in $\mathcal{S}_{fix}(\mathcal{A})$ if and only if there is a winning derivation of the second in $\boldsymbol{\gamma}^{st}_{fix}(\mathcal{G}, \mathcal{A})$.

Proof. The proof of the direct implication proceeds by translation of a derivation π of

$$\emptyset \ \vdash \ \mu F_i \ : \ \alpha \ :: \ \kappa(F_i)$$

in $\mathcal{S}_{fix}(\mathcal{A})$ to a derivation $\{\!|\, \pi \,|\!\}$ of

$$F_i \ : \ \bigwedge_{\{1\}} \ \boxdot_\varepsilon \ (\!|\, \alpha \,|\!) \ :: \ \kappa(F_i) \ \vdash \ F_i \ : \ (\!|\, \alpha \,|\!) \ :: \ \kappa(F_i)$$

in $\boldsymbol{\gamma}^{st}_{fix}(\mathcal{G}, \mathcal{A})$. We define the translation $\{\!|\, \pi \,|\!\}$ as follows. The derivation π in $\mathcal{S}_{fix}(\mathcal{A})$ is of the shape

$$\dfrac{\begin{array}{c} \pi_{j_k} \\ \vdots \\ \emptyset \ \vdash \ \mu_k(F_1, \ldots, F_n).\, \langle\, \mathcal{R}(F_1), \ldots, \mathcal{R}(F_n) \,\rangle \ : \ \beta_{j_k} \ :: \ \kappa(F_k) \quad \text{for every } 1 \leq k \leq n \text{ and } j_k \in J_k \\ F_1 \ : \ \{(c_{j_1}, \beta_{j_1}) \mid j_1 \in J_1\} \ :: \ \sigma_1, \ldots, \ F_n \ : \ \{(c_{j_n}, \beta_{j_n}) \ :: \ \sigma_n \mid j_n \in J_n\} \ \vdash \ \mathcal{R}(F_i) \ : \ \alpha \ :: \ \sigma_i \end{array}}{\emptyset \ \vdash \ \mu_i(F_1, \ldots, F_n).\, \langle\, \mathcal{R}(F_1), \ldots, \mathcal{R}(F_n) \,\rangle \ : \ \alpha \ :: \ \sigma_i}$$

and we can use Corollary 6 to translate the subproof of

$$F_1 \ : \ \{(c_{j_1}, \beta_{j_1}) \mid j_1 \in J_1\} \ :: \ \sigma_1, \ldots, \ F_n \ : \ \{(c_{j_n}, \beta_{j_n}) \ :: \ \sigma_n \mid j_n \in J_n\} \ \vdash \ \mathcal{R}(F_i) \ : \ \alpha \ :: \ \sigma_i$$

into a proof $\pi_{\mathcal{R}}$ of $\boldsymbol{\gamma}^{st}(\mathcal{A})$. We then define $\{\!|\, \pi \,|\!\}$ as:

$$\dfrac{\dfrac{\begin{array}{c} \{\!|\, \pi_{j_k} \,|\!\} \\ \vdots \\ F_k \ : \ \boxdot_\varepsilon \ (\!|\, \beta_{j_k} \,|\!) \ \vdash \ F_k \ : \ (\!|\, \beta_{j_k} \,|\!) \end{array}}{fix} \qquad \dfrac{\begin{array}{c} \{\!|\, \pi_{j_{k'}} \,|\!\} \\ \vdots \\ F_{k'} \ : \ \boxdot_\varepsilon \ (\!|\, \beta_{j_{k'}} \,|\!) \ \vdash \ F_{k'} \ : \ (\!|\, \beta_{j_{k'}} \,|\!) \end{array}}{fix}}{\begin{array}{c} \pi_{\mathcal{R}} \\ \ldots, F_k \ : \ \bigwedge_{j_k \in J_k} \ \boxdot_{c_{j_k}} \ (\!|\, \beta_{j_k} \,|\!), \ldots \ \vdash \ \mathcal{R}(F_i) \ : \ (\!|\, \alpha \,|\!) \end{array}}{fix}$$

$$\dfrac{\ldots, F_k \ : \ \bigwedge_{j_k \in J_k} \ \boxdot_{c_{j_k}} \ (\!|\, \beta_{j_k} \,|\!), \ldots \ \vdash \ \mathcal{R}(F_i) \ : \ (\!|\, \alpha \,|\!)}{F_i \ : \ \bigwedge_{\{1\}} \ \boxdot_\varepsilon \ (\!|\, \alpha \,|\!) \ \vdash \ F_i \ : \ (\!|\, \alpha \,|\!)}$$

Note that we replace the Axiom leaves of $\pi_{\mathcal{R}}$ introducing non-terminals by appropriate instances of the *fix* rule.

By Proposition 16, the maximal color seen along the finite path leading from the root of $\{\!|\, \pi \,|\!\}$ to the node

$$F_k \,:\, \circledcirc_\varepsilon \, (\!|\, \beta_{j_k} \,|\!) \;\vdash\; F_k \,:\, (\!|\, \beta_{j_k} \,|\!)$$

is precisely the color c_{j_k}, which is also the one of the node

$$\emptyset \;\vdash\; \mu_k(F_1, \ldots, F_n).\, \langle\, \mathcal{R}(F_1), \ldots, \mathcal{R}(F_n) \,\rangle \,:\, \beta_{j_k} \,::\, \kappa(F_k)$$

in π. From this observation, it is easy to relate the infinite branches of both π and $\{\!|\, \pi \,|\!\}$ and to notice that they have the same maximal color seen infinitely often.

The converse direction of the proof proceeds similarly, by the reverse translation of a proof of $\boldsymbol{\gamma}^{st}_{fix}(\mathcal{G}, \mathcal{A})$ into a proof of $\mathcal{S}_{fix}(\mathcal{A})$. □

A consequence of these results is the following theorem of "semantic soundness-and-completeness", which establishes a perfect correspondence between our finitary interpretation $[\![\mathcal{G}]\!]_{\mathcal{A}}$ of the higher-order recursion scheme $\mathcal{G}$ in the Scott semantics, and the set of accepting states of the automaton $\mathcal{A}$:

Theorem 32. *An alternating parity tree automaton $\mathcal{A}$ has an accepting run-tree with initial state q_0 over the value tree $\langle\, \mathcal{G} \,\rangle$ of a productive higher-order recursion scheme $\mathcal{G}$ if and only if $q_0 \in [\![\mathcal{G}]\!]_{\mathcal{A}}$.*

Proof. Let $\mathcal{G} = \langle \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$ be a higher-order recursion scheme and $\mathcal{G}^{\beta\eta} = \langle \Sigma, \mathcal{N}, \mathcal{R}^{\beta\eta}, S \rangle$ be its β-normal η-long form. Suppose that $\mathcal{N} = \{F_1 = S, \ldots, F_n\}$. By Corollary 7, we have that $q_0 \in [\![\mathcal{G}]\!]_{\mathcal{A}}$ if and only if there is a winning derivation of the sequent

$$\emptyset \;\vdash\; \mu_1(F_1, \ldots, F_n).\, \langle\, \mathcal{R}(F_1), \ldots, \mathcal{R}(F_n) \,\rangle \,:\, q_0 \,::\, o$$

in $\mathcal{S}_{fix}(\mathcal{A})$. By Proposition 36, this is equivalent to the existence of a winning derivation of the sequent

$$\emptyset \;\vdash\; \mu_1(F_1, \ldots, F_n).\, \langle\, \mathcal{R}^{\beta\eta}(F_1), \ldots, \mathcal{R}^{\beta\eta}(F_n) \,\rangle \,:\, q_0 \,::\, o$$

proved without subtyping in $\mathcal{S}_{fix}(\mathcal{A})$. By Proposition 37, this is equivalent to the fact that the sequent

$$S \,:\, \bigwedge_{\{1\}} \circledcirc_\varepsilon \; q \,::\, o \;\vdash\; S \,:\, q \,::\, o$$

has a winning proof in the type system $\boldsymbol{\gamma}^{st}_{fix}(\mathcal{G}^{\beta\eta}, \mathcal{A})$. This is equivalent to the fact that Eve has a winning strategy in the game $\mathsf{Adamic}^{st}(\mathcal{G}^{\beta\eta}, \mathcal{A})$, and, by Theorem 20, this is in turn equivalent to the fact that $\mathcal{A}$ has an accepting run-tree with initial state q_0 over the value tree $\langle\, \mathcal{G}^{\beta\eta} \,\rangle$. We conclude using Proposition 35, which states that $\langle\, \mathcal{G} \,\rangle = \langle\, \mathcal{G}^{\beta\eta} \,\rangle$. □

Note that it would be interesting to obtain a more general and dual version of Theorem 32, in the spirit of the theorems of §9.2 and of Conjecture 1. In this generalization relying on Church encodings, the interpretation of higher-order recursion schemes (and of the associated λY-terms) would be independent of the automaton of interest. However, as explained earlier, a difficulty here is that considering the terminals of Σ as variables and no longer as constants requires to use parameters when computing the fixpoint $\mathbf{Y}$ in the model. In the presence of subtyping and of cartesian products, this seems to forbid the use of typing rules for the operators μ_i, as parameters coming from different components of a cartesian product could be present in the model. This would be an obstruction to their presentation as the context of only one rule μF_i, which supposes a projection on only one component of the product.

By Corollary 7, checking whether $q_0 \in [\![\mathcal{G}]\!]_{\mathcal{A}}$ is equivalent to checking whether there exists a derivation of the sequent $\emptyset \vdash \mu S : q_0 :: o$ in the colored intersection type system $\mathcal{S}_{fix}(\mathcal{A})$. Since the interpretation of simple types in **FinScottL** is finite, only finitely many intersection types and contexts may occur in such a derivation. Hence, searching for a derivation of the sequent $\emptyset \vdash \mu S : q_0 :: o$ reduces in this case to solving a finite parity game whose nodes are precisely all the sequents that may appear in a derivation tree. Since the winner of a parity game is decidable by Theorem 5, we immediately obtain:

Corollary 8. *The local model-checking problem is decidable.*

Complexity. The complexity of local higher-order model-checking already appears in Ong's first proof of decidability [Ong06] and is, as we explained earlier, n-EXPTIME, where n is the order of the higher-order recursion scheme of interest. It makes sense to give a bit more details: in his article, the complexity is

$$exp_n\left(O\left(|\mathcal{G}^t| \cdot |Q| \cdot |\Omega(Q)|\right)\right)$$

where $|\mathcal{G}^t|$ is the size of the HORS obtained from $\mathcal{G}$ by, roughly speaking, taking its β-normal η-long form[3], $|Q|$ is the size of the set of states of the automaton of interest, and $|\Omega(Q)|$ the one of its set of colors. The function exp_n corresponds to the iteration of n applications of the exponential.

The problem can also be solved by working directly with the recursion scheme of interest, as in [KO09] for instance, where the complexity is

$$O\left(|\mathcal{R}|^{1+c|\Omega(Q)|} \cdot exp_n\left(O\left(A \cdot |Q| \cdot |\Omega(Q)|\right)\right)\right)$$

Here, A is the maximal arity of a non-terminal of $\mathcal{N}$ or of a terminal of Σ, Q is the set of states of the automaton of interest, $|\mathcal{R}|$ the number of rewrite rules of the recursion scheme, and c is a constant whose value is approximately $\frac{1}{3}$ and which comes from the application of Schewe's algorithm for solving parity games [Sch07]. Interestingly, this complexity result refines Ong's original one by several aspects:

[3] In fact, Ong also inserts special symbols @ to freeze redexes, but this does not essentially changes the size of the scheme. See [Ong06] or [Gre10] for more details on the transformation and its importance in Ong's approach, based on game semantics.

- it only considers the size of the higher-order recursion scheme, computed as its number of rules: this is notably independent on whether we consider it in β-normal η-long form, or in the slightly different frozen form originally introduced by Ong,
- and it makes clearly appear that the problem is *polynomial* in the size of the higher-order recursion scheme, and that its exponential component only depends on the maximal arity of a symbol and on the size of the automaton of interest. Roughly speaking,

  the local higher-order model-checking problem is polynomial in the size of the higher-order recursion scheme, and n-exponential on the size of the alternating parity automaton of interest.

  This explains why the several practical model-checkers designed in higher-order model-checking can be used in practice in spite of the seemingly huge complexity of the problem. It should be noted that the first tools, namely TRecS [Kob09a], GTRecS [Kob11], GTRecS2, C-SHORe [BCHS13], and TravMC [NRO12] only considered properties specified by tree automata with trivial acceptance conditions.

  However, the tools TRecS-APT [FIK13] and TravMC2 [NO14] consider all alternating *parity* automata and therefore support the verification of all properties expressible in the bisimulation-invariant fragment of monadic second-order logic.

Our approach allows to find a similar complexity: deciding the problem amounts to solving a parity game over the fragment of the model allowing to interpret all subterms of the recursion scheme, so that we can restrict on the interpretation of types up to some fixed arity and order. This gives a finite parity game, which is n-exponential in the maximal arity occurring in the HORS and in the size of the set of states of the automaton: recall that we can understand the interpretation of a type as the set of colored, idempotent intersection types refining it. The solution of the game is then in $NP \cap coNP$, so that the problem is in n-EXPTIME. An interesting connection with linear logic could be investigated here, as the intersection operator corresponds to the exponential modality. The order of a HORS indeed corresponds to the maximal depth of exponential boxes nested between fixpoint calls, and it would be nice to relate this to characterizations in light linear logic or in elementary linear logic of the n-EXPTIME complexity class, as investigated for instance by Baillot in [Bai11].

Extensional collapses. Since we explained in §5.4 that the Scott model of linear logic is the extensional collapse of its relational semantics, as proved in [Ehr12b], this relation has been guiding our extension of both semantics with coloring, and countable multiplicities in the quantitative case of the relational model. Note in particular that the definition of the coloring comonad $\Box$ in the Scott semantics is precisely obtained as a collapse of its relational counterpart: by replacing multisets with sets, and by saturating the relation to obtain a downward-closed one. We believe that the models we obtained are related by an extensional collapse relation:

Conjecture 2. *We conjecture that the notion of extensional collapse formulated by Ehrhard [Ehr12b] for cartesian closed categories from Bucciarelli's axiomatization [Buc97] can be extended to such categories with a Conway operator, in such a way that the colored Scott model of linear logic is the extensional collapse of its colored, infinitary relational semantics.*

Note that a corollary of this conjecture would be that Conjecture 1 holds. In a first time, we believe that it would be interesting to take a syntactic approach to understand the problem better, and to start from $\mathcal{S}_{fix}(\mathcal{A})$ together with its non-idempotent counterpart – with infinite multiplicities. We believe that the latter system can be used to capture the infinitary colored relational semantics of λY-terms. Focusing on β-normal η-long forms allows not to consider subtyping in the idempotent system, and investigating the collapse would probably reduce in this type-theoretic setting to relating non-idempotent intersection types with the idempotent intersection types obtained by forgetting the multiplicities of use of types.

10.7 Decidability of the selection problem

Now that we obtained the decidability of the local higher-order model-checking problem, we can go further and prove that the selection problem defined by Carayol and Serre in [CS12] is decidable. We focus on its automata-theoretic reformulation due to Haddad [Had13a]:

> *Given a higher-order regular tree $\langle\, \mathcal{G} \,\rangle$ computed by a higher-order recursion scheme $\mathcal{G}$, an alternating parity automaton $\mathcal{A}$, and a state q of this automaton from which $\langle\, \mathcal{G} \,\rangle$ is accepted, can we compute a higher-order recursion scheme $\mathcal{G}_q$ producing a winning run-tree $\langle\, \mathcal{G}_q \,\rangle$ of $\mathcal{A}$ from q over $\langle\, \mathcal{G} \,\rangle$?*

We obtain a new proof of the decidability of this problem, which appears already in [CS12] and in [Had13a,Had13b]. A difference in our approach is that we consider *alternating* parity tree automata, and not just non-deterministic ones. While these two classes of automata are equivalent by Theorem 4, our approach based on linear logic makes it natural to consider alternating automata.

Theorem 33. *The selection problem is decidable.*

Proof. To prove the decidability of the local higher-order model-checking problem (Corollary 8), we used Theorem 5, which implies the existence of a *memoryless* winning strategy on the finite parity game used to build a winning derivation tree typing $\mathcal{G}$ with the initial state q_0. In this setting, winning strategies correspond to winning derivation trees of the intersection type system $\mathcal{S}_{fix}(\mathcal{A})$, and memoryless strategies correspond to derivation trees admitting a finite representation using backtracking pointers.

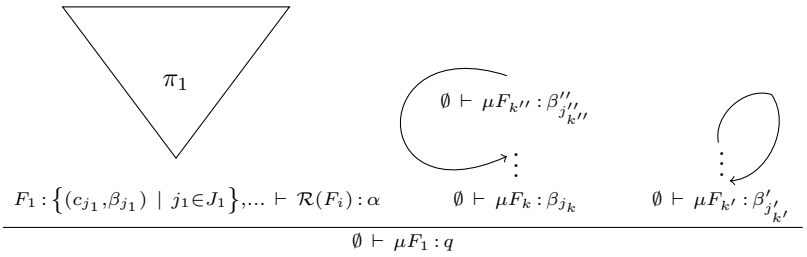
Let $\mathcal{G} = \langle \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$ be a productive higher-order recursion scheme, $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, \Omega \rangle$ be an alternating parity tree automaton, and $q \in Q$ be a state

from which $\langle\, \mathcal{G} \,\rangle$ is accepted by $\mathcal{A}$. We write $\mathcal{N} \;=\; \{\, S = F_1, \ldots, F_n \,\}$. By Theorem 32 and Corollary 7, there exists a winning derivation of the sequent

$$\emptyset \;\vdash\; \mu F_1 \;:\; q \;::\; o \tag{10.9}$$

in $\mathcal{S}_{fix}(\mathcal{A})$. As we just explained, Theorem 5 allows to deduce the existence of regular derivation of the sequent (10.9) — this meaning that this derivation can be obtained as the unfolding of a finite derivation with backtracking pointers.

We fix such a finite representation π. It is typically of the following form:



(10.10)

We suppose that each occurrence of a sequent

$$\emptyset \;\vdash\; \mu F_k \;:\; \beta_k \;::\; \kappa(F_k) \tag{10.11}$$

in π is precisely located by an integer we refer to as *occ*. This allows us to distinguish between different nodes of π labeled with a same sequent (10.11). In (10.11), we say that F_k has type β_k in *occ*.

We now explain how we define the annotated higher-order recursion scheme

$$\mathcal{G}_q \;\;=\;\; \langle \Sigma_q, \mathcal{N}_q, \mathcal{R}_q, S^q \rangle$$

Note that this definition depends on the choice of π, even though we do not write it explicitly to lighten notations. We define

$$\Sigma_q \;\;=\;\; \left\{\, a^\alpha \mid \alpha \;=\; u_1 \to \cdots \to u_{\mathrm{ar}(a)} \to q \;::\; o^{\mathrm{ar}(a)} \to o \,\right\}$$

We then set

$$\mathcal{N}_q \;\;\;=\;\;\; \left\{\, F^\beta_{i,occ} \mid F_i \in \mathcal{N} \text{ and } F_i \text{ has type } \beta \text{ in } occ \,\right\}$$

Let $F^\beta_{i,occ} \;\in\; \mathcal{N}_q$. Suppose that

$$\beta \;\;=\;\; \bigwedge_{i_1 \in I_1} \sigma_{1 i_1} \to \cdots \to \bigwedge_{i_n \in I_n} \sigma_{1 i_n} \to q \;\;::\;\; \kappa_1 \to \cdots \to \kappa_n \to o \tag{10.12}$$

then

$$\kappa(F^\beta_{i,occ}) \;\;=\;\; \underbrace{\kappa_1 \to \cdots \to \kappa_1}_{|I_1| \text{ times}} \to \cdots \to \underbrace{\kappa_n \to \cdots \to \kappa_n}_{|I_n| \text{ times}} \to o$$

Accordingly, we label the variables in $\mathcal{G}_q$: they will be of the form φ^β, where φ is a variable of the higher-order recursion scheme $\mathcal{G}$ and β is as in (10.12). The variable φ^β has the same simple type as $F^\beta_{i,occ}$.

It remains to define the set $\mathcal{R}_q$ of rewrite rules for $\mathcal{G}_q$. We first introduce the term transformation $\{\!|\ \cdot\ |\!\}^\alpha_{\pi,occ}$, which maps a λ-term without abstractions and of set of constants $\Sigma \uplus \mathcal{N}$ to a λ-term without abstractions and of set of constants $\Sigma_q \uplus \mathcal{N}_q$. Consider the occurrence occ of a non-terminal F_i. In π, it occurs as in (10.10), and it introduces the term $t = \mathcal{R}(F_i)$, itself typed by the subproof π_1 rooted at the left successor of the node corresponding to occ. The transformation $\{\!|\ \cdot\ |\!\}^\alpha_{\pi,occ}$ rewrites t as follows:

- $\{\!|\, F_i \,|\!\}^\alpha_{\pi,occ} \;=\; F^\alpha_{i,occ'}$, where occ' is the occurrence corresponding to the non-terminal of interest in the proof π,
- $\{\!|\, a \,|\!\}^\alpha_{\pi,occ} \;=\; a^\alpha$,
- $\{\!|\, x \,|\!\}^\alpha_{\pi,occ} \;=\; x^\alpha$,
- $\{\!|\, M\ N \,|\!\}^\beta_{\pi,occ} \;=\; \{\!|\, M \,|\!\}^{\{(c_1,\alpha_1),\dots,(c_n,\alpha_n)\}\to\beta}_{\pi,occ}\ \{\!|\, N \,|\!\}^{\alpha_1}_{\pi,occ}\ \cdots\ \{\!|\, N \,|\!\}^{\alpha_n}_{\pi,occ}$ where $\{(c_1,\alpha_1),\dots,(c_n,\alpha_n)\}\to\beta$ is the type of M in the subproof π_1.

Now, for every occurrence occ of a non-terminal F_i with type

$$\beta \;=\; \{(c_{11},\alpha_{11}),\dots,(c_{1k_1},\alpha_{1k_1})\}\to\cdots\to\{(c_{n1},\alpha_{n1}),\dots,(c_{nk_n},\alpha_{nk_n})\}\to q$$

we define the corresponding rewriting rule as follows:

$$F^\beta_{i,occ} \;=\; \lambda x_1^{\alpha_{11}}.\ \dots\ \lambda x_1^{\alpha_{1k_1}}.\ \dots\ \lambda x_n^{\alpha_{n1}}.\ \dots\ \lambda x_n^{\alpha_{nk_n}}.\, \{\!|\, \mathcal{R}(F_i) \,|\!\}^q_{\pi,occ}$$

To ease reading, we sometimes use a vectorial notation for terms: the rewriting rule associated with $F^\beta_{i,occ}$ is then written

$$F^\beta_{i,occ} \;=\; \lambda\vec{x_1}.\ \dots\ \lambda\vec{x_n}.\, \{\!|\, \mathcal{R}(F_i) \,|\!\}^q_{\pi,occ}$$

and in the definition of $\{\!|\ \cdot\ |\!\}^\alpha_{\pi,occ}$, we write

$$\{\!|\, M\ N \,|\!\}^\beta_{\pi,occ} \;=\; \{\!|\, M \,|\!\}^{\{(c_1,\alpha_1),\dots,(c_n,\alpha_n)\}\to\beta}_{\pi,occ}\ \overrightarrow{\{\!|\, N \,|\!\}^{\alpha_i}_{\pi,occ}}$$

With these notations, the rules of $\mathcal{R}_q$ appear as a « vectorialization » of the rules of $\mathcal{R}$. In fact the only change is that, following the intuition that the intersection operator on types lifts to higher-order the alternating behavior of APT, we perform the duplications of the APT directly in the terms defining $\mathcal{G}$, by using the information on duplication contained by the proof π.

We claim that $\langle\, \mathcal{G}_q \,\rangle$ is a winning run-tree of $\mathcal{A}$ over $\langle\, \mathcal{G} \,\rangle$, with the mild difference that symbols are annotated with transitions of $\mathcal{A}$ rather than simply with states. It suffices to change the labels of the symbols of Σ in $\mathcal{G}_q$ and only to remember the return state of the transitions decorating the symbols to obtain a run-tree. Another way, due to Haddad [Had13b], is to turn every labeled terminal a^α into the non-terminal A^α, with the rewriting rule $A^\alpha \to a^q$, where

$$\alpha \;=\; \beta_1\to\cdots\to\beta_n\to q.$$

□

Chapter 11

Contributions and perspectives

The purpose of this thesis was to connect higher-order model-checking to a series of advanced ideas in contemporary semantics, like linear logic and its relational semantics, indexed linear logic, distributive laws, parametric comonads and tensorial logic. All these ingredients met and combined surprisingly well with higher-order model-checking. The approach revealed in particular that the traditional treatment of inductive-coinductive reasoning based on colors is secretly founded on the same comonadic principles as the exponential modality of linear logic. This discovery led us to the definition of both qualitative and quantitative denotational models of linear logic in which higher-order recursion schemes can be interpreted in a way that is for the first time meaningful to higher-order model-checking. The correspondence between our finitary and colored Scott-relational interpretation of linear logic and higher-order model-checking is established in two main technical steps: first, we identify our finitary interpretation to a modal intersection type system which improves the system originally defined by Kobayashi and Ong; then, we adapt to our modal intersection type system the soundness-and-completeness proof by Kobayashi and Ong in order to obtain the expected correspondence. We obtain in this way a model in which the interpretation of a higher-order recursion scheme with respect to a given alternating parity tree automaton is precisely the set of states from which this automaton accepts the infinite value tree of the scheme. The finiteness of the semantics permitted us to solve not only the local model-checking problem, but also the selection problem.

To conclude this thesis, we list our contributions and the potential research directions that arise from the semantic analysis of higher-order model-checking conducted in this document. Most of these contributions appear in our articles with Melliès [GM15a, GM15b, GM15c, GM15d]. We present them following their order of introduction in the manuscript.

We start this thesis with an essentially pedagogical contribution, which consists in the exposition of the core ingredients of higher-order model-checking in Part I. Our aim in this part, and mainly in the first two chapters of this thesis, was to give a self-contained introduction to alternating parity tree automata and related logics over trees, as well as to higher-order recursion schemes and to the equivalent λ-terms with recursion, in order to state the three main problems

of higher-order model-checking: local and global model-checking, and selection.

A coinductive approach to higher-order model-checking. Our first, yet very preliminary scientific contribution is presented in Chapter 4, in which we introduce coinduction quite informally, and give a coinductive framework for the evaluation of higher-order recursion schemes adapted from the one of Czajka [Cza15] for infinitary λ-terms, and which is also related to Endrullis and Polonsky's approach [EP11]. We formulate this framework from the idea that HORS, being equivalent to λY-terms, can be understood as *regular* simply-typed infinitary λ-terms over a signature of constants. Their normalization can therefore be studied using the coinductive concepts considered in infinitary rewriting theory, instead of using the traditional supremum of finite approximations of the value tree.

Given a HORS $\mathcal{G}$ and an alternating tree automaton $\mathcal{A}$, we also define a non-deterministic coinductive rewriting relation $\rightarrow^{\infty}_{\mathcal{G},\mathcal{A}}$ which computes the run-trees of $\mathcal{A}$ over the tree $\langle\, \mathcal{G} \,\rangle$ produced by the infinitary head normalization of $\mathcal{G}$. This synchronization of the computation of $\langle\, \mathcal{G} \,\rangle$ by head reduction with the execution of the alternating tree automaton $\mathcal{A}$ is reminiscent of Salvati and Walukiewicz's approach using Krivine machines [SW14], but it is presented here using coinduction. To our knowledge, coinductive rewriting has never been considered for higher-order recursion schemes. We believe that it brings new tools and techniques which can be of interest in higher-order model-checking. The motivation underlying our coinductive formulation of $\rightarrow^{\infty}_{\mathcal{G},\mathcal{A}}$ is the following one: although we did not proceed in this way in the manuscript, we believe that it is possible to establish the soundness-and-completeness theorem (Theorem 19 in §6.4) directly on the type system with infinite derivations $\boldsymbol{\gamma}_{fix}(\mathcal{G},\mathcal{A})$, by taking advantage of the coinductive (rather than inductive) formulation of $\rightarrow^{\infty}_{\mathcal{G},\mathcal{A}}$. The proof would be in the same mood as the proof of coinductive subject reduction given for simple types in §4.3. In this coinductive framework, a natural temptation would be to extend the soundness-and-completeness theorem to the general case of infinitary λ-terms, which would be undecidable. Lemma 11 suggests however that the extension could only hold on a fragment of these terms, and it would be interesting to have a better understanding of the phenomenas it reveals.

Another interest of this coinductive approach is that we believe that it would lead to a conceptually cleaner proof, which could be easier to adapt to other classes of automata than APT, but which could also allow a neat proof in the case of non-idempotent intersection types with countable multiplicities. A last point is that, as explained in §6.5, we conjecture that the completeness proof could be more natural when devised coinductively.

In a broader perspective, the proofs and translations on infinite objects appearing in this thesis could be reformulated in a coinductive manner. This would pave the way towards an implementation of higher-order model-checking in proof assistants like Coq, in which coinduction is the typical reasoning mechanism for dealing with infinitary structures and proofs using them.

A modal reformulation of the Kobayashi-Ong type system. Our main technical contribution, and the cornerstone of our work, lies in our study of the type system $KO(\mathcal{A})$ of Kobayashi and Ong, and in its reformulation as the

modal type system $\boldsymbol{\gamma}(\mathcal{A})$. The minor alterations we perform on the original coloring operation have major consequences, as they disclose that the coloring of alternating parity automata behaves algebraically as a parametric comonad in the sense of Melliès. This discovery is the source of the semantic constructions of Part III, and leads us to introduce colored tensorial logic. Another important and related contribution is our reformulation of the game $\mathit{Adamic}(\mathcal{G}, \mathcal{A})$ as the type system $\boldsymbol{\gamma}_{fix}(\mathcal{G}, \mathcal{A})$: while Kobayashi and Ong's original game focused on decidability, and therefore emphasized on the uniformity of plays Eve could achieve in a strategy, our reformulation allows a connection with proof theory, deeply linked to tensorial logic and game semantics. We believe that the introduction of the parity condition on derivation trees is conceptually interesting; it notably motivates the definition of both parity fixpoints of Part III.

Since the coloring operation of APT corresponds to a parametric comonad, and that the associated parity condition can be formulated directly over typing derivations or in the semantics by means of a corresponding Conway operator, it makes sense to wonder to which classes of tree automata the type-theoretic and semantic soundness-and-completeness theorems could be accommodated. A first natural question would be to consider the class of ω-regular winning conditions introduced by Tsukada and Ong [TO14]; another would be to investigate the more general case of winning conditions expressed as Borel sets. It would also make sense to consider automata handling coeffects as counters or probabilities, which could certainly be modeled in appropriate relational semantics – whose cardinality could become a hurdle to decidability, but nevertheless allow compositional approaches.

An infinitary model of linear logic. We explain in Chapter 5 how non-idempotent intersection types are connected to the relational semantics of linear logic, and how this allows to solve the model-checking problem for the simple case of lambda-terms without recursion and of alternating tree automata without parity condition. Once the comonadic nature of the coloring operation of APT revealed, it is natural to consider an extension of these relational semantics in which we may interpret terms with recursion consistently with the parity condition of APT. We first give an axiomatization of the equations expected of an appropriate fixpoint operator, by translating into the language of linear logic the notion of Conway operator of Bloom and Esik [BÉ96]. This leads to our third contribution, which is an infinitary variant of these familiar relational semantics of linear logic, extended with a coloring comonad and an inductive-coinductive Conway operator based on a parity acceptance condition.

An important aspect of this contribution is that we give a semantics which is *independent* of the automaton of interest, and is only parametrized by a set of states and a set of colors. This independence follows from a study of the dual nature of higher-order recursion schemes and of alternating tree automata, at the light of linear logic.

We conjecture that these semantics capture the higher-order model-checking problem, for that the interaction – simply computed, due to the duality we previously disclosed, by a relational composition – of a λ-term with recursion computing a tree and of an alternating parity tree automaton contains the initial state of the automaton if and only if it accepts the tree generated by the term. We see two ways to prove this conjecture. The first one would be to prove

that an extension with a countable multiplicity and a modal coloring annotation of the non-idempotent intersection type system reflecting the usual, finitary relational semantics characterizes the colored, infinitary relational semantics. We would then have to adapt the proof of the soundness-and-completeness theorem to the case of non-idempotent intersection types with a countable multiplicity. The second option would be to prove Conjecture 2 and to use the resulting extensional collapse to deduce the result from the colored Scott semantics.

Another question would be to reformulate the inductive-coinductive fixpoint operator simply by using the inductive and the coinductive fixpoints we introduced on the infinitary relational semantics, and by applying one or the other when interpreting terms, depending on the current color, in the spirit of the finitary construction of Melliès [Mel14a] and of Salvati and Walukiewicz [SW15a]. Of course, the same question holds for the finitary semantics.

Colored tensorial logic. In addition to the semantic constructions it allows, the discovery of the modal nature of the coloring operation of alternating parity tree automata has logical consequences, which were not explored in this thesis. Our fourth contribution, the introduction of colored tensorial logic, is a first step towards a natural connection between the infinitary colored relational semantics of linear logic and an associated extension of dialogue games with parity conditions. Seeing the infinitary relational model as a dialogue category [Mel09, Mel16a, Mel12, Mel16b], this connection paves the way towards an extension of the existing triptych: « tensorial logic, dialogue games, dialogue categories » with infinite interactions regulated by an inductive-coinductive fixpoint policy.

Alternatively, it would make sense to introduce an infinitary extension of tensorial logic with both an inductive fixpoint operator μ and a coinductive one ν. This would probably lead to an equivalent logical framework, which could then be related to Santocanale's calculus of *circular proofs* [San02a, San02b, FS13], in which proofs of infinitary depth obtained by iteration of an inductive and of a coinductive fixpoint operators are considered. It should be pointed out that these circular proofs were introduced in relation with parity games. Connections may also appear with $\mu MALL$, an extension of the multiplicative-additive fragment of linear logic with inductive and coinductive fixpoint operators considered by Baelde in [Bae12]. This variant of colored tensorial logic with inductive and coinductive fixpoints could in turn be connected to an infinitary game semantics obtained from an extension of dialogue games with inductive and coinductive fixpoints, in the spirit of Clairambault's work [Cla09, Cla13].

Yet another potential connection with logical aspects would be to study extensions of *differential linear logic* [ER05,Ehr16]. As stressed in §9.1, the reason to interpret non-deterministic, and more generally alternating tree automata in the relational semantics of linear logic instead of carrying our analysis further in linear logic itself is its lack of non-determinism. Differential linear logic can be considered as a non-deterministic extension of linear logic deeply related to the relational semantics, but on the logical side, so that it could be interesting to study its extension with infinitary multisets, coloring and fixpoints as well.

Finitary models and decidability. Obtaining decidability results for the higher-order model-checking problems requires to adapt our infinitary approach to a *finitary* model of linear logic. As suggested by Ehrhard's extensional collapse result, our fifth contribution is to consider the Scott semantics of linear logic and to extend it with a coloring modality and an inductive-coinductive fixpoint operator, both adapted from the case of the infinitary relational semantics. Our approach provides a rigorous and compositional treatment of higher-order model-checking, and adapts to the inductive-coinductive framework of MSO logic a nice and well-established connection between linear logic, Scott domains, and intersection types. From this connection follows the decidability of the local higher-order model-checking problem, and of the selection problem. We explain the complexity results obtained by previous approaches as the size of the fragment of the model – or of the set of intersection types – whose exploration is required to build a denotation – or, equivalently, a typing derivation – for the higher-order recursion scheme of interest.

Perspectives include quite important but also technical questions. A first question would be to extend our approach to a framework in which the interpretation of constants is not fixed, but parametrized by means of a Church encoding, as we did for the infinitary semantics in Chapter 9. Another question would be to obtain a reformulation of the parity fixpoint using the traditional inductive and coinductive ones, as already discussed for the infinitary model; and a last one would be to give a more direct proof of the semantic soundness-and-completeness theorem by adapting the theorem for $\boldsymbol{\gamma}^{st}_{fix}(\mathcal{G}, \mathcal{A})$ to a variant with subtyping, instead of relying on β-normal η-long forms to eliminate it.

Several more elaborate perspectives arise as well. The first one would be to look for a relation between the complexity obtained using this finitary approach and light linear logics, in which characterizations of the n-EXPTIME complexity classes arise and notably make use of a measure of the depth of nesting of exponential boxes, which is related in our case to the order of the recursion scheme of interest. The second question is motivated by the existing relation between the relational semantics of linear logic and game semantics: is there a model of game semantics corresponding to the Scott semantics? Is it enough to consider an exponential construction on games which opens copies of a dialogue game only when it corresponds to an interaction which was not played earlier, and uses some memory to directly give the result without opening a new copy of the arena otherwise?

Extensional collapses. On this last point, we only formulate perspectives. Our semantic investigation of higher-order model-checking was deeply inspired by Ehrhard's extensional collapse result [Ehr12b], which states that the collapse of the relational model of linear logic corresponds to its Scott semantics. Connections with associated idempotent and non-idempotent type systems [BE01, dC09, Ter12, Ehr12a] were essential to our study, and seem to hold when these systems are extended with a coloring modality and a parity fixpoint operator building infinite derivations. A first point would be to check that such type-theoretic connections indeed hold, and to use them to formulate a notion of extensional collapse adapted to the case of our study. Obtaining a collapse result would allow to deduce a semantic soundness-and-completeness theorem for the infinitary case directly for the finitary one, solving in this way

Conjecture 1.

This idea of extensional collapse also rises an interesting question regarding *continuity*: we claimed that the interpretation of λY-terms in our infinitary, colored relational semantics was *not* continuous, for that the discrimination of witness trees with respect to the parity condition is only performed on infinitary trees, and not on their finite approximations. On the other hand, the interpretation in finite models can be considered as continuous, as the computation of the denotation of a term ends after a finite amount of steps. Type-theoretically, this comes from the memoryless determinacy of parity games, which leads to looking only for « regular » derivations, always typing a given sequent in the same way. Suggesting that a collapse relation exists between our finitary and infinitary models leads to wonder whether such a characterization could be given in the infinitary semantics: we conjecture that if there is a derivation typing a λY-term in these semantics, then there exists a « regular » derivation, in a bit more refined sense since the multiplicities occurring in the contexts need to be updated between two proofs of a « similar » sequent. In some sense, in addition to the backtracking pointers used to represent regular proofs in the finitary case, some context needs to be added to the one of the sequent of interest every time the loop is visited. It would be interesting to formalize this notion, and to understand how it relates to continuity.

The last point comes from the perspective of an extension of the soundness-and-completeness result to other classes of tree automata, as automata with counters or probabilities for instance. As in the approach we took in this thesis, it is easier to extend the relational semantics in a first time, and then to devise an appropriate companion Scott semantics. In the event of the existence of an extensional collapse relation between the two resulting models, the finitary model could be used to *approximate* quantitative properties of the relational semantics in a smaller – and perhaps decidable, in some situations – denotational model.

Bibliography

[Acz88] Peter Aczel. *Non-well-founded Sets*. Number 14 in Lecture Notes. Center for the Study of Language and Information, Stanford University, 1988. (Cited on page 86.)

[AdMO05] Klaus Aehlig, Jolie G. de Miranda, and C.-H. Luke Ong. The monadic second order theory of trees given by arbitrary level-two recursion schemes is decidable. In Pawel Urzyczyn, editor, *TLCA '05*, volume 3461 of *LNCS*, pages 39–54. Springer, 2005. URL: `http://dx.doi.org/10.1007/11417170_5`. (Cited on pages 15 and 25.)

[Aeh06] Klaus Aehlig. A finite semantics of simply-typed lambda terms for infinite runs of automata. In Ésik [Ési06], pages 104–118. URL: `http://dx.doi.org/10.1007/11874683_7`. (Cited on pages 15, 17, 18, 25, 28, and 103.)

[Aeh07] Klaus Aehlig. A finite semantics of simply-typed lambda terms for infinite runs of automata. *Logical Methods in Computer Science*, 3(3), 2007. URL: `http://dx.doi.org/10.2168/LMCS-3(3:1)2007`. (Cited on page 25.)

[AN01] A. Arnold and D. Niwinski. *Rudiments of &mgr;-calculus*. Studies in Logic and the Foundations of Mathematics. Elsevier Science, 2001. URL: `https://books.google.fr/books?id=MkWZaiECJvQC`. (Cited on pages 35 and 46.)

[Bae12] David Baelde. Least and greatest fixed points in linear logic. *ACM Trans. Comput. Log.*, 13(1):2, 2012. URL: `http://doi.acm.org/10.1145/2071368.2071370`. (Cited on pages 30 and 276.)

[Bai11] Patrick Baillot. Elementary linear logic revisited for polynomial time and an exponential time hierarchy. In Hongseok Yang, editor, *APLAS '11*, volume 7078 of *LNCS*, pages 337–352. Springer, 2011. URL: `http://dx.doi.org/10.1007/978-3-642-25318-8_25`. (Cited on page 268.)

[Bar84] H.P. Barendregt. *The lambda calculus: its syntax and semantics*. Studies in logic and the foundations of mathematics. North-Holland, 1984. (Cited on pages 56, 58, and 59.)

[Bar93] Michael Barr. Terminal coalgebras in well-founded set theory. *Theoretical Computer Science*, 114(2):299 – 315, 1993. (Cited on page 86.)

[BCHS13] Christopher H. Broadbent, Arnaud Carayol, Matthew Hague, and Olivier Serre. C-shore: a collapsible approach to higher-order verification. In Greg Morrisett and Tarmo Uustalu, editors, *ICFP '13*, pages 13–24. ACM, 2013. URL: http://doi.acm.org/10.1145/2500365.2500589. (Cited on page 268.)

[BCOS10] Christopher H. Broadbent, Arnaud Carayol, C.-H. Luke Ong, and Olivier Serre. Recursion schemes and logical reflection. In *LICS '10*, pages 120–129. IEEE Computer Society, 2010. URL: http://dx.doi.org/10.1109/LICS.2010.40. (Cited on pages 26, 27, 79, and 80.)

[BÉ93] S.L. Bloom and Z. Ésik. *Iteration theories: the equational logic of iterative processes*. EATCS monographs on theoretical computer science. Springer-Verlag, 1993. URL: http://books.google.fr/books?id=Hf3uAAAAMAAJ. (Cited on page 201.)

[BÉ96] Stephen L. Bloom and Zoltán Ésik. Fixed-point operations on CCC's. Part I. *TCS*, 155, 1996. (Cited on pages 201, 255, and 275.)

[BE00] Antonio Bucciarelli and Thomas Ehrhard. On phase semantics and denotational semantics in multiplicative-additive linear logic. *Ann. Pure Appl. Logic*, 102(3):247–282, 2000. URL: http://dx.doi.org/10.1016/S0168-0072(99)00040-8. (Cited on pages 15, 20, 21, 24, 104, 115, 234, 237, and 238.)

[BE01] Antonio Bucciarelli and Thomas Ehrhard. On phase semantics and denotational semantics: the exponentials. *Ann. Pure Appl. Logic*, 109(3):205–241, 2001. URL: http://dx.doi.org/10.1016/S0168-0072(00)00056-7. (Cited on pages 15, 20, 21, 24, 104, 111, 115, 234, 237, 238, and 277.)

[Bec69] Jon Beck. *Seminar on Triples and Categorical Homology Theory: ETH 1966/67*, chapter Distributive laws, pages 119–140. Springer Berlin Heidelberg, Berlin, Heidelberg, 1969. URL: http://dx.doi.org/10.1007/BFb0083084. (Cited on page 223.)

[BEM97] Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability analysis of pushdown automata: Application to model-checking. In Antoni W. Mazurkiewicz and Józef Winkowski, editors, *CONCUR '97*, volume 1243 of *LNCS*, pages 135–150. Springer, 1997. URL: http://dx.doi.org/10.1007/3-540-63141-0_10. (Cited on page 13.)

[BH15] Henning Basold and Helle Hvid Hansen. Well-definedness and observational equivalence for inductive-coinductive programs. 2015. URL: http://cs.ru.nl/~hbasold/publications/ObsEq.pdf. (Cited on page 92.)

[Bie95] Gavin M. Bierman. What is a categorical model of intuitionistic linear logic? In Mariangiola Dezani-Ciancaglini and Gordon D. Plotkin, editors, *TLCA '95*, volume 902 of *LNCS*, pages 78–93. Springer, 1995. URL: http://dx.doi.org/10.1007/BFb0014046. (Cited on page 111.)

[BO09] William Blum and C.-H. Luke Ong. The safe lambda calculus. *Logical Methods in Computer Science*, 5(1), 2009. URL: `http://arxiv.org/abs/0901.2399`. (Cited on page 25.)

[Böh68] Corrado Böhm. Alcune proprietà delle forme β-η-normali nel $\lambda-$K calcolo. *Publicazioni dell'Istituto per le Applicazioni del Calcolo*, 696, 1968. (Cited on page 59.)

[Buc97] Antonio Bucciarelli. Logical relations and lambda -theories. *Advances in Theory and Formal Methods of Computing, proceedings of the 3rd Imperial College Workshop*, pages 47–48, 1997. (Cited on page 269.)

[BW15] Julian Bradfield and Igor Walukiewicz. The mu-calculus and model-checking. In H. Veith E. Clarke, T. Henzinger, editor, *Handbook of Model Checking*. Springer-Verlag, 2015. (Cited on pages 35, 45, and 46.)

[Cau02] Didier Caucal. On infinite terms having a decidable monadic theory. In Krzysztof Diks and Wojciech Rytter, editors, *MFCS '02*, volume 2420 of *LNCS*, pages 165–176. Springer, 2002. URL: `http://dx.doi.org/10.1007/3-540-45687-2_13`. (Cited on pages 15 and 25.)

[CCF58] Haskell Brooks Curry, William Craig, and Robert Feys. Combinatory logic, 1, 1958. (Cited on page 59.)

[CDG+07] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 2007. release October, 12th 2007. (Cited on pages 35 and 51.)

[CDHL84] M. Coppo, M. Dezani-Ciancaglini, F. Honsell, and G. Longo. Extended Type Structures and Filter Lambda Models. In *Logic Colloquium 82*, 1984. (Cited on page 244.)

[CE81] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In Dexter Kozen, editor, *Logics of Programs, Workshop*, volume 131 of *LNCS*, pages 52–71. Springer, 1981. URL: `http://dx.doi.org/10.1007/BFb0025774`. (Cited on page 13.)

[CES10] Alberto Carraro, Thomas Ehrhard, and Antonino Salibra. Exponentials with infinite multiplicities. In Anuj Dawar and Helmut Veith, editors, *CSL '10*, volume 6247 of *LNCS*, pages 170–184. Springer, 2010. URL: `http://dx.doi.org/10.1007/978-3-642-15205-4_16`. (Cited on page 30.)

[Chu32] A. Church. *A Set of Postulates for the Foundation of Logic*, volume 33 of *Annals of Mathematics*. 1932. (Cited on page 55.)

[Cla09] Pierre Clairambault. Least and greatest fixpoints in game semantics. In Luca de Alfaro, editor, *FOSSACS '09*, volume 5504 of *LNCS*, pages 16–31. Springer, 2009. URL: http://dx.doi.org/10.1007/978-3-642-00596-1_3. (Cited on page 276.)

[Cla13] Pierre Clairambault. Strong functors and interleaving fixpoints in game semantics. *RAIRO - Theor. Inf. and Applic.*, 47(1):25–68, 2013. URL: http://dx.doi.org/10.1051/ita/2012028. (Cited on page 276.)

[Cou90] Bruno Courcelle. Recursive applicative program schemes. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 459–492. 1990. (Cited on page 68.)

[Cou97] B. Courcelle. Handbook of graph grammars and computing by graph transformation. chapter The Expression of Graph Properties and Graph Transformations in Monadic Second-order Logic, pages 313–400. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1997. URL: http://www.labri.fr/perso/courcell/CoursMaster/GraphGrammarsBook(1997).pdf. (Cited on page 40.)

[CS12] Arnaud Carayol and Olivier Serre. Collapsible pushdown automata and labeled recursion schemes: Equivalence, safety and effective selection. In *LICS '12* [DBL12], pages 165–174. URL: http://dx.doi.org/10.1109/LICS.2012.73. (Cited on pages 15, 27, 79, 80, and 269.)

[Cur08] Pierre-Louis Curien. Introduction to linear logic and ludics, part i. http://www.arxiv.org/abs/cs/0501035v1, 2008. (Cited on pages 65 and 67.)

[CW03] Arnaud Carayol and Stefan Wöhrle. The caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In Paritosh K. Pandya and Jaikumar Radhakrishnan, editors, *FSTTCS '03*, volume 2914 of *LNCS*, pages 112–123. Springer, 2003. URL: http://dx.doi.org/10.1007/978-3-540-24597-1_10. (Cited on page 25.)

[Cza15] Lukasz Czajka. Coinductive techniques in infinitary lambda-calculus. *CoRR*, abs/1501.04354, 2015. URL: http://arxiv.org/abs/1501.04354. (Cited on pages 87, 88, 90, 93, 94, 95, and 274.)

[Dam77a] Werner Damm. Higher type program schemes and their tree languages. In Hans Tzschach, H. Waldschmidt, and Hermann K.-G. Walter, editors, *Theoretical Computer Science, 3rd GI-Conference*, volume 48 of *LNCS*, pages 51–72. Springer, 1977. URL: http://dx.doi.org/10.1007/3-540-08138-0_5. (Cited on pages 14 and 68.)

[Dam77b] Werner Damm. Languages defined by higher type program schemes. In Arto Salomaa and Magnus Steinby, editors, *ICALP '77*, volume 52 of *LNCS*, pages 164–179. Springer, 1977. URL:

`http://dx.doi.org/10.1007/3-540-08342-1_13`. (Cited on pages 14 and 68.)

[Dam82] Werner Damm. The io- and oi-hierarchies. *Theoretical Computer Science*, 20(2):95 – 207, 1982. (Cited on pages 25, 68, and 76.)

[DBL05] *LICS '05*. IEEE Computer Society, 2005. URL: `http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=10087`. (Cited on pages 287 and 289.)

[DBL12] *LICS '12*. IEEE Computer Society, 2012. (Cited on pages 282 and 290.)

[dC09] Daniel de Carvalho. Execution time of lambda-terms via denotational semantics and intersection types. *CoRR*, abs/0905.4251, 2009. URL: `http://arxiv.org/abs/0905.4251`. (Cited on pages 20, 21, 104, 115, and 277.)

[DF80] W. Damm and E. Fehr. A schematalogical approach to the analysis of the procedure concept in algol-languages. *Proc. 5ème Colloque sur les Arbres en Algèbre et en Programmation*, 1:130–134, 1980. (Cited on page 78.)

[DP90] Brian A. Davey and Hilary A. Priestley. *Introduction to lattices and order*. Cambridge University Press, 1990. (Cited on pages 35 and 42.)

[Ehr12a] Thomas Ehrhard. Collapsing non-idempotent intersection types. In Patrick Cégielski and Arnaud Durand, editors, *CSL '12*, volume 16 of *LIPIcs*, pages 259–273. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012. URL: `http://dx.doi.org/10.4230/LIPIcs.CSL.2012.259`. (Cited on pages 15, 20, 21, 24, 117, 118, 138, 244, 249, 253, and 277.)

[Ehr12b] Thomas Ehrhard. The scott model of linear logic is the extensional collapse of its relational model. *Theor. Comput. Sci.*, 424:20–45, 2012. URL: `http://dx.doi.org/10.1016/j.tcs.2011.11.027`. (Cited on pages 15, 21, 31, 118, 241, 268, 269, and 277.)

[Ehr14] Thomas Ehrhard. Lambda-calcul : syntaxe et sémantique. *Cours au MPRI*, 2014. URL: `http://www.pps.univ-paris-diderot.fr/~ehrhard/master2/mpri-13-14.pdf`. (Cited on page 115.)

[Ehr16] Thomas Ehrhard. A semantical introduction to differential linear logic. *to appear in MSCS*, 2016. (Cited on page 276.)

[EP11] Jörg Endrullis and Andrew Polonsky. Infinitary rewriting coinductively. In Nils Anders Danielsson and Bengt Nordström, editors, *TYPES '11*, volume 19 of *LIPIcs*, pages 16–27. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011. URL: `http://dx.doi.org/10.4230/LIPIcs.TYPES.2011.16`. (Cited on pages 87, 94, and 274.)

[ER05] Thomas Ehrhard and Laurent Regnier. Differential interaction nets. *Electr. Notes Theor. Comput. Sci.*, 123:35–74, 2005. URL: http://dx.doi.org/10.1016/j.entcs.2004.06.060. (Cited on page 276.)

[ES77] Joost Engelfriet and Erik Meineche Schmidt. IO and OI. I. *J. Comput. Syst. Sci.*, 15(3):328–353, 1977. URL: http://dx.doi.org/10.1016/S0022-0000(77)80034-2. (Cited on page 68.)

[ES78] Joost Engelfriet and Erik Meineche Schmidt. IO and OI. II. *J. Comput. Syst. Sci.*, 16(1):67–99, 1978. URL: http://dx.doi.org/10.1016/0022-0000(78)90051-X. (Cited on page 68.)

[Ési06] Zoltán Ésik, editor. *CSL '06*, volume 4207 of *LNCS*. Springer, 2006. (Cited on pages 279 and 288.)

[FIK13] Koichi Fujima, Sohei Ito, and Naoki Kobayashi. Practical alternating parity tree automata model checking of higher-order recursion schemes. In Chung-chieh Shan, editor, *APLAS '13*, volume 8301 of *LNCS*, pages 17–32. Springer, 2013. URL: http://dx.doi.org/10.1007/978-3-319-03542-0_2. (Cited on page 268.)

[FS13] Jérôme Fortier and Luigi Santocanale. Cuts for circular proofs: semantics and cut-elimination. In Simona Ronchi Della Rocca, editor, *CSL '13*, volume 23 of *LIPIcs*, pages 248–262. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013. URL: http://dx.doi.org/10.4230/LIPIcs.CSL.2013.248. (Cited on page 276.)

[Gir87] Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987. URL: http://dx.doi.org/10.1016/0304-3975(87)90045-4. (Cited on page 15.)

[GL15] Jean Goubault-Larrecq. Logique et informatique II: Aspects logiques. *Cours aux ENS d'Ulm et de Cachan*, 2015. URL: http://www.lsv.ens-cachan.fr/~goubault/Lambda/types.pdf. (Cited on page 64.)

[GM15a] Charles Grellois and Paul-André Melliès. Finitary semantics of linear logic and higher-order model-checking. In Giuseppe F. Italiano, Giovanni Pighizzini, and Donald Sannella, editors, *MFCS '15*, volume 9234 of *LNCS*, pages 256–268. Springer, 2015. URL: http://dx.doi.org/10.1007/978-3-662-48057-1_20. (Cited on pages 28, 80, and 273.)

[GM15b] Charles Grellois and Paul-André Melliès. Indexed linear logic and higher-order model checking. In Jakob Rehof, editor, *ITRS '14*, volume 177 of *EPTCS*, pages 43–52, 2015. URL: http://dx.doi.org/10.4204/EPTCS.177.4. (Cited on pages 20, 28, 115, 237, 238, and 273.)

[GM15c] Charles Grellois and Paul-André Melliès. An infinitary model of linear logic. In Pitts [Pit15], pages 41–55. URL: http://dx.doi.org/10.1007/978-3-662-46678-0_3. (Cited on pages 28, 200, and 273.)

[GM15d] Charles Grellois and Paul-André Melliès. Relational semantics of linear logic and higher-order model checking. In Kreutzer [Kre15], pages 260–276. URL: http://dx.doi.org/10.4230/LIPIcs.CSL.2015.260. (Cited on pages 22, 23, 28, and 273.)

[Gol05] Mayer Goldberg. On the recursive enumerability of fixed-point combinators, 2005. URL: http://www.brics.dk/RS/05/1/BRICS-RS-05-1.pdf. (Cited on page 60.)

[Gre10] Charles Grellois. Game semantics of higher-order recursion schemes establishes the decidability of mso model-checking. Master thesis, available on: http://research.grellois.fr/doc/grellois-game-semantics-and-ho-model-checking.pdf, 2010. (Cited on pages 26 and 267.)

[GTL89] Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and Types*. C.U.P., 1989. (Cited on page 192.)

[GTW02] E. Grädel, W. Thomas, and T. Wilke. *Automata, Logics, and Infinite Games: A Guide to Current Research*. LNCS. Springer, 2002. URL: https://books.google.fr/books?id=2-Vl9Za93xIC. (Cited on page 35.)

[Had12] Axel Haddad. IO vs OI in higher-order recursion schemes. In Dale Miller and Zoltán Ésik, editors, *FICS '12*, volume 77 of *EPTCS*, pages 23–30, 2012. URL: http://dx.doi.org/10.4204/EPTCS.77.4. (Cited on page 76.)

[Had13a] Axel Haddad. Model checking and functional program transformations. In Seth and Vishnoi [SV13], pages 115–126. URL: http://dx.doi.org/10.4230/LIPIcs.FSTTCS.2013.115. (Cited on pages 27, 80, and 269.)

[Had13b] Axel Haddad. *Shape-Preserving Transformations of Higher-Order Recursion Schemes*. Thèse de doctorat, Université Paris Diderot - Paris 7, 2013. URL: http://www.liafa.univ-paris-diderot.fr/~haddad/Shape-PreservingTransformationsofHigher-OrderRecursionSchemes.pdf. (Cited on pages 15, 27, 76, 79, 81, 136, 269, and 271.)

[Has99] Masahito Hasegawa. *Models of Sharing Graphs: A Categorical Semantics of Let and Letrec*. Number 1192 in Distinguished dissertations series. Springer-Verlag, 1999. (Cited on pages 201 and 203.)

[HMOS08] Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier Serre. Collapsible pushdown automata and recursion schemes. In *LICS '08*, pages 452–461. IEEE Computer Society, 2008. URL: http://dx.doi.org/10.1109/LICS.2008.34. (Cited on pages 26, 27, and 80.)

[Hue76] Gérard Huet. *Résolution d'équations dans des langages d'ordre 1,2,...,ω*. Thèse de doctorat, Université Paris 7, 1976. URL: http://yquem.inria.fr/~huet/PUBLIC/Huet1976.pdf. (Cited on page 250.)

[HVP05] Haruo Hosoya, Jerome Vouillon, and Benjamin C. Pierce. Regular expression types for XML. *ACM Trans. Program. Lang. Syst.*, 27(1):46–90, 2005. URL: `http://doi.acm.org/10.1145/1053468.1053470`. (Cited on page 17.)

[Ind76] Klaus Indermark. Schemes with recursion on higher types. In Antoni W. Mazurkiewicz, editor, *MFCS '76*, volume 45 of *LNCS*, pages 352–358. Springer, 1976. URL: `http://dx.doi.org/10.1007/3-540-07854-1_198`. (Cited on page 68.)

[JSV96] André Joyal, Ross Street, and Dominic Verity. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119:447–468, 4 1996. URL: `http://journals.cambridge.org/article_S0305004100074338`. (Cited on page 201.)

[JW95] David Janin and Igor Walukiewicz. Automata for the modal mu-calculus and related results. In Jirí Wiedermann and Petr Hájek, editors, *MFCS '95*, volume 969 of *LNCS*, pages 552–562. Springer, 1995. URL: `http://dx.doi.org/10.1007/3-540-60246-1_160`. (Cited on page 50.)

[JW96] David Janin and Igor Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In Ugo Montanari and Vladimiro Sassone, editors, *CONCUR '96*, volume 1119 of *LNCS*, pages 263–277. Springer, 1996. URL: `http://dx.doi.org/10.1007/3-540-61604-7_60`. (Cited on page 39.)

[KNU01] Teodor Knapik, Damian Niwinski, and Pawel Urzyczyn. Deciding monadic theories of hyperalgebraic trees. In *TLCA '01*, pages 253–267, 2001. URL: `http://dx.doi.org/10.1007/3-540-45413-6_21`. (Cited on pages 15, 24, and 25.)

[KNU02] Teodor Knapik, Damian Niwinski, and Pawel Urzyczyn. Higher-order pushdown trees are easy. In Nielsen and Engberg [NE02], pages 205–222. URL: `http://dx.doi.org/10.1007/3-540-45931-6_15`. (Cited on pages 15 and 25.)

[KNUW05] Teodor Knapik, Damian Niwinski, Pawel Urzyczyn, and Igor Walukiewicz. Unsafe grammars and panic automata. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *ICALP '05*, volume 3580 of *LNCS*, pages 1450–1461. Springer, 2005. URL: `http://dx.doi.org/10.1007/11523468_117`. (Cited on pages 15 and 25.)

[KO] Naoki Kobayashi and C.-H. Luke Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes (journal version, in preparation). (Cited on pages 31, 121, 136, 137, 145, 147, 151, 161, 162, and 177.)

[KO09] Naoki Kobayashi and C.-H. Luke Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion

schemes. In *LICS '09*, 2009. (Cited on pages 22, 24, 26, 28, 29, 30, 31, 61, 79, 80, 104, 121, 122, 124, 125, 128, 129, 136, 137, 151, 236, and 267.)

[Kob09a] Naoki Kobayashi. Model-checking higher-order functions. In António Porto and Francisco Javier López-Fraguas, editors, *PPDP '09*, pages 25–36. ACM, 2009. URL: `http://doi.acm.org/10.1145/1599410.1599415`. (Cited on page 268.)

[Kob09b] Naoki Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In Zhong Shao and Benjamin C. Pierce, editors, *POPL '09*, pages 416–428. ACM, 2009. URL: `http://doi.acm.org/10.1145/1480881.1480933`. (Cited on pages 17, 18, 25, 26, 27, 28, 30, 104, and 122.)

[Kob11] Naoki Kobayashi. A practical linear time algorithm for trivial automata model checking of higher-order recursion schemes. In Martin Hofmann, editor, *FOSSACS '11*, volume 6604 of *LNCS*, pages 260–274. Springer, 2011. URL: `http://dx.doi.org/10.1007/978-3-642-19805-2_18`. (Cited on page 268.)

[Kob13] Naoki Kobayashi. Model checking higher-order programs. *J. ACM*, 60(3):20, 2013. URL: `http://doi.acm.org/10.1145/2487241.2487246`. (Cited on page 138.)

[Kre15] Stephan Kreutzer, editor. *CSL '15*, volume 41 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. URL: `http://www.dagstuhl.de/dagpub/978-3-939897-90-3`. (Cited on pages 285 and 292.)

[Kri07] Jean-Louis Krivine. A call-by-name lambda-calculus machine. *Higher-Order and Symbolic Computation*, 20(3):199–207, 2007. URL: `http://dx.doi.org/10.1007/s10990-007-9018-9`. (Cited on page 100.)

[KS12] Dexter Kozen and Alexandra Silva. Practical coinduction. Technical Report `http://hdl.handle.net/1813/30510`, Computing and Information Science, Cornell University, November 2012. (Cited on page 87.)

[Lev06] Paul Blain Levy. Call-by-push-value: Decomposing call-by-value and call-by-name. *Higher-Order and Symbolic Computation*, 19(4):377–414, 2006. URL: `http://dx.doi.org/10.1007/s10990-006-0480-6`. (Cited on page 59.)

[Mar75] Donald A. Martin. Borel Determinacy. pages 363–371, 1975. (Cited on page 52.)

[Mel05] Paul-André Melliès. Asynchronous games 4: A fully complete model of propositional linear logic. In *LICS '05* [DBL05], pages 386–395. URL: `http://dx.doi.org/10.1109/LICS.2005.6`. (Cited on page 236.)

[Mel06a] Paul-André Melliès. Asynchronous games 2: The true concurrency of innocence. *Theor. Comput. Sci.*, 358(2-3):200–228, 2006. URL: http://dx.doi.org/10.1016/j.tcs.2006.01.016. (Cited on page 236.)

[Mel06b] Paul-André Melliès. Functorial boxes in string diagrams. In Ésik [Ési06], pages 1–30. (Cited on pages 23, 129, 132, 140, 151, and 237.)

[Mel09] Paul-André Melliès. Categorical semantics of linear logic. In *Interactive models of computation and program behaviour*, pages 1–196. 2009. (Cited on pages 111, 117, 141, 226, and 276.)

[Mel12] Paul-André Melliès. Game semantics in string diagrams. In *LICS '12*, pages 481 – 490. ACM SIGACT and IEEE Computer Society, IEEE Computer Society, 2012. (Cited on pages 15, 141, 236, 237, and 276.)

[Mel14a] Paul-André Melliès. Linear logic and higher-order model-checking. Talk at Institut Henri Poincaré, http://www.pps.univ-paris-diderot.fr/~mellies/slides/workshop-IHP-model-checking.pdf, 2014. (Cited on pages 28, 29, and 276.)

[Mel14b] Paul-André Melliès. The parametric continuation monad. *Math. Struct. Comp. Sci.*, 2014. (Cited on pages 23, 132, 140, 151, and 237.)

[Mel16a] Paul-André Melliès. Dialogue categories and chiralities. *To appear in PRIMS*, 2016. (Cited on pages 141 and 276.)

[Mel16b] Paul-André Melliès. *Une étude micrologique de la négation.* Habilitation à diriger des recherches, Université Paris 7, 2016. (Cited on pages 15, 141, and 276.)

[Mil78] Robin Milner. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17(3):348 – 375, 1978. URL: http://www.sciencedirect.com/science/article/pii/0022000078900144. (Cited on page 61.)

[Miq01] Alexandre Miquel. *Le calcul des constructions implicites : syntaxe et sémantique.* PhD thesis, Université Paris 7, 2001. (Cited on page 30.)

[Mon03] Raphaël Montelatici. Polarized proof nets with cycles and fixpoints semantics. In Martin Hofmann, editor, *TLCA '03*, volume 2701 of *LNCS*, pages 256–270. Springer, 2003. URL: http://dx.doi.org/10.1007/3-540-44904-3_18. (Cited on page 30.)

[MS85] David E. Muller and Paul E. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theor. Comput. Sci.*, 37:51–75, 1985. URL: http://dx.doi.org/10.1016/0304-3975(85)90087-8. (Cited on page 15.)

[MS95] David E. Muller and Paul E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of rabin, mcnaughton and safra. *Theor. Comput. Sci.*, 141(1&2):69–107, 1995. URL: `http://dx.doi.org/10.1016/0304-3975(94)00214-4`. (Cited on page 51.)

[MT10] Paul-André Melliès and Nicolas Tabareau. Resource modalities in tensor logic. *Ann. Pure Appl. Logic*, 161(5):632–653, 2010. URL: `http://dx.doi.org/10.1016/j.apal.2009.07.018`. (Cited on pages 15, 140, 235, 236, and 237.)

[MV05] Paul-André Melliès and Jerome Vouillon. Recursive polymorphic types and parametricity in an operational framework. In *LICS '05* [DBL05], pages 82–91. URL: `http://dx.doi.org/10.1109/LICS.2005.42`. (Cited on page 94.)

[NE02] Mogens Nielsen and Uffe Engberg, editors. *FOSSACS '02*, volume 2303 of *LNCS*. Springer, 2002. (Cited on pages 286 and 291.)

[Niv72] M. Nivat. On the interpretation of recursive program schemes. In *Symposia Matematica*, 1972. (Cited on pages 14 and 67.)

[Niw88] Damian Niwinski. Fixed points vs. infinite generation. In *(LICS '88)*, pages 402–409. IEEE Computer Society, 1988. URL: `http://dx.doi.org/10.1109/LICS.1988.5137`. (Cited on pages 39 and 45.)

[NO14] Robin P. Neatherway and C.-H. Luke Ong. Travmc2: higher-order model checking for alternating parity tree automata. In Neha Rungta and Oksana Tkachuk, editors, *2014 International Symposium on Model Checking of Software, SPIN*, pages 129–132. ACM, 2014. URL: `http://doi.acm.org/10.1145/2632362.2632381`. (Cited on page 268.)

[NRO12] Robin P. Neatherway, Steven J. Ramsay, and C.-H. Luke Ong. A traversal-based algorithm for higher-order model checking. In Peter Thiemann and Robby Bruce Findler, editors, *ICFP '12*, pages 353–364. ACM, 2012. URL: `http://doi.acm.org/10.1145/2364527.2364578`. (Cited on page 268.)

[Ong] C.-H. Luke Ong. On model-checking trees generated by higher-order recursion schemes (long version). (Cited on page 25.)

[Ong06] C.-H. Luke Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS '06*, pages 81–90. IEEE Computer Society, 2006. (Cited on pages 15, 25, 26, 80, and 267.)

[Ong15] C.-H. Luke Ong. Normalisation by traversals. *CoRR*, abs/1511.02629, 2015. URL: `http://arxiv.org/abs/1511.02629`. (Cited on page 25.)

[OT12] C.-H. Luke Ong and Takeshi Tsukada. Two-level game semantics, intersection types, and recursion schemes. In Artur Czumaj,

Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *ICALP '12*, volume 7392 of *LNCS*, pages 325–336. Springer, 2012. URL: http://dx.doi.org/10.1007/978-3-642-31585-5_31. (Cited on page 27.)

[Par12] Pawel Parys. On the significance of the collapse operation. In *LICS '12* [DBL12], pages 521–530. URL: http://dx.doi.org/10.1109/LICS.2012.62. (Cited on pages 25 and 26.)

[Pit15] Andrew M. Pitts, editor. *FoSSaCS '15*, volume 9034 of *LNCS*. Springer, 2015. URL: http://dx.doi.org/10.1007/978-3-662-46678-0. (Cited on pages 284 and 292.)

[Plo75] G.D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1(2):125 – 159, 1975. (Cited on page 61.)

[POM14] Tomas Petricek, Dominic A. Orchard, and Alan Mycroft. Coeffects: a calculus of context-dependent computation. In *ICFP*, 2014. (Cited on page 140.)

[QS82] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in CESAR. In Mariangiola Dezani-Ciancaglini and Ugo Montanari, editors, *International Symposium on Programming '82*, volume 137 of *LNCS*, pages 337–351. Springer, 1982. URL: http://dx.doi.org/10.1007/3-540-11494-7_22. (Cited on page 13.)

[Rab69] Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the AMS*, 141:pp. 1–35, 1969. URL: http://www.jstor.org/stable/1995086. (Cited on pages 15 and 38.)

[Ram13] Steven Ramsay. *Intersection Types and Higher-Order Model Checking*. Phd thesis, University of Oxford, 2013. URL: https://www2.warwick.ac.uk/fac/sci/dcs/people/steven_ramsay/dissertation.pdf. (Cited on page 138.)

[Rey93] JohnC. Reynolds. The discoveries of continuations. *LISP and Symbolic Computation*, 6(3-4):233–247, 1993. URL: http://dx.doi.org/10.1007/BF01019459. (Cited on page 61.)

[Ré13] Didier Rémy. Type systems for programming languages. *MPRI lecture notes*, 2013. URL: http://gallium.inria.fr/~remy/mpri/2013/cours.pdf. (Cited on page 64.)

[Sal09] Sylvain Salvati. Recognizability in the simply typed lambda-calculus. In Hiroakira Ono, Makoto Kanazawa, and Ruy J. G. B. de Queiroz, editors, *WoLLIC '09*, volume 5514 of *LNCS*, pages 48–60. Springer, 2009. URL: http://dx.doi.org/10.1007/978-3-642-02261-6_5. (Cited on pages 17 and 28.)

[Sal10] Sylvain Salvati. On the membership problem for non-linear abstract categorial grammars. *Journal of Logic, Language and Information*, 19(2):163–183, 2010. URL: `http://dx.doi.org/10.1007/s10849-009-9110-0`. (Cited on pages 138, 245, 249, 250, and 252.)

[San02a] Luigi Santocanale. μ-bicomplete categories and parity games. *ITA*, 36(2):195–227, 2002. URL: `http://dx.doi.org/10.1051/ita:2002010`. (Cited on page 276.)

[San02b] Luigi Santocanale. A calculus of circular proofs and its categorical semantics. In Nielsen and Engberg [NE02], pages 357–371. URL: `http://dx.doi.org/10.1007/3-540-45931-6_25`. (Cited on page 276.)

[San09] Davide Sangiorgi. On the origins of bisimulation and coinduction. *ACM Trans. Program. Lang. Syst.*, 31(4), 2009. URL: `http://doi.acm.org/10.1145/1516507.1516510`. (Cited on page 86.)

[Sch07] Sven Schewe. Solving parity games in big steps. In Vikraman Arvind and Sanjiva Prasad, editors, *FSTTCS '07*, volume 4855 of *LNCS*, pages 449–460. Springer, 2007. URL: `http://dx.doi.org/10.1007/978-3-540-77050-3_37`. (Cited on pages 52 and 267.)

[SdV02] Paula Severi and Fer-Jan de Vries. An extensional böhm model. In Sophie Tison, editor, *RTA '02*, volume 2378 of *LNCS*, pages 159–173. Springer, 2002. URL: `http://dx.doi.org/10.1007/3-540-45610-4_12`. (Cited on page 94.)

[See89] R.A.G. Seely. Linear logic, *-autonomous categories and cofree coalgebras. In *In Categories in Computer Science and Logic*, pages 371–382. AMS, 1989. (Cited on pages 111 and 203.)

[Ser13] Olivier Serre. *Playing with Trees and Logic*. Habilitation à diriger des recherches, Université Paris Diderot - Paris 7, 2013. URL: `http://www.liafa.univ-paris-diderot.fr/~serre/papers/HDR.pdf`. (Cited on pages 68, 70, and 81.)

[SP00] Alex K. Simpson and Gordon D. Plotkin. Complete axioms for categorical fixed-point operators. In *LICS '00*, pages 30–41. IEEE Computer Society, 2000. URL: `http://doi.ieeecomputersociety.org/10.1109/LICS.2000.855753`. (Cited on pages 200, 201, and 203.)

[Sta02] R. Statman. On the lambda y calculus. In *LICS '02*, pages 159–166, 2002. (Cited on page 76.)

[SV13] Anil Seth and Nisheeth K. Vishnoi, editors. *FSTTCS '13*, volume 24 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013. URL: `http://drops.dagstuhl.de/portals/extern/index.php?semnr=13018`. (Cited on pages 285 and 292.)

[SW11] Sylvain Salvati and Igor Walukiewicz. Krivine machines and higher-order schemes. In Luca Aceto, Monika Henzinger, and Jirí Sgall, editors, *ICALP '11*, volume 6756 of *LNCS*, pages 162–173. Springer, 2011. URL: http://dx.doi.org/10.1007/978-3-642-22012-8_12. (Cited on page 26.)

[SW12] Sylvain Salvati and Igor Walukiewicz. Recursive schemes, krivine machines, and collapsible pushdown automata. In Alain Finkel, Jérôme Leroux, and Igor Potapov, editors, *Reachability Problems - RP 2012*, volume 7550 of *LNCS*, pages 6–20. Springer, 2012. URL: http://dx.doi.org/10.1007/978-3-642-33512-9_2. (Cited on pages 78 and 257.)

[SW13a] Sylvain Salvati and Igor Walukiewicz. Evaluation is msol-compatible. In Seth and Vishnoi [SV13], pages 103–114. URL: http://dx.doi.org/10.4230/LIPIcs.FSTTCS.2013.103. (Cited on page 27.)

[SW13b] Sylvain Salvati and Igor Walukiewicz. Using models to model-check recursive schemes. In Masahito Hasegawa, editor, *TLCA '13*, volume 7941 of *LNCS*, pages 189–204. Springer, 2013. URL: http://dx.doi.org/10.1007/978-3-642-38946-7_15. (Cited on page 28.)

[SW14] Sylvain Salvati and Igor Walukiewicz. Krivine machines and higher-order schemes. *Inf. Comput.*, 239:340–355, 2014. URL: http://dx.doi.org/10.1016/j.ic.2014.07.012. (Cited on pages 26, 80, 100, and 274.)

[SW15a] Sylvain Salvati and Igor Walukiewicz. A model for behavioural properties of higher-order programs. In Kreutzer [Kre15], pages 229–243. URL: http://dx.doi.org/10.4230/LIPIcs.CSL.2015.229. (Cited on pages 24, 28, 29, 80, and 276.)

[SW15b] Sylvain Salvati and Igor Walukiewicz. Typing weak MSOL properties. In Pitts [Pit15], pages 343–357. URL: http://dx.doi.org/10.1007/978-3-662-46678-0_22. (Cited on pages 28 and 29.)

[Tai67] W. W. Tait. Intensional interpretations of functionals of finite type i. *The Journal of Symbolic Logic*, 32(2):pp. 198–212, 1967. (Cited on page 64.)

[Tar55] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Math.*, 5(2):285–309, 1955. (Cited on page 41.)

[Ter12] Kazushige Terui. Semantic evaluation, intersection types and complexity of simply typed lambda calculus. In Ashish Tiwari, editor, *RTA '12*, volume 15 of *LIPIcs*, pages 323–338. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012. URL: http://dx.doi.org/10.4230/LIPIcs.RTA.2012.323. (Cited on pages 20, 21, 24, 117, 138, 244, 249, 253, and 277.)

[Tho96] Wolfgang Thomas. Languages, automata, and logic. In *HANDBOOK OF FORMAL LANGUAGES*, pages 389–455. Springer, 1996. (Cited on page 35.)

[TO] Takeshi Tsukada and C.-H. Luke Ong. Compositional higher-order model checking via ω-regular games over böhm trees (long version). URL: `http://www-kb.is.s.u-tokyo.ac.jp/~tsukada/papers/effect-arena-long.pdf`. (Cited on page 27.)

[TO14] Takeshi Tsukada and C.-H. Luke Ong. Compositional higher-order model checking via ω-regular games over böhm trees. In Thomas A. Henzinger and Dale Miller, editors, *CSL-LICS '14*, pages 78:1–78:10. ACM, 2014. URL: `http://doi.acm.org/10.1145/2603088.2603133`. (Cited on pages 27, 80, and 275.)

[Tur37] A. M. Turing. The p-function in $\lambda - K$-conversion. *Journal of Symbolic Logic*, 2(4):164–164, December 1937. URL: `http://homepage.mac.com/a.eppendahl/work/others/pflkc/`. (Cited on page 60.)

[Wad71] C.P. Wadsworth. *Semantics and Pragmatics of the Lambda-calculus*. University of Oxford, 1971. URL: `https://books.google.de/books?id=kl1QIQAACAAJ`. (Cited on page 59.)

[Wal95] Igor Walukiewicz. Completeness of kozen's axiomatisation of the propositional mu-calculus. In *LICS '95*, pages 14–24. IEEE Computer Society, 1995. URL: `http://dx.doi.org/10.1109/LICS.1995.523240`. (Cited on page 46.)

[Wal96] Igor Walukiewicz. Pushdown processes: Games and model checking. In Rajeev Alur and Thomas A. Henzinger, editors, *CAV '96*, volume 1102 of *LNCS*, pages 62–74. Springer, 1996. URL: `http://dx.doi.org/10.1007/3-540-61474-5_58`. (Cited on page 13.)

[Wal01] Igor Walukiewicz. Pushdown processes: Games and model-checking. *Inf. Comput.*, 164(2):234–263, 2001. URL: `http://dx.doi.org/10.1006/inco.2000.2894`. (Cited on page 26.)

[Wil01] Thomas Wilke. Alternating tree automata, parity games, and modal μ-calculus. *Bull. Belg. Math. Soc. Simon Stevin*, 8(2):359–391, 2001. URL: `http://projecteuclid.org/euclid.bbms/1102714178`. (Cited on pages 35 and 51.)

[Win98] Glynn Winskel. A linear metalanguage for concurrency. In *AMAST '98*, 1998. (Cited on page 117.)

[Zie98] Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theor. Comput. Sci.*, 200(1-2):135–183, 1998. URL: `http://dx.doi.org/10.1016/S0304-3975(98)00009-7`. (Cited on page 52.)

Index