Contributions to the Design of Safe Complex Systems

Rabéa Ameur Boulifa

UNIVERSITÉ CÔTE D'AZUR

École Doctorale STIC

Sciences et Technologies de l'Information et de la Communication

Contributions to the Design of Safe Complex Systems

Contributions à la Conception des Systèmes Complexes Sûrs

Mémoire de Synthèse présenté à l'Université Côte d'Azur pour l'obtention d'une

HABILITATION À DIRIGER LES RECHERCHES

Spécialité Informatique

par

Rabéa AMEUR-BOULIFA

soutenue le 27 Janvier 2023

Jury:

Rapporteurs	Pr. Olga Kouchnarenko	Université de Franche-Comté, France
	Pr. Laure Gonnord	Institut Polytechnique Besançon, France
	Pr. Gwen Salaün	Université Grenoble Alpes, France
Examinateurs	Pr Ileana Ober	Université Paul Sabatier, France
	DR Robert De Simone	INRIA Sophia-Antipolis, France
Invité	DR Eric Madelaine	INRIA Sophia-Antipolis, France

Contents

List of Figures

À Yemma

Chapter 1

Introduction – Français

La simplicité n'a pas besoin d'être simple, mais du complexe resserré et synthétisé.

Alfred Jarry.

1.1 Résumé

Mes travaux de recherche s'inscrivent dans un cadre qui vise à développer des approches formelles pour aider à concevoir des systèmes complexes (distribués et embarqués) à large échelle, avec un bon niveau de sûreté et de sécurité. Plus précisément, mes travaux consistent à proposer des outils théoriques et pratiques permettant de simplifier la modélisation et la vérification des systèmes et des applications à concevoir.

Il est bien connu que la complexité des systèmes, notamment des systèmes concurrents, les problèmes de fiabilité et les contraintes de délais de mise sur le marché sont des exemples de certains défis actuels qui poussent les méthodologies de conception existantes à leurs limites. Les systèmes distribués en particulier se caractérisent par un grand nombre d'entités, parfois hétérogènes, en interaction. Elles sont souvent décrites avec différents modèles (matériels et logiciels), différents langages de programmation et de description, et utilisent divers moyens de communication. Il arrive aussi que ces systèmes soient en perpétuelle évolution, les entités sont conçues dans l'objectif d'évoluer et se veulent extensibles afin de prendre en charge des fonctionnalités non prévues initialement. Leur étude nécessite alors de combiner plusieurs points de vue (différents niveaux de spécification et d'abstraction, différents formalismes,...).

Confronté à des modèles excessivement complexes de par leurs tailles, et par la variété des interactions, le défi est l'intégration rigoureuse de ces différents aspects d'un système au sein d'une démarche unifiée de modélisation et de vérification. Les modèles utilisés jusqu'alors pour la modélisation et la vérification des comportements des systèmes montrent aujourd'hui leurs limites. Par exemple, l'aspect variabilité ou paramétrique d'un système multi-entités devient difficilement "représentable" dans ces modèles. Et le caractère "extensiblité" n'en est pas moins difficile à représenter.

Pour faire face à ces problèmes, *nous proposons un cadre théorique et pratique, autour d'un formalisme appelé pNets, qui facilite la modélisation et la vérification des systèmes complexes (et concurrents). Le cadre pNets fournit une approche qui permet une conception rigoureuse et incrémentale des systèmes grâce à une modélisation modulaire, hiérarchique, compositionnelle et symbolique.*

Des pNets...

Dans le cadre de mes travaux de doctorat [33], nous avons défini un formalisme permettant la spécification compositionnelle de systèmes complexes. Ce formalisme, nommé *pNets* (pour *parameterized Networks of automata*), permet d'exprimer le modèle sémantique comportemental d'un système à travers la composition hiérarchique des modèles de ses sous-systèmes, qui eux sont représentés par des systèmes de transition étiquetés paramétrés nommés *pLTSs* (pour *parameterized labelled transition systems*). Le modèle des pNets étend les modèles des algèbres de processus traditionnels pour:

- modéliser aisément des familles de processus, mais aussi les interactions entre les éléments de ces familles en se basant directement sur l'index de la source et de la cible, sans passer par des canaux de communication artificiels.

- prendre en compte la composition hiérarchique des processus, à un niveau sémantique très expressif: nous avons conçu pour cela une extension des vecteurs de synchronisation d'Arnold et Nivat [14], permettant un codage sémantique flexible de modes de synchronisation très variés, plutôt que de nous limiter à un choix restreint d'opérateurs de parallélisme.

- introduire un codage explicite des données, tant sous forme de communication "value-passing", que pour la description de topologies paramétrées de processus.

- définir des techniques permettant de prouver en pratique l'équivalence comportementale de deux systèmes donnés.

Le modèle *pNets* et la composition hiérarchique des pLTSs, permet une spécification de bas niveau d'un système pour exprimer et réfléter son comportement global à travers la spécification de ses sous-systèmes. L'aspect paramétré, permet entre autres de raisonner sur des systèmes de taille variable voire infinie, ce qui est crucial pour la conception et la programmation distribuées. Les modèles paramétrés peuvent être instanciés ou assemblés pour construire de plus grands systèmes. L'autre propriété importante pour ces modèles est la "compositionalité". Outre la modélisation qui est compositionnelle, la vérification peut être aussi accomplie de manière compositionnelle; ce qui signifie que les propriétés logiques, et les équivalences, peuvent être vérifiées localement, et seront garanties préservées par composition.

Cette contribution à la théorie des systèmes concurrents a été utilisée, dans un premier temps, pour la spécification des applications Java, en particulier des applications implémentées avec le middleware ProActive [33]. Les modèles pNets ont servi alors pour définir des procédures de génération automatique de modèles comportementaux pour les objets actifs de la bibliothèque ProActive, avec leurs mecanismes de communication: les queues de requêtes asynchrones ainsi que les futurs de première classe [23, 32]. Ils ont aussi été utilisé pour la modélisation et l'analyse de plusieurs cas d'étude réalistes [7, 8, 15, 64] dont la complexité dépassait largement les possibilités des outils de vérification de l'époque.

Par la suite, dans le cadre d'autres travaux (e.g. [20]), le modèle pNets a été utilisé pour définir et exprimer la sémantique des applications développées par une approche à base de composants. Celles-ci sont des assemblages de composants avec leur structure d'encapsulation, leurs possibilités de reconfiguration dynamique, et dotées parfois de contrôleurs non-fonctionnels. Naturellement, l'approche pNets qui est par essence une approche compositionnelle se prête particulièrement bien pour l'analyse de ce type d'applications à base de composants, notamment de composants logiciels. À travers les travaux menés sur les modèles de composants, nous avons montré que les modèles pNets pouvaient exprimer formellement la sémantique comportementale des composants Fractal (e.g. [21, 34]). Nous avons étendu les modèles et créé une nouvelle version [10] pour sous-tendre la sémantique des modèles de composants GCM (Grid Component Model) (e.g. [25, 50]), et plus généralement, pour exprimer la sémantique des langages de programmation à base de composants asynchrones (adaptables ou autonomes) qui supportent les mecanismes pour structurer les applications, et d'autres caractéristiques comme les futurs de première classe et leurs stratégies de mise à jour ainsi que les communications de groupe.

Une plateforme, nommée VerCors (pour VERification de modèles pour COmposants Répartis communicants, sûrs et Sécurisés), a été construite sur la base du formalisme pNets (e.g. [19, 44, 73]). L'outil offre un éditeur graphique pour la spécification des architectures de composants, accessible aux non-experts; il offre également des passerelles vers les outils de minimisation et de model-checking de CADP (e.g. [61]). VerCors utilise le formalisme pNets comme sémantique de base pour la modélisation du comportement des systèmes, mais utilise comme langage intermédiaire

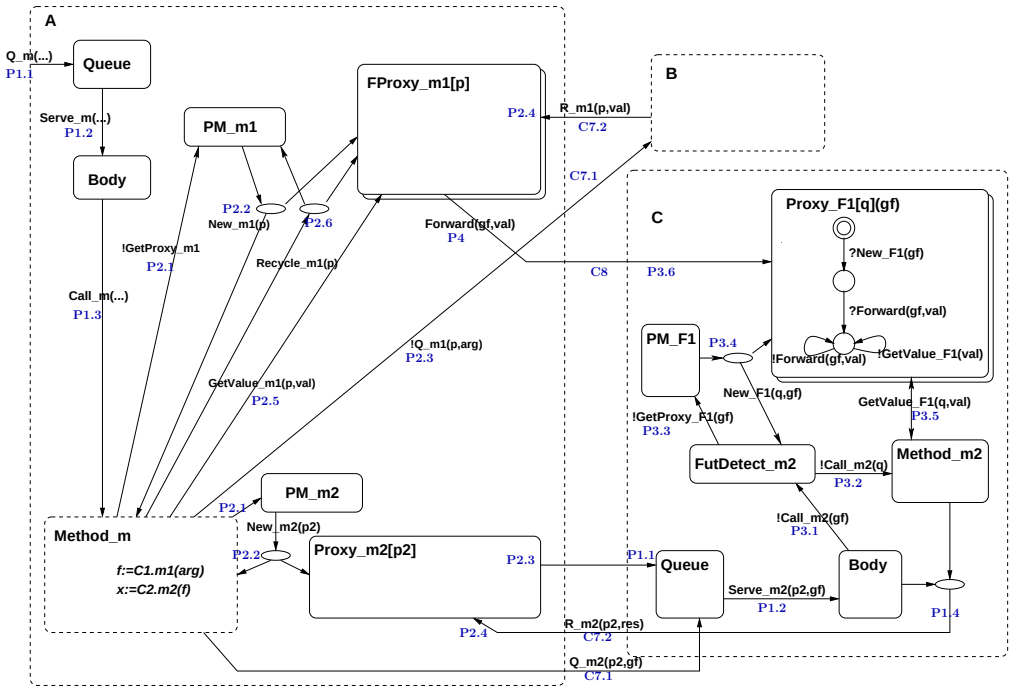


Figure 1.1: Structure pNets d'un composant GCM avec sa Queue de requêtes asynchrones et ses Futurs (Figure extraite de [10])

le langage appelé Fiacre[1](pour Format Intermédiaire pour les Architectures de Composants Répartis Embarqués) [29]. Ce langage pivot, a été défini avec les partenaires du projet appelé également Fiacre, pour servir comme langage formel commun à plusieurs outils de vérification, il est lui même basé sur la notion de processus et de constructeurs de composition hiérarchique.

Vers les Open pNets.

Tout en poursuivant l'objectif premier de la modélisation et la vérification des systèmes paramétrés, extensibles et compositionnelles, dans nos travaux de ces dernières années, la phase deux visait à accroître la puissance du formalisme de spécification pNets, pour capturer les aspects de l'évolution du comportement des composants et supporter l'idée de systèmes partiellement spécifiés. La sémantique comportemantale des pNets, ainsi que les algorithmes de vérification, sont adaptés pour l'analyse des systèmes fermés (clos), c'est-à-dire les systèmes statiques et non évolutifs. Cependant, ils ne permettent pas la modélisation de systèmes dont la spécification est partielle ou incomplète. C'est pour cette raison que la notion d'open pNets a été introduite, pour offrir la possibilité de représenter l'ouverture d'un système et l'incomplétude d'une spécification. Les open pNets sont des pNets avec des trous permettant de décrire en outre des processus inconnus. Dans ce formalisme, les indices de processus sont utilisés comme moyen pour encoder des composants non spécifiés (inconnus) et dont les comportements sont décrits que par l'ensemble des actions acceptés par le composant (la sorte du trou), sans aucune spécification sur l'ordre dans lequel elles seront exécutées.

L'extension de la théorie des pNets à la théorie des open pNets permet de raisonner sur des systèmes complexes ouverts ou partiellement spécifiés. Les travaux sur les Open pNets ont démarré sur l'initiative entre autres d'Eric Madelaine (chercheur INRIA), Ludovic Henrio (chercheur CNRS – ENS Lyon) et (Min Zhang (Chercheuse ECNU – Université Shangaï) [75].

[1]Fiacre sur `http://projects.laas.fr/fiacre/`

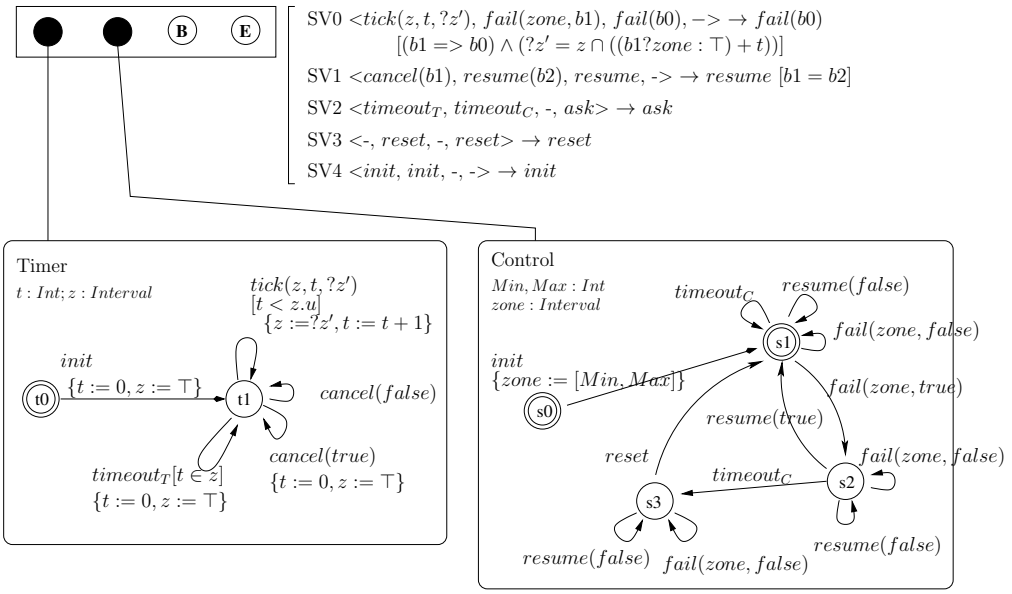


Figure 1.2: Un exemple d'open pNet modélisant un Failure Monitor (Figure extraite de [103])

Nous les avons poursuivis pour définir une théorie pour raisonner sur l'équivalence des systèmes, selon les notions de bisimulation symbolique et d'équivalence comportementale. Nos principales contributions à cette recherche sont les suivantes:

1. La définition de la notion d'open automata (d'automates ouverts): une algèbre d'automates paramétrés avec des trous qui constitue une sémantique de base pour les systèmes concurrents ouverts, notamment les pNets ouverts. Aussi, nous avons défini des règles de traduction des pNets ouverts en automates ouverts.

2. La définition de la théorie permettant de raisonner sur les pNets ouverts selon la notion d'équivalence comportementale. En particulier, nous avons avons défini la notion de relation de bisimulation entre automates ouverts. Le travail s'est fait en deux étapes: dans un premier temps, nous avons adapté la relation de bisimulation forte déjà définie et introduite dans [75] pour permettre de montrer que cette relation est une relation d'équivalence. Nous avons également établi une preuve complète que cette relation de bisimulation est satisfait les propriétés de compositionnalité. Dans l'étape suivante, soucieux de minimiser l'effort de vérification des modèles et d'ignorer les étapes de calcul, i.e. les actions qui ne modifient pas le comportement "visible" d'un système, nous avons introduit la notion de transition ouverte faible pour définir la théorie de bisimulation faible. Des algorithmes ont été développés pour le calcul de la sémantique symbolique en termes d'automates ouverts; ces derniers sont utilisés pour l'analyse d'une étude de cas réaliste: l'analyse du logiciel de contrôle embarqué des satellites [103, 104].

3. Le modèle d'automate ouvert défini peut être considéré comme une description partielle sur un système global, où les détails inutiles des autres composants et de l'environnement externe ont été abstraits. À partir de ce modèle initial partiel (abstrait), les détails des composants peuvent être introduits par raffinement de composants séparés, et les descriptions résultantes peuvent être synthétisées par composition en un modèle détaillé du système global. Pour garantir une substituabilité sûre d'une version abstraite d'un composant par sa version raffinée dans une telle démarche, nous proposons une définition du raffinement des automates ouverts en terme de simulation. Nous

avons donc défini une relation de simulation entre les open pNets qui joue le même rôle que la simulation entre les systèmes de transition classiques, avec des contraintes particulières et des conditions supplémentaires pour traiter les sortes des trous. La relation simulation définie sur les automates ouverts capture le principe selon lequel un composant en raffine un autre s'il présente moins de trous, ou s'il impose plus d'ordre sur les actions de ses trous. Bien qu'elle ne donne pas des garanties sur la préservation des propriétés de sûreté, la relation de simulation permet de garantir qu'une implémentation est bien une extension d'une spécification, une propriété particulièrement utile dans certains cadres, notamment en programmation.

1.2 Structure du Document

Le reste du document est organisé comme suit:

Le chapitre 3 rassemble des éléments de comparaison avec d'autres travaux de recherche directement liés à notre travail: approches de conception – de modélisation et de vérification – compositionnelle permettant de spécifier et de vérifier des systèmes de manière compositionnelle; nous discuterons des modèles symboliques qui permettent de vérifier des systèmes paramétrés, mais aussi des systèmes sensibles aux données; et enfin nous discuterons la notion de raffinement qui permet de relier des spécifications à des implémentations correctes.

Le chapitre 4 décrit notre théorie de spécification compositionnelle. Celle-ci basée sur le formalisme pNets pour le codage des systèmes fermés (clos) et sur les Open pNets pour le codage des systèmes ouverts. Ces modèles adaptés pour décrire la sémantique des systèmes distribués, et supporte parfaitement l'approche compositionnelle. L'article principal sur le modèle open pNets qui a été soumis au journal Logical and Algebraic Methods in Programming, JLAMP [11] est inclus dans le chapitre.

Puis le chapitre 5 présente une autre contribution au développement de modèles systèmes compositionnels à travers une approche incrémentale. Les open automata sont considérés comme des spécifications partielles de systèmes, qui peuvent être raffinées par composition. Les détails techniques de l'approche sont donnés dans l'article inclus: un papier en cours d'une une soumission.

Enfin, le document sera terminé par une analyse de l'état actuel de ce travail. Ce dernier ouvre plusieurs directions pour des travaux futurs que nous prévoyons d'explorer dans les années à venir.

Chapter 2

Introduction – English

Everything should be made as simple as possible, but not simpler.

Albert Einstein. Oct. 1977.

2.1 Summary

My research work is part of a framework that aims to develop formal approaches to help design complex systems with a good level of safety and security. More precisely, my work consists in proposing a theoretical and practical framework allowing to facilitate the modelling and verification of complex distributed and embedded systems.

It is well known that the complexity of systems, especially concurrent systems, reliability issues and time-to-market constraints are examples of some of the current challenges that push existing system design methodologies to their limits. These systems are characterised by a large number of interacting heterogeneous entities, described through different models (hardware and software), various programming languages, and use various means of communication. Moreover, these systems are often in perpetual evolution, the entities are designed with the objective of evolving and are intended to be extensible in order to cope with unplanned or unanticipated functionalities. Their analysis requires the combination of several points of view (different levels of specification and abstraction, various formalisms,...).

Faced with the complexity of models through their size, and the variety of interaction mechanisms, the challenge is the rigorous integration of these different aspects of a system within an unified and rigorous approach. In other words, the challenge is to define a procedure for integrating several perspectives within a single model. The models used so far for the modelling and behavioural verification of systems have revealed their limitations. For instance, most formalisms are not able to "represent" the variability or parametric aspect of a multi-entities system. The "extensibility" (and infinite) aspect is no less difficult to model.

To address these issues, *we propose a theoretical framework, referred to as pNets, that can help to model and verify complex (and concurrent) systems. The pNets framework provides an approach that supports incremental design through symbolic compositional hierarchical modelling.*

From pNets...

During my PhD research [33], we defined a symbolic compositional hierarchical model called *pNets* (for parameterized Networks of automata) for systems modelling and verification. The pNets formalism allows to express the behavioural semantics of a system through hierarchical composition of the models of its subsystems, which are represented by parameterised labelled transition systems (pLTSs). The pNets model extends the classical models of process algebras in order to:

- ease the modelling of process families, and the interactions between the elements of these families, based purely on the source and target indices, without using artificial communication channels.

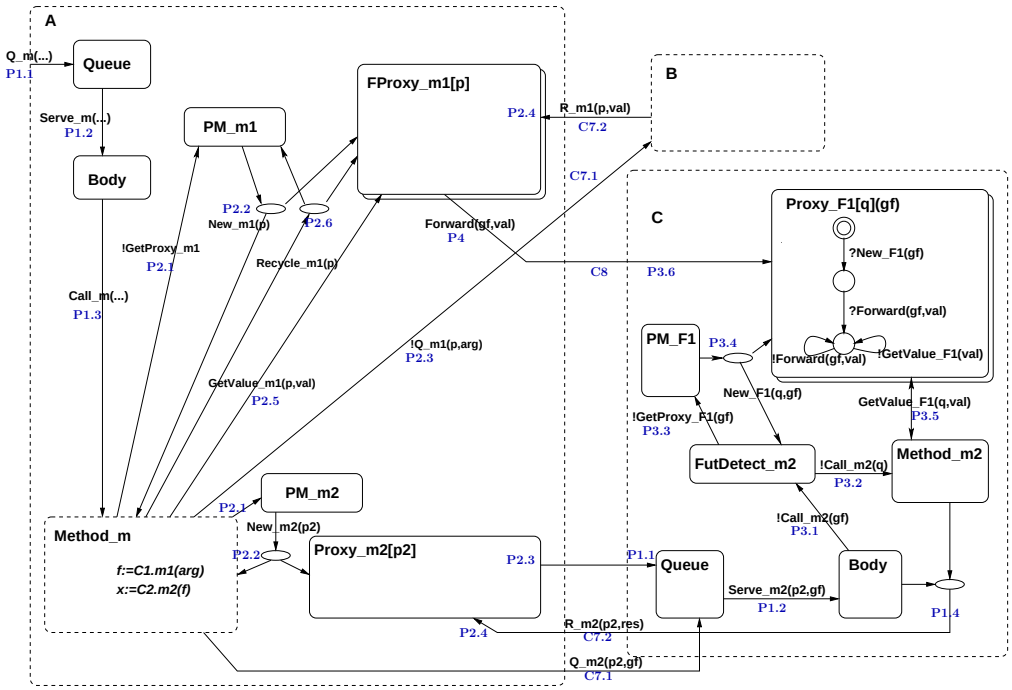


Figure 2.1: Structure pNets of a GCM component with its Queue of asynchronous requests and its Futures (Figure retrieved from [10])

- take into account the hierarchical composition of processes, at a high expressive semantic level: we have designed for this purpose an extension of Arnold and Nivat's synchronization vectors [14], providing a flexible formalism for encoding a wide variety of synchronization modes, rather than being limited to a restricted choice of parallelism operators.

- include explicit coding of data in models, both as communication parameters for concurrent value-passing systems, and as parameters for the description of parameterised process topologies.

- provide techniques and tools based on equivalence-checking notion, for determining the establishing equivalence of systems.

The pNets and pLTSs models allow the low-level beahavioural specification of a system through specification of its subsystems. The parameterised feature allows us to reason about variable-sized systems, or even potentially infinite-sized systems, which is a crucial aspect for distributed design and programming. The parametrized models can be instantiated, or assembled to build larger systems. Another key feature of pNets models is the "compositionality". This formalism allows to build a models in a compositional and hierarchical manner, but also allows to perform compositional verification, which gives a significant advantage for the verification technique that relies on model checking.

Our contribution to the theory of concurrent systems has been used, in the early work, for the specification of Java applications, which are developed over ProActive middleware [33]. The pNets models were then used to develop procedures for the automatic generation of behavioural models for active objects within ProActive library, with their various mechanisms of communication and features, as asynchronous request queues and first class futures [23,32]. The potential of pNets model is demonstrated through many convincing and realistic case studies [7,8,15,64], in which the complexity of some case studies was far beyond the capabilities of the existing tools.

Subsequently, in other work [20], the models have been used in the area of component-based software engineering. The pNets formalism is used to express the semantics of applications developed by a component-based approach. These applications are basically assemblies of components – program units – with their specific characteristics as the notion of encapsulation structure, dynamic reconfiguration possibilities, and sometimes with non-functional controllers. Naturally, the pNets approach, which is basically a compositional approach, is particularly well-suited to the analysis of such applications. In this context, we have shown that pNet models can formally express the behavioural semantics of Fractal component models [21,34]. Further in this line, we have extended an earlier version (given in [10]) to cover the complete behavioural semantics of the Grid Component Model (GCM) [25, 50], and more generally to address a broader class of (adaptable or autonomous) component-based programming languages that support mechanisms for structuring applications and others features like first-class futures with their update strategies and also group communications.

A platform, called VerCors [19], has been built on the pNets formalism; the platform [44, 73] offers an user-friendly graphical editor for the specification and modelling of component architectures; and it is directly plugged to CADP minimisation and model-checking tools [61], that is used for the verification. The pNets models are used as the basic model for the description of systems, but at the implementation level the models are translated into Fiacre[1]intermediate language (for Intermediate Format for Embedded Distributed Component Architectures) [29]. The Fiacre language, that is supported by several verification tools, is also based on the notion of processes and hierarchical composition constructors.

The pNets model allows us to analyse full-size distributed systems. The parametric nature of the model and the properties of compositionability of the equivalence relations are thus the main strengths of our approach.

[1]Fiacre on `http://projects.laas.fr/fiacre/`

Towards Open pNets.

While pursuing the primary objective of modelling and verifying parameterised and extensible systems in a compositional manner, in a second phase of our research, we have tackled the problem of partial (or incomplete) specifications. pNets models, in their initial version, do not allow modelling of a partially specified system. The formalism and the verification algorithms based on the pNets were developed for the analysis of closed systems, i.e. the systems that are static and fully specified. But it appears that sometimes the behaviour of some sub-systems is unknown and there is still a need to analyse such systems.

For this reason, the notion of open pNets was introduced, to address this issue, i.e. to offer t the possibility of modelling open systems and the incomplete specification. Open pNets are pNets with holes that gives the ability to model open systems. The holes are placeholders inside the hierarchical structure that can be filled by sub-systems, i.e. they play the role of unknown processes. Each hole has an index as its identity and the set of actions that it can perform. Thus, the extension of pNets towards Open pNets provide a means to deal with partially specified systems and to reason on them. The research on Open pNets was initiated in collaboration with Eric Madelaine (INRIA researcher), Ludovic Henrio (CNRS - ENS Lyon researcher) and (Min Zhang (Chercheuse ECNU – Université Shangaï) [75], we have pursued them to develop a theory for reasoning about the equivalence of systems. Our main contributions to this research are the following:

1. The definition of the notion of open automata: an algebra of automata with parameters and holes that gives a basic semantics for concurrent systems, in particular for open pNets. Moreover, we have defined rules allowing to derive open automata from open pNets.

2. The definition of the theory for reasoning about open pNets: the behavioural reasoning through equivalences and preorders that provides a means for relating different systems expressed in the formalism. A notion of strong bisimulation over open automata has been defined and introduced

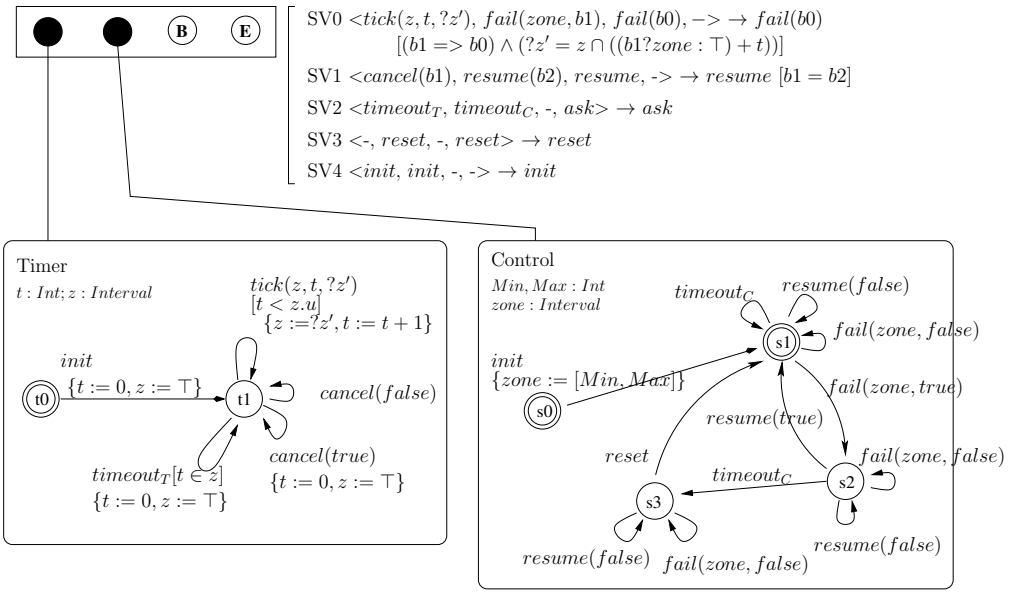


Figure 2.2: Example of open pNet encoding of the Failure Monitor architecture (Figure retrieved from [103])

in [75], so we extended this relation with an additional property in order to get an equivalence relation. Furthermore, we have developed a complete proof demonstrating that the bisimulation relation is compositional. Afterwards, we introduced the theory of weak bisimulation for open automata, and its properties. The theory relies on the definition of the notion of weak open transitions that are derived from transitions of the open automaton by concatenating invisible action transitions with one (visible or not) action transition. We showed that under certain condition of non-observability of actions, weak bisimulation relation is compositional. Algorithms been have developed for computing the symbolic semantics in term of open automata; they are used for the analysis of a realistic use-case based on the on-board control software of satellites [103, 104].

3. The open automaton model can be seen as a partial description of an overall system, where unnecessary details of other components and the external environment have been abstracted. From this initial partial (abstract) model, the details of components can be introduced separately by refinement, and the resulting models from the composition composition can be viewed as detailed model of the overall system. In order to ensure a safe substitutability of an abstract version of a component by its refined version in such an approach, we propose a definition of a refinement relation of open automata in terms of simulation. The simulation relation over open automata is in some way similar to the simulation over the classical labelled transition systems, with symbolic evaluation of the guards and transitions, but it also provides how to deal with the holes. It captures the principle that one component refines another if it has fewer holes, or if it introduces more order over the actions of the holes. Although, the proposed does not provide guarantees on the preservation of the safety properties, it provides a means to show that an implementation is an extension of some given specification, a particularly useful property in certain settings, as in programming context.

Open pNets model allows us to reason on composition operators as well as on full-size distributed open systems.

2.2 Organisation of the manuscript

The rest of the document is organized as follows:

Chapter 3 gathers elements of comparison with other research works directly related to our work: design – modelling and verification – compositional approaches to building and verifying systems; we will discuss symbolic models that allow us to express parametrised systems and to verify data-sensitive systems; and finally we will introduce the notion of refinement that allows us to relate specifications to theirs implementations.

Chapter 4 presents our theory of compositional specification. This is based on the pNets formalism for encoding closed systems and on open pNets for encoding open systems. Both models are convenient languages for expressing the semantics of distributed systems, and the description languages for modelling component-based applications. The main article on the Open pNets model, which is accepted to the journal Logical and Algebraic Methods in Programming (JLAMP), is included in the chapter.

Chapter 5 presents another contribution to the development of compositional system models through an incremental approach. Open automata are considered as partial system specifications, which can be refined by composition. Establishing refinement relations between systems is an important mean for verifying their correctness. Relying on open automata, we define refinement relations for the comparison of systems specified as pNets. Open automata are labelled transition systems with parameters and holes, which are adopted as a semantic view of open pNets. This part is illustrated by the paper published in the International Conference on Software Engineering and Formal Methods (SEFM'23).

Finally, the document will be concluded with an analysis of the current state of this work. This opens up several directions for future work which we plan to explore in the coming years.

Chapter 3

Related Works

Process description languages, as CCS [95] and CSP [76], are useful tools for the design of concurrent and distributed systems because of their compositional nature and their well-established semantic theory. Yet, it well-known that the major problem in applying automatic verification techniques to analyse even moderate sized concurrent systems is the potential combinational explosion of the state-space, arising from the combination of independent subsystems. Nevertheless, many progress has been made to combat this problem; Indeed, various techniques have been developed to tackle the state-space explosion, either by applying *compositional* reasoning which relies on "divide-and-conquer" strategies that (de)compose a global system into local concurrent subsystems (processes) and address each subsystem separately, by applying model checking based on *symbolic* models which instead of reasoning about the pure version of process algebras, it is done on their abstracted semantic version which allows to perform the analysis on the input values, or by applying an *incremental* approach which advocates starting from a system specification and deriving the corresponding implementation using refinement. But there is very little or no support that combine all these techniques. Before discussing our approach in the context of modelling and verification of concurrent systems, it is actually interesting to relate it to all these existing techniques.

In this chapter we give an overview of research on approaches for overcoming the state space problem, which related to the advocated approach through different perspectives, commenting the relations and differences when possible.

Compositional systems

Compositional Modelling.

The increasing complexity of distributed systems has led to the use of compositional techniques for their design, modelling and verification. The compositional approach is based on the "divide-and-conquer" principle. It allows a system to be divided into more manageable parts (sub-systems) that can be studied and modified relatively independently. This approach is effective in several area. For instance, in the field of software engineering, compositional programming approach is considered as entirely apart branch.
Component-based software approach is widely used by the programmers of applications on computing grids or clouds (e.g., [24,25,97]). Component models, like Fractal [38] and GCM [43], provide a structured programming paradigm, and more efficient means to ensure variety and re-usability of programs. From the modelling side, component models provide a convenient formalism (and abstraction) for specifying and verifying the correctness of systems: they provide structuring entities easing the verification. The concept of compositionality of models is widely used in several modelling languages, such as process algebras and UML [98] languages. Some formalisms offer some advanced constructs, as for Statecharts [71] that includes specific features such as hierarchy and broadcast communication. The hierarchical structure of statecharts (which is fundamentally a network property) introduces a new dimension to conventional compositional models, it extends traditional state machines with features such as multi-level (nested) hierarchy on states. pNets are also hierarchical structures, whereas in statecharts the

states are organised in a hierarchy, in pNets, the processes are hierarchical.

Compositionality.

The ultimate aim of compositional techniques is to provide a means for reasoning about the behaviour of a large system based on its subsystems. Verifying concurrent systems with a large number of subsystems can be notoriously hard to achieve due to the state explosion problem. Compositional verification can provide a powerful algorithm for state-space reduction.

The theoretical idea of the compositional verification approach has been introduced by Winskel in [112] thirty years ago. Later, Andersan demonstrated [13] its practical use and its potential in overcoming state-space explosion through the analysis of Milner's Scheduler use-case [91]. Since then, there have been several works in the field (e.g. [2, 5, 90]). In the literature, there are various forms of compositional verification. Amongst them, the approaches referred to as *compositional minimisation* or *compositional reachability analysis* (e.g. [47, 78, 108]). In these approaches, a system can be minimized by replacing a component (or process) of a system with an abstraction, simpler than the corresponding component while still preserving the property that is checked. Another alternative, leading to a reduction of the state-space, is the use of the interface constraints technique (e.g. [46,67,81]); this technique allows to impose restrictions on the component behaviour, when the latter cannot be minimised in isolation from its environment, i.e., from the other components. Compositional minimisation approaches rely on equivalence-checking technique for verification. This technique consists in checking behavioural equivalence and preorder relations between the labelled transition systems that model the behaviour of systems. The pNets verification approach is part of the general compositional minimisation approach and is based on model checking technique.

We find interesting to cite here the compositional reasoning approach, referred to as *assume-guarantee* approach (e.g. [12, 66, 107]). This approach also aims at overcoming state-explosion, although different from the previous one and therefore from the approach adopted with pNets, since it extends the system model with assumptions about the environment. It is based on model checking technique, but the verification of a global system is conducted on a conjunction of its subsystems (local specifications), including additional assumptions that should constrain the system behaviour.

In addition to the fact that the compositional verification approaches allow the analysis of systems by (de)composition, they offer a significant benefit for the verification of their properties. Actually, most of them support the quotienting methods (e.g. [78, 84]); these methods for reducing state-space are based on the principle of preventing the construction and the exploration of the whole set of states of a system, by gradually moving specification of components (subsystems) from the corresponding model into the properties to be verified.

Compositional technique has been used with success for the verification of concurrent systems and embedded software on many (even industrial) case-studies (e.g. [82, 86, 113]). In [63] the authors have reported and highlighted an extensive list of case studies. Despite its great potential in overcoming state-space explosion as demonstrated in many convincing case studies, compositional approach is not used widely enough in practice. There are very few practical tools that support such an approach (e.g. [48, 62, 68]) because modelling (and therefore verification) still faces other limitations that pose significant issues for scalability and decidability, such as the problem of data transmission. This is specifically what we target here.

Symbolic and data-sensitive systems

Another alternative used for overcoming the state explosion problem is the symbolic representation of the system. When searching the literature for the keyword *symbolic approach*, many results are related to the *Symbolic Bisimulation* notion. This notion refers to a domain that is not related to our topic. Indeed, Symbolic Bisimulation refers to the computation of state-space. An approach introduced in the early 1980s [57, 106], which is based on the symbolic (rather then enumerative) representation of the set of states

of a system. This approach has produced tools, that were widely used in practice with the advent of binary decision diagrams (BDDs) [39, 40].

The notion *symbolic approach*, that is closed to our research, allows reasoning about processes, which manipulate and exchange values. This feature of exchanging data between processes, and which is referred to as *value-passing*, was introduced by Milner [93] on CCS and Hoare [76] on CSP. In those works, the notion of data values is only expressed at the language level, which appears as a feature of its syntax. This additional feature for the transmission of data (with possibly infinite value domains), is considered as input values and has impacted on the already established theories for the processes calculi. As pointed out in [85] this leads to potentially non finite-branching transitions on which algorithms establishing behavioural equivalences fail to be decidable. In the literature, it has been recognized that an attractive solution to this problem, is to deal with data (inputs) symbolically, i.e., to rely on symbolic models and semantics for establishing equivalences.

The pioneering work in this area is done by Hennessy and Lin [72]. In their seminal paper [79], the authors introduced a notion of symbolic transition graphs and developed the associated symbolic bisimulation with various forms (early and late, strong and weak). The main idea of their approach is to assign to each action (over a transition) a formula describing the symbolic values used in the action. The notion of symbolic semantic has been used for establishing behavioural equivalence over various formalisms as for full LOTOS (e.g. [42]) or pi-calculus (e.g. [31, 41, 54, 87]), even for quantum processes (e.g. [59]).

Due the symbolic nature of the constructs of open pNets, our approach also fits into this field of symbolic semantics. However, all the above works are based on models definitely different from ours, and none of them allows systems to be as much parameterised as open pNets.

Open systems and refinement

Like the behavioural bisimulation equivalences, the refinement orderings provide a basis for compositional reasoning about concurrent systems [49]. The notion of refinement captures the relation between a specification and an implementation of the same component. Refinement entails that one system can be considered as a more precise version of the specification, featuring all the specified behaviours with more concrete details. An elegant way to check whether an implementation meets a specification is to build a refinement (preorder) relation that relates their state models; It captures the idea of more behaviour and reflects the fact that an implementation is "at least as good as" the specification.

The notion of refinement was initially introduced to allow reasoning on the correction of programmes [56], then it was formalised as theory (e.g. [17]), and then used in different verification frameworks (e.g. [1]). Its particular strengths it adds a further dimension to compositionality, and has even given rise to a complete methodology for designing systems through the well known B Method [3].

In the process algebra setting, the refinement is addressed either by the verification of inclusion between the sets of traces recognized by processes, or by the simulation-based approach (e.g. [27, 28, 80, 92]). These approaches ensure that all behaviours of the specification must belong to the behaviours of the implementation. Most works on the refinement in the literature focus on closed systems [70], whose behaviour is fully defined and completely determined by labelled transition systems. They cannot be used as such to deal with open systems whose behaviour depends on their interaction with the environment, and which require reasoning about unspecified behaviours.

There is few work that has addressed the refinement of open systems (e.g. [55, 114]). Defining refinement of open systems as trace inclusion is addressed as a notion of subtyping in type theory (e.g. [35, 65]). Such refinement is instead based on an interface-oriented approach, to give the ability to characterise behaviour it allows the expression of (internal and external) choices. The refinement of open systems is also tackled in terms of alternating simulation (e.g. [6,52]), which deals with game-based models. Alternating simulation that is originating from the game theory [51] allows the study of relations between individual components by viewing them as alternating transition systems [110]. In particular, a refinement of game-based automata expresses

that the refined component can offer more services (input actions) and fewer service demands (output actions). The difference with the open pNets is mainly attributable to the operation of composition which is rather specialised in the case of game approach [52], whereas in our approach it is inclusive. In order to establish the condition of composability, we use the notion of comparability of holes (similar to the notion of compatibility of game approach), that is explicitly encompassed in the definition of composition.

The result of our preliminary investigation and research on open automata confirms our belief that this formalism can provide theoretical foundations and a great support for the vertical dimension of compositional verification. It would therefore be interesting to have a further exploration, in particular to examine other semantics of (bi)simulations and refinements (e.g. those highlighted in [58]).

Chapter 4

A Theory for the Composition of Open Systems

4.1 Summary

In computer science a parallel and distributed system consists of several cooperating components, which may be called processes. In order to study such systems, dealing with interactions, concurrency theory is generally used, which is the fundamental theory of interacting, parallel and distributed systems. Process calculi (or process algebra) is usually considered as an approach to concurrency theory [18, 60], as it provides means to describe such systems, and means to talk about parallel composition. Besides this, it allows reasoning about such systems using basic laws. Behind the theory of process algebra and its mathematical models there exist many variants of modelling languages, among them CCS [91, 93], CSP [37], LOTOS [30] and MEIJE [16] that are used to model concurrent systems and applied to solve real-life problems in various areas, including telecommunication protocols, distributed software, and hardware circuits.

Along the same lines as this research, for the least fifteen years, we have defined a behavioural semantic model extending the existing semantic models for concurrent and distributed systems. This model called pNets (as parameterised Networks of synchronised automata) [22, 23] allows:

- hierarchical composition of processes, at a very expressive semantic level, through synchronisation vectors, allowing to express a wide range of classical synchronisation patterns and various synchronization modes, instead of the limited choice of parallel operators. Synchronization vectors allow also to express data transmission.

- explicit handling of data, parameters can be used as local variables inside parametrized labelled transition systems (pLTSs) and for "value-passing communication"; they can also be used to define families of pNets of variable size and for the description of indexed process topologies.

In technical terms, pNets are constructors for hierarchical behavioural structures: a pNet is formed by other pNets, or pLTSs at the leaves of the hierarchy tree. A composite pNet consists of a set of sub-pNets, each exposing a set of actions according to its sort. The synchronisation between a global action of the pNet and actions of the sub-pNets is achieved through synchronisation vectors. The expressiveness of the pNet model has been demonstrated through real-world and real-size use-cases [7,9], among the more complex examples include the specification and formal analysis of the Chilean electronic invoicing system [15]. The pNets model has also been shown that it is well-suited for formalising crucial aspects of distributed components: like reconfiguration and diverse mechanisms of communication (e.g. [8, 10, 34]). More importantly, it has been demonstrated that pNets can provide a theoretical background and a very rich environment for building heterogeneous component-based applications [10, 21]. In particular, the pNets formalism has been used as a low level semantic framework for expressing the behaviour of various component models: as semantics for the Fractal model [38] which is a

standard component model for programming components for the Grid, and for the GCM model [4] and its GCM/ProActive [43] with their specific features, including first-class futures and group communication.

Subsequently, the pNets formalism has been proven to be a particularly well-suited model for specifying open systems. In pNets formalism, and more generally in process algebra languages, the term system (and sub-system) means the behaviour of a process, i.e. a system is anything showing behaviour, like the execution of a software system. Behaviour is determined by actions or events that the system can perform, but also by the order in which they can be executed (and possibly other aspects of the execution). The pNets formalism allows us to deal with a notion of closed systems, whose behaviour is totally determined, meaning that the behaviour of each process is fully specified. But sometimes the specification of certain components (processes) of a system is unknown, which makes the specification of the global system partial. These types of systems, whose behaviour is partially specified, are called open, in contrast with closed systems. Their modelling requires a means of expressing both the known and unknown parts. pNets formalism lacks constructs to express such systems.

Consequently,

This has led to the transformation of the formalism from a (closed) pNets to an Open pNets. Similarly to pNets, open pNets model offers means of describing and specifying systems in a compositional and symbolic manner, but also means of talking about partially specified systems, i.e. about open systems. Technically speaking, similar (closed) pNets the structure of open pNets is hierarchical, while their leaves can be either automata (pLTSs) or "holes", playing the role of process parameters. The holes are used as placeholders for unknown processes, of which we only specify the set of possible actions that it can perform. They can be instantiated by pNets through composition mechanism. The open pNets formalism was introduced by Henrio, Madelaine and Minh [74, 75]. The expressiveness of language and its ability to encode open systems has been shown in several examples. In particular, the way of coding the operators of various classical process algebras.

Example

To illustrate the expressive power and semantic completeness of this formalism, consider the following LOTOS specification: `P>>(acc(v);Q)` that includes the Enable and Prefix operators of the LOTOS language. For the readers not familiar with LOTOS, the enable operator, denoted `>>`, expresses sequential composition, it is seen as a special case of parallel composition. The behaviour of the process on its left must terminate successfully in order for the process on the right to be enabled. The waiting is obtained on a special gate δ. In Figure 4.1 we show the encoding of the formula using pNets. This pNet is composed of: a toplevel node PN_1 encoding the Enable operator, a pLTS C_1 acting as the controller of the node; a hole representing the process variable P; another node PN_2 representing the Prefix operator, with its controller C_2, and its hole Q.

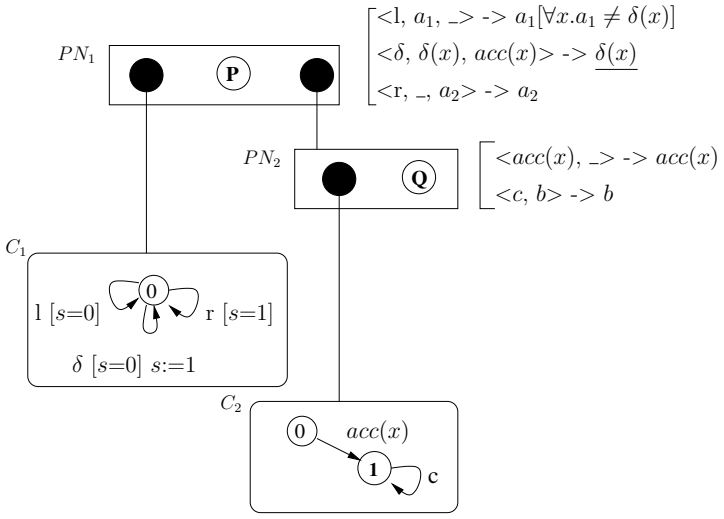


Figure 4.1: Open pNet encoding LOTOS expression

The root pNet node PN_1 has 3 sub-pNets: its controller C_1, the hole P and the sub-pNet PN_2. For this pNet system, the sort Action includes the actions of all processes, i.e. both the actions of holes and controllers: $\{l, \delta, r, a_1, a_2, b, c, acc\}$. Three synchronisation vectors define the possible synchronisations between the actions of the subnets. The first vector "$< l, a_1, \_ > - > a_1$" transmits an action a_1 of P to the upper level. Here the "l" action is related to the corresponding transition of the controller C_1, "_" means that the corresponding subnet is not involved

in this synchronisation, and there is a guard meaning this synchronisation can only occur if a_1 is not a δ. The second synchronisation vector composes an action $\delta(x)$ from P with an $acc(x)$ from Q, transmitting a value x before Q takes the control; The vector result $\underline{\delta(x)}$ is a "synchronized" action, that cannot be further synchronized at upper levels. This notion of synchronized actions is a useful generalization of the notion of internal actions, that will be convenient for observing internal events during model-checking.

Note that unlike most process calculi that use a "state-oriented" encoding style, the pNet model allows two encoding styles, "state-oriented" and "data-oriented", which are used to encode the two controllers. Controller C_2 is "state-oriented": it has two states to indicate the position of the control point in the Prefix expression $(acc(v); Q)$. The second one is "data-oriented": controller C_1 has a single state, with a state variable s encoding the change of control point from the left to the right subnet.

Comparing processes is at the core of the study of process algebra. Indeed, in various application areas, there is a desire to compare process models, for instance, to relate a discovered model to an existing reference model. It is widely recognized that behavioural equivalence is the most basic notion of the concurrency theory, as reflected by its very rich literature. A board spectrum of notions of equivalence (e.g., trace equivalence, bisimulation, branching bisimulation, etc.) are provided (e.g. [105]). The various types of equivalence relations capture the different theoretical semantics of concurrency, and allow to decide about the behavioural equivalence of processes on various levels. Trace equivalence (e.g. [37,77]) and bisimulation equivalence (e.g. [94, 99]) characterise the two extremes of the spectrum, in between them there is a whole lattice of equivalence relations [109].

To establish the theoretical foundations of open pNets, naturally we considered their semantics. We examined the notion of behavioural equivalence (equality) over open pNets models, and the compositionality properties that they offer. As a first step in our research, we focused mainly on strong bisimulation and weak bisimulation as they are the central notions of equivalence in process theory.

The results of our investigations and research in this field are summarised below:

- Bisimulation equivalences for open pNets are obtained by introducing open labelled transitions systems (open automata) as their semantic model: open transitions in open automata carry some sort of guarded action labels. Another way to consider these is that each represents a potentially infinite set transitions.

- A notion of strong bisimulation is defined, whose key feature is the possibility to simulate a single open transition with a set of open transitions. This bisimulation called FH-bisimulation (FH as Formal Hypotheses) stating that the bisimulation relation is achievable after imposing a formal assumption on the unknown components of the system.

- We have proven the composability of the theory. This key property states: *If two systems are proven equivalent they will be undistinguishable by their context, and they will also be undistinguishable when their holes are filled with equivalent systems.*

- We designed a weak bisimulation theory for open automata, with the study of its key properties (the technical details of this work are given and discussed in the paper included in the following section).

4.2 Paper Accepted to JLAMP

Compositional Equivalences Based on Open pNets

Rabéa Ameur-Boulifa

LTCI, Télécom Paris, Institut Polytechnique de Paris, France

Ludovic Henrio*

Univ Lyon, EnsL, UCBL, CNRS, Inria, LIP, F-69342, LYON Cedex 07, France.

Eric Madelaine

INRIA Sophia Antipolis Méditérannée, UCA, BP 93, 06902 Sophia Antipolis, France

Abstract

Establishing equivalences between programs is crucial both for verifying correctness of programs and for justifying optimisations and program transformations. There exist several equivalence relations for programs, and bisimulations are among the most versatile of these equivalences. Among bisimulations one distinguishes strong bisimulation that requires that each action of a program is simulated by a single action of the equivalent program, and weak bisimulation that allows some of the actions to be invisible, and thus not simulated.

pNet is a generalisation of automata that model open systems. They feature variables and hierarchical composition. Open pNets are pNets with holes, i.e. placeholders that can be filled later by sub-systems. However, there is no standard tool for defining the semantics of an open system in this context. This article first defines *open automata* that are labelled transition systems with parameters and holes. Relying on open automata, it then defines bisimilarity relations for the comparison of systems specified as pNets. We first present a strong bisimilarity for open pNets called FH-bisimilarity. Next we offer an equivalence relation similar to the classical *weak bisimulation* equivalence, and study its properties. Among these properties we are interested in compositionality: if two systems are proven equivalent they will be indistinguishable by their context, and they will also be indistinguishable when their holes are filled with equivalent systems. We identify sufficient conditions to ensure compositionality of strong and weak bisimulation. The contributions of this article are illustrated using a transport protocol as running example.

Keywords: Bisimulation, compositionality, automata, semantics.

*Corresponding author

Email addresses: rabea.ameur-boulifa@telecom-paris.fr (Rabéa Ameur-Boulifa), ludovic.henrio@cnrs.fr (Ludovic Henrio), eric.madelaine@inria.fr (Eric Madelaine)

1. Introduction

In the nineties, several works extended the basic behavioural models based on labelled transition systems to address value-passing or parameterised systems, using various symbolic encodings of the transitions [16, 37]. These works use the term *parameter* to designate variables whose value have a strong influence the system structure and behaviour. In parameterised systems, parameters can typically be the number of processes in the system or the way they interact. In [34, 26], Lin, Ingolfsdottir and Hennessy developed a full hierarchy of bisimulation equivalences, together with a proof system, for value passing CCS, including notions of symbolic behavioural semantics and various symbolic bisimulations (early and late, strong and weak, and their congruent versions). They also extended this work to models with explicit assignments [40]. Separately Rathke [28] defined another symbolic semantics for a parameterised broadcast calculus, together with strong and weak bisimulation equivalences, and developed a symbolic model-checker based on a tableau method for these processes. Thirty years later, no verification platform use this kind of approaches to provide proof methods for value-passing processes or open process expressions, perhaps because of the difficulty to apply these methods on industrial systems.

This article provides a theoretical background that allows us to implement such a verification platform. We build upon the concept of pNets that we have employed to give a behavioural semantics of distributed components and verify the correctness of distributed applications in the past 15 years. pNets is a low level semantic framework for expressing the behaviour of various classes of distributed languages, and as a common internal format for our tools. pNets support the specification of parameterised hierarchical labelled transition systems: labelled transition systems with parameters can be combined hierarchically.

We develop here a semantics for a model of interacting processes with parameters and holes. Our approach is originally inspired from Structured Operational Semantics with conditional premises as in [21, 47]. But we aim at a more constructive and implementable approach to compute the semantics (intuitively transitions including first order predicates) and to check equivalences for these open systems. The main interest of our symbolic approach is to define a method to prove properties directly on open structures; these properties will then be preserved by any correct instantiation of the holes. As a consequence, our model allows us to reason about composition operators as well as about realistic distributed systems. The parametric nature of the model and the properties of compositionality of the equivalence relations are thus the main strengths of our approach.

pNets. pNet is a convenient model to model concurrent systems in a hierarchical and parameterised way. The coordination between processes is expressed as synchronisation vectors that allow for the definition of complex and expressive synchronisation patterns. Open pNets are pNets for which some elements in the

hierarchy are still undefined, such undefined elements are called *holes*. A hole can be *filled* later by providing another pNet. This article first defines pNets and illustrates with an example how they can be used to provide the model of a communicating system.

A semantics for open pNets based on open automata. The semantics of pNets can be expressed as a translation to a labelled transition system (LTS), but only if the pNet has no parameter and no hole. Adding parameters to a LTS is quite standard [40] but enabling holes inside LTSs is not a standard notion.

To define a semantics for open pNets we thus need LTSs that have both standard variable parameters, and process parameters, i.e. holes that can be filled by processes. We call such LTSs with parameters and holes *open automata*. The main goal of this article is to define the theory behind open automata and to use them to provide a semantics and prove compositionality properties for open pNets. The transitions of open automata are much more complex than transitions of an LTS as the firing of a transition depends on parameters and actions that are symbolic. This article defines the notion of *open transition*, namely a transition that is symbolic in terms of parameters and coordinated actions.

Beware that even if open transitions may look similar to the notion of Transition System Specification [24, 23] and other forms of SOS rules, they are *not* structural rules, but rules defining the behaviour of the global states of the system.

Unlike pNets, open automata are not hierarchical structures, we consider them here as a mathematical structure that is convenient for formal reasoning but not adapted to the definition of a complex and structured system. Open transitions are expressed in terms of logics while the communication in pNets is specified as synchronisation vectors that specify synchronised actions. Open automata could alternatively be seen as an algebra that can be studied independently from its application to pNets but their compositionality properties make more sense in a hierarchical model like pNets.

Previous works and contribution

While most of our previous works relied on closed, fully-instantiated semantics [7, 2, 29], it is only recently that we could design a first version of a parameterised semantics for pNets with a strong bisimulation equivalence [30]. This article builds upon this previous parameterised semantics and provides a clean and complete version of the semantics with a slightly simplified formalism that makes proofs easier. It also adds a notion of global state to automata. Moreover, in [30] the study of compositionality was only partial, and in particular the proof that bisimularity is an equivalence is one new contribution of this article and provides a particularly interesting insight on the semantic model we use. The new formalism allowed us to extend the work and define weak bisimulation for open automata, which is entirely new. This allows us to define a weak bisimulation equivalence for open pNets with valuable compositionality properties. To summarise, the contribution of this paper are the following:

- The definition of open automata: an algebra of parameterised automata with holes, and a strong bisimulation relation. This is an adaptation of [30] with an additional result stating that strong FH-bisimilarity is indeed an equivalence relation.
- A semantics for open pNets expressed as translation to open automata. This is an adaptation of [30] with a complete proof that strong FH-bisimilarity is compositional.
- A theory of weak bisimulation for open automata, and a study of its properties. It relies on the definition of weak open transitions that are derived from transitions of the open automaton by concatenating invisible action transitions with one (visible or not) action transition. The precise and sound definition of the concatenation is also a major contribution of this article.
- A resulting weak FH-bisimilarity equivalence for open pNets and a simple static condition on synchronisation vectors inside pNets that is sufficient to ensure that weak FH-bisimilarity is compositional.
- An illustrative example based on a simple transport protocol, showing the construction of the weak open transitions, and the proof of weak FH-bisimulation.

What is new about open automata bisimulation?

Bisimulation over a symbolic and open model like open pNets or open automata is different from the classical notion of bisimulation because it cannot rely on the equality over a finite set of action labels. Classical bisimulations require to exhibit, for each transition of one system, a transition of the other system that simulates it. Instead, bisimulation for open automata relies on the simulation of each open transition of one automaton by *a set of* open transitions of the other one, that should cover all the cases where the original transition can be triggered. This is similar to the early and late symbolic bisimulation equivalences for value-passing CCS [27], though we use more general definitions in our setting.

Compositionality of bisimilarity in our model comes from the specification of the interactions, including actions of the holes. This is quite different from the works on contextual equivalences, e.g. [37, 38]; we will provide a detailed comparison in Section 6. In pNets, synchronisation vectors define the possible interactions between the pNet that fills the hole and the surrounding pNets. In open automata, this is reflected by symbolic hypotheses that depend on the actions of the holes. This additional specification is the price to pay to obtain the compositionality of bisimilarity that cannot be guaranteed in traditional process algebras.

This approach also allows us to specify a sufficient condition on allowed transitions to make weak bisimilarity compositional; namely it is not possible to synchronise on invisible actions from the holes or prevent them to occur. This

is loosely related to works on the syntactic conditions on SOS rules to check whether weak bisimulation is a congruence for some process algebra operators [24]. Our approach is semantical and more global: our sufficient condition applies to all the synchronisations at a given composition level of an (open) system and not on individual rules. It is expressed on the open automaton (see Definition 15).

Structure

This article is organised as follows. Section 2 provides the definition of pNets and introduces the notations used in this paper, including the definition of open pNets. Section 3 defines open automata, i.e. automata with parameters and transitions conditioned by the behaviour of "holes"; a strong bisimulation equivalence for open automata is also presented in this section. Section 4 gives the semantics of open pNets expressed as open automata, and states compositionality properties of strong bisimularity for open pNets. Section 5 defines a weak bisimulation equivalence on open automata and derives weak bisimilarity for pNets, together with compositionality properties of weak bisimilarity. Finally, Section 6 discusses related works and Section 7 concludes the paper.

2. Background and Notations

This section introduces the notations we will use in this article, and recalls the definition of pNets [30] with an informal semantics of the pNet constructs. The only significant difference compared to our previous definitions (from [30]) is that we remove here the restriction that was stating that variables should be local to a state of a labelled transition system.

2.1. Notations

Term algebra. Our models rely on a notion of parameterised actions, which are symbolic expressions using data types and variables. As our model aims at encoding the low-level behaviour of possibly very different programming languages, we do not want to impose one specific algebra for denoting actions, nor any specific communication mechanism. So we leave the constructors of the algebra that will be used to build expressions and actions unspecified. Moreover, we use a generic *action interaction* mechanism, based on (some sort of) unification between two or more action expressions, to express various kinds of communication or synchronisation mechanisms.

Formally, we assume the existence of a term algebra $\mathbb{T}$, and denote as Σ the signature of the data and action constructors. Within $\mathbb{T}$, we distinguish a set of data expressions $\mathbb{E}$, including a set of boolean expressions $\mathbb{B}$ ($\mathbb{B} \subseteq \mathbb{E}$), and a set of action expressions called the action algebra $\mathbb{A}$, with $\mathbb{A} \subseteq \mathbb{T}, \mathbb{E} \cap \mathbb{A} = \emptyset$; naturally action terms will use data expressions as sub-terms[1]. The function $vars(t)$ identifies the set of variables in a term $t \in \mathbb{T}$.

[1] In our tools, we use datatypes for the different kinds of terms. In this article, we use different sets of variables for terms of different kinds.

We let e_i range over expressions ($e_i \in \mathbb{E}$), a range over action labels, op be operators, and x_i and y_i range over variable names. We additionally rely on a set of action names, ranged over by a, b, ...

We define two kinds of parameterised actions. The first kind supports two kinds of parameters: input parameters that are variables and output parameters that can be any expression. The second kind makes no distinction between input and output parameters. The actions that distinguish input variables will be used in the definition of *pLTS* below and are defined as follows:

$\alpha \in \mathbb{A}$	$::=$	$a(p_1, \ldots, p_n)$	action terms
p_i	$::=$	$?x \mid e_i$	parameters (input var or expression)
e_i	$::=$	$\mathit{Value} \mid x \mid op(e_1, .., e_n)$	Expressions

The *input variables* in an action term are those marked with a *?*. We additionally impose that each input variable does not appear anywhere else in the same action term: $p_i = ?x \Rightarrow \forall j \neq i.\, x \notin \mathit{vars}(p_j)$. We define $iv(t)$ as the set of input variables of a term t (without the '?' marker). Input variables are used in guards and to update the local state, they can only appear in well-identified expressions. Action algebras can encode naturally usual point-to-point message passing calculi (using $a(?x_1, ..., ?x_n)$ for inputs, $a(v_1, .., v_n)$ for outputs), but they also allow for more general synchronisation mechanisms, like gate negotiation in Lotos, or broadcast communications.

The set of actions that do not distinguish input variables is denoted $\mathbb{A}_S$, it will be used in synchronisation vectors of pNets:

$$\alpha \in \mathbb{A}_S \quad ::= \quad a(e_1, \ldots, e_n)$$

Indexed sets. This article extensively uses indexed structures (maps) over some countable indexed sets. The indices can typically be integers, bounded or not. We use indexed sets in pNets because we want to consider a set of processes, and specify separately how to synchronise them. Roughly this could also be realised using tuples, however indexed sets are more general, can be infinite, and give a more compact representation than using the position in a possibly long tuple.

An indexed family is denoted as follows: $t_i^{i \in I}$ is a family of elements t_i indexed over the set I. Such a family is equivalent to the mapping $(i \mapsto t_i)^{i \in I}$, and we will also use mapping notations to manipulate indexed sets. To specify the set over which the structure is indexed, indexed structures are always denoted with an exponent of the form $i \in I$.

Consequently, $t_i^{i \in I}$ defines first I the set over which the family is indexed, and then t_i the elements of the family. For example $t_i^{i \in \{3\}}$ is the mapping with a single entry t_3 at index 3; exceptionally, for mappings with only a few entries we use the notation $(3 \mapsto t_3)$ instead. In this article, sentences of the form "there exists $t_i^{i \in I}$" means there exist I and a function that maps each element of I to a term t_i.

When this is not ambiguous, we shall abuse notations for sets, and typically write "indexed set over I" when formally we should speak of multisets, and

"$x \in A_i^{i\in I}$" to mean $\exists i \in I.\, x = A_i$. To simplify equations, an indexed set can be denoted $\bar{t}$ instead of $t_i^{i\in I}$ when I is irrelevant or clear from the context.

The disjoint union on sets is $\uplus$ and we only use $A \uplus B$ when A and B are disjoint. We extend it to union of indexed sets provided they are indexed over disjoint families; is $\uplus$ is then defined by the merge of the two sets. The elements of the union of two indexed sets are then accessed by using an index of one of the two joined families. The subtraction operation on indexed sets is $\setminus$, it reduces the set of indexes such that $\mathrm{dom}(A \setminus B) = \mathrm{dom}(A) \setminus B$.

Substitutions. This article also uses substitutions. Applying a substitution inside a term t is denoted $t\{\!\{y_i \leftarrow e_i\}\!\}^{i\in I}$ and consists in replacing in parallel all the occurrences of variables y_i in the term t by the terms e_i. Note that a substitution is defined by a partial function that is applied on the variables inside a term. We let *Post* range over partial functions that are used as substitution and use the notation $\{y_i \leftarrow e_i\}^{i\in I}$ to define such a partial function[2]. These partial functions are sometimes called *substitution functions* in the following. Thus, $\{\!\{Post\}\!\}$ is the operation that applies, in a parallel manner, the substitution defined by the partial function *Post*. $\odot$ is a composition operator on these partial functions, such that for any term t we have: $t\{\!\{Post \odot Post'\}\!\} = (t\{\!\{Post'\}\!\})\{\!\{Post\}\!\}$. This property must also be valid when the substitution does not operate on all variables. We thus define a composition operation as follows:

$$\begin{aligned}(x_k \leftarrow e_k)^{k\in K} \odot (x'_{k'} \leftarrow e'_{k'})^{k'\in K'} =& (x_k \leftarrow e_k\{\!\{(x'_{k'} \leftarrow e'_{k'})^{k'\in K'}\}\!\})^{k\in K} \\ & \cup (x'_{k'} \leftarrow e'_{k'})^{k'\in K''}\end{aligned}$$

where $K'' = \{k' \in K' | x'_{k'} \notin \{x_k\}^{k\in K}\}$

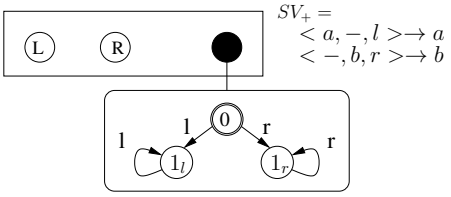
2.2. The principles of Parameterised Networks (pNets)

pNets are tree-like structures, where the leaves are either *parameterised labelled transition systems (pLTSs)*, expressing the behaviour of basic processes, or *holes*, used as placeholders for unknown processes. Every node of the tree is a pNet, it acts as a synchronising artefacts, using a set of *synchronisation vectors* that express the possible synchronisation between the parameterised actions of a subset of the sub-trees. The pNets model is hierarchical in the structure of the processes, in contrast to the Statecharts formalism [25], which is widely used to model high-level behaviour, that organises the states (but not processes) in a hierarchy.

We introduce the notion of pNets through a simple example below, and define formally pLTSs and pNets afterwards:

Example 1 (CCS choice). Here is the encoding of a choice operator.

[2] When using this notation, we suppose, without loss of generality that each y_i is different.



It consists of one pNet (Definition 2 below) with two holes and a sub-net. The pNet is represented by the top box with three circles and two synchronisation vectors on the right. The sub-net is a pLTS that is represented by the bottom box. Each hole is represented by an empty disc, when the hole is filled it becomes a black disc. The left hole is indexed L the right hole R. The sub-net is an labelled transition system (LTS) with three states and emitting actions l and r.

The behaviour of the pNet is defined with synchronisation vectors also shown on the figure. In the examples, we write them on the form $< a, -, l > \rightarrow a$. This states that if the first hole L performs the action a and the third sub-net, i.e. the LTS, performs the action l, both of them progress synchronously, and an action a is emitted by the pNet. The symbol $-$ at the second position denotes that the second hole does nothing. On the formal side, numbering and ordering the vectors is cumbersome, this is why we adopt indexed families of actions. The LTS is sometimes called the "control part", it controls the evolution of the rest of the pNet. The first action of one of the holes decides which branch of the LTS is activated; all subsequent actions will be performed by the same side.

2.3. Parameterised Labelled Transition systems (pLTS)

A pLTS is a labelled transition system with variables; variables can be used inside states, actions, guards, and assignments. Note that we make no assumption on finiteness of the set of states nor on finite branching of the transition relation. Compared to our previous works [30, 2] make variables global, which makes the model easier to use.

Definition 1 (pLTS). A pLTS is a tuple $pLTS \triangleq \langle\!\langle S, s_0, V, \rightarrow \rangle\!\rangle$ where:

- S is a set of states.
- $s_0 \in S$ is the initial state.
- V is a set of global variables for the pLTS.
- $\rightarrow \subseteq S \times L \times S$ is the transition relation and L is the set of labels. Labels have the form:
  $\langle \alpha,\ e_b,\ (x_j := e_j)^{j \in J} \rangle$, where $\alpha \in \mathbb{A}$ is a parameterised action, $e_b \in \mathbb{B}$ is a guard, and the variables x_j (that are pairwise distinct) are assigned the expressions $e_j \in \mathbb{E}$. If $s \xrightarrow{\langle \alpha,\ e_b,\ (x_j := e_j)^{j \in J} \rangle} s' \in \rightarrow$ then $vars(\alpha) \backslash iv(\alpha) \subseteq V$, $vars(e_b) \subseteq V \cup vars(\alpha)$, and $\forall j \in J.\ (vars(e_j) \subseteq V \cup iv(\alpha) \wedge x_j \in V)$.

A set of assignments between two states is performed in parallel so that their order do not matter and they all use the values of variables before the transition or the values received as action parameters.

2.4. Parameterised Networks (pNets)

Now we define pNet nodes as constructors for hierarchical behavioural structures. A pNet has a set of sub-pNets that can be either pNets or pLTSs, and a set of holes, playing the role of process parameters. A pNet is thus a composition operator that can receive processes as parameters; it expresses how the actions of the sub-processes synchronise.

Each sub-pNet exposes a set of actions, called *internal actions*. The synchronisation between global actions exposed by the pNet and internal actions of its sub-pNets is given by *synchronisation vectors*: a synchronisation vector synchronises one or several internal actions, and exposes a single resulting global action.

We now define the structure of pNets, the following definition relies on the definition of holes, leaves and sorts formalised below in Definition 3. Informally, holes are process parameters, leaves provide the set of pLTSs at the leaves of the hierarchical structure of a pNet, and sorts give the signature of a pNet, i.e. the actions it exposes.

Definition 2 (pNets). A pNet P is a hierarchical structure where leaves are pLTSs and holes

$$P \triangleq pLTS \mid \langle\!\langle P_i^{i\in I}, Sort_j^{j\in J}, SV_k^{k\in K} \rangle\!\rangle$$

We denote $vars(P)$ the set of variables used by the pLTSs inside P and $Sort(P)$ the signature of the actions emitted by P; both are defined below, in Definition 3. A pNet is composed of the following:

- I is a set of indices and $P_i^{i\in I}$ is the family of sub-pNets indexed over I. $vars(P_i)$ and $vars(P_j)$ must be disjoint for $i \neq j$.
- J is a set of indices, called *holes*. I and J are *disjoint*: $I \cap J = \emptyset$, $I \cup J \neq \emptyset$.
- $Sort_j \subseteq \mathbb{A}_S$ is a set of action terms, denoting the *sort* of hole j.
- $SV_k^{k\in K}$ is a set of synchronisation vectors.
  $\forall k \in K.\, SV_k = \alpha_l^{l\in I_k \uplus J_k} \rightarrow \alpha'_k[e_k]$ where $\alpha'_k \in \mathbb{A}_S$, $I_k \subseteq I$, $J_k \subseteq J$, $\forall i \in I_k.\, \alpha_i \in Sort(P_i)$, $\forall j \in J_k.\, \alpha_j \in Sort_j$, and $vars(\alpha'_k) \subseteq \bigcup_{l\in I_k \uplus J_k} vars(\alpha_l)$. The global action of a vector SV_k is α'_k. $e_k \in \mathbb{B}$ is a guard associated to the vector such that $vars(e_k) \subseteq \bigcup_{l\in I_k \uplus J_k} vars(\alpha_l)$.

Synchronisation vectors are identified modulo renaming of variables that appear in their action terms, e.g. the vectors $< a(x), b(x) > \rightarrow \tau$ and $< a(y), b(y) > \rightarrow \tau$ are equivalent.

The preceding definition relies on the auxiliary functions defined below:

Definition 3 (Sorts, holes, leaves, variables of pNets).

- The sort of a pNet is its signature, i.e. the set of actions in $\mathbb{A}_S$ it can perform, where each action signature is an action label plus the arity of the action.

$$\begin{array}{l}\mathrm{Sort}(\langle\!\langle S, s_0, V, \rightarrow \rangle\!\rangle) = \{\mathrm{Sort}(\alpha) | s \xrightarrow{\langle \alpha,\ e_b,\ (x_j := e_j)^{j\in J}\rangle} s' \in \rightarrow\} \\ \mathrm{Sort}(\langle\!\langle \overline{P}, \overline{Sort}, \overline{SV} \rangle\!\rangle) = \{\mathrm{Sort}(\alpha') |\, \overline{\alpha} \rightarrow \alpha'[e_b] \in \overline{SV}\} \\ \mathrm{Sort}(\alpha(p_1, .., p_n)) = (\alpha, n)\end{array}$$

- The set of variables of a pNet P, denoted $vars(P)$ is disjoint union the set of variables of all pLTSs that compose P.

- The set of holes $\mathrm{Holes}(P)$ of a pNet is the set of indices of the holes of the pNet itself plus the indices of all the holes of its sub-pNets. It is defined inductively (we suppose that those index sets are disjoints):

$$\begin{array}{l}\mathrm{Holes}(\langle\!\langle S, s_0, V, \rightarrow \rangle\!\rangle) = \emptyset \\ \mathrm{Holes}(\langle\!\langle P_i^{i\in I}, Sort_j^{j\in J}, \overline{SV} \rangle\!\rangle) = J \uplus \bigcup_{i\in I} \mathrm{Holes}(P_i) \\ \forall i \in I.\ \mathrm{Holes}(P_i) \cap J = \emptyset \\ \forall i_1, i_2 \in I.\, i_1 \neq i_2 \Rightarrow \mathrm{Holes}(P_{i_1}) \cap \mathrm{Holes}(P_{i_2}) = \emptyset\end{array}$$

- The set of leaves of a pNet is the set of all pLTSs occurring in the structure, as an indexed family of the form $\mathrm{Leaves}(P) = \langle\!\langle P_i \rangle\!\rangle^{i\in L}$.

$$\begin{array}{l}\mathrm{Leaves}(\langle\!\langle S, s_0, V, \rightarrow \rangle\!\rangle) = \emptyset \\ \mathrm{Leaves}(\langle\!\langle P_i^{i\in I}, \overline{Sort}, \overline{SV} \rangle\!\rangle) = \biguplus_{i\in I} \mathrm{Leaves}(P_i) \uplus \{i \mapsto P_i | P_i \text{ is a } pLTS\}\end{array}$$

For example, the controller of Example 1 has the sort $\{l, r\}$ and holes $\{L, R\}$. Note that $\mathrm{Holes}(P)$is a set of indexes because holes are characterized only by their indices, while entities at the leaves ar pLTSs and thus $\mathrm{Leaves}(P)$ is a set of pLTSs. A pNet Q is *closed* if it has no hole: $\mathrm{Holes}(Q) = \emptyset$; else it is said to be *open*. Sort comes naturally with a compatibility relation that is similar to a type-compatibility check. We simply say that two sorts are compatible if they consist of the same actions with the same arity. In practice, it is sufficient to check the equality of the two sets of action signatures of the two sorts[3].

The informal semantics of pNets is as follows. pLTSs behave more or less like classical automata with conditional branching and variables. The actions on the *pLTS*s can send or receive values, potentially modifying the value of variables. pNets are synchronisation entities: a pNet node composes several sub-pNets and defines how the sub-pNets interact, where a sub-pNet is either

[3]A more complex compatibility relation could be defined, but this is out of the scope of this article.

a pNet or a pLTS. The synchronisation between sub-pNets is defined by synchronisation vectors (originally introduced in [4]) that express how an action of a sub-pNet can be synchronised with actions of other sub-pNet, and how the resulting synchronised action is visible from outside of the pNet. The synchronisation mechanism is very expressive, including pattern-matching/unification between the parameterized actions within the vector, and an additional predicate over their variables. Consider a pNet node that assembles several pLTSs, the synchronisation vectors specify the way that transitions of the composed pNet are built from the transitions of the sub-nets. This can be seen as "conditional transitions" of the pNet, or alternatively, as a syntax to encode structural operational semantics (SOS rules) [44] of the system: each vector expresses not only the actions emitted by the pNet but also what transitions of the composed pLTSs must occur to trigger this global transition. Synchronisation vectors can also express the exportation of an action of a sub-pNet to the next level, or to hide an interaction and make it non-observable. Finally, a pNet can leave sub-pNets undefined and instead declare holes with a well-defined signature. Holes can then be filled with a sub-pNet. This is defined as follows.

Definition 4 (pNet composition). An open pNet: $P = \langle\!\langle P_i^{i\in I}, Sort_j^{j\in J}, \overline{SV} \rangle\!\rangle$ can be (partially) filled by providing a pNet Q to fill one of its holes. Suppose $j_0 \in J$ and $\mathrm{Sort}(Q) \subseteq Sort_{j_0}$, then:

$$P\,[Q]_{j_0} = \langle\!\langle P_i^{i\in I} \uplus \{j_0 \mapsto Q\}, Sort_j^{j\in J\setminus\{j_0\}}, \overline{SV} \rangle\!\rangle$$

pNets are composition entities equipped with a rich synchronisation mechanism: synchronisation vectors allow the expression of synchronisation between any number of entities and at the same time the passing of data between processes. Their strongest feature is that the data emitted by processes can be used inside the synchronisation vector to do addressing: it is easy to synchronise a process indexed by n with the action $a(v, n)$ of another process. This is very convenient to model systems and encode futures or message routing.

pNets have been used to model distributed components using the Grid Component Model, illustrating the expressiveness of the model [2]. These works show that pNets are convenient to express the behaviour of a system in a compositional way. Unfortunately, the semantics of pNets and the existing tools at that point were only able to deal with a closed and completely instantiated system: pNets could be used as composition operators in the definition of the semantics, which was sufficient to perform finite-state model checking on a closed system, but there was no theory for the use of pNets as operators and no tool for proving properties on open system. Consequently, much of the formalisation efforts did not use holes and the interplay between holes, sorts, and synchronisation vector was not formalised. In previous works [2], only closed pNets were equipped with a semantics, which was defined as labelled transition systems. The theory of pNets as operators for open systems is given in the following sections. Comparing formally the existing direct operational semantics and the semantics derived from open automata in the current article would be an interesting partial proof

of soundness for our semantics. The proof could only be partial as the formal semantics that exists only consider closed and fully instantiated pNets. Proving an equivalence between the semantics presented in this article and the operational one shown in [2] is outside the scope of this article because we focus here on the modelling of holes that were not considered in the previous semantics. It is however easy to see that, in case there is no hole the structure of the open automaton that defines the semantics here is very close to the pLTS that is used to define the semantics, even though the formalisms are slightly different.

2.5. Running example

To illustrate this work, we use a simple communication protocol, that provides safe transport of data between two processes, over unsafe media.

Figure 1 (left) shows the example principle, which corresponds to the hierarchical structure of a pNet: two unspecified processes P and Q (holes) communicate messages, with a data value argument, through the two protocol entities. Process P sends a `p-send(m)` message to the *Sender*; this communication is denoted as `in(m)`. At the other end, process Q receives the message from the *Receiver*. The holes P and Q can also have other interactions with their environment, represented here by actions `p-a` and `q-b`. The underlying network is modelled by a medium entity transporting messages from the sender to the receiver, and that is able to detect transport errors and signal them to the sender. The return *ack* message from *Receiver* to *Sender* is supposed to be safe. The final transmission of the message to the recipient (the hole Q) includes the value of the "error counter" *ec*.

Figure 1 (right) shows a graphical view of the pNet *SimpleProtocolSpec* that specifies the system. The pNet is made of the composition of two pNets: a *SimpleSystem* node, and a *PerfectBuffer* sub-pNet. The full system implementation should be equivalent (e.g. weakly bisimilar) to this *SimpleProtocolSpec*. The pNet has a tree-like structure. The root node of the tree *SimpleSystem* is the top level of the pNet structure. It acts as the parallel operator. It consists of three nodes: two holes P and Q and one sub-pNet, denoted *PerfectBuffer*. Nodes of the tree are synchronised using four synchronisation vectors, that express the possible synchronisations between the parameterised actions of a subset of the nodes. For instance, in the vector $<$ `p-send(m)`, `in(m)`, _ $>\!\!\rightarrow$ `in(m)` only P and *PerfectBuffer* nodes are involved in the synchronisation. The synchronisation between these processes occurs when process P performs `p-send(m)` action sending a message, and the *PerfectBuffer* accepts the message through an `in(m)` action at the same time; the result that will be returned at upper level is the action `in(m)`.

Figure 2 shows the pNet model of the protocol implementation, called *SimpleProtocolImpl*. When the *Medium* detects an error (modelled by a local τ action), it sends back a `m-error` message to the *Sender*. The *Sender* increments its local error counter *ec*, and resends the message (including *ec*) to the *Medium*, that will, eventually, transmit m, ec to the *Receiver*.

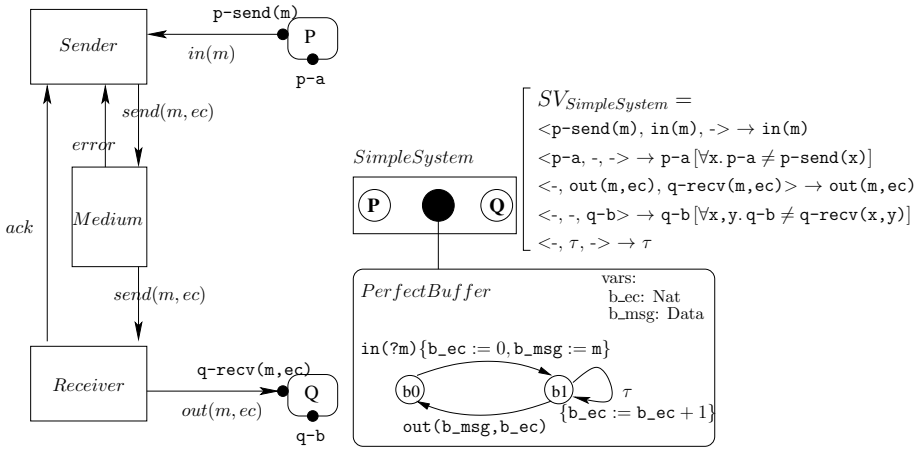


Figure 1: pNet structure of the example and its specification expressed as a pNet called *SimpleProtocolSpec*

3. A model of process composition

The semantics of open pNets will be defined as an open automaton. An open automaton is an automaton where each transition composes transitions of several LTSs with action of some holes, the transition occurs if some predicates hold, and can involve a set of state modifications. This section defines open automata and a bisimulation theory for them. This section is an improved version of the formalism described in [30], extending the automata with a notion of global variable, which makes the state of the automaton more explicit. We also adopt a semantics and logical interpretation of the automata that intuitively can be stated as follows: "if a transition belongs to an open automaton, any refinement of this transition also belongs to the automaton". Our open automata are clearly inspired by the work of De Simone on formatting of SOS rules [16]. A precise comparison with related works can be found in Section 6.

3.1. Open automata

Open automata (OA) are not composition structures but they are made of transitions that are dependent of the actions of the holes, and they can use variables (potentially with only symbolic values).

Definition 5 (Open transitions). An *open transition* (OT) over a set J of holes with sorts $Sort_j^{j\in J}$, a set V of variables, and a set of states $\mathcal{S}$ is a structure of the form:

$$\frac{\beta_j^{j\in J'}, Pred, Post}{s \xrightarrow{\alpha} s'}$$

where $J' \subseteq J$ is the set of holes involved in the transition; $s, s' \in \mathcal{S}$ are states of the automaton; and β_j is a transition of the hole j, with $\mathrm{Sort}(\beta_j) \subseteq Sort_j$. α is

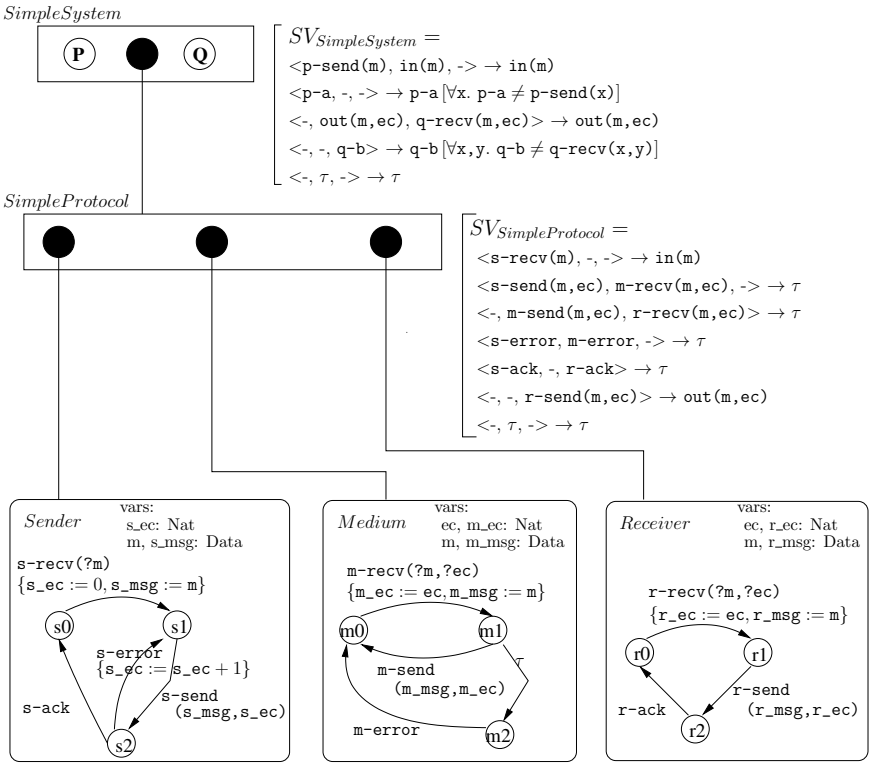


Figure 2: The *SimpleProtocolImpl* pNet resulting from the composition of the *SimpleSystem* and the *SimpleProtocol* pNets.

the resulting action of this open transition. *Pred* is a predicate, *Post* is a set of assignments that are effective after the open transition, and are represented as a substitution function: $(x_k \leftarrow e_k)^{k \in K}$. Predicates and expressions of an open transition can refer to the variables inside V and the different terms β_j and α. More precisely:

$$vars(Pred) \subseteq V \cup vars(\alpha) \cup \bigcup_{j \in J'} vars(\beta_j) \quad \wedge$$
$$\forall k.\, x_k \in V \quad \wedge \quad \forall k.\, vars(e_k) \subseteq V \cup vars(\alpha) \cup \bigcup_{j \in J'} vars(\beta_j)$$

The assignments are applied simultaneously because the variables in V can be in both sides (x_ks are distinct). Open transitions are identified modulo logical equivalence on their predicate.

It is important to understand the difference between the red dotted rule and a classical inference rule. They correspond to two different logical levels. On one side, classical (black) inference rules act at the mathematical level of the paper

proofs (as e.g. the rules in Definition 13). They use an expressive logic (like any other computer science article). On the other side, open transition rules (with dotted lines) are logical implications that belong to the open automata algebra. Their logic has a specific syntax that can be mechanized; this logic includes the boolean expressions $\mathbb{B}$, boolean operators, and term equality.

An open automaton is an automaton where transitions are open transitions.

Definition 6 (Open automaton). An *open automaton* is a structure $A = \langle\!\langle J, \mathcal{S}, s_0, V, \mathcal{T} \rangle\!\rangle$ where:

- J is a set of indices.
- $\mathcal{S}$ is a set of states and s_0 is an initial state belonging to $\mathcal{S}$.
- V is a set of variables of the automaton and each $v \in V$ may have an initial value $init(v)$.
- $\mathcal{T}$ is a set of open transitions and for each $t \in \mathcal{T}$ there exists J' with $J' \subseteq J$, such that t is an open transition over J' and $\mathcal{S}$.

While the definition and usage of the open transition can be considered purely syntactically, we take in this article a semantics and logical understanding of open automata. We see open transitions as logical formulas with a constrained syntax and logics rather than purely syntactical terms. Consequently, the open transition sets in open automata are closed by a simple form of refinement that allows us to refine the predicate, or substitute any free variable by an expression. Formally, for each predicate $Pred$ for each partial function $Post$, if $V \cap \mathrm{dom}(Post) = \emptyset$, we have:

$$\frac{\overline{\beta}, Pred', Post'}{s \xrightarrow{\alpha} s'} \in \mathcal{T} \quad \Longrightarrow \quad \frac{\overline{\beta}\{\!\{Post\}\!\}, Pred'\{\!\{Post\}\!\} \wedge Pred, Post \odot Post'}{s \xrightarrow{\alpha\{\!\{Post\}\!\}} s'} \in \mathcal{T}$$

Because of the semantic interpretation of open automata, the set of open transition of an open automaton is infinite (for example because every free variable can be substituted by any term). This raises an issue when a finite representation is needed, which is the case both in our tools, and when writing examples. When needed, we can rely on a *canonical representation* of the open automaton, provided that a finite subset of the open transitions is sufficient to generate, by substitution, the other ones. Thus, we use this canonical representation in our examples. In the following, we will abusively write that we define an "open automaton" when we provide its canonical representation.

Another aspect of the semantic interpretation is that we consider terms up to semantic equivalence, i.e. equivalence of two predicates $Pred$ and $Pred'$ can be denoted $Pred = Pred'$, where the $=$ symbol is interpreted semantically.

Though the definition is simple, the fact that transitions are complex structures relating events must not be underestimated. The first element of theory for open automata, i.e. the definition of a strong bisimulation, is given below.

3.2. Bisimulation for open Automata

We define now a bisimulation relation tailored to open automata and their parametric nature. This relation relates states of the open automata and guarantees that the related states are observationally equivalent, i.e. equivalent states can trigger transitions with identical action labels. Its key characteristics are 1) the introduction of predicates in the bisimulation relation: the relation between states may depend on the value of the variables; 2) bisimulation relates elements of the open transitions and takes into account predicates over variables, actions of the holes, and state modifications. We name it FH-bisimulation, as a short cut for the "Formal Hypotheses" over the holes behaviour manipulated in the transitions, but also as a reference to the work of De Simone [16], that pioneered this idea. Indeed, our definition uses both hypotheses on the behaviour of holes, as in [16], and symbolic manipulation of action expressions, as in symbolic bisimulations of [27].

One of the original aspects of FH-bisimulation is due to the symbolic nature of open automata. Indeed, a single state of the automaton represents a potentially infinite number of concrete states, depending on the value of the automaton variables, and a single open transition of the automaton may also be instantiated with an unbounded number of values for the transition parameters. Consequently it would be too restrictive to impose that each transition of one automaton is matched by exactly one transition of the bisimilar automaton. Thus the definition of bisimulation requires that, for each open transition of one automaton, there exists a matching set of open transitions covering the original one. Indeed depending on the value of action parameters or automaton variables, different open transitions might simulate the same one.

The parametric nature of the automata entails a second original aspect of FH-bisimulation: the nature of the bisimulation relation itself. A classical relation between states can be seen as a function mapping pairs of state to a boolean value (true if the states are related, false if they are not). An FH-bisimulation relation maps pairs of states to boolean expressions that use variables of the two systems. Formally, a relation over the states of two open automata $\langle\!\langle J, \mathcal{S}_1, s_0, V_1, \mathcal{T}_1 \rangle\!\rangle$ and $\langle\!\langle J, \mathcal{S}_2, t_0, V_2, \mathcal{T}_2 \rangle\!\rangle$ has the signature $\mathcal{S}_1 \times \mathcal{S}_2 \to \mathbb{B}$. We suppose without loss of generality that the variables of the two open automata are disjoint. We adopt a notation similar to standard relations and denote it $\mathcal{R} = \{(s, t | \mathit{Pred}_{s,t})\}$, where: 1) For any pair $(s, t) \in \mathcal{S}_1 \times \mathcal{S}_2$, there is a single $(s, t | \mathit{Pred}_{s,t}) \in \mathcal{R}$ stating that s and t are related if $\mathit{Pred}_{s,t}$ is True, i.e. the states are related when the value of the automata variables satisfy the predicate $\mathit{Pred}_{s,t}$. 2) The free variables of $\mathit{Pred}_{s,t}$ belong to V_1 and V_2, i.e. $\mathit{vars}(\mathit{Pred}_{s,t}) \subseteq V_1 \cup V_2$. FH-bisimulation is defined formally[4]:

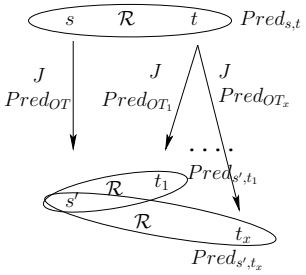
Definition 7 (Strong FH-bisimulation).
Suppose $A_1 = \langle\!\langle J, \mathcal{S}_1, s_0, V_1, \mathcal{T}_1 \rangle\!\rangle$ and $A_2 = \langle\!\langle J, \mathcal{S}_2, t_0, V_2, \mathcal{T}_2 \rangle\!\rangle$ are open automata with identical holes of the same sort, with disjoint sets of variables ($V_1 \cap V_2 = \emptyset$).

[4]In this article, we denote β_{jx} a double indexed set, instead of the classical $\beta_{j,\,x}$. Indeed the standard notation would be too heavy in our case.

Then $\mathcal{R}$ is an FH-bisimulation if and only if for all states $s \in \mathcal{S}_1$ and $t \in \mathcal{S}_2$, $(s, t|Pred_{s,t}) \in \mathcal{R}$, we have the following:

- For any open transition OT in $\mathcal{T}_1$:

$$\frac{\beta_j^{j \in J'}, Pred_{OT}, Post_{OT}}{s \xrightarrow{\alpha} s'}$$



  there exists an indexed set of open transitions $OT_x^{x \in X} \subseteq \mathcal{T}_2$:

$$\frac{\beta_{jx}^{j \in J_x}, Pred_{OT_x}, Post_{OT_x}}{t \xrightarrow{\alpha_x} t_x}$$

  such that $\forall x.\, J' = J_x$ and there exists some $Pred_{s',t_x}$ such that $(s', t_x|Pred_{s',t_x}) \in \mathcal{R}$ and

$$Pred_{s,t} \wedge Pred_{OT} \implies \bigvee_{x \in X} (\forall j.\beta_j = \beta_{jx} \wedge Pred_{OT_x} \wedge \alpha = \alpha_x \wedge Pred_{s',t_x} \{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\})$$

- and symmetrically any open transition from t in $\mathcal{T}_2$ can be covered by a set of transitions from s in $\mathcal{T}_1$.

Two open automata are FH-bisimilar if there exists an FH-bisimulation that relates their initial states[5]. We call this relation *FH-bisimilarity*.

Classically, $Pred_{s',t_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\}$ applies in parallel the substitution defined by the partial functions $Post_{OT}$ and $Post_{OT_x}$ (parallelism is crucial inside each $Post$ set but not between $Post_{OT}$ and $Post_{OT_x}$ that are independent), applying the assignments of the involved rules. We can prove that bisimilarity is an equivalence relation.

Example 2. The simulation of one transition by many others is one non-standard aspect of this definition. This is made necessary by the parameterised nature of our model. Consider the following open transition.

$$\frac{\overline{\beta}, True, \{\!\{y \leftarrow x\}\!\}}{s_1 \xrightarrow{\alpha(x)} s_1'}$$

Bisimulation should allow it to be matched by the two following ones (depending on the value of x), to prove that the relation $\mathcal{R} = \{(s_1, s_2, True), (s_1', s_2', True)\}$ is a bisimulation.

$$\frac{\overline{\beta}, x \geq 0, \{\!\{y \leftarrow x\}\!\}}{s_2 \xrightarrow{\alpha(x)} s_2'} \qquad \frac{\overline{\beta}, x < 0, \{\!\{y \leftarrow x\}\!\}}{s_2 \xrightarrow{\alpha(x)} s_2'}$$

[5] In other words, the predicate relation associated to the initial states is *True*.

This example illustrates the necessity of multiple transitions in the definition of bisimulation in a naive and minimalistic way. It can easily be extended into a non-trivial example with more states and different usage of the variables.

Theorem 1 (FH-bisimularity is an equivalence). *FH-bisimilarity is reflexive, symmetric and transitive.*

The proof of this theorem can be found in [3]. The only non-trivial part of the proof is the proof of transitivity. It relies on the following elements. First, the transitive composition of two relations with predicate is defined; this is not exactly standard as it requires to define the right predicate for the transitive composition and producing a single predicate to relate any two states. Then the fact that one open transition is simulated by a family of open transitions leads to a doubly indexed family of simulating open transition; this needs particular care, also because of the use of renaming (*Post*) when proving that the predicates satisfy the definition (property on $Pred_{s,t} \wedge Pred_{OT}$ in the definition).

Finite versus infinite open automata, and decidability

As mentioned in Definition 15, we adopt here a semantic view on open automata. More precisely, in [31], we define *semantic open automata* (infinite as in Definition 6), and *structural open automata* (finite) that can be generated as the semantics of pNets (see Definition 9), and used in their implementation. Then we define an alternative version of our bisimulation, called structural FH-bisimulation, based on structural open automata, and prove that the *semantic* and *structural* FH-bisimulations coincide. In the sequel, all mentions of finite automata, and algorithms for bisimulations, implicitly refer to their *structural* versions.

If we assume that everything is finite (states and transitions in the open automata), then it is easy to prove that it is decidable whether a relation is a FH-bisimulation, provided the logic of the predicates is decidable (a proof of this claim can be found in [30]). Formally:

Theorem 2 (Decidability of FH-bisimulation). *Let A_1 and A_2 be finite open automata and $\mathcal{R}$ a relation over their states $\mathcal{S}_1$ and $\mathcal{S}_2$ constrained by a set of predicates. Assume that the predicate inclusion is decidable over the action algebra $\mathbb{A}$. Then it is decidable whether the relation $\mathcal{R}$ is an FH-bisimulation.*

4. Semantics of Open pNets

This section defines the semantics of an open pNet via translation into an open automaton. In this translation, the states of the open automaton are obtained as products of the states of the pLTSs at the leaves of the composition. The predicates on the transitions are obtained both from the predicates on the transitions of the pLTSs, and from the synchronisation vectors involved in the transition.

The definition of bisimulation for open automata allows us to derive a bisimilarity relation for open pNets. As pNets are composition structures, it then makes sense to prove compositionality lemmas: we prove that the composition of strongly bisimilar pNets are themselves bisimilar.

4.1. Deriving an open automaton from an open pNet

To derive an open automaton from a pNet, we first describe the set of states of the automaton. Then we show the construction rule for transitions of the automaton, which relies on the derivation of predicates unifying synchronisation vectors and the actions of the pNets involved in a given synchronisation.

States of open pNets are tuples of states. We denote them as $\triangleleft \dots \triangleright$ for distinguishing tuple states from other tuples.

Definition 8 (States of open pNets). A state of an open pNet is a (not necessarily finite) tuple of the states of its leaves.

For any pNet P, let Leaves$(P) = \langle\!\langle S_i, s_{i0}, V, \rightarrow_i \rangle\!\rangle^{i \in L}$ be the set of pLTS at its leaves, then $States(P) = \{\triangleleft s_i^{i \in L} \triangleright \,|\forall i \in L. s_i \in S_i\}$. A pLTS being its own single leave: $States(\langle\!\langle S, s_0, V, \rightarrow \rangle\!\rangle) = \{\triangleleft s \triangleright \,|s \in S\}$.

The initial state is defined as: $InitState(P) = \triangleleft {s_{i0}}^{i \in L} \triangleright$.

To be precise, the state of each pLTS is entirely characterized by both the state of the automaton, and the values of its variables V.

Predicates. We define a predicate $Pred_{sv}$ relating a synchronisation vector (of the form $(\alpha'_i)^{i \in I}, (\beta'_j)^{j \in J} \rightarrow \alpha'[e_b]$), the actions of the involved sub-pNets and the resulting actions. This predicate verifies:

$$Pred_{sv}\Big(\big((\alpha'_i)^{i \in I}, (\beta'_j)^{j \in J} \rightarrow \alpha'[e_b]\big), \alpha_i^{i \in I}, \beta_j^{j \in J}, \alpha\Big) \Leftrightarrow$$
$$\forall i \in I.\, \alpha_i = \alpha'_i \wedge \forall j \in J.\, \beta_j = \beta'_j \wedge \alpha = \alpha' \wedge e_b$$

Somehow, this predicate entails a verification of satisfiability in the sense that if the predicate $Pred_{sv}$ is not satisfiable, then the transition associated with the synchronisation will not occur in the considered state, or equivalently will occur with a *False* precondition. If the action families do not match or if there is no valuation of variables such that the above formula can be ensured then the predicate is undefined.

The definition of this predicate is not constructive. In our tool [46], we construct a logical formula encoding the matching and unification condition involved, and we let an SMT engine (in the current implementation Z3 [35]) decide its satisfiability.

Example 3 (An open-transition). At the upper level, the *SimpleSystem* pNet of Figure 2 has 2 holes and *SimpleProtocol* as a sub-pNet, itself containing 3 pLTSs. One of its possible open transitions (synchronizing the hole P with the *Sender* within the *SimpleProtocol*) is:

$$\frac{s \xrightarrow{\langle \alpha,\ e_b,\ (x_j := e_j)^{j\in J}\rangle} s' \in \rightarrow}{\langle\!\langle S, s_0, \rightarrow \rangle\!\rangle \models \dfrac{\emptyset, e_b, \{x_j \leftarrow e_j\}^{j\in J}}{\lhd s \rhd \xrightarrow{\alpha} \lhd s' \rhd}} \quad \textbf{Tr1}$$

and

$$\frac{\begin{array}{c}\text{Leaves}(\langle\!\langle P_m^{m\in I}, \overline{Sort}, SV_k^{k\in K}\rangle\!\rangle) = pLTS_l^{l\in L} \quad k \in K \\ SV_k = (\alpha'_m)^{m\in I_1 \uplus I_2 \uplus J} \rightarrow \alpha'[e_b] \\ \forall m \in I_1.\ P_m \models \dfrac{\beta_j^{j\in J_m}, Pred_m, Post_m}{\lhd s_i^{i\in L_m} \rhd \xrightarrow{\alpha_m} \lhd (s'_i)^{i\in L_m} \rhd} \\ \forall m \in I_2.\ P_m \models \dfrac{\emptyset, Pred_m, Post_m}{\lhd s_m \rhd \xrightarrow{\alpha_m} \lhd s'_m \rhd} \qquad J' = \biguplus_{m\in I_1} J_m \uplus J \\ Pred = \bigwedge_{m\in I_1 \uplus I_2} Pred_m \wedge Pred_{sv}(SV_k, \alpha_m^{m\in I_1\uplus I_2}, \beta_j^{j\in J}, \alpha) \\ \forall i \in L \setminus \left(\biguplus_{m\in I_1} L_m \uplus I_2\right).\ s'_i = s_i \qquad \text{fresh}(\alpha'_m, \alpha', \beta_j^{j\in J}, \alpha)\end{array}}{\langle\!\langle P_m^{m\in I}, \overline{Sort}, SV_k^{k\in K}\rangle\!\rangle \models \dfrac{\beta_j^{j\in J'}, Pred, \biguplus_{m\in I_1\uplus I_2} Post_m}{\lhd s_i^{i\in L} \rhd \xrightarrow{\alpha} \lhd (s'_i)^{i\in L} \rhd}} \quad \textbf{Tr2}$$

Figure 3: Rules **Tr1** and **Tr2** defining the semantics of open pNets

$$OT_1 = \frac{\{\texttt{P} \mapsto \texttt{p-send(m)}\}, [\texttt{m=m'}], (\texttt{s\_msg} \leftarrow \texttt{m})}{\lhd s_0, m_0, r_0 \rhd \xrightarrow{\texttt{in(m')}} \lhd s_1, m_0, r_0 \rhd}$$

The global states here are triples, the product of states of the 3 pLTSs (holes have no state). The assignment performed by the open transition uses the variable `m` from the action of hole `P` to set the value of the sender variable named `s_msg`.

We build the semantics of open pNets as an open automaton over the states given by Definition 8. The open transitions first project the global state into states of the leaves, then apply pLTS transitions on these states, and compose them with the sort of the holes. The semantics instantiates fresh variables using the predicate $\text{fresh}(x)$, additionally, for an action α, $\text{fresh}(\alpha)$ means all variables in α are fresh.

Definition 9 (Semantics of open pNets). The semantics of a pNet P is an open automaton $A = \langle\!\langle \mathit{Holes}(P), \mathit{States}(P), \mathit{InitState}(P), \mathit{vars}(P), \mathcal{T} \rangle\!\rangle$ where $\mathcal{T}$ is the smallest set of open transitions such that $\mathcal{T} = \{OT \mid P \models OT\}$ and $P \models OT$ is defined by the rules in Figure 4.1

- The rule **Tr1** for a pLTS checks that the guard is verified and transforms assignments into post-conditions.
- The rule **Tr2** deals with pNet nodes: for each possible synchronisation vector (of index k) applicable to the rule subject, the premises include one *open transition* for each sub-pNet involved, one possible *action* for each hole involved, and the predicate relating these with the resulting action of the vector. The sub-pNets involved are split between two sets, I_2 for sub-pNets that are pLTSs (with open transitions obtained by rule **Tr1**), and I_1 for the sub-pNets that are not pLTSs (with open transitions obtained by rule **Tr2**), J is the set of holes involved in the transition[6][7].

A key to understand **Tr2** is that the open transitions are expressed in terms of the leaves and holes of the whole pNet structure, i.e. a flattened view of the pNet. For example, L is the index set of the Leaves, L_m the index set of the leaves of one sub-pNet indexed m, so all L_m are disjoint subsets of L. Thus the states in the open transitions, at each level, are tuples including states of all the leaves of the pNet, not only those involved in the chosen synchronisation vector.

Note that the construction is symbolic, and each open transition deduced expresses a whole family of behaviours, for any possible value of the variables.

In [30], we have shown a detailed example of the construction of a complex open transition, building a deduction tree using rules **Tr1** and **Tr2**. We have also shown in [30] that an open pNet with finite synchronisation sets, finitely many leaves and holes, and each pLTS at leaves having a finite number of states and (symbolic) transitions, induces a finite automaton. The algorithm for building such an automaton can be found in [45].

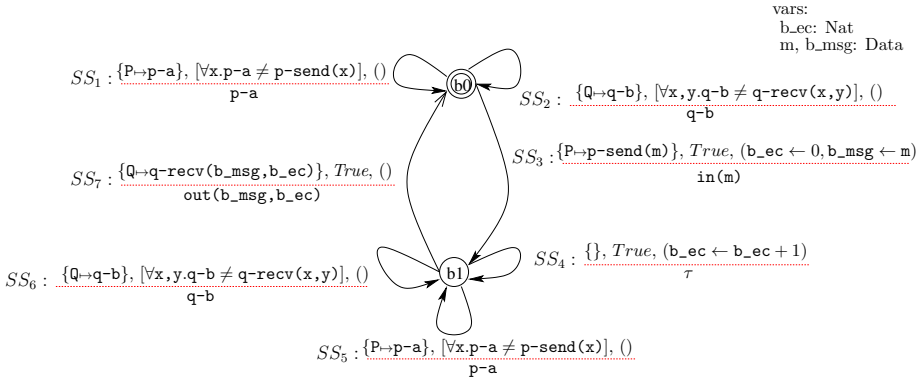


Figure 4: Open automaton for *SimpleProtocolSpec*

[6]Formally, if $SV_k = (\alpha')_m^{m \in M} \rightarrow \alpha'[e_b]$ is a synchronisation vector of P then $J = M \cap \text{Holes}(P)$, $I_2 = M \cap \text{Leaves}(P)$, $I_1 = M \setminus J \setminus I_2$

[7]We could replace I_1 and I_2 by their formal definition in **Tr2** but the rule would be more difficult to read.

Example

Figure 4 shows the open automaton computed from the *SimpleProtocolSpec* pNet given in Figure 1. For later references, we name SS_i the transitions of this (strong) specification automaton while transitions of the *SimpleProtocolImpl* pNet are labelled SI_i. In the figures we annotate each open automaton with the set of its variables.

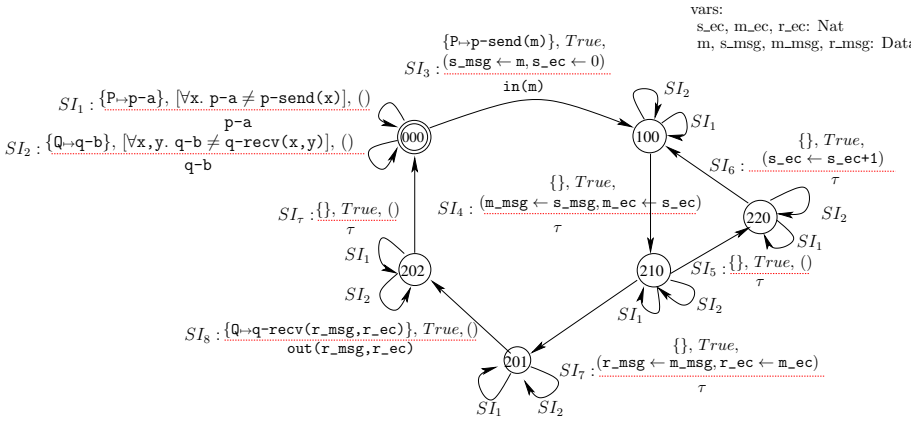


Figure 5: Open automaton for *SimpleProtocolImpl*

Figure 5 shows the open automaton of *SimpleProtocolImpl* from Figure 2. In this drawing, we have short labels for states, representing $\triangleleft s_0, m_0, r_0 \triangleright$ by `000`. Note that open transitions are denoted $\texttt{SI}_i$ and tau open transition by $\texttt{SI}_\tau$. The resulting behaviour is quite simple: we have a main loop including receiving a message from P and transmitting the same message to Q, with some intermediate τ actions from the internal communications between the protocol processes. In most of the transitions, you can observe that data is propagated between the successive pLTS variables (holding the message, and the error counter value). On the right of the figure, there is a loop of τ actions ($\texttt{SI}_4$, $\texttt{SI}_5$ and $\texttt{SI}_6$) showing the handling of errors and the incrementation of the error counter.

4.2. pNet Composition Properties: composition of open transitions

The semantics of open pNets allows us to prove two crucial properties relating pNet composition with pNet semantics: open transition of a composed pNet can be decomposed into open transitions of its composing sub-pNets, and conversely, from the open transitions of sub-pNets, an open transition of the composed pNet can be built.

We start with a decomposition property: from one open transition of $P[Q]_{j_0}$, we exhibit corresponding behaviours of P and Q, and determine the relation between their predicates.

Lemma 1 (Open transition decomposition). *Consider two pNets P and Q that are not pLTSs*[8]*. Let* $\text{Leaves}(Q) = p_l^{l\in L_Q}$ *and suppose:*

$$P[Q]_{j_0} \models \frac{\beta_j^{j\in J}, Pred, Post}{\lhd s_i^{i\in L} \rhd \xrightarrow{\alpha} \lhd s_i'^{i\in L} \rhd}$$

with $J \cap \text{Holes}(Q) \neq \emptyset$ *or* $\exists i \in L_Q.\, s_i \neq s_i'$*, i.e. Q takes part in the reduction. Then there exist α_Q, $Pred'$, $Pred''$, $Post'$, $Post''$ s.t.:*

$$P \models \frac{\beta_j^{j\in(J\setminus\text{Holes}(Q))\cup\{j_0\}}, Pred', Post'}{\lhd s_i^{i\in L\setminus L_Q} \rhd \xrightarrow{\alpha} \lhd s_i'^{i\in L\setminus L_Q} \rhd}$$

$$\text{and} \qquad Q \models \frac{\beta_j^{j\in J\cap\text{Holes}(Q)}, Pred'', Post''}{\lhd s_i^{i\in L_Q} \rhd \xrightarrow{\alpha_Q} \lhd s_i'^{i\in L_Q} \rhd}$$

and $Pred \iff Pred' \wedge Pred'' \wedge \alpha_Q = \beta_{j_0}$, $Post = Post' \uplus Post''$ *where $Post''$ is the restriction of $Post$ over variables of Q.*

Lemma 2 is combining an open transition of P with an open transition of Q, and building a corresponding transition of $P[Q]_{j_0}$ by assembling their elements.

Lemma 2 (Open transition composition). *Suppose* $j_0 \in J$ *and:*

$$P \models \frac{\beta_j^{j\in J}, Pred, Post}{\lhd s_i^{i\in L} \rhd \xrightarrow{\alpha} \lhd s_i'^{i\in L} \rhd} \qquad \text{and} \qquad Q \models \frac{\beta_j^{j\in J_Q}, Pred', Post'}{\lhd s_i^{i\in L_Q} \rhd \xrightarrow{\alpha_Q} \lhd s_i'^{i\in L_Q} \rhd}$$

Then, we have:

$$P[Q]_{j_0} \models \frac{\beta_j^{(j\in J\setminus\{j_0\})\uplus J_Q}, Pred \wedge Pred' \wedge \alpha_Q = \beta_{j_0}, Post \uplus Post'}{\lhd s_i^{i\in L\uplus L_Q} \rhd \xrightarrow{\alpha} \lhd s_i'^{i\in L\uplus L_Q} \rhd}$$

Note that this does not mean that any two pNets can be composed and produce an open transition. Indeed, the predicate $Pred \wedge Pred' \wedge \alpha_Q = \beta_{j_0}$ is often not satisfiable, in particular if the action α_Q cannot be matched with β_{j_0}. Note also that β_{j_0} is only used as an intermediate term inside formulas in the composed open transition: it does not appear as global action, and will not appear as an action of a hole.

4.3. Bisimulation for open pNets – a composable bisimulation theory

As our symbolic operational semantics provides an open automaton, we can apply the notion of strong (symbolic) bisimulation on automata to open pNets.

Definition 10 (FH-bisimulation for open pNets). Two pNets are FH-bisimilar if their associated open automata are bisimilar.

[8] A similar lemma can be proven for a pLTS Q

We can now prove that pNet composition preserves FH-bisimilarity. More precisely, one can define two preservation properties, namely 1) when one hole of a pNet is filled by two bisimilar other (open) pNets; and 2) when the same hole in two bisimilar pNets are filled by the same pNet, in other words, composing a pNet with two bisimilar contexts. The general case will be obtained by transitivity of the bisimilarity relation (Theorem 1).

Theorem 3 (Congruence). *Consider an open pNet* $P = \langle\!\langle P_i^{i\in I}, Sort_j^{j\in J}, \overline{SV}\rangle\!\rangle$. *Let* $j_0 \in J$ *be a hole. Let* Q *and* Q' *be two FH-bisimilar pNets such that*[9] $\mathrm{Sort}(Q) = \mathrm{Sort}(Q') = Sort_{j_0}$. *Then* $P[Q]_{j_0}$ *and* $P[Q']_{j_0}$ *are FH-bisimilar.*

Theorem 4 (Context equivalence). *Consider two open pNets* $P = \langle\!\langle P_i^{i\in I}, Sort_j^{j\in J}, \overline{SV}\rangle\!\rangle$ *and* $P' = \langle\!\langle {P'}_i^{i\in I}, Sort_j^{j\in J}, \overline{SV'}\rangle\!\rangle$ *that are FH-bisimilar (they thus have the same holes). Let* $j_0 \in J$ *be a hole, and* Q *be a pNet such that* $\mathrm{Sort}(Q) = Sort_{j_0}$. *Then* $P[Q]_{j_0}$ *and* $P'[Q]_{j_0}$ *are FH-bisimilar.*

Finally, the previous theorems can be composed to state a general theorem about composability and FH-bisimilarity.

Theorem 5 (Composability). *Consider two FH-bisimilar pNets with an arbitrary number of holes, when replacing, inside those two original pNets, a subset of the holes by FH-bisimilar pNets, we obtain two FH-bisimilar pNets.*

This theorem is quite powerful, as it somehow implies that the theory of open pNets can be used to study properties of process composition. Open pNets can indeed be applied to study process operators and process algebras, as shown in [30] where compositional properties are extremely useful. In the case of interaction protocols [13], compositionality of bisimulation can justify abstractions used in some parts of the application.

5. Weak bisimulation

Weak symbolic bisimulation [26] was introduced to relate transition systems that have indistinguishable behaviour, with respect to some definition of *internal actions* that are considered local to some subsystem, and consequently cannot be observed, nor used for synchronisation with their context. The notion of non-observable actions varies in different contexts, e.g. *tau* in CCS [42, 43], and *i* in Lotos [11]. We could define classically a set of *internal/non-observable actions* depending on a specific action algebra. However in this paper, to simplify the notations, we will simply use τ as the single non-observable action; the generalisation of our results to a set of non-observable actions is trivial. Naturally, a non-observable action cannot be synchronised with actions of other

[9] Note that $\mathrm{Sort}(Q) = \mathrm{Sort}(Q')$ is ensured by strong bisimilarity.

systems in its environment. We show here that under such assumption of non-observability of τ actions, see Definition 11, we can define a weak bisimulation relation that is compositional, in the sense of open pNet composition. In this section we will first define a notion of weak open transition similar to open transition. In fact a weak open transition is made of several open transitions labelled as non-observable transitions, plus potentially one observable open transition. This allows us to define weak open automata, and a weak bisimulation relation based on these weak open automata. Finally, we apply this weak bisimulation to open pNets, obtain a weak bisimilarity relation for open pNets, and prove that this relation has compositional properties.

5.1. Preliminary definitions and notations

We first specify in terms of open transition, what it means for an action to be non-observable. We first define (in Definition 11) systems that cannot observe τ actions of sub-systems; namely pNets that cannot change their state, or emit an observable action when one of its holes emits a τ action.

More precisely, we state that τ is not observable if the automaton always allows any τ transition from holes, and additionally the global transition resulting from a τ action of a hole is a τ transition not changing the pNet's state. We define $\mathrm{Id}(V)$ as the identity function on the set of variables V.

Definition 11 (Non-observability of τ actions for open automata). An open automaton $A = \langle\!\langle J, \mathcal{S}, s_0, V, \mathcal{T} \rangle\!\rangle$ *cannot observe* τ *actions* if and only if for all j in J and s in $\mathcal{S}$ we have:

1. $$\frac{(j \mapsto \tau),\ \mathit{True},\ \mathrm{Id}(V)}{s \xrightarrow{\tau} s} \in \mathcal{T}$$

   and

2. for all β_j, J, α, s, s', *Pred*, *Post* such that

   $$\frac{\beta_j^{j \in J},\ \mathit{Pred},\ \mathit{Post}}{s \xrightarrow{\alpha} s'} \in \mathcal{T}$$

   If there exists j such that $\beta_j = \tau$ then we have:

   $$\alpha = \tau \wedge s = s' \wedge \mathit{Pred} = \mathit{True} \wedge \mathit{Post} = \mathrm{Id}(V) \wedge J = \{j\}$$

The first statement of the definition states that the open automaton must allow a hole to do a silent action at any time, and must not observe it, i.e. it cannot change its internal state because a hole did a τ transition. The second statement ensures that there cannot be in the open automaton other transitions that would be able to observe a τ action from a hole: statement (2) states that all the open transitions where a hole does a τ action must be of the shape given in statement (1). In this second statement, the condition $J = \{j\}$ is a bit restrictive, it could

safely be replaced by $\forall j \in J. \beta_j = \tau$, allowing the other holes to perform τ transitions too (because these τ actions cannot be observed). This possible synchronisation of τ actions would not be a problem as condition 1 still ensures that each process can do a τ separately.

By definition, one weak open transition contains several open transitions, where each open transition can require an observable action from a given hole, the same hole might have to emit several observable actions for a single weak open transition to occur. Consequently, for a weak open transition to trigger, a sequence of actions from a given hole may be required.

Thus, we let γ range over sequences of action terms and use $\oplus$ as the concatenation operator that appends sequences of action terms: given two sequences of action terms $\gamma \oplus \gamma'$ concatenates the two sequences. The operation is lifted to indexed sets of sequences: at each index i, $\overline{\gamma_1} \oplus \overline{\gamma_2}$ concatenates the sequences of actions at index i of $\overline{\gamma_1}$ and the one at index i of $\overline{\gamma_2}$[10]. $[a]$ denotes a sequence with a single element.

As required actions are now sequences of observable actions, we need an operator to build them from set of actions that occur in open transitions, i.e. an operator that takes a set of actions performed by one hole and produces a sequence of observable actions.

Thus we define $(\overline{\beta})^{\nabla}$ as the mapping $\overline{\beta}$ with only observable actions of the holes in I, but where each element is either empty or a list of length 1:

$$(\beta_i^{i \in I})^{\nabla} = [\beta_i]^{i \in I'} \text{ where } I' = \{i | i \in I \wedge \beta_i \neq \tau\}$$

As an example the $(\overline{\beta})^{\nabla}$ built from the transition OT_1 in Example 3, page 19 is $\texttt{P}\mapsto[\texttt{p-send(m)}]$. Remark that in our simple example no τ transition involves any visible action from a hole, so we have no β sequences of length longer than 1 in the weak automaton.

5.2. Weak open transition definition

Because of the non-observability property (Definition 11), it is possible to add any number of τ transitions of the holes before or after any open transition freely. This property justifies the fact that we can abstract away from τ transitions from holes in the definition of a weak open transition. We define weak open transitions similarly to open transitions except that holes can perform *sequences of observable actions* instead of single actions (observable or not). Compared to the definition of open transition, this small change has a significant impact as a single weak transition is the composition of several transitions of the holes.

Definition 12 (Weak open transition (WOT)). A weak open transition over a set J of holes with sorts $Sort_j^{j \in J}$ and a set of states $\mathcal{S}$ is a structure of the form:

$$\frac{\gamma_j^{j \in J'}, Pred, Post}{s \overset{\alpha}{\Rightarrow} s'}$$

[10] One of the two sequences is empty when $i \notin \text{dom}(\overline{\gamma_1})$ or $i \notin \text{dom}(\overline{\gamma_2})$.

$$\frac{}{s \overset{\emptyset, \mathit{True}, \mathrm{Id}(V)}{\overset{\tau}{\Rightarrow}} s} \in \mathcal{WT} \quad \textbf{WT1} \qquad \text{and} \qquad \frac{s \overset{\overline{\beta}, \mathit{Pred}, \mathit{Post}}{\overset{\alpha}{\longrightarrow}} s' \in \mathcal{T}}{s \overset{(\overline{\beta})^{\nabla}, \mathit{Pred}, \mathit{Post}}{\overset{\alpha}{\Rightarrow}} s' \in \mathcal{WT}} \textbf{WT2}$$

and

$$\frac{\begin{array}{c} s \overset{\overline{\gamma_1}, \mathit{Pred}_1, \mathit{Post}_1}{\overset{\tau}{\Rightarrow}} s_1 \in \mathcal{WT} \qquad s_1 \overset{\overline{\gamma_2}, \mathit{Pred}_2, \mathit{Post}_2}{\overset{\alpha}{\Rightarrow}} s_2 \in \mathcal{WT} \\ s_2 \overset{\overline{\gamma_3}, \mathit{Pred}_3, \mathit{Post}_3}{\overset{\tau}{\Rightarrow}} s' \in \mathcal{WT} \qquad \overline{\gamma} = \overline{\gamma_1} \oplus \overline{\gamma_2}\{\!\{\mathit{Post}_1\}\!\} \oplus \overline{\gamma_3}\{\!\{\mathit{Post}_2 \odot \mathit{Post}_1\}\!\} \\ \alpha' = \alpha\{\!\{\mathit{Post}_1\}\!\} \qquad \mathit{Pred} = \mathit{Pred}_1 \wedge \mathit{Pred}_2\{\!\{\mathit{Post}_1\}\!\} \wedge \mathit{Pred}_3\{\!\{\mathit{Post}_2 \odot \mathit{Post}_1\}\!\} \end{array}}{s \overset{\overline{\gamma}, \mathit{Pred}, \mathit{Post}_3 \odot \mathit{Post}_2 \odot \mathit{Post}_1}{\overset{\alpha'}{\Rightarrow}} s' \in \mathcal{WT}} \textbf{WT3}$$

Figure 6: Weak transition definition

Where $J' \subseteq J$, $s, s' \in \mathcal{S}$ and γ_j is a list of transitions of the hole j, with each element of the list in Sort_j. α is an action label denoting the resulting action of this open transition. *Pred* and *Post* are defined similarly to Definition 5. We use $\mathcal{WT}$ to range over sets of weak open transitions.

A weak open automaton $\langle\!\langle J, \mathcal{S}, s_0, V, \mathcal{WT} \rangle\!\rangle$ is similar to an open automaton except that $\mathcal{WT}$ is a set of weak open transitions over J and $\mathcal{S}$.

A weak open transition labelled α can be seen as a sequence of open transitions that are all labelled τ except one that is labelled α; however conditions on predicates, effects, and states must be verified for this sequence to be fired.

We are now able to build a weak open automaton from an open automaton. This is done in a way that resembles the process of τ saturation: we add τ open transitions before or after another open transition, regardless of whether it is observable or not.

Definition 13 (Building a weak open automaton).
Let $A = \langle\!\langle J, \mathcal{S}, s_0, V, \mathcal{T} \rangle\!\rangle$ be an open automaton. The weak open automaton *derived* from A is an open automaton $\langle\!\langle J, \mathcal{S}, s_0, V, \mathcal{WT} \rangle\!\rangle$ where $\mathcal{WT}$ is derived from $\mathcal{T}$ by saturation, applying the rules of Figure 6.

Rule **WT1** states that it is always possible to perform a non-observable transition, where the state is unchanged and the holes perform no action. Rule **WT2** states that each open transition is a weak open transition. Finally, Rule **WT3** allows any number of τ transitions before or after any weak open transition. This rule carefully composes predicates, effects, and actions of the holes. Indeed, predicate Pred_2 manipulates variables of s_1 that result from the first weak

open transition. Their values thus depend on the initial state but also on the effect (as a substitution function $Post_1$) of the first weak open transition. In the same manner, $Pred_3$ must be applied the substitution defined by the composition $Post_2 \odot Post_1$. Similarly, effects on variables must be applied to obtain the global effect of the composed weak open transition, to observable actions of the holes, and to the global action of the weak open transition.

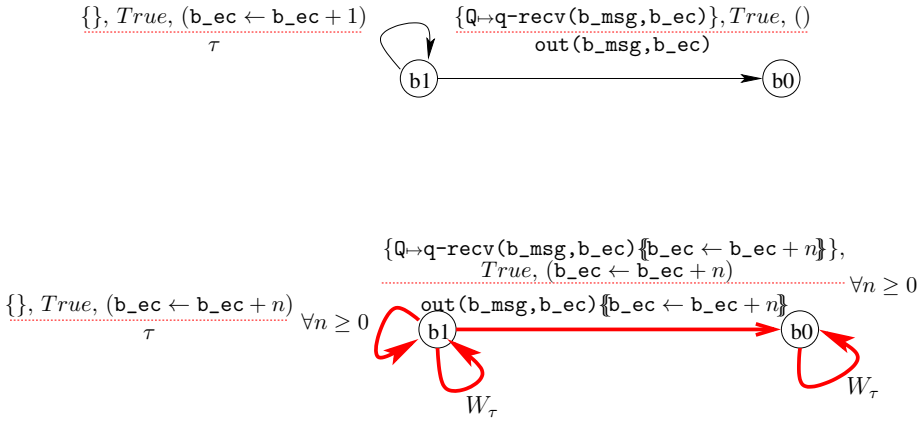


Figure 7: Construction of an example of weak open transition

Example 4 (A weak open-transition). Figure 7 shows the construction of one of the weak transitions of the open automaton of *SimpleProtocolSpec*. On the top we show the subset of the original open automaton (from Figure 4) considered here, and at the bottom the generated weak transition. For readability, we abbreviate the weak open transitions encoded by $\frac{\{\}, True, ()}{s \overset{\tau}{\Rightarrow} s'}$ as W_τ. The weak open transition shown here is the transition delivering the result of the algorithm to hole Q by applying rules: **WT1**,**WT2**, and **WT3**. First rule **WT1** adds a WT_τ loop on each state. Rule **WT2** transforms each 2 OTs into WOTs. Then consider application of Rule **WT3** on a sequence of 3 WOTs. $\frac{\{\}, True, (\texttt{b\_ec} \leftarrow \texttt{b\_ec} + 1)}{\texttt{b1} \overset{\tau}{\Rightarrow} \texttt{b1}}$; $\frac{\{\}, True, (\texttt{b\_ec} \leftarrow \texttt{b\_ec} + 1)}{\texttt{b1} \overset{\tau}{\Rightarrow} \texttt{b1}}$; $\frac{\{\}, True, ()}{\texttt{b1} \overset{\tau}{\Rightarrow} \texttt{b1}}$. The result will be: $\frac{\{\}, True, (\texttt{b\_ec} \leftarrow \texttt{b\_ec} + 2)}{\texttt{b1} \overset{\tau}{\Rightarrow} \texttt{b1}}$. We can iterate this construction an arbitrary number of times, getting for any natural number n a weak open transition: $\frac{\emptyset, True, (\texttt{ec} \leftarrow \texttt{ec} + n)}{\texttt{b1} \overset{\tau}{\Rightarrow} \texttt{b1}} \forall n \geq 0$. Finally, applying again **WT3**, and using the central open transition having `out(b_msg,b_ec)` as α, we get the resulting weak open transition between b1 and b0 (as shown in Figure 7). Applying the substitutions finally yields the weak transitions family WS_7 in Figure 8.

Example 5 (Weak open automata). Figures 8 and 9 respectively show the

weak automata of *SimpleProtocolSpec* and *SimpleProtocolImpl*. We encode weak open transitions by WS on the specification model and by WI on the implementation model.

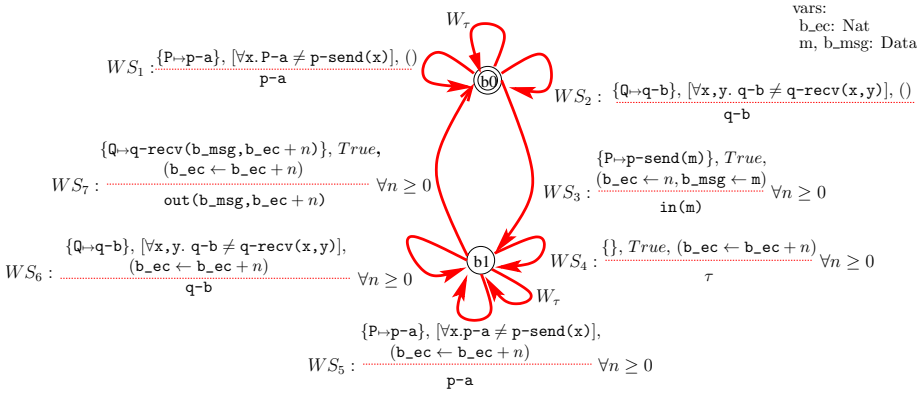


Figure 8: Weak Open Automaton of *SimpleProtocolSpec*

For readability, we only give names to the weak open transitions of *SimpleProtocolImpl* in Figure 9; we detail some of these transitions below and the full list is included in the extended version [3] . Let us point out that the weak OT loops (WI_1,WI_2 and W_τ) on state 000 are also present in all other states, we did not repeat them. Additionally, many WOTs are similar, and numbered accordingly as 3, 3a, 3b, 3c and 8, 8a, 8b, 8c respectively: they only differ by their respective source or target states; the "variant" WOTs appear in blue in Figure 9.

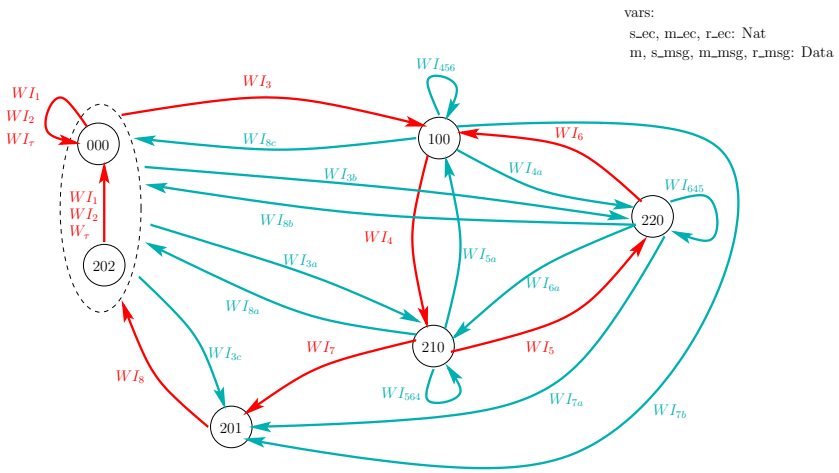


Figure 9: Weak Open Automaton of *SimpleProtocolImpl*

Now let us give some details about the construction of the weak automaton of the *SimpleProtocolImpl* pNet, obtained by application of the weak rules as explained above. We concentrate on weak open transitions WI_3 and WI_4. Let us denote as $post_n$ the effect (as a substitution function) of the strong open transitions SI_n from Figure 5:

$post_3 = (\texttt{s\_msg} \leftarrow \texttt{m}, \texttt{s\_ec} \leftarrow 0)$
$post_4 = (\texttt{m\_msg} \leftarrow \texttt{s\_msg}, \texttt{m\_ec} \leftarrow \texttt{s\_ec})$
$post_5 = ()$
$post_6 = (\texttt{s\_ec} \leftarrow \texttt{s\_ec+1})$

Then the effect of one single $100 \xrightarrow{OT_4} 210 \xrightarrow{OT_5} 220 \xrightarrow{OT_6} 100$ loop is[11]:

$$post_{456} = post_6 \odot post_5 \odot post_4 = (\texttt{s\_ec} \leftarrow \texttt{s\_ec} + 1)$$

So if we denote $post_{456*}$ any iteration of this loop, we get $post_{456*} = (\texttt{s\_ec} \leftarrow \texttt{s\_ec} + n)$ for any $n \geq 0$, and the *Post* of the weak OT WI_3 is:
$Post_3 = post_{456*} \odot post_3 = (\texttt{s\_msg} \leftarrow m, \texttt{s\_ec} \leftarrow n), \forall n \geq 0$ and *Post* of WI_{3a} is:
$post_4 \odot post_{456*} \odot post_3 = (\texttt{m\_msg} \leftarrow m, \texttt{m\_ec} \leftarrow n), \forall n \geq 0.$

We can now show some of the weak OTs of Figure 9 (the full table is included in the extended version [3]). As we have seen above, the effect of rule WT_3 when a silent action have an effect on the variable *ec* will generate an infinite family of WOTs, depending on the number of iterations through the loops. We denote these families using a "meta-variable" n, ranging over `Nat`.

$$WI_1 = \frac{\{\texttt{P}\mapsto\texttt{p-a}\}, [\forall \texttt{x}.\texttt{p-a} \neq \texttt{p-send(x)}], ()}{s \overset{\texttt{p-a}}{\Longrightarrow} s} \text{ (for any } s \in S)$$

$$\forall n \geq 0.\ WI_3(n) = \frac{\{\texttt{P}\mapsto\texttt{p-send(m)}\},\ True,\ (\texttt{s\_msg} \leftarrow \texttt{m}, \texttt{s\_ec} \leftarrow n)}{000 \overset{\texttt{in(m)}}{\Longrightarrow} 100}$$

$$\forall n \geq 0.\ WI_4(n) = \frac{\{\},\ True,\ (\texttt{m\_msg} \leftarrow \texttt{s\_msg}, \texttt{m\_ec} \leftarrow \texttt{s\_ec}+n, \texttt{s\_ec} \leftarrow \texttt{s\_ec}+n)}{100 \overset{\tau}{\Rightarrow} 210}$$

$$\forall n \geq 0.\ WI_{456}(n) = \frac{\{\},\ True,\ (\texttt{s\_ec} \leftarrow \texttt{s\_ec} + n)}{100 \overset{\tau}{\Rightarrow} 100}$$

The *Post* of the weak OT WI_{6a} is:

$$\begin{aligned} Post_{6a} &= post_4 \odot post_{456*} \odot post_6 \\ &= (\texttt{m\_msg} \leftarrow \texttt{s\_msg}, \texttt{m\_ec} \leftarrow \texttt{s\_ec}) \odot (\texttt{s\_ec} \leftarrow \texttt{s\_ec}+n) \odot (\texttt{s\_ec} \leftarrow \texttt{s\_ec}+1) \\ &= (\texttt{m\_msg} \leftarrow \texttt{s\_msg}, \texttt{m\_ec} \leftarrow \texttt{s\_ec} + 1 + n, \texttt{s\_ec} \leftarrow \texttt{s\_ec} + 1 + n) \end{aligned}$$

So we get:

$$\forall n \geq 0.\ WI_{6a}(n) = \frac{\{\},\ True,\ (\texttt{m\_ec} \leftarrow \texttt{s\_ec} + 1 + n, \texttt{s\_ec} \leftarrow \texttt{s\_ec} + 1 + n)}{220 \overset{\tau}{\Rightarrow} 210}$$

[11] when showing the result of *Posts* composition, we will omit the identity substitution functions introduced by the $\odot$ definition in page 7

5.3. Composition properties: composition of weak open transitions

We now have two different semantics for open pNets: a strong semantics, defined as an open automaton, and as a weak semantics, defined as a weak open automaton. Like the open automaton, the weak open automaton features valuable composition properties. We can exhibit a composition property and a decomposition property that relate open pNet composition with their semantics, defined as weak open automata. These are however technically more complex than the ones for open automata because each hole performs a set of actions, and thus a composed transition is the composition of one transition of the top-level pNet and a sequence of transitions of the sub-pNet that fills its hole. Composition and decomposition properties can be found as Lemma 6, Lemma 7, and Lemma 8 in [3].

5.4. Weak FH-bisimulation

For defining a bisimulation relation between weak open automata, two options are possible. One option is that we define a simulation similar to the strong simulation but based on weak open automata, this would look like the FH-simulation but would need to be adapted to weak open transitions. Alternatively, we could define directly and classically a weak FH-simulation as a relation between two open automata, relating the open transitions of the first one with the transitions of the weak open automaton derived from the second one.

The definition below specifies how a set of weak open transitions can simulate an open transition, and under which condition; this is used to relate, by weak FH-bisimulation, two open automata by reasoning on the weak open automata that can be derived from the strong ones.

Definition 14 (Weak FH-bisimulation).
Let $A_1 = \langle\!\langle J, \mathcal{S}_1, s_0, V_1, \mathcal{T}_1 \rangle\!\rangle$ and $A_2 = \langle\!\langle J, \mathcal{S}_2, t_0, V_2, \mathcal{T}_2 \rangle\!\rangle$ be open automata with disjoint sets of variables. Let $\langle\!\langle J, \mathcal{S}_1, s_0, V_1, \mathcal{WT}_1 \rangle\!\rangle$ and $\langle\!\langle J, \mathcal{S}_2, t_0, V_2, \mathcal{WT}_2 \rangle\!\rangle$ be the weak open automata derived from A_1 and A_2 respectively. Let $\mathcal{R}$ a relation over $\mathcal{S}_1$ and $\mathcal{S}_2$, as in Definition 7.

Then $\mathcal{R}$ is a weak FH-bisimulation iff for any states $s \in \mathcal{S}_1$ and $t \in \mathcal{S}_2$ such that $(s, t|Pred_{s,t}) \in \mathcal{R}$, we have the following:

- For any open transition OT in $\mathcal{T}_1$:

$$\frac{\beta_j^{j \in J'}, Pred_{OT}, Post_{OT}}{s \xrightarrow{\alpha} s'}$$

there exists an indexed set of weak open transitions $WOT_x^{x \in X} \subseteq \mathcal{WT}_2$:

$$\frac{\gamma_{jx}^{j \in J_x}, Pred_{OT_x}, Post_{OT_x}}{t \xRightarrow{\alpha_x} t_x}$$

such that $\forall x.\{j \in J' | \beta_j \neq \tau\} = J_x, (s', t_x | Pred_{s',t_x}) \in \mathcal{R}$; and

$$Pred_{s,t} \wedge Pred_{OT} \implies \bigvee_{x \in X} \left(\forall j \in J_x . (\beta_j)^{\nabla} = \gamma_{jx} \wedge Pred_{OT_x} \wedge \alpha = \alpha_x \wedge Pred_{s',t_x} \{\!\{ Post_{OT} \uplus Post_{OT_x} \}\!\} \right)$$

- and symmetrically any open transition from t in $\mathcal{T}_2$ can be covered by a set of weak transitions from s in $\mathcal{WT}_1$.

Two open automata are *weak FH-bisimilar* if there exists a weak FH-bisimulation relation that relates their initial states. This relation is called *weak FH-bisimilarity*. Two pNets are weak FH-bisimilar if their associated open automata are weakly bisimilar.

Compared to strong bisimulation, except the obvious use of weak open transitions to simulate an open transition, the condition on predicate is slightly changed concerning actions of the holes. Indeed only the visible actions of the holes must be compared and they form a list of actions, but of length at most one.

Our first important result is that weak FH-bisimilarity is an equivalence in the same way as strong FH-bisimilarity.

Theorem 6 (Weak FH-bisimilarity is an equivalence). *Weak FH-bisimilarity is reflexive, symmetric and transitive.*

The proof is detailed in [3], it follows a similar pattern as the proof that strong FH-bisimilarity is an equivalence, but technical details are different, and in practice we rely on a variant of the definition of weak FH-bisimilarity; this equivalent version simulates a *weak* open transition with a set of weak open transition. The careful use of the best definition of weak FH-bisimilarity makes the proof similar to the strong FH-bisimilarity case.

Proving bisimulation in practice

In practice, we are dealing with finite representations of the (infinite) open automata. In [31], we defined a slightly modified definition of the "coverage" proof obligation, in the case of strong FH-bisimulation. This modification is required to manage in a finite way all possible instantiations of an OT. In the case of weak FH-bisimulation, the proof obligation from Definition 14 becomes:

$$\forall fv_{OT}. \left\{ Pred_{s,t} \wedge Pred_{OT} \implies \bigvee_{x \in X} \left[\exists fv_{OT_x} . \left(\forall j \in J_x . (\beta_j)^{\nabla} = \gamma_{jx} \wedge Pred_{OT_x} \wedge \alpha = \alpha_x \wedge Pred_{s',t_x} \{\!\{ Post_{OT} \uplus Post_{OT_x} \}\!\} \right) \right] \right\}$$

where fv_{OT} denotes the set of free variables of all expressions in OT.

5.5. Weak FH-bisimilation for open pNets

Before defining a weak open automaton for the semantics of open pNets, it is necessary to state under which condition a pNet is unable to observe silent actions of its holes. In the setting of pNets this can simply be expressed as a condition on the synchronisation vectors. Precisely, the set of synchronisation vectors must contain vectors that let silent actions go through the pNet, i.e. synchronisation vectors where one hole does a τ transition, and the global visible action is a τ. Additionally, no other synchronisation vector must be able to react on a silent action from a hole, i.e. if a synchronisation vector observes a τ from a hole it cannot synchronise it with another action nor emit an action that is not τ. This is formalised as follows:

Definition 15 (Non-observability of silent actions for pNets). A pNet $\langle\!\langle P_i^{i\in I}, Sort_j^{j\in J}, \overline{SV}\rangle\!\rangle$ *cannot observe silent actions* if it verifies: $\forall i \in I \uplus J.\, (i{\mapsto}\tau) \to \tau[True] \in \overline{SV}$ and

$$\forall \left((\alpha_i)^{i\in I'} \to \alpha'[e_b] \in \overline{SV} \right), \forall i \in I' \cap J.\, \alpha_i = \tau \implies \alpha' = \tau \ \wedge \ I' = \{i\}$$

With this definition, it is easy to check that the open automaton that gives the semantics of such an open pNet cannot observe silent actions in the sense of Definition 11.

Property 1 (Non-observability of silent actions). *The semantics of a pNet, as provided in Definition 9, that cannot observe silent actions is an open automaton that cannot observe silent actions.*

Under this condition, it is safe to define the weak open automaton that provides a weak semantics to a given pNet. This is simply obtained by applying Definition 13 to generate a weak open automaton from the open automaton that is the strong semantics of the open pNet, as provided by Definition 9.

Definition 16 (Semantics of pNets as a weak open automaton). Let A be the open automaton expressing the semantics of an open pNet P; let $\langle\!\langle J, \mathcal{S}, s_0, V, \mathcal{WT}\rangle\!\rangle$ be the weak open automaton derived from A; we call this weak open automaton the weak semantics of the pNet P. Then, we denote $P \models WOT$ whenever $WOT \in \mathcal{WT}$.

From the definition of the weak open automata of pNets, we can now study the properties of weak bisimulation concerning open pNets.

5.6. Properties of weak bisimulation for open pNets

When silent actions cannot be observed, weak FH-bisimilarity is a congruence for open pNets: if P and Q are weakly bisimilar to P' and Q' then the composition of P and Q is weakly bisimilar to the composition of P' and Q', where composition is the hole replacement operator: $P[Q]_j$ and $P'[Q']_j$ are weak FH-bisimilar. This can be shown by proving the two following theorems.

The detailed proof of these theorem can be found in [3]. The proof strongly relies on the fact that weak FH-bisimulation is an equivalence, but also on the composition properties for open automata.

Theorem 7 (Congruence for weak FH-bisimilarity). *Consider an open pNet P that cannot observe silent actions, of the form $P = \langle\!\langle P_i^{i\in I}, Sort_j^{j\in J}, \overline{SV}\rangle\!\rangle$. Let $j_0 \in J$ be a hole. Let Q and Q' be two weak FH-bisimilar pNets such that*[12] $\mathrm{Sort}(Q) = \mathrm{Sort}(Q') \subseteq Sort_{j_0}$. *Then $P[Q]_{j_0}$ and $P[Q']_{j_0}$ are weak FH-bisimilar.*

Theorem 8 (Context equivalence for weak FH-bisimilarity). *Consider two open pNets $P = \langle\!\langle P_i^{i\in I}, Sort_j^{j\in J}, \overline{SV}\rangle\!\rangle$ and $P' = \langle\!\langle {P'}_i^{i\in I}, Sort_j^{j\in J}, \overline{SV'}\rangle\!\rangle$ that are weak FH-bisimilar (recall they must have the same holes to be FH-bisimilar) and that cannot observe silent actions. Let $j_0 \in J$ be a hole, and Q be a pNet such that $\mathrm{Sort}(Q) \subseteq Sort_{j_0}$. Then $P[Q]_{j_0}$ and $P'[Q]_{j_0}$ are weak FH-bisimilar.*

Finally, the previous theorems can be composed to state a general theorem about composability and weak FH-bisimilarity.

Theorem 9 (Composability of weak FH-bisimilarity). *Consider two weak FH-bisimilar pNets with an arbitrary number of holes, such that the two pNets cannot observe silent actions. When replacing, inside those two original pNets, a subset of the holes by weak FH-bisimilar pNets, we obtain two weak FH-bisimilar pNets.*

Example 6 (CCS Choice). Consider the + operator of CCS, shown in Example 1. The pNet does not satisfy Definition 15. Indeed, if a or b is τ then the + operator can observe the τ transition. It is well-known that weak bisimularity is not a congruence in CCS, this corresponds to the fact that the + operator can observe the τ transitions. Thus, even if we can define a weak FH-bisimilarity for CCS with + it does not verify the necessary requirements for being a congruence.

On the other side, the parallel operator defined similarly *satisfies* Definition 15, and indeed bisimilarity is a congruence for the parallel operator in CCS.

Running example

In Section 5 we have shown the full saturated weak automaton for both *SimpleProtocolSpec* and *SimpleProtocolImpl*. We will show here how we can check if some given relation between these two automata is a weak FH-bisimulation.

Preliminary remarks:

- Both pNets trivially verify the "non-observability" condition: the vectors having τ as an action of a sub-net are of the form "$< -, \tau, - > \rightarrow \tau$".

[12] Note that $\mathrm{Sort}(Q) = \mathrm{Sort}(Q')$ is ensured by weak FH-bisimilarity.

- We must take care of variable name conflicts: in our example, the variables of the 2 systems already have different names, but the action parameters occurring in the transitions (`m`, `msg`, `ec`) are the same, that is not correct. In the tools, this is managed by the static semantic layer; in the example, we rename the only conflicting variables m into $m1$ for *SimpleProtocolSpec*, and $m2$ for *SimpleProtocolImpl*.

Now consider the relation $\mathcal{R}$ defined by the following triples:

SimpleProtocolSpec states	*SimpleProtocolImpl* states	Predicate
b0	000	True
b0	202	True
b1	100	`b_msg = s_msg` $\wedge$ `b_ec = s_ec`
b1	210	`b_msg = m_msg` $\wedge$ `b_ec = m_ec`
b1	220	`b_msg = s_msg` $\wedge$ `b_ec = s_ec`
b1	201	`b_msg = r_msg` $\wedge$ `b_ec = r_ec`

Checking that $\mathcal{R}$ is a weak FH-bisimulation means checking, for each of these triples, that each (strong) OT of one the states corresponds to a set of WOTs of the other, using the conditions from Definition 14. We give here one example: consider the second triple from the table, and transition SS_3 from state `b0`. Its easy to guess that it will correspond to $WI_3(0)$ of state 202 (and equivalently state 000, see Figure 9):

$$SS_3 = \frac{\{\texttt{P}\mapsto\texttt{p-send(m1)}\}, True, (\texttt{b\_msg} \leftarrow \texttt{m1}, \texttt{b\_ec} \leftarrow 0)}{\texttt{b0} \xrightarrow{\texttt{in(m1)}} \texttt{b1}}$$

$$WI_3(0) = \frac{\{\texttt{P}\mapsto\texttt{p-send(m2)}\}, True, (\texttt{s\_msg} \leftarrow \texttt{m2}, \texttt{s\_ec} \leftarrow 0)}{000 \xrightarrow{\texttt{in(m2)}} 100}$$

Let us check formally the conditions:

- Their sets of active (non-silent) holes is the same: $J' = J_x = \{\texttt{P}\}$.
- Triple $(\texttt{b1}, 100, \texttt{b\_msg = s\_msg} \wedge \texttt{b\_ec = s\_ec})$ is in $\mathcal{R}$.
- The verification condition
  $\forall fv_{OT}.\{Pred \wedge Pred_{OT}$
  $\Longrightarrow \bigvee_{x\in X} \left[\exists fv_{OT_x}.\left(\forall j \in J_x.(\beta_j)^{\nabla} = \gamma_{jx} \wedge Pred_{OT_x} \wedge \alpha = \alpha_x \wedge Pred_{s',t_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\}\right)\right]\}$

  Gives us:

  $\forall \texttt{m1}.\{True \wedge True \implies \exists \texttt{m2}.$
  $(\texttt{[p-send(m1)]} = \texttt{[p-send(m2)]} \wedge True \wedge \texttt{in(m1)} = \texttt{in(m2)} \wedge$
  $(\texttt{b\_msg = s\_msg} \wedge \texttt{b\_ec = s\_ec})\{\!\{(\texttt{b\_msg} \leftarrow \texttt{m1}, \texttt{b\_ec} \leftarrow 0) \uplus (\texttt{s\_msg} \leftarrow \texttt{m2}, \texttt{s\_ec} \leftarrow 0)\}\!\})\}$

  That is reduced to:

  $\forall \texttt{m1}.\exists \texttt{m2}.(\texttt{p-send(m1)} = \texttt{p-send(m2)} \wedge \texttt{in(m1)} = \texttt{in(m2)} \wedge \texttt{m1} = \texttt{m2} \wedge 0 = 0)$

  That is a tautology.

6. Related Works

To the best of our knowledge, there are not many research works on Weak Bisimulation Equivalences between such complicate system models (open, symbolic, data-aware, with loops and assignments). We give a brief overview of other related publications, focussing first on Open and Compositional approaches, then on Symbolic Bisimulation for data-sensitive systems.

Open and compositional systems

In [36, 35], the authors investigate several methodologies for the compositional verification of software systems, with the aim to verify reconfigurable component systems. To improve scaling and compositionality, the authors decompose the verification problem that is to be resolved by a SMT (satisfiability modulo theory) solver into independent sub-problems on independent sets of variables. These works clearly highlight the interest of incremental and compositional verification in a very general setting. In our own work on open pNets, adding more structure to the composition model, we show how to enforce a compositional proof system that is more versatile than independent sets of variables as the composition is structured and allows arbitrary synchronisations between sub-entities. Our theory has also been encoded into an SMT solver and it would be interesting to investigate how the examples of evolving systems studied by Johnson et al. could be encoded into pNet and verified by our framework. However, the models of Johnson et al. are quite different from ours, in particular they are much less structured, and translating them is clearly outside the scope of this article.

In previous work [20], we also have shown how (closed) pNet models could be used to encode and verify finite instances of reconfigurable component systems.

Methodologies for reasoning about abstract semantics of open systems can be found in [5, 6, 18], authors introduce behavioural equivalences for open systems from various symbolic approaches. Working in the setting of process calculi, some close relations exist with the work of the authors of [5, 6], where both approaches are based on some kinds of labelled transition systems. The distinguishing feature of their approach is that the transitions systems are labelled with logical formulae that provides an abstract characterization of the structure that a hole must possess and of the actions it can perform in order to allow a transition to fire. Logical formulae are suitable formalisms that capture the general class of components that can act as the placeholders of the system during its evolution. In our approach we purposely leave the algebra of action terms undefined but the only operation we allow on action of holes is the comparison with other actions. Defining properly the interaction between a logical formulae in the action and the logics of the pNet composition seems very difficult. mCRL2 [22] is another effective model for specifying and proving properties of concurrent systems. mCRL2 has an established tool-suite and share similarities with pNets. However, pNets feature hierarchical composition with more structure than mCRL2 that composes processes with a parallel operator. Synchronisation of processes is expressed very differently; it is difficult

to precisely compare multi-actions of mCRL2 with synchronisation vectors of pNets but synchronisation vector ogf pNets enforce a synchronisation based on the structure while in mCRL2 synchronisation is specified in a versatile, flexible, but less structured way.

In the same vein as context systems [38], pNets is a formalism for modular and possibly incomplete description of concurrent systems. The two formalisms are however different as the theory of contexts relies on a form of rewrite rules, while pNets rely on parametric automata to express the system behaviour. pNets have similar features as context systems [38] and static constructs [33]. Indeed all these approaches allow for modular and possibly incomplete description and structural composition of systems. The main originality of pNets compared to these other compositional approaches is the parameterised nature of the specification, which enables reasoning on value-passing systems but also on rich synchronisations that depend on the value of parameters.

Decomposition techniques

Quotienting of process algebras [38] and decomposition techniques for mCRL2 [39] share similarities with our approach; they propose to overcome the state-space-explosion problem by decomposing formulas to be verified according to the process composition. The decomposed problem must be equivalent to the original one. However these techniques are expressed in a very different setting from ours and it is difficult to precisely relate them to the more structural and parameterised point of view we adopt here. We could try to apply such automatic decomposition techniques to open pNets, but deriving a decomposition for systems synchronised in a very parameterised way like we do requires further investigations. Both parallel composition [38] and mCRL2 [39] feature a concrete verification setting where decomposition is useful, while open automata provide a more general setting that could be used to represent both frameworks and hopefully generalise process decomposition results of [38, 39].

Logical and semantics approaches

Among the approaches for modelling open systems, one can cite [8] that uses transition conditions depending on an external environment, and introduce bisimulation relations based on this approach. The approach of [8] is highly based on logics and their bisimulation theory is richer than ours in this aspect, while our theory is highly structural and focuses on relation between structure and equivalence. Also, we see composition as a structural operation putting systems together, and do not focus on the modelling of an unknown outside world. Overall we believe that the two approaches are complementary but comparing precisely the two different bisimulation theories is not trivial.

There is also a clear relation with the seminal works on rule formats for Structured Operational Semantics, e.g. De Simone format, GSOS, and conditional rules with or without negative premises [16, 10, 24, 47]. The Open pNets model provides a way to define operators similar to these rules formats, but with quite different aim and approach. A formal comparison would be interesting, though not trivial. What we can say easily is that: the pNet format

syntactically encompasses De Simone, GSOS, and conditional premises rules. Then our compositionality result is more powerful than their classical results, but this is not a surprise, as we rely on a (sufficient) syntactic hypothesis on a particular system, rather than the general rules defining an operator. Last, we intentionally do not accept negative premises, that would be more to put into practice in our implementation. This extension could be studied in future work.

Symbolic and data-sensitive systems

As mentioned in the *Introduction*, we were substantially inspired by the works of Lin et al. [34, 26, 40]. They developed the theory of symbolic transition graphs (STG), and the associated symbolic (early and late, strong and weak) bisimulations. Moreover, they studied STGs with assignments as a model for message-passing processes. Our work extends those contributions in several ways: first our models are compositional, and our bisimulations come with effective conditions for being preserved by pNet composition (i.e. congruent), even for the weak version. This result is more general than the bisimulation congruences for value-passing CCS in [34]. Then our settings for management of data types are much less restrictive, thanks to our use of satisfiability engines, while Lin's algorithms were limited to data-independent systems.

In a similar way, [1] presents a notion of "data-aware" bisimulations on data graphs, in which computation of such bisimulations is studied based on XPath logical language extended with tests for data equality.

Research related to the keyword "Symbolic Bisimulation" refer to two very different domains, namely BDD-like techniques for modelling and computing finite-state bisimulations, that are not related to our topic; and symbolic semantics for data-dependant or high-order systems, that are very close in spirit to our approach. In this last area, we can mention Calder's work [15], that defines a symbolic semantic for full Lotos, with a symbolic bisimulation over it; Borgstrom et al., Liu et al, Delaune et al. and Buscemi et al. providing symbolic semantics and equivalence for different variants of pi calculus respectively [12, 17, 41, 14]; and more recently Feng et al. provide a symbolic bisimulation for quantum processes [19]. All the above works are based on models definitely different from ours, and none of them allows system to be as much parameterised as open pNets; this additional expressiveness is due to the open and symbolic nature of our constructs.

7. Conclusion and discussion

pNets (Parameterised Networks of Automata) is a formalism adapted to the representation of the behaviour of parallel or distributed systems. One strength of pNets is their parameterised nature, making them suitable for to the representation of systems of arbitrary size, and making the modelling of parameterised systems possible. Parameters are also crucial to reason about interaction protocols that can address one entity inside an indexed set of processes. pNets have been successfully used to represent behavioural specification

of parallel and distributed components and verify their correctness [2, 29]. VCE is the specification and verification platform that uses pNets as an intermediate representation. In this platform we have developed tool support for computing the symbolic semantics in term of open automata; this is presented in [45, 46], together with a case-study based on the on-board control software of satellites. In [9] we present how to encode reactive systems from the BIP specification language and check their temporal properties using VCE. In [31, 32] we describe our strong bisimulation algorithms, with illustration on the equivalence of different encodings of operators.

Open pNets are pNets with holes; they are adapted to represent processes parameterised by the behaviour of other processes, like composition operators or interaction protocols that synchronise the actions of processes that can be provided afterwards. Open pNets are hierarchical composition of automata with holes and parameters. We defined here a semantics for open pNets and a complete bisimulation theory for them. The semantics of open pNets relies on the definition of open automata that are automata with holes and parameters, but no hierarchy. Open automata are a flattened view of the pNet; their behaviour is expressed as open transitions that allow for a more semantic interpretation of process parameters (holes) than pNets. In the end, open automata are labelled transition systems with parameters and holes, a notion that is useful to define semantics, but makes less sense for the high level modelling of a system, compared to pNets. Open automata is the formalism that makes it possible to define FH-bisimilarity.

This article defines a strong and a weak bisimulation relation that are adapted to parameterised systems and hierarchical composition. FH-bisimulation handles pNet parameters in the sense that two states might be or not in relation depending on the value of parameters. Strong FH-bisimilarity is compositional in the sense that it is maintained when composing processes. We also identified a simple and realistic condition on the semantics of non-observable actions that allows weak FH-bisimilarity to be also compositional. Overall we believe that this article paved the way for a solid theoretical foundation for compositional verification of parallel and distributed systems.

The pNets formalism supports the refinement checking at the automaton level through a simulation, with symbolic evaluation of guards and transitions. The definition of simulation on open automata should be stronger than a classical simulation since it matches a transition with a family of transitions. Such a relation should be able to check the refinement by taking into account state duplication, transition removal, guard strengthening, variable modification. Additionally, composition of pNets gives the possibility to either add new holes to a system or fill holes. A useful simulation relation should thus support the comparison of automata that do not have the same number of holes. Designing such a simulation relation is a non-trivial extension that we leave for future work.

We are currently looking at further properties of FH-bisimulation, but also the relations with existing equivalences on both closed and open systems. In particular, our model being significantly different from those considered in [34], it would be interesting to compare our “FH” family of bisimulations with the

hierarchy of symbolic bisimulations from those authors. We also plan to apply open pNets to the study of complex composition operators in a symbolic way, for example in the area of parallel skeletons, or distributed algorithms.

Recently we published preliminary work on methods for checking weak FH-bisimulation [48]. The challenges here, in the context of our symbolic systems, are not so much algorithmic complexity, as was the case with classical weak bisimulation on finite models, but decidability and termination. The naive approach, using an explicit construction of the weak transition, may in itself introduce non-termination, so we prefer a direct implementation of the weak bisimulation definition, without constructing the weak automata beforehand, but searching *on demand* to construct the required weak transitions. We illustrate this approach on a simple error-correcting transport protocol case-study. Beside, we explore in [49] more pragmatic approaches using weak bisimulation preserving (pattern-based) reduction rules.

References

[1] Sergio Abriola, Pablo Barceló, Diego Figueira, and Santiago Figueira. Bisimulations on data graphs. *Journal of Artificial Intelligence Research*, 61:171–213, 2018.

[2] Rabéa Ameur-Boulifa, Ludovic Henrio, Oleksandra Kulankhina, Eric Madelaine, and Alexandra Savu. Behavioural semantics for asynchronous components. *Journal of Logical and Algebraic Methods in Programming*, 89:1–40, 2017.

[3] Rabéa Ameur-Boulifa, Ludovic Henrio, and Eric Madelaine. Compositional equivalences based on open pnets. *CoRR*, abs/2007.10770, 2020.

[4] André Arnold. Synchronised behaviours of processes and rational relations. *Acta Informatica*, 17:21–29, 1982.

[5] Paolo Baldan, Andrea Bracciali, and Roberto Bruni. Bisimulation by unification. In *AMAST 2002 - Algebraic Methodology and Software Technology, 9th International Conference*, volume 2422 of *Lecture Notes in Computer Science*, pages 254–270. Springer, 2002.

[6] Paolo Baldan, Andrea Bracciali, and Roberto Bruni. A semantic framework for open processes. *Theoretical Computer Science*, 389(3):446–483, 2007.

[7] Tomás Barros, Rabéa Ameur-Boulifa, Antonio Cansado, Ludovic Henrio, and Eric Madelaine. Behavioural models for distributed fractal components. *Annales des Télécommunications*, 64(1-2):25–43, 2009.

[8] Harsh Beohar, Barbara König, Sebastian Küpper, and Alexandra Silva. Conditional transition systems with upgrades. *Science of Computer Programming*, 186:102320, 2020.

[9] Simon Bliudze, Ludovic Henrio, and Eric Madelaine. Verification of concurrent design patterns with data. In Hanne Riis Nielson and Emilio Tuosto, editors, *COORDINATION 2019 - 21st International Conference on Coordination Models and Languages*, volume LNCS-11533, pages 161–181. Springer International Publishing, 2019.

[10] Bard Bloom, Sorin Istrail, and Albert R. Meyer. Bisimulation can't be traced. *Journal of the ACM*, 42(1):232–268, 1995.

[11] Tommaso Bolognesi and Ed Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks*, 14(1):25–59, 1987.

[12] Johannes Borgström, Sébastien Briais, and Uwe Nestmann. Symbolic Bisimulation in the Spi Calculus. In *CONCUR 2004 - Concurrency Theory, 15th International Conference*, volume 3170 of *Lecture Notes in Computer Science*, pages 161–176. Springer, 2004.

[13] Rabéa Ameur Boulifa, Raluca Halalai, Ludovic Henrio, and Eric Madelaine. Verifying safety of fault-tolerant distributed components. In *FACS 2011 - 8th International Symposium on Formal Aspects of Component Software*, Lecture Notes in Computer Science. Springer, 2011.

[14] Maria Grazia Buscemi and Ugo Montanari. Open Bisimulation for the Concurrent Constraint Pi-Calculus. In *ESOP 2008 - Programming Languages and Systems, 17th European Symposium on Programming*, volume 4960 of *Lecture Notes in Computer Science*, pages 254–268. Springer, 2008.

[15] Muffy Calder and Carron Shankland. A symbolic semantics and bisimulation for full LOTOS. In *FORTE 2001 - 21st International Conference on Formal Techniques for Networked and Distributed Systems*, pages 185–200. Springer, 2001.

[16] Robert De Simone. Higher-level synchronising devices in MEIJE-SCCS. *Theoretical Computer Science*, 37:245–267, 1985.

[17] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Symbolic Bisimulation for the Applied Pi Calculus. In *FSTTCS 2007 - Foundations of Software Technology and Theoretical Computer Science, 27th International Conference*, volume 4855 of *Lecture Notes in Computer Science*, pages 133–145. Springer, 2007.

[18] Jérémy Dubut. Bisimilarity of Diagrams. In *RAMiCS 2020 - Relational and Algebraic Methods in Computer Science, 18th International Conference*, volume 12062 of *Lecture Notes in Computer Science*, pages 65–81. Springer, 2020.

[19] Yuan Feng, Yuxin Deng, and Mingsheng Ying. Symbolic bisimulation for quantum processes. *ACM Transactions on Computational Logic (TOCL)*, 15(2):14, 2014.

[20] Nuno Gaspar, Ludovic Henrio, and Eric Madelaine. Formally reasoning on a reconfigurable component-based system - A case study for the industrial world. In *FACS 2013 - Formal Aspects of Component Software, 10th International Symposium,*, volume 8348 of *Lecture Notes in Computer Science*, pages 137–156. Springer, 2013.

[21] Jan Friso Groote. Transition system specifications with negative premises. *Theoretical Computer Science*, 118(2):263–299, 1993.

[22] Jan Friso Groote, Jeroen J. A. Keiren, Bas Luttik, Erik P. de Vink, and Tim A. C. Willemse. Modelling and analysing software in mcrl2. In Farhad Arbab and Sung-Shik Jongmans, editors, *Formal Aspects of Component Software*, pages 25–48, Cham, 2020. Springer International Publishing.

[23] Jan Friso Groote, Mohammad Reza Mousavi, and Michel A. Reniers. A hierarchy of SOS rule formats. *Electronic Notes in Theoretical Computer Science*, 156(1):3–25, 2006. Proceedings of the Second Workshop on Structural Operational Semantics.

[24] Jan Friso Groote and Frits Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100(2):202–260, 1992.

[25] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming.*, 8(3):231–274, 1987.

[26] Matthew Hennessy and Huimin Lin. Symbolic bisimulations. *Theoretical Computer Science*, 138(2):353–389, 1995.

[27] Matthew Hennessy and Humin Lin. Symbolic bisimulations. *Theoretical Computer Science*, 138(2):353–389, 1995. Meeting on the mathematical foundation of programing semantics.

[28] Matthew Hennessy and Julian Rathke. Bisimulations for a calculus of broadcasting systems. *Theoretical Computer Science*, 200(1-2):225–260, 1998.

[29] Ludovic Henrio, Oleksandra Kulankhina, Siqi Li, and Eric Madelaine. Integrated environment for verifying and running distributed components. In *FASE 2016 - 19th International Conference on Fundamental Approaches to Software Engineering*, pages 66–83. Springer Berlin Heidelberg, 2016.

[30] Ludovic Henrio, Eric Madelaine, and Min Zhang. A Theory for the Composition of Concurrent Processes. In *FORTE 2016 - 36th International Conference on Formal Techniques for Distributed Objects, Components, and Systems*, volume 9688, pages 175 – 194. Springer, 2016.

[31] Zechen Hou and Eric Madelaine. Symbolic Bisimulation for Open and Parameterized Systems. In *PEPM 2020 - Workshop on Partial Evaluation and Program Manipulation*. ACM SIGPLAN, 2020.

[32] Zechen Hou, Eric Madelaine, Jing Liu, and Yuxin Deng. Symbolic Bisimulation for Open and Parameterized Systems - Extended version. Research Report RR-9304, Inria & Université Cote d'Azur, CNRS, I3S, Sophia Antipolis, France ; East China Normal University (Shanghai), November 2019.

[33] Hans Hüttel and Kim Guldstrand Larsen. The use of static constructs in A modal process logic. In *Logic at Botik '89, Symposium on Logical Foundations of Computer Science*, volume 363 of *Lecture Notes in Computer Science*, pages 163–180. Springer, 1989.

[34] Anna Ingólfsdóttir and Huimin Lin. A symbolic approach to value-passing processes. In *Handbook of Process Algebra*, pages 427–478. North-Holland/Elsevier, 2001.

[35] Kenneth Johnson and Radu Calinescu. Efficient Re-resolution of SMT Specifications for Evolving Software Architectures. In *QoSA 2014 - 10th International ACM Sigsoft Conference on Quality of Software Architectures*, pages 93–102. ACM, 2014.

[36] Kenneth Johnson, Radu Calinescu, and Shinji Kikuchi. An incremental verification framework for component-based software systems. In *CBSE 2013 - 16th International ACM Sigsoft Symposium on Component-based Software Engineering*, pages 33–42. ACM, 2013.

[37] Kim G. Larsen. A context dependent equivalence between processes. *Theoretical Computer Science*, 49:184–215, 1987.

[38] Kim Guldstrand Larsen and Liu Xinxin. Compositionality through an operational semantics of contexts. *Journal of Logic and Computation*, 1(6):761–795, 1991.

[39] Maurice Laveaux and Tim A. C. Willemse. Decomposing monolithic processes in a process algebra with multi-actions. In *ICE*, volume 347 of *EPTCS*, pages 57–76, 2021.

[40] Huimin Lin. Symbolic transition graph with assignment. In *CONCUR'96 - Concurrency Theory, 7th International Conference*, volume 1119, pages 50–65. Springer Berlin Heidelberg, 1996.

[41] Jia Liu and Huimin Lin. A Complete Symbolic Bisimulation for Full Applied Pi Calculus. In *SOFSEM 2010 - 36th Conference on Current Trends in Theory and Practice of Computer Science*, volume 5901, pages 552–563. Springer, 2010.

[42] Robin Milner. *A Calculus of Communicating Systems*. Springer-Verlag, Berlin, Heidelberg, 1982.

[43] Robin Milner. *Communication and Concurrency*. Int. Series in Computer Science. Prentice-Hall, Englewood Cliffs, New Jersey, 1989. SU Fisher Research 511/24.

[44] Gordon D. Plotkin. A structural approach to operational semantics. *Journal of Logic and Algebraic Programming*, 60-61:17–139, 2004.

[45] Xudong Qin, Simon Bliudze, Eric Madelaine, and Min Zhang. Using SMT engine to generate symbolic automata. In *AVOCS 2018 - 18th International Workshop on Automated Verification of Critical Systems*, volume 076. Electronic Communications of the EASST, 2018.

[46] Xudong Qin, Simon Bliudze, Eric Madelaine, and Min Zhang. Using SMT engine to generate Symbolic Automata -Extended version. Research Report RR-9177, Inria & Université Cote d'Azur, CNRS, I3S, Sophia Antipolis, France ; inria, June 2018.

[47] Robert Jan Van Glabbeek. The meaning of negative premises in transition system specifications II. *The Journal of Logic and Algebraic Programming*, 60-61:229–258, 2004.

[48] Biyang Wang, Eric Madelaine, and Min Zhang. New symbolic model and equivalences checking for open automata. In *SMC 2021 - IEEE International Conference on Systems, Man, and Cybernetics*, pages 2360–2367. IEEE, 2021.

[49] Biyang Wang, Eric Madelaine, and Min Zhang. Symbolic Weak Equivalences: Extension, Algorithms, and Minimization - Extended version. Research Report RR-9389, Inria, Université Cote d'Azur, CNRS, I3S, Sophia Antipolis, France; East China Normal University (Shanghai), 2021.

Compositional Equivalences Based on Open pNets

Rabéa Ameur-Boulifa

LTCI, Télécom Paris, Institut Polytechnique de Paris, France

Ludovic Henrio*

Univ Lyon, EnsL, UCBL, CNRS, Inria, LIP, F-69342, LYON Cedex 07, France.

Eric Madelaine

INRIA Sophia Antipolis Méditérannée, UCA, BP 93, 06902 Sophia Antipolis, France

Abstract

Establishing equivalences between programs is crucial both for verifying correctness of programs and for justifying optimisations and program transformations. There exist several equivalence relations for programs, and bisimulations are among the most versatile of these equivalences. Among bisimulations one distinguishes strong bisimulation that requires that each action of a program is simulated by a single action of the equivalent program, and weak bisimulation that allows some of the actions to be invisible, and thus not simulated.

pNet is a generalisation of automata that model open systems. They feature variables and hierarchical composition. Open pNets are pNets with holes, i.e. placeholders that can be filled later by sub-systems. However, there is no standard tool for defining the semantics of an open system in this context. This article first defines *open automata* that are labelled transition systems with parameters and holes. Relying on open automata, it then defines bisimilarity relations for the comparison of systems specified as pNets. We first present a strong bisimilarity for open pNets called FH-bisimilarity. Next we offer an equivalence relation similar to the classical *weak bisimulation* equivalence, and study its properties. Among these properties we are interested in compositionality: if two systems are proven equivalent they will be indistinguishable by their context, and they will also be indistinguishable when their holes are filled with equivalent systems. We identify sufficient conditions to ensure compositionality of strong and weak bisimulation. The contributions of this article are illustrated using a transport protocol as running example.

Keywords: Bisimulation, compositionality, automata, semantics.

*Corresponding author

Email addresses: rabea.ameur-boulifa@telecom-paris.fr (Rabéa Ameur-Boulifa), ludovic.henrio@cnrs.fr (Ludovic Henrio), eric.madelaine@inria.fr (Eric Madelaine)

1. Introduction

In the nineties, several works extended the basic behavioural models based on labelled transition systems to address value-passing or parameterised systems, using various symbolic encodings of the transitions [16, 37]. These works use the term *parameter* to designate variables whose value have a strong influence the system structure and behaviour. In parameterised systems, parameters can typically be the number of processes in the system or the way they interact. In [34, 26], Lin, Ingolfsdottir and Hennessy developed a full hierarchy of bisimulation equivalences, together with a proof system, for value passing CCS, including notions of symbolic behavioural semantics and various symbolic bisimulations (early and late, strong and weak, and their congruent versions). They also extended this work to models with explicit assignments [40]. Separately Rathke [28] defined another symbolic semantics for a parameterised broadcast calculus, together with strong and weak bisimulation equivalences, and developed a symbolic model-checker based on a tableau method for these processes. Thirty years later, no verification platform use this kind of approaches to provide proof methods for value-passing processes or open process expressions, perhaps because of the difficulty to apply these methods on industrial systems.

This article provides a theoretical background that allows us to implement such a verification platform. We build upon the concept of pNets that we have employed to give a behavioural semantics of distributed components and verify the correctness of distributed applications in the past 15 years. pNets is a low level semantic framework for expressing the behaviour of various classes of distributed languages, and as a common internal format for our tools. pNets support the specification of parameterised hierarchical labelled transition systems: labelled transition systems with parameters can be combined hierarchically.

We develop here a semantics for a model of interacting processes with parameters and holes. Our approach is originally inspired from Structured Operational Semantics with conditional premises as in [21, 47]. But we aim at a more constructive and implementable approach to compute the semantics (intuitively transitions including first order predicates) and to check equivalences for these open systems. The main interest of our symbolic approach is to define a method to prove properties directly on open structures; these properties will then be preserved by any correct instantiation of the holes. As a consequence, our model allows us to reason about composition operators as well as about realistic distributed systems. The parametric nature of the model and the properties of compositionality of the equivalence relations are thus the main strengths of our approach.

pNets. pNet is a convenient model to model concurrent systems in a hierarchical and parameterised way. The coordination between processes is expressed as synchronisation vectors that allow for the definition of complex and expressive synchronisation patterns. Open pNets are pNets for which some elements in the

hierarchy are still undefined, such undefined elements are called *holes*. A hole can be *filled* later by providing another pNet. This article first defines pNets and illustrates with an example how they can be used to provide the model of a communicating system.

A semantics for open pNets based on open automata. The semantics of pNets can be expressed as a translation to a labelled transition system (LTS), but only if the pNet has no parameter and no hole. Adding parameters to a LTS is quite standard [40] but enabling holes inside LTSs is not a standard notion.

To define a semantics for open pNets we thus need LTSs that have both standard variable parameters, and process parameters, i.e. holes that can be filled by processes. We call such LTSs with parameters and holes *open automata*. The main goal of this article is to define the theory behind open automata and to use them to provide a semantics and prove compositionality properties for open pNets. The transitions of open automata are much more complex than transitions of an LTS as the firing of a transition depends on parameters and actions that are symbolic. This article defines the notion of *open transition*, namely a transition that is symbolic in terms of parameters and coordinated actions.

Beware that even if open transitions may look similar to the notion of Transition System Specification [24, 23] and other forms of SOS rules, they are *not* structural rules, but rules defining the behaviour of the global states of the system.

Unlike pNets, open automata are not hierarchical structures, we consider them here as a mathematical structure that is convenient for formal reasoning but not adapted to the definition of a complex and structured system. Open transitions are expressed in terms of logics while the communication in pNets is specified as synchronisation vectors that specify synchronised actions. Open automata could alternatively be seen as an algebra that can be studied independently from its application to pNets but their compositionality properties make more sense in a hierarchical model like pNets.

Previous works and contribution

While most of our previous works relied on closed, fully-instantiated semantics [7, 2, 29], it is only recently that we could design a first version of a parameterised semantics for pNets with a strong bisimulation equivalence [30]. This article builds upon this previous parameterised semantics and provides a clean and complete version of the semantics with a slightly simplified formalism that makes proofs easier. It also adds a notion of global state to automata. Moreover, in [30] the study of compositionality was only partial, and in particular the proof that bisimularity is an equivalence is one new contribution of this article and provides a particularly interesting insight on the semantic model we use. The new formalism allowed us to extend the work and define weak bisimulation for open automata, which is entirely new. This allows us to define a weak bisimulation equivalence for open pNets with valuable compositionality properties. To summarise, the contribution of this paper are the following:

- The definition of open automata: an algebra of parameterised automata with holes, and a strong bisimulation relation. This is an adaptation of [30] with an additional result stating that strong FH-bisimilarity is indeed an equivalence relation.
- A semantics for open pNets expressed as translation to open automata. This is an adaptation of [30] with a complete proof that strong FH-bisimilarity is compositional.
- A theory of weak bisimulation for open automata, and a study of its properties. It relies on the definition of weak open transitions that are derived from transitions of the open automaton by concatenating invisible action transitions with one (visible or not) action transition. The precise and sound definition of the concatenation is also a major contribution of this article.
- A resulting weak FH-bisimilarity equivalence for open pNets and a simple static condition on synchronisation vectors inside pNets that is sufficient to ensure that weak FH-bisimilarity is compositional.
- An illustrative example based on a simple transport protocol, showing the construction of the weak open transitions, and the proof of weak FH-bisimulation.

What is new about open automata bisimulation?

Bisimulation over a symbolic and open model like open pNets or open automata is different from the classical notion of bisimulation because it cannot rely on the equality over a finite set of action labels. Classical bisimulations require to exhibit, for each transition of one system, a transition of the other system that simulates it. Instead, bisimulation for open automata relies on the simulation of each open transition of one automaton by *a set of* open transitions of the other one, that should cover all the cases where the original transition can be triggered. This is similar to the early and late symbolic bisimulation equivalences for value-passing CCS [27], though we use more general definitions in our setting.

Compositionality of bisimilarity in our model comes from the specification of the interactions, including actions of the holes. This is quite different from the works on contextual equivalences, e.g. [37, 38]; we will provide a detailed comparison in Section 6. In pNets, synchronisation vectors define the possible interactions between the pNet that fills the hole and the surrounding pNets. In open automata, this is reflected by symbolic hypotheses that depend on the actions of the holes. This additional specification is the price to pay to obtain the compositionality of bisimilarity that cannot be guaranteed in traditional process algebras.

This approach also allows us to specify a sufficient condition on allowed transitions to make weak bisimilarity compositional; namely it is not possible to synchronise on invisible actions from the holes or prevent them to occur. This

is loosely related to works on the syntactic conditions on SOS rules to check whether weak bisimulation is a congruence for some process algebra operators [24]. Our approach is semantical and more global: our sufficient condition applies to all the synchronisations at a given composition level of an (open) system and not on individual rules. It is expressed on the open automaton (see Definition 15).

Structure

This article is organised as follows. Section 2 provides the definition of pNets and introduces the notations used in this paper, including the definition of open pNets. Section 3 defines open automata, i.e. automata with parameters and transitions conditioned by the behaviour of "holes"; a strong bisimulation equivalence for open automata is also presented in this section. Section 4 gives the semantics of open pNets expressed as open automata, and states compositionality properties of strong bisimularity for open pNets. Section 5 defines a weak bisimulation equivalence on open automata and derives weak bisimilarity for pNets, together with compositionality properties of weak bisimilarity. Finally, Section 6 discusses related works and Section 7 concludes the paper.

2. Background and Notations

This section introduces the notations we will use in this article, and recalls the definition of pNets [30] with an informal semantics of the pNet constructs. The only significant difference compared to our previous definitions (from [30]) is that we remove here the restriction that was stating that variables should be local to a state of a labelled transition system.

2.1. Notations

Term algebra. Our models rely on a notion of parameterised actions, which are symbolic expressions using data types and variables. As our model aims at encoding the low-level behaviour of possibly very different programming languages, we do not want to impose one specific algebra for denoting actions, nor any specific communication mechanism. So we leave the constructors of the algebra that will be used to build expressions and actions unspecified. Moreover, we use a generic *action interaction* mechanism, based on (some sort of) unification between two or more action expressions, to express various kinds of communication or synchronisation mechanisms.

Formally, we assume the existence of a term algebra $\mathbb{T}$, and denote as Σ the signature of the data and action constructors. Within $\mathbb{T}$, we distinguish a set of data expressions $\mathbb{E}$, including a set of boolean expressions $\mathbb{B}$ ($\mathbb{B} \subseteq \mathbb{E}$), and a set of action expressions called the action algebra $\mathbb{A}$, with $\mathbb{A} \subseteq \mathbb{T}, \mathbb{E} \cap \mathbb{A} = \emptyset$; naturally action terms will use data expressions as sub-terms[1]. The function $vars(t)$ identifies the set of variables in a term $t \in \mathbb{T}$.

[1] In our tools, we use datatypes for the different kinds of terms. In this article, we use different sets of variables for terms of different kinds.

We let e_i range over expressions ($e_i \in \mathbb{E}$), a range over action labels, *op* be operators, and x_i and y_i range over variable names. We additionally rely on a set of action names, ranged over by a, b, ...

We define two kinds of parameterised actions. The first kind supports two kinds of parameters: input parameters that are variables and output parameters that can be any expression. The second kind makes no distinction between input and output parameters. The actions that distinguish input variables will be used in the definition of *pLTS* below and are defined as follows:

$\alpha \in \mathbb{A}$	::=	$a(p_1, \ldots, p_n)$	action terms
p_i	::=	$?x \mid e_i$	parameters (input var or expression)
e_i	::=	$\mathit{Value} \mid x \mid op(e_1, .., e_n)$	Expressions

The *input variables* in an action term are those marked with a *?*. We additionally impose that each input variable does not appear anywhere else in the same action term: $p_i = ?x \Rightarrow \forall j \neq i. x \notin \mathit{vars}(p_j)$. We define $iv(t)$ as the set of input variables of a term t (without the '?' marker). Input variables are used in guards and to update the local state, they can only appear in well-identified expressions. Action algebras can encode naturally usual point-to-point message passing calculi (using $a(?x_1, ..., ?x_n)$ for inputs, $a(v_1, .., v_n)$ for outputs), but they also allow for more general synchronisation mechanisms, like gate negotiation in Lotos, or broadcast communications.

The set of actions that do not distinguish input variables is denoted $\mathbb{A}_S$, it will be used in synchronisation vectors of pNets:

$$\alpha \in \mathbb{A}_S \quad ::= \quad a(e_1, \ldots, e_n)$$

Indexed sets. This article extensively uses indexed structures (maps) over some countable indexed sets. The indices can typically be integers, bounded or not. We use indexed sets in pNets because we want to consider a set of processes, and specify separately how to synchronise them. Roughly this could also be realised using tuples, however indexed sets are more general, can be infinite, and give a more compact representation than using the position in a possibly long tuple.

An indexed family is denoted as follows: $t_i^{i \in I}$ is a family of elements t_i indexed over the set I. Such a family is equivalent to the mapping $(i \mapsto t_i)^{i \in I}$, and we will also use mapping notations to manipulate indexed sets. To specify the set over which the structure is indexed, indexed structures are always denoted with an exponent of the form $i \in I$.

Consequently, $t_i^{i \in I}$ defines first I the set over which the family is indexed, and then t_i the elements of the family. For example $t_i^{i \in \{3\}}$ is the mapping with a single entry t_3 at index 3; exceptionally, for mappings with only a few entries we use the notation $(3 \mapsto t_3)$ instead. In this article, sentences of the form "there exists $t_i^{i \in I}$" means there exist I and a function that maps each element of I to a term t_i.

When this is not ambiguous, we shall abuse notations for sets, and typically write "indexed set over I" when formally we should speak of multisets, and

"$x \in A_i^{i\in I}$" to mean $\exists i \in I.\, x = A_i$. To simplify equations, an indexed set can be denoted $\overline{t}$ instead of $t_i^{i\in I}$ when I is irrelevant or clear from the context.

The disjoint union on sets is $\uplus$ and we only use $A \uplus B$ when A and B are disjoint. We extend it to union of indexed sets provided they are indexed over disjoint families; is $\uplus$ is then defined by the merge of the two sets. The elements of the union of two indexed sets are then accessed by using an index of one of the two joined families. The subtraction operation on indexed sets is $\setminus$, it reduces the set of indexes such that $\mathrm{dom}(A \setminus B) = \mathrm{dom}(A) \setminus B$.

Substitutions. This article also uses substitutions. Applying a substitution inside a term t is denoted $t\{\!\{y_i \leftarrow e_i\}\!\}^{i\in I}$ and consists in replacing in parallel all the occurrences of variables y_i in the term t by the terms e_i. Note that a substitution is defined by a partial function that is applied on the variables inside a term. We let *Post* range over partial functions that are used as substitution and use the notation $\{y_i \leftarrow e_i\}^{i\in I}$ to define such a partial function[2]. These partial functions are sometimes called *substitution functions* in the following. Thus, $\{\!\{Post\}\!\}$ is the operation that applies, in a parallel manner, the substitution defined by the partial function *Post*. $\odot$ is a composition operator on these partial functions, such that for any term t we have: $t\{\!\{Post \odot Post'\}\!\} = (t\{\!\{Post'\}\!\})\{\!\{Post\}\!\}$. This property must also be valid when the substitution does not operate on all variables. We thus define a composition operation as follows:

$$\begin{aligned}(x_k \leftarrow e_k)^{k\in K} \odot (x'_{k'} \leftarrow e'_{k'})^{k'\in K'} = &(x_k \leftarrow e_k\{\!\{(x'_{k'} \leftarrow e'_{k'})^{k'\in K'}\}\!\})^{k\in K} \\ &\cup (x'_{k'} \leftarrow e'_{k'})^{k'\in K''}\end{aligned}$$

where $K'' = \{k' \in K' | x'_{k'} \notin \{x_k\}^{k\in K}\}$

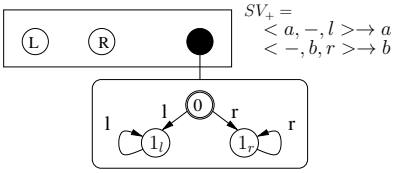
2.2. The principles of Parameterised Networks (pNets)

pNets are tree-like structures, where the leaves are either *parameterised labelled transition systems (pLTSs)*, expressing the behaviour of basic processes, or *holes*, used as placeholders for unknown processes. Every node of the tree is a pNet, it acts as a synchronising artefacts, using a set of *synchronisation vectors* that express the possible synchronisation between the parameterised actions of a subset of the sub-trees. The pNets model is hierarchical in the structure of the processes, in contrast to the Statecharts formalism [25], which is widely used to model high-level behaviour, that organises the states (but not processes) in a hierarchy.

We introduce the notion of pNets through a simple example below, and define formally pLTSs and pNets afterwards:

Example 1 (CCS choice). Here is the encoding of a choice operator.

[2]When using this notation, we suppose, without loss of generality that each y_i is different.



It consists of one pNet (Definition 2 below) with two holes and a sub-net. The pNet is represented by the top box with three circles and two synchronisation vectors on the right. The sub-net is a pLTS that is represented by the bottom box. Each hole is represented by an empty disc, when the hole is filled it becomes a black disc. The left hole is indexed L the right hole R. The sub-net is an labelled transition system (LTS) with three states and emitting actions l and r.

The behaviour of the pNet is defined with synchronisation vectors also shown on the figure. In the examples, we write them on the form $< a, -, l > \rightarrow a$. This states that if the first hole L performs the action a and the third sub-net, i.e. the LTS, performs the action l, both of them progress synchronously, and an action a is emitted by the pNet. The symbol $-$ at the second position denotes that the second hole does nothing. On the formal side, numbering and ordering the vectors is cumbersome, this is why we adopt indexed families of actions. The LTS is sometimes called the "control part", it controls the evolution of the rest of the pNet. The first action of one of the holes decides which branch of the LTS is activated; all subsequent actions will be performed by the same side.

2.3. Parameterised Labelled Transition systems (pLTS)

A pLTS is a labelled transition system with variables; variables can be used inside states, actions, guards, and assignments. Note that we make no assumption on finiteness of the set of states nor on finite branching of the transition relation. Compared to our previous works [30, 2] make variables global, which makes the model easier to use.

Definition 1 (pLTS). A pLTS is a tuple $pLTS \triangleq \langle\!\langle S, s_0, V, \rightarrow \rangle\!\rangle$ where:

- S is a set of states.
- $s_0 \in S$ is the initial state.
- V is a set of global variables for the pLTS.
- $\rightarrow \subseteq S \times L \times S$ is the transition relation and L is the set of labels. Labels have the form:
  $\langle \alpha,\ e_b,\ (x_j := e_j)^{j \in J} \rangle$, where $\alpha \in \mathbb{A}$ is a parameterised action, $e_b \in \mathbb{B}$ is a guard, and the variables x_j (that are pairwise distinct) are assigned the expressions $e_j \in \mathbb{E}$. If $s \xrightarrow{\langle \alpha,\ e_b,\ (x_j := e_j)^{j \in J} \rangle} s' \in \rightarrow$ then $vars(\alpha) \backslash iv(\alpha) \subseteq V$, $vars(e_b) \subseteq V \cup vars(\alpha)$, and $\forall j \in J.\ (vars(e_j) \subseteq V \cup iv(\alpha) \wedge x_j \in V)$.

A set of assignments between two states is performed in parallel so that their order do not matter and they all use the values of variables before the transition or the values received as action parameters.

2.4. Parameterised Networks (pNets)

Now we define pNet nodes as constructors for hierarchical behavioural structures. A pNet has a set of sub-pNets that can be either pNets or pLTSs, and a set of holes, playing the role of process parameters. A pNet is thus a composition operator that can receive processes as parameters; it expresses how the actions of the sub-processes synchronise.

Each sub-pNet exposes a set of actions, called *internal actions*. The synchronisation between global actions exposed by the pNet and internal actions of its sub-pNets is given by *synchronisation vectors*: a synchronisation vector synchronises one or several internal actions, and exposes a single resulting global action.

We now define the structure of pNets, the following definition relies on the definition of holes, leaves and sorts formalised below in Definition 3. Informally, holes are process parameters, leaves provide the set of pLTSs at the leaves of the hierarchical structure of a pNet, and sorts give the signature of a pNet, i.e. the actions it exposes.

Definition 2 (pNets). A pNet P is a hierarchical structure where leaves are pLTSs and holes

$$P \triangleq pLTS \mid \langle\!\langle P_i^{i\in I}, Sort_j^{j\in J}, SV_k^{k\in K} \rangle\!\rangle$$

We denote $vars(P)$ the set of variables used by the pLTSs inside P and $Sort(P)$ the signature of the actions emitted by P; both are defined below, in Definition 3. A pNet is composed of the following:

- I is a set of indices and $P_i^{i\in I}$ is the family of sub-pNets indexed over I. $vars(P_i)$ and $vars(P_j)$ must be disjoint for $i \neq j$.
- J is a set of indices, called *holes*. I and J are *disjoint*: $I \cap J = \emptyset$, $I \cup J \neq \emptyset$.
- $Sort_j \subseteq \mathbb{A}_S$ is a set of action terms, denoting the *sort* of hole j.
- $SV_k^{k\in K}$ is a set of synchronisation vectors.
  $\forall k \in K.\, SV_k = \alpha_l^{l\in I_k \uplus J_k} \rightarrow \alpha'_k[e_k]$ where $\alpha'_k \in \mathbb{A}_S$, $I_k \subseteq I$, $J_k \subseteq J$, $\forall i \in I_k.\, \alpha_i \in Sort(P_i)$, $\forall j \in J_k.\, \alpha_j \in Sort_j$, and $vars(\alpha'_k) \subseteq \bigcup_{l\in I_k \uplus J_k} vars(\alpha_l)$. The global action of a vector SV_k is α'_k. $e_k \in \mathbb{B}$ is a guard associated to the vector such that $vars(e_k) \subseteq \bigcup_{l\in I_k \uplus J_k} vars(\alpha_l)$.

Synchronisation vectors are identified modulo renaming of variables that appear in their action terms, e.g. the vectors $< a(x), b(x) > \rightarrow \tau$ and $< a(y), b(y) > \rightarrow \tau$ are equivalent.

The preceding definition relies on the auxiliary functions defined below:

Definition 3 (Sorts, holes, leaves, variables of pNets).

- The sort of a pNet is its signature, i.e. the set of actions in $\mathbb{A}_S$ it can perform, where each action signature is an action label plus the arity of the action.

$$\begin{array}{l}\mathrm{Sort}(\langle\!\langle S, s_0, V, \rightarrow \rangle\!\rangle) = \{\mathrm{Sort}(\alpha) | s \xrightarrow{\langle \alpha,\ e_b,\ (x_j := e_j)^{j\in J}\rangle} s' \in \rightarrow\} \\ \mathrm{Sort}(\langle\!\langle \overline{P}, \overline{Sort}, \overline{SV} \rangle\!\rangle) = \{\mathrm{Sort}(\alpha') | \, \overline{\alpha} \rightarrow \alpha'[e_b] \in \overline{SV}\} \\ \mathrm{Sort}(\alpha(p_1, .., p_n)) = (\alpha, n)\end{array}$$

- The set of variables of a pNet P, denoted $vars(P)$ is disjoint union the set of variables of all pLTSs that compose P.

- The set of holes $\mathrm{Holes}(P)$ of a pNet is the set of indices of the holes of the pNet itself plus the indices of all the holes of its sub-pNets. It is defined inductively (we suppose that those index sets are disjoints):

$$\begin{array}{l}\mathrm{Holes}(\langle\!\langle S, s_0, V, \rightarrow \rangle\!\rangle) = \emptyset \\ \mathrm{Holes}(\langle\!\langle P_i^{i\in I}, Sort_j^{j\in J}, \overline{SV} \rangle\!\rangle) = J \uplus \bigcup_{i\in I} \mathrm{Holes}(P_i) \\ \forall i \in I.\ \mathrm{Holes}(P_i) \cap J = \emptyset \\ \forall i_1, i_2 \in I.\, i_1 \neq i_2 \Rightarrow \mathrm{Holes}(P_{i_1}) \cap \mathrm{Holes}(P_{i_2}) = \emptyset\end{array}$$

- The set of leaves of a pNet is the set of all pLTSs occurring in the structure, as an indexed family of the form $\mathrm{Leaves}(P) = \langle\!\langle P_i \rangle\!\rangle^{i\in L}$.

$$\begin{array}{l}\mathrm{Leaves}(\langle\!\langle S, s_0, V, \rightarrow \rangle\!\rangle) = \emptyset \\ \mathrm{Leaves}(\langle\!\langle P_i^{i\in I}, \overline{Sort}, \overline{SV} \rangle\!\rangle) = \biguplus_{i\in I} \mathrm{Leaves}(P_i) \uplus \{i \mapsto P_i | P_i \text{ is a } pLTS\}\end{array}$$

For example, the controller of Example 1 has the sort $\{l, r\}$ and holes $\{L, R\}$. Note that $\mathrm{Holes}(P)$is a set of indexes because holes are characterized only by their indices, while entities at the leaves ar pLTSs and thus $\mathrm{Leaves}(P)$ is a set of pLTSs. A pNet Q is *closed* if it has no hole: $\mathrm{Holes}(Q) = \emptyset$; else it is said to be *open*. Sort comes naturally with a compatibility relation that is similar to a type-compatibility check. We simply say that two sorts are compatible if they consist of the same actions with the same arity. In practice, it is sufficient to check the equality of the two sets of action signatures of the two sorts[3].

The informal semantics of pNets is as follows. pLTSs behave more or less like classical automata with conditional branching and variables. The actions on the *pLTS*s can send or receive values, potentially modifying the value of variables. pNets are synchronisation entities: a pNet node composes several sub-pNets and defines how the sub-pNets interact, where a sub-pNet is either

[3]A more complex compatibility relation could be defined, but this is out of the scope of this article.

a pNet or a pLTS. The synchronisation between sub-pNets is defined by synchronisation vectors (originally introduced in [4]) that express how an action of a sub-pNet can be synchronised with actions of other sub-pNet, and how the resulting synchronised action is visible from outside of the pNet. The synchronisation mechanism is very expressive, including pattern-matching/unification between the parameterized actions within the vector, and an additional predicate over their variables. Consider a pNet node that assembles several pLTSs, the synchronisation vectors specify the way that transitions of the composed pNet are built from the transitions of the sub-nets. This can be seen as "conditional transitions" of the pNet, or alternatively, as a syntax to encode structural operational semantics (SOS rules) [44] of the system: each vector expresses not only the actions emitted by the pNet but also what transitions of the composed pLTSs must occur to trigger this global transition. Synchronisation vectors can also express the exportation of an action of a sub-pNet to the next level, or to hide an interaction and make it non-observable. Finally, a pNet can leave sub-pNets undefined and instead declare holes with a well-defined signature. Holes can then be filled with a sub-pNet. This is defined as follows.

Definition 4 (pNet composition). An open pNet: $P = \langle\!\langle P_i^{i\in I}, Sort_j^{j\in J}, \overline{SV} \rangle\!\rangle$ can be (partially) filled by providing a pNet Q to fill one of its holes. Suppose $j_0 \in J$ and $\mathrm{Sort}(Q) \subseteq Sort_{j_0}$, then:

$$P\,[Q]_{j_0} = \langle\!\langle P_i^{i\in I} \uplus \{j_0 \mapsto Q\}, Sort_j^{j\in J\setminus\{j_0\}}, \overline{SV} \rangle\!\rangle$$

pNets are composition entities equipped with a rich synchronisation mechanism: synchronisation vectors allow the expression of synchronisation between any number of entities and at the same time the passing of data between processes. Their strongest feature is that the data emitted by processes can be used inside the synchronisation vector to do addressing: it is easy to synchronise a process indexed by n with the action $a(v, n)$ of another process. This is very convenient to model systems and encode futures or message routing.

pNets have been used to model distributed components using the Grid Component Model, illustrating the expressiveness of the model [2]. These works show that pNets are convenient to express the behaviour of a system in a compositional way. Unfortunately, the semantics of pNets and the existing tools at that point were only able to deal with a closed and completely instantiated system: pNets could be used as composition operators in the definition of the semantics, which was sufficient to perform finite-state model checking on a closed system, but there was no theory for the use of pNets as operators and no tool for proving properties on open system. Consequently, much of the formalisation efforts did not use holes and the interplay between holes, sorts, and synchronisation vector was not formalised. In previous works [2], only closed pNets were equipped with a semantics, which was defined as labelled transition systems. The theory of pNets as operators for open systems is given in the following sections. Comparing formally the existing direct operational semantics and the semantics derived from open automata in the current article would be an interesting partial proof

of soundness for our semantics. The proof could only be partial as the formal semantics that exists only consider closed and fully instantiated pNets. Proving an equivalence between the semantics presented in this article and the operational one shown in [2] is outside the scope of this article because we focus here on the modelling of holes that were not considered in the previous semantics. It is however easy to see that, in case there is no hole the structure of the open automaton that defines the semantics here is very close to the pLTS that is used to define the semantics, even though the formalisms are slightly different.

2.5. Running example

To illustrate this work, we use a simple communication protocol, that provides safe transport of data between two processes, over unsafe media.

Figure 1 (left) shows the example principle, which corresponds to the hierarchical structure of a pNet: two unspecified processes P and Q (holes) communicate messages, with a data value argument, through the two protocol entities. Process P sends a `p-send(m)` message to the *Sender*; this communication is denoted as `in(m)`. At the other end, process Q receives the message from the *Receiver*. The holes P and Q can also have other interactions with their environment, represented here by actions `p-a` and `q-b`. The underlying network is modelled by a medium entity transporting messages from the sender to the receiver, and that is able to detect transport errors and signal them to the sender. The return *ack* message from *Receiver* to *Sender* is supposed to be safe. The final transmission of the message to the recipient (the hole Q) includes the value of the "error counter" *ec*.

Figure 1 (right) shows a graphical view of the pNet *SimpleProtocolSpec* that specifies the system. The pNet is made of the composition of two pNets: a *SimpleSystem* node, and a *PerfectBuffer* sub-pNet. The full system implementation should be equivalent (e.g. weakly bisimilar) to this *SimpleProtocolSpec*. The pNet has a tree-like structure. The root node of the tree *SimpleSystem* is the top level of the pNet structure. It acts as the parallel operator. It consists of three nodes: two holes P and Q and one sub-pNet, denoted *PerfectBuffer*. Nodes of the tree are synchronised using four synchronisation vectors, that express the possible synchronisations between the parameterised actions of a subset of the nodes. For instance, in the vector $<$ `p-send(m)`, `in(m)`, $\_$ $>\!\!\rightarrow$ `in(m)` only P and *PerfectBuffer* nodes are involved in the synchronisation. The synchronisation between these processes occurs when process P performs `p-send(m)` action sending a message, and the *PerfectBuffer* accepts the message through an `in(m)` action at the same time; the result that will be returned at upper level is the action `in(m)`.

Figure 2 shows the pNet model of the protocol implementation, called *SimpleProtocolImpl*. When the *Medium* detects an error (modelled by a local τ action), it sends back a `m-error` message to the *Sender*. The *Sender* increments its local error counter *ec*, and resends the message (including *ec*) to the *Medium*, that will, eventually, transmit m, ec to the *Receiver*.

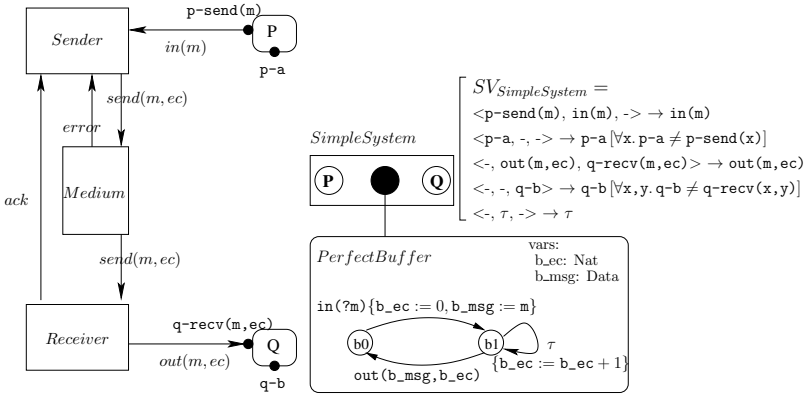


Figure 1: pNet structure of the example and its specification expressed as a pNet called *SimpleProtocolSpec*

3. A model of process composition

The semantics of open pNets will be defined as an open automaton. An open automaton is an automaton where each transition composes transitions of several LTSs with action of some holes, the transition occurs if some predicates hold, and can involve a set of state modifications. This section defines open automata and a bisimulation theory for them. This section is an improved version of the formalism described in [30], extending the automata with a notion of global variable, which makes the state of the automaton more explicit. We also adopt a semantics and logical interpretation of the automata that intuitively can be stated as follows: "if a transition belongs to an open automaton, any refinement of this transition also belongs to the automaton". Our open automata are clearly inspired by the work of De Simone on formatting of SOS rules [16]. A precise comparison with related works can be found in Section 6.

3.1. Open automata

Open automata (OA) are not composition structures but they are made of transitions that are dependent of the actions of the holes, and they can use variables (potentially with only symbolic values).

Definition 5 (Open transitions). An *open transition* (OT) over a set J of holes with sorts $Sort_j^{j \in J}$, a set V of variables, and a set of states $\mathcal{S}$ is a structure of the form:

$$\frac{\beta_j^{j \in J'}, Pred, Post}{s \xrightarrow{\alpha} s'}$$

where $J' \subseteq J$ is the set of holes involved in the transition; $s, s' \in \mathcal{S}$ are states of the automaton; and β_j is a transition of the hole j, with $\mathrm{Sort}(\beta_j) \subseteq Sort_j$. α is

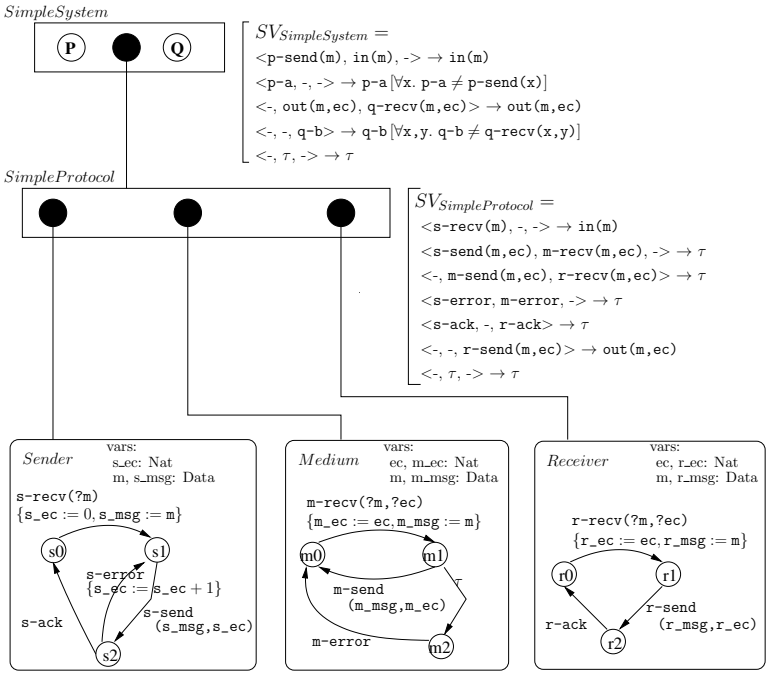


Figure 2: The *SimpleProtocolImpl* pNet resulting from the composition of the *SimpleSystem* and the *SimpleProtocol* pNets.

the resulting action of this open transition. *Pred* is a predicate, *Post* is a set of assignments that are effective after the open transition, and are represented as a substitution function: $(x_k \leftarrow e_k)^{k \in K}$. Predicates and expressions of an open transition can refer to the variables inside V and the different terms β_j and α. More precisely:

$$vars(Pred) \subseteq V \cup vars(\alpha) \cup \bigcup_{j \in J'} vars(\beta_j) \quad \wedge$$
$$\forall k.\, x_k \in V \quad \wedge \quad \forall k.\, vars(e_k) \subseteq V \cup vars(\alpha) \cup \bigcup_{j \in J'} vars(\beta_j)$$

The assignments are applied simultaneously because the variables in V can be in both sides (x_ks are distinct). Open transitions are identified modulo logical equivalence on their predicate.

It is important to understand the difference between the red dotted rule and a classical inference rule. They correspond to two different logical levels. On one side, classical (black) inference rules act at the mathematical level of the paper

proofs (as e.g. the rules in Definition 13). They use an expressive logic (like any other computer science article). On the other side, open transition rules (with dotted lines) are logical implications that belong to the open automata algebra. Their logic has a specific syntax that can be mechanized; this logic includes the boolean expressions $\mathbb{B}$, boolean operators, and term equality.

An open automaton is an automaton where transitions are open transitions.

Definition 6 (Open automaton). An *open automaton* is a structure $A = \langle\langle J, \mathcal{S}, s_0, V, \mathcal{T}\rangle\rangle$ where:

- J is a set of indices.
- $\mathcal{S}$ is a set of states and s_0 is an initial state belonging to $\mathcal{S}$.
- V is a set of variables of the automaton and each $v \in V$ may have an initial value $init(v)$.
- $\mathcal{T}$ is a set of open transitions and for each $t \in \mathcal{T}$ there exists J' with $J' \subseteq J$, such that t is an open transition over J' and $\mathcal{S}$.

While the definition and usage of the open transition can be considered purely syntactically, we take in this article a semantics and logical understanding of open automata. We see open transitions as logical formulas with a constrained syntax and logics rather than purely syntactical terms. Consequently, the open transition sets in open automata are closed by a simple form of refinement that allows us to refine the predicate, or substitute any free variable by an expression. Formally, for each predicate $Pred$ for each partial function $Post$, if $V \cap \mathrm{dom}(Post) = \emptyset$, we have:

$$\frac{\overline{\beta}, Pred', Post'}{s \xrightarrow{\alpha} s'} \in \mathcal{T} \quad \Longrightarrow \quad \frac{\overline{\beta}\{\!\{Post\}\!\}, Pred'\{\!\{Post\}\!\} \wedge Pred, Post \odot Post'}{s \xrightarrow{\alpha\{\!\{Post\}\!\}} s'} \in \mathcal{T}$$

Because of the semantic interpretation of open automata, the set of open transition of an open automaton is infinite (for example because every free variable can be substituted by any term). This raises an issue when a finite representation is needed, which is the case both in our tools, and when writing examples. When needed, we can rely on a *canonical representation* of the open automaton, provided that a finite subset of the open transitions is sufficient to generate, by substitution, the other ones. Thus, we use this canonical representation in our examples. In the following, we will abusively write that we define an "open automaton" when we provide its canonical representation.

Another aspect of the semantic interpretation is that we consider terms up to semantic equivalence, i.e. equivalence of two predicates $Pred$ and $Pred'$ can be denoted $Pred = Pred'$, where the $=$ symbol is interpreted semantically.

Though the definition is simple, the fact that transitions are complex structures relating events must not be underestimated. The first element of theory for open automata, i.e. the definition of a strong bisimulation, is given below.

3.2. Bisimulation for open Automata

We define now a bisimulation relation tailored to open automata and their parametric nature. This relation relates states of the open automata and guarantees that the related states are observationally equivalent, i.e. equivalent states can trigger transitions with identical action labels. Its key characteristics are 1) the introduction of predicates in the bisimulation relation: the relation between states may depend on the value of the variables; 2) bisimulation relates elements of the open transitions and takes into account predicates over variables, actions of the holes, and state modifications. We name it FH-bisimulation, as a short cut for the "Formal Hypotheses" over the holes behaviour manipulated in the transitions, but also as a reference to the work of De Simone [16], that pioneered this idea. Indeed, our definition uses both hypotheses on the behaviour of holes, as in [16], and symbolic manipulation of action expressions, as in symbolic bisimulations of [27].

One of the original aspects of FH-bisimulation is due to the symbolic nature of open automata. Indeed, a single state of the automaton represents a potentially infinite number of concrete states, depending on the value of the automaton variables, and a single open transition of the automaton may also be instantiated with an unbounded number of values for the transition parameters. Consequently it would be too restrictive to impose that each transition of one automaton is matched by exactly one transition of the bisimilar automaton. Thus the definition of bisimulation requires that, for each open transition of one automaton, there exists a matching set of open transitions covering the original one. Indeed depending on the value of action parameters or automaton variables, different open transitions might simulate the same one.

The parametric nature of the automata entails a second original aspect of FH-bisimulation: the nature of the bisimulation relation itself. A classical relation between states can be seen as a function mapping pairs of state to a boolean value (true if the states are related, false if they are not). An FH-bisimulation relation maps pairs of states to boolean expressions that use variables of the two systems. Formally, a relation over the states of two open automata $\langle\!\langle J, \mathcal{S}_1, s_0, V_1, \mathcal{T}_1 \rangle\!\rangle$ and $\langle\!\langle J, \mathcal{S}_2, t_0, V_2, \mathcal{T}_2 \rangle\!\rangle$ has the signature $\mathcal{S}_1 \times \mathcal{S}_2 \to \mathbb{B}$. We suppose without loss of generality that the variables of the two open automata are disjoint. We adopt a notation similar to standard relations and denote it $\mathcal{R} = \{(s, t | \mathit{Pred}_{s,t})\}$, where: 1) For any pair $(s, t) \in \mathcal{S}_1 \times \mathcal{S}_2$, there is a single $(s, t | \mathit{Pred}_{s,t}) \in \mathcal{R}$ stating that s and t are related if $\mathit{Pred}_{s,t}$ is True, i.e. the states are related when the value of the automata variables satisfy the predicate $\mathit{Pred}_{s,t}$. 2) The free variables of $\mathit{Pred}_{s,t}$ belong to V_1 and V_2, i.e. $\mathit{vars}(\mathit{Pred}_{s,t}) \subseteq V_1 \cup V_2$. FH-bisimulation is defined formally[4]:

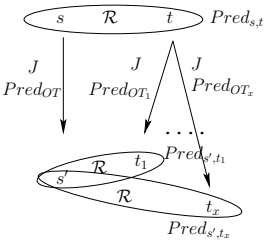
Definition 7 (Strong FH-bisimulation).
Suppose $A_1 = \langle\!\langle J, \mathcal{S}_1, s_0, V_1, \mathcal{T}_1 \rangle\!\rangle$ and $A_2 = \langle\!\langle J, \mathcal{S}_2, t_0, V_2, \mathcal{T}_2 \rangle\!\rangle$ are open automata with identical holes of the same sort, with disjoint sets of variables ($V_1 \cap V_2 = \emptyset$).

[4]In this article, we denote β_{jx} a double indexed set, instead of the classical $\beta_{j,\,x}$. Indeed the standard notation would be too heavy in our case.

Then $\mathcal{R}$ is an FH-bisimulation if and only if for all states $s \in \mathcal{S}_1$ and $t \in \mathcal{S}_2$, $(s, t|Pred_{s,t}) \in \mathcal{R}$, we have the following:

- For any open transition OT in $\mathcal{T}_1$:

$$\frac{\beta_j^{j \in J'}, Pred_{OT}, Post_{OT}}{s \xrightarrow{\alpha} s'}$$



  there exists an indexed set of open transitions $OT_x^{x \in X} \subseteq \mathcal{T}_2$:

$$\frac{\beta_{jx}^{j \in J_x}, Pred_{OT_x}, Post_{OT_x}}{t \xrightarrow{\alpha_x} t_x}$$

  such that $\forall x.\, J' = J_x$ and there exists some $Pred_{s',t_x}$ such that $(s', t_x|Pred_{s',t_x}) \in \mathcal{R}$ and

$$Pred_{s,t} \wedge Pred_{OT} \implies \bigvee_{x \in X} (\forall j. \beta_j = \beta_{jx} \wedge Pred_{OT_x} \wedge \alpha = \alpha_x \wedge Pred_{s',t_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\})$$

- and symmetrically any open transition from t in $\mathcal{T}_2$ can be covered by a set of transitions from s in $\mathcal{T}_1$.

Two open automata are FH-bisimilar if there exists an FH-bisimulation that relates their initial states[5]. We call this relation *FH-bisimilarity*.

Classically, $Pred_{s',t_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\}$ applies in parallel the substitution defined by the partial functions $Post_{OT}$ and $Post_{OT_x}$ (parallelism is crucial inside each $Post$ set but not between $Post_{OT}$ and $Post_{OT_x}$ that are independent), applying the assignments of the involved rules. We can prove that bisimilarity is an equivalence relation.

Example 2. The simulation of one transition by many others is one non-standard aspect of this definition. This is made necessary by the parameterised nature of our model. Consider the following open transition.

$$\frac{\overline{\beta}, True, \{\!\{y \leftarrow x\}\!\}}{s_1 \xrightarrow{\alpha(x)} s_1'}$$

Bisimulation should allow it to be matched by the two following ones (depending on the value of x), to prove that the relation $\mathcal{R} = \{(s_1, s_2, True), (s_1', s_2', True)\}$ is a bisimulation.

$$\frac{\overline{\beta}, x \geq 0, \{\!\{y \leftarrow x\}\!\}}{s_2 \xrightarrow{\alpha(x)} s_2'} \qquad \frac{\overline{\beta}, x < 0, \{\!\{y \leftarrow x\}\!\}}{s_2 \xrightarrow{\alpha(x)} s_2'}$$

[5] In other words, the predicate relation associated to the initial states is *True*.

This example illustrates the necessity of multiple transitions in the definition of bisimulation in a naive and minimalistic way. It can easily be extended into a non-trivial example with more states and different usage of the variables.

Theorem 1 (FH-bisimularity is an equivalence). *FH-bisimilarity is reflexive, symmetric and transitive.*

The proof of this theorem can be found in [3]. The only non-trivial part of the proof is the proof of transitivity. It relies on the following elements. First, the transitive composition of two relations with predicate is defined; this is not exactly standard as it requires to define the right predicate for the transitive composition and producing a single predicate to relate any two states. Then the fact that one open transition is simulated by a family of open transitions leads to a doubly indexed family of simulating open transition; this needs particular care, also because of the use of renaming ($Post$) when proving that the predicates satisfy the definition (property on $Pred_{s,t} \wedge Pred_{OT}$ in the definition).

Finite versus infinite open automata, and decidability

As mentioned in Definition 15, we adopt here a semantic view on open automata. More precisely, in [31], we define *semantic open automata* (infinite as in Definition 6), and *structural open automata* (finite) that can be generated as the semantics of pNets (see Definition 9), and used in their implementation. Then we define an alternative version of our bisimulation, called structural FH-bisimulation, based on structural open automata, and prove that the *semantic* and *structural* FH-bisimulations coincide. In the sequel, all mentions of finite automata, and algorithms for bisimulations, implicitly refer to their *structural* versions.

If we assume that everything is finite (states and transitions in the open automata), then it is easy to prove that it is decidable whether a relation is a FH-bisimulation, provided the logic of the predicates is decidable (a proof of this claim can be found in [30]). Formally:

Theorem 2 (Decidability of FH-bisimulation). *Let A_1 and A_2 be finite open automata and $\mathcal{R}$ a relation over their states $\mathcal{S}_1$ and $\mathcal{S}_2$ constrained by a set of predicates. Assume that the predicate inclusion is decidable over the action algebra $\mathbb{A}$. Then it is decidable whether the relation $\mathcal{R}$ is an FH-bisimulation.*

4. Semantics of Open pNets

This section defines the semantics of an open pNet via translation into an open automaton. In this translation, the states of the open automaton are obtained as products of the states of the pLTSs at the leaves of the composition. The predicates on the transitions are obtained both from the predicates on the transitions of the pLTSs, and from the synchronisation vectors involved in the transition.

The definition of bisimulation for open automata allows us to derive a bisimilarity relation for open pNets. As pNets are composition structures, it then makes sense to prove compositionality lemmas: we prove that the composition of strongly bisimilar pNets are themselves bisimilar.

4.1. Deriving an open automaton from an open pNet

To derive an open automaton from a pNet, we first describe the set of states of the automaton. Then we show the construction rule for transitions of the automaton, which relies on the derivation of predicates unifying synchronisation vectors and the actions of the pNets involved in a given synchronisation.

States of open pNets are tuples of states. We denote them as $\triangleleft \ldots \triangleright$ for distinguishing tuple states from other tuples.

Definition 8 (States of open pNets). A state of an open pNet is a (not necessarily finite) tuple of the states of its leaves.

For any pNet P, let $\text{Leaves}(P) = \langle\!\langle S_i, s_{i0}, V, \rightarrow_i \rangle\!\rangle^{i \in L}$ be the set of pLTS at its leaves, then $States(P) = \{\triangleleft s_i^{i \in L} \triangleright \,|\forall i \in L. s_i \in S_i\}$. A pLTS being its own single leave: $States(\langle\!\langle S, s_0, V, \rightarrow \rangle\!\rangle) = \{\triangleleft s \triangleright \,|s \in S\}$.

The initial state is defined as: $InitState(P) = \triangleleft {s_{i0}}^{i \in L} \triangleright$.

To be precise, the state of each pLTS is entirely characterized by both the state of the automaton, and the values of its variables V.

Predicates. We define a predicate $Pred_{sv}$ relating a synchronisation vector (of the form $(\alpha'_i)^{i \in I}, (\beta'_j)^{j \in J} \rightarrow \alpha'[e_b]$), the actions of the involved sub-pNets and the resulting actions. This predicate verifies:

$$Pred_{sv}\Big(\big((\alpha'_i)^{i \in I}, (\beta'_j)^{j \in J} \rightarrow \alpha'[e_b]\big), \alpha_i^{i \in I}, \beta_j^{j \in J}, \alpha\Big) \Leftrightarrow$$
$$\forall i \in I.\, \alpha_i = \alpha'_i \wedge \forall j \in J.\, \beta_j = \beta'_j \wedge \alpha = \alpha' \wedge e_b$$

Somehow, this predicate entails a verification of satisfiability in the sense that if the predicate $Pred_{sv}$ is not satisfiable, then the transition associated with the synchronisation will not occur in the considered state, or equivalently will occur with a *False* precondition. If the action families do not match or if there is no valuation of variables such that the above formula can be ensured then the predicate is undefined.

The definition of this predicate is not constructive. In our tool [46], we construct a logical formula encoding the matching and unification condition involved, and we let an SMT engine (in the current implementation Z3 [35]) decide its satisfiability.

Example 3 (An open-transition). At the upper level, the *SimpleSystem* pNet of Figure 2 has 2 holes and *SimpleProtocol* as a sub-pNet, itself containing 3 pLTSs. One of its possible open transitions (synchronizing the hole P with the *Sender* within the *SimpleProtocol*) is:

$$\frac{s \xrightarrow{\langle \alpha,\ e_b,\ (x_j := e_j)^{j\in J}\rangle} s' \in \rightarrow}{\langle\!\langle S, s_0, \rightarrow \rangle\!\rangle \models \dfrac{\emptyset, e_b, \{x_j \leftarrow e_j\}^{j\in J}}{\lhd s \rhd \xrightarrow{\alpha} \lhd s' \rhd}} \quad \textbf{Tr1}$$

and

$$\frac{\begin{array}{c}\text{Leaves}(\langle\!\langle P_m^{m\in I}, \overline{Sort}, SV_k^{k\in K}\rangle\!\rangle) = pLTS_l^{l\in L} \quad k \in K \\ SV_k = (\alpha'_m)^{m\in I_1 \uplus I_2 \uplus J} \rightarrow \alpha'[e_b] \\ \forall m \in I_1.\ P_m \models \dfrac{\beta_j^{j\in J_m}, Pred_m, Post_m}{\lhd s_i^{i\in L_m} \rhd \xrightarrow{\alpha_m} \lhd (s'_i)^{i\in L_m} \rhd} \\ \forall m \in I_2.\ P_m \models \dfrac{\emptyset, Pred_m, Post_m}{\lhd s_m \rhd \xrightarrow{\alpha_m} \lhd s'_m \rhd} \qquad J' = \biguplus_{m\in I_1} J_m \uplus J \\ Pred = \bigwedge_{m\in I_1 \uplus I_2} Pred_m \wedge Pred_{sv}(SV_k, \alpha_m^{m\in I_1\uplus I_2}, \beta_j^{j\in J}, \alpha) \\ \forall i \in L \setminus \left(\biguplus_{m\in I_1} L_m \uplus I_2\right).\ s'_i = s_i \qquad \text{fresh}(\alpha'_m, \alpha', \beta_j^{j\in J}, \alpha)\end{array}}{\langle\!\langle P_m^{m\in I}, \overline{Sort}, SV_k^{k\in K}\rangle\!\rangle \models \dfrac{\beta_j^{j\in J'}, Pred, \biguplus_{m\in I_1\uplus I_2} Post_m}{\lhd s_i^{i\in L} \rhd \xrightarrow{\alpha} \lhd (s'_i)^{i\in L} \rhd}} \quad \textbf{Tr2}$$

Figure 3: Rules **Tr1** and **Tr2** defining the semantics of open pNets

$$OT_1 = \frac{\{\texttt{P}\mapsto\texttt{p-send(m)}\}, [\texttt{m=m'}], (\texttt{s\_msg} \leftarrow \texttt{m})}{\lhd s_0, m_0, r_0 \rhd \xrightarrow{\texttt{in(m')}} \lhd s_1, m_0, r_0 \rhd}$$

The global states here are triples, the product of states of the 3 pLTSs (holes have no state). The assignment performed by the open transition uses the variable `m` from the action of hole `P` to set the value of the sender variable named `s_msg`.

We build the semantics of open pNets as an open automaton over the states given by Definition 8. The open transitions first project the global state into states of the leaves, then apply pLTS transitions on these states, and compose them with the sort of the holes. The semantics instantiates fresh variables using the predicate $\text{fresh}(x)$, additionally, for an action α, $\text{fresh}(\alpha)$ means all variables in α are fresh.

Definition 9 (Semantics of open pNets). The semantics of a pNet P is an open automaton $A = \langle\!\langle Holes(P), States(P), InitState(P), vars(P), \mathcal{T}\rangle\!\rangle$ where $\mathcal{T}$ is the smallest set of open transitions such that $\mathcal{T} = \{OT \mid P \models OT\}$ and $P \models OT$ is defined by the rules in Figure 4.1

- The rule **Tr1** for a pLTS checks that the guard is verified and transforms assignments into post-conditions.
- The rule **Tr2** deals with pNet nodes: for each possible synchronisation vector (of index k) applicable to the rule subject, the premises include one *open transition* for each sub-pNet involved, one possible *action* for each hole involved, and the predicate relating these with the resulting action of the vector. The sub-pNets involved are split between two sets, I_2 for sub-pNets that are pLTSs (with open transitions obtained by rule **Tr1**), and I_1 for the sub-pNets that are not pLTSs (with open transitions obtained by rule **Tr2**), J is the set of holes involved in the transition[6][7].

A key to understand **Tr2** is that the open transitions are expressed in terms of the leaves and holes of the whole pNet structure, i.e. a flattened view of the pNet. For example, L is the index set of the Leaves, L_m the index set of the leaves of one sub-pNet indexed m, so all L_m are disjoint subsets of L. Thus the states in the open transitions, at each level, are tuples including states of all the leaves of the pNet, not only those involved in the chosen synchronisation vector.

Note that the construction is symbolic, and each open transition deduced expresses a whole family of behaviours, for any possible value of the variables.

In [30], we have shown a detailed example of the construction of a complex open transition, building a deduction tree using rules **Tr1** and **Tr2**. We have also shown in [30] that an open pNet with finite synchronisation sets, finitely many leaves and holes, and each pLTS at leaves having a finite number of states and (symbolic) transitions, induces a finite automaton. The algorithm for building such an automaton can be found in [45].

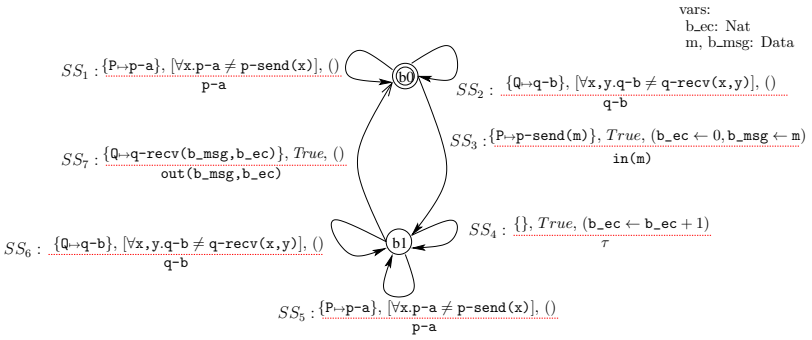


Figure 4: Open automaton for *SimpleProtocolSpec*

[6]Formally, if $SV_k = (\alpha')_m^{m \in M} \rightarrow \alpha'[e_b]$ is a synchronisation vector of P then $J = M \cap \text{Holes}(P)$, $I_2 = M \cap \text{Leaves}(P)$, $I_1 = M \setminus J \setminus I_2$

[7]We could replace I_1 and I_2 by their formal definition in **Tr2** but the rule would be more difficult to read.

Example

Figure 4 shows the open automaton computed from the *SimpleProtocolSpec* pNet given in Figure 1. For later references, we name SS_i the transitions of this (strong) specification automaton while transitions of the *SimpleProtocolImpl* pNet are labelled SI_i. In the figures we annotate each open automaton with the set of its variables.

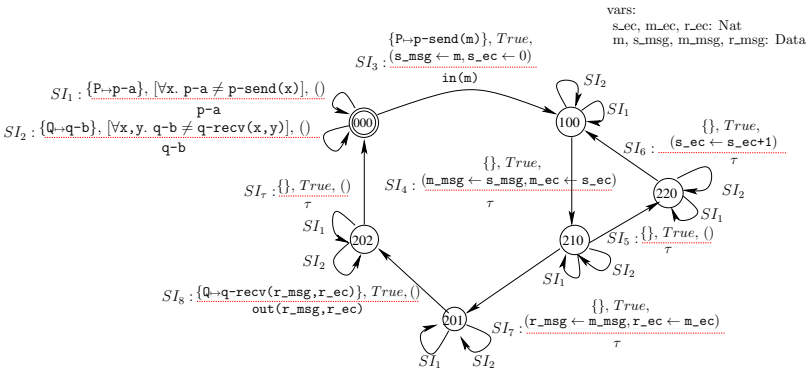


Figure 5: Open automaton for *SimpleProtocolImpl*

Figure 5 shows the open automaton of *SimpleProtocolImpl* from Figure 2. In this drawing, we have short labels for states, representing $\triangleleft s_0, m_0, r_0 \triangleright$ by 000. Note that open transitions are denoted $\mathtt{SI}_i$ and tau open transition by $\mathtt{SI}_\tau$. The resulting behaviour is quite simple: we have a main loop including receiving a message from P and transmitting the same message to Q, with some intermediate τ actions from the internal communications between the protocol processes. In most of the transitions, you can observe that data is propagated between the successive pLTS variables (holding the message, and the error counter value). On the right of the figure, there is a loop of τ actions ($\mathtt{SI}_4$, $\mathtt{SI}_5$ and $\mathtt{SI}_6$) showing the handling of errors and the incrementation of the error counter.

4.2. pNet Composition Properties: composition of open transitions

The semantics of open pNets allows us to prove two crucial properties relating pNet composition with pNet semantics: open transition of a composed pNet can be decomposed into open transitions of its composing sub-pNets, and conversely, from the open transitions of sub-pNets, an open transition of the composed pNet can be built.

We start with a decomposition property: from one open transition of $P[Q]_{j_0}$, we exhibit corresponding behaviours of P and Q, and determine the relation between their predicates.

Lemma 1 (Open transition decomposition). *Consider two pNets P and Q that are not pLTSs*[8]*. Let* $\mathrm{Leaves}(Q) = p_l^{l\in L_Q}$ *and suppose:*

$$P[Q]_{j_0} \models \frac{\beta_j^{j\in J}, \mathit{Pred}, \mathit{Post}}{\lhd s_i^{i\in L} \rhd \xrightarrow{\alpha} \lhd s_i'^{i\in L} \rhd}$$

with $J \cap \mathrm{Holes}(Q) \neq \emptyset$ *or* $\exists i \in L_Q.\, s_i \neq s_i'$*, i.e. Q takes part in the reduction. Then there exist α_Q, Pred', Pred'', Post', Post'' s.t.:*

$$P \models \frac{\beta_j^{j\in(J\setminus\mathrm{Holes}(Q))\cup\{j_0\}}, \mathit{Pred}', \mathit{Post}'}{\lhd s_i^{i\in L\setminus L_Q} \rhd \xrightarrow{\alpha} \lhd s_i'^{i\in L\setminus L_Q} \rhd}$$

$$\textit{and} \qquad Q \models \frac{\beta_j^{j\in J\cap\mathrm{Holes}(Q)}, \mathit{Pred}'', \mathit{Post}''}{\lhd s_i^{i\in L_Q} \rhd \xrightarrow{\alpha_Q} \lhd s_i'^{i\in L_Q} \rhd}$$

and $\mathit{Pred} \iff \mathit{Pred}' \wedge \mathit{Pred}'' \wedge \alpha_Q = \beta_{j_0}$*,* $\mathit{Post} = \mathit{Post}' \uplus \mathit{Post}''$ *where Post'' is the restriction of Post over variables of Q.*

Lemma 2 is combining an open transition of P with an open transition of Q, and building a corresponding transition of $P[Q]_{j_0}$ by assembling their elements.

Lemma 2 (Open transition composition). *Suppose $j_0 \in J$ and:*

$$P \models \frac{\beta_j^{j\in J}, \mathit{Pred}, \mathit{Post}}{\lhd s_i^{i\in L} \rhd \xrightarrow{\alpha} \lhd s_i'^{i\in L} \rhd} \qquad \textit{and} \qquad Q \models \frac{\beta_j^{j\in J_Q}, \mathit{Pred}', \mathit{Post}'}{\lhd s_i^{i\in L_Q} \rhd \xrightarrow{\alpha_Q} \lhd s_i'^{i\in L_Q} \rhd}$$

Then, we have:

$$P[Q]_{j_0} \models \frac{\beta_j^{(j\in J\setminus\{j_0\})\uplus J_Q}, \mathit{Pred} \wedge \mathit{Pred}' \wedge \alpha_Q = \beta_{j_0}, \mathit{Post} \uplus \mathit{Post}'}{\lhd s_i^{i\in L\uplus L_Q} \rhd \xrightarrow{\alpha} \lhd s_i'^{i\in L\uplus L_Q} \rhd}$$

Note that this does not mean that any two pNets can be composed and produce an open transition. Indeed, the predicate $\mathit{Pred} \wedge \mathit{Pred}' \wedge \alpha_Q = \beta_{j_0}$ is often not satisfiable, in particular if the action α_Q cannot be matched with β_{j_0}. Note also that β_{j_0} is only used as an intermediate term inside formulas in the composed open transition: it does not appear as global action, and will not appear as an action of a hole.

4.3. Bisimulation for open pNets – a composable bisimulation theory

As our symbolic operational semantics provides an open automaton, we can apply the notion of strong (symbolic) bisimulation on automata to open pNets.

Definition 10 (FH-bisimulation for open pNets). Two pNets are FH-bisimilar if their associated open automata are bisimilar.

[8] A similar lemma can be proven for a pLTS Q

We can now prove that pNet composition preserves FH-bisimilarity. More precisely, one can define two preservation properties, namely 1) when one hole of a pNet is filled by two bisimilar other (open) pNets; and 2) when the same hole in two bisimilar pNets are filled by the same pNet, in other words, composing a pNet with two bisimilar contexts. The general case will be obtained by transitivity of the bisimilarity relation (Theorem 1).

Theorem 3 (Congruence). *Consider an open pNet* $P = \langle\!\langle P_i^{i\in I}, Sort_j^{j\in J}, \overline{SV}\rangle\!\rangle$. *Let* $j_0 \in J$ *be a hole. Let* Q *and* Q' *be two FH-bisimilar pNets such that*[9] $\mathrm{Sort}(Q) = \mathrm{Sort}(Q') = Sort_{j_0}$. *Then* $P[Q]_{j_0}$ *and* $P[Q']_{j_0}$ *are FH-bisimilar.*

Theorem 4 (Context equivalence). *Consider two open pNets* $P = \langle\!\langle P_i^{i\in I}, Sort_j^{j\in J}, \overline{SV}\rangle\!\rangle$ *and* $P' = \langle\!\langle {P'}_i^{i\in I}, Sort_j^{j\in J}, \overline{SV'}\rangle\!\rangle$ *that are FH-bisimilar (they thus have the same holes). Let* $j_0 \in J$ *be a hole, and* Q *be a pNet such that* $\mathrm{Sort}(Q) = Sort_{j_0}$. *Then* $P[Q]_{j_0}$ *and* $P'[Q]_{j_0}$ *are FH-bisimilar.*

Finally, the previous theorems can be composed to state a general theorem about composability and FH-bisimilarity.

Theorem 5 (Composability). *Consider two FH-bisimilar pNets with an arbitrary number of holes, when replacing, inside those two original pNets, a subset of the holes by FH-bisimilar pNets, we obtain two FH-bisimilar pNets.*

This theorem is quite powerful, as it somehow implies that the theory of open pNets can be used to study properties of process composition. Open pNets can indeed be applied to study process operators and process algebras, as shown in [30] where compositional properties are extremely useful. In the case of interaction protocols [13], compositionality of bisimulation can justify abstractions used in some parts of the application.

5. Weak bisimulation

Weak symbolic bisimulation [26] was introduced to relate transition systems that have indistinguishable behaviour, with respect to some definition of *internal actions* that are considered local to some subsystem, and consequently cannot be observed, nor used for synchronisation with their context. The notion of non-observable actions varies in different contexts, e.g. *tau* in CCS [42, 43], and *i* in Lotos [11]. We could define classically a set of *internal/non-observable actions* depending on a specific action algebra. However in this paper, to simplify the notations, we will simply use τ as the single non-observable action; the generalisation of our results to a set of non-observable actions is trivial. Naturally, a non-observable action cannot be synchronised with actions of other

[9] Note that $\mathrm{Sort}(Q) = \mathrm{Sort}(Q')$ is ensured by strong bisimilarity.

systems in its environment. We show here that under such assumption of non-observability of τ actions, see Definition 11, we can define a weak bisimulation relation that is compositional, in the sense of open pNet composition. In this section we will first define a notion of weak open transition similar to open transition. In fact a weak open transition is made of several open transitions labelled as non-observable transitions, plus potentially one observable open transition. This allows us to define weak open automata, and a weak bisimulation relation based on these weak open automata. Finally, we apply this weak bisimulation to open pNets, obtain a weak bisimilarity relation for open pNets, and prove that this relation has compositional properties.

5.1. Preliminary definitions and notations

We first specify in terms of open transition, what it means for an action to be non-observable. We first define (in Definition 11) systems that cannot observe τ actions of sub-systems; namely pNets that cannot change their state, or emit an observable action when one of its holes emits a τ action.

More precisely, we state that τ is not observable if the automaton always allows any τ transition from holes, and additionally the global transition resulting from a τ action of a hole is a τ transition not changing the pNet's state. We define $\mathrm{Id}(V)$ as the identity function on the set of variables V.

Definition 11 (Non-observability of τ actions for open automata). An open automaton $A = \langle\!\langle J, \mathcal{S}, s_0, V, \mathcal{T} \rangle\!\rangle$ *cannot observe* τ *actions* if and only if for all j in J and s in $\mathcal{S}$ we have:

1. $$\frac{(j \mapsto \tau),\ \mathit{True},\ \mathrm{Id}(V)}{s \xrightarrow{\tau} s} \in \mathcal{T}$$

   and

2. for all β_j, J, α, s, s', *Pred*, *Post* such that

   $$\frac{\beta_j^{j \in J},\ \mathit{Pred},\ \mathit{Post}}{s \xrightarrow{\alpha} s'} \in \mathcal{T}$$

   If there exists j such that $\beta_j = \tau$ then we have:

   $$\alpha = \tau \wedge s = s' \wedge \mathit{Pred} = \mathit{True} \wedge \mathit{Post} = \mathrm{Id}(V) \wedge J = \{j\}$$

The first statement of the definition states that the open automaton must allow a hole to do a silent action at any time, and must not observe it, i.e. it cannot change its internal state because a hole did a τ transition. The second statement ensures that there cannot be in the open automaton other transitions that would be able to observe a τ action from a hole: statement (2) states that all the open transitions where a hole does a τ action must be of the shape given in statement (1). In this second statement, the condition $J = \{j\}$ is a bit restrictive, it could

safely be replaced by $\forall j \in J. \beta_j = \tau$, allowing the other holes to perform τ transitions too (because these τ actions cannot be observed). This possible synchronisation of τ actions would not be a problem as condition 1 still ensures that each process can do a τ separately.

By definition, one weak open transition contains several open transitions, where each open transition can require an observable action from a given hole, the same hole might have to emit several observable actions for a single weak open transition to occur. Consequently, for a weak open transition to trigger, a sequence of actions from a given hole may be required.

Thus, we let γ range over sequences of action terms and use $\oplus$ as the concatenation operator that appends sequences of action terms: given two sequences of action terms $\gamma \oplus \gamma'$ concatenates the two sequences. The operation is lifted to indexed sets of sequences: at each index i, $\overline{\gamma_1} \oplus \overline{\gamma_2}$ concatenates the sequences of actions at index i of $\overline{\gamma_1}$ and the one at index i of $\overline{\gamma_2}$[10]. $[a]$ denotes a sequence with a single element.

As required actions are now sequences of observable actions, we need an operator to build them from set of actions that occur in open transitions, i.e. an operator that takes a set of actions performed by one hole and produces a sequence of observable actions.

Thus we define $(\overline{\beta})^{\nabla}$ as the mapping $\overline{\beta}$ with only observable actions of the holes in I, but where each element is either empty or a list of length 1:

$$(\beta_i^{i \in I})^{\nabla} = [\beta_i]^{i \in I'} \text{ where } I' = \{i | i \in I \wedge \beta_i \neq \tau\}$$

As an example the $(\overline{\beta})^{\nabla}$ built from the transition OT_1 in Example 3, page 19 is $\texttt{P}\mapsto[\texttt{p-send(m)}]$. Remark that in our simple example no τ transition involves any visible action from a hole, so we have no β sequences of length longer than 1 in the weak automaton.

5.2. Weak open transition definition

Because of the non-observability property (Definition 11), it is possible to add any number of τ transitions of the holes before or after any open transition freely. This property justifies the fact that we can abstract away from τ transitions from holes in the definition of a weak open transition. We define weak open transitions similarly to open transitions except that holes can perform *sequences of observable actions* instead of single actions (observable or not). Compared to the definition of open transition, this small change has a significant impact as a single weak transition is the composition of several transitions of the holes.

Definition 12 (Weak open transition (WOT)). A weak open transition over a set J of holes with sorts $Sort_j^{j \in J}$ and a set of states $\mathcal{S}$ is a structure of the form:

$$\frac{\gamma_j^{j \in J'}, Pred, Post}{s \overset{\alpha}{\Rightarrow} s'}$$

[10] One of the two sequences is empty when $i \notin \text{dom}(\overline{\gamma_1})$ or $i \notin \text{dom}(\overline{\gamma_2})$.

$$\frac{}{\underset{s \overset{\tau}{\Rightarrow} s}{\emptyset, \mathit{True}, \mathrm{Id}(V)} \in \mathcal{WT}}\ \textbf{WT1} \qquad \text{and} \qquad \frac{\underset{s \overset{\alpha}{\rightarrow} s'}{\overline{\beta}, \mathit{Pred}, \mathit{Post}} \in \mathcal{T}}{\underset{s \overset{\alpha}{\Rightarrow} s'}{(\overline{\beta})^{\nabla}, \mathit{Pred}, \mathit{Post}} \in \mathcal{WT}}\ \textbf{WT2}$$

and

$$\frac{\underset{s \overset{\tau}{\Rightarrow} s_1}{\overline{\gamma_1}, \mathit{Pred}_1, \mathit{Post}_1} \in \mathcal{WT} \qquad \underset{s_1 \overset{\alpha}{\Rightarrow} s_2}{\overline{\gamma_2}, \mathit{Pred}_2, \mathit{Post}_2} \in \mathcal{WT} \qquad \underset{s_2 \overset{\tau}{\Rightarrow} s'}{\overline{\gamma_3}, \mathit{Pred}_3, \mathit{Post}_3} \in \mathcal{WT} \qquad \overline{\gamma} = \overline{\gamma_1} \oplus \overline{\gamma_2}\{\!\{\mathit{Post}_1\}\!\} \oplus \overline{\gamma_3}\{\!\{\mathit{Post}_2 \odot \mathit{Post}_1\}\!\} \quad \alpha' = \alpha\{\!\{\mathit{Post}_1\}\!\} \quad \mathit{Pred} = \mathit{Pred}_1 \wedge \mathit{Pred}_2\{\!\{\mathit{Post}_1\}\!\} \wedge \mathit{Pred}_3\{\!\{\mathit{Post}_2 \odot \mathit{Post}_1\}\!\}}{\underset{s \overset{\alpha'}{\Rightarrow} s'}{\overline{\gamma}, \mathit{Pred}, \mathit{Post}_3 \odot \mathit{Post}_2 \odot \mathit{Post}_1} \in \mathcal{WT}}\ \textbf{WT3}$$

Figure 6: Weak transition definition

Where $J' \subseteq J$, $s, s' \in \mathcal{S}$ and γ_j is a list of transitions of the hole j, with each element of the list in Sort_j. α is an action label denoting the resulting action of this open transition. *Pred* and *Post* are defined similarly to Definition 5. We use $\mathcal{WT}$ to range over sets of weak open transitions.

A weak open automaton $\langle\!\langle J, \mathcal{S}, s_0, V, \mathcal{WT} \rangle\!\rangle$ is similar to an open automaton except that $\mathcal{WT}$ is a set of weak open transitions over J and $\mathcal{S}$.

A weak open transition labelled α can be seen as a sequence of open transitions that are all labelled τ except one that is labelled α; however conditions on predicates, effects, and states must be verified for this sequence to be fired.

We are now able to build a weak open automaton from an open automaton. This is done in a way that resembles the process of τ saturation: we add τ open transitions before or after another open transition, regardless of whether it is observable or not.

Definition 13 (Building a weak open automaton).
Let $A = \langle\!\langle J, \mathcal{S}, s_0, V, \mathcal{T} \rangle\!\rangle$ be an open automaton. The weak open automaton *derived* from A is an open automaton $\langle\!\langle J, \mathcal{S}, s_0, V, \mathcal{WT} \rangle\!\rangle$ where $\mathcal{WT}$ is derived from $\mathcal{T}$ by saturation, applying the rules of Figure 6.

Rule **WT1** states that it is always possible to perform a non-observable transition, where the state is unchanged and the holes perform no action. Rule **WT2** states that each open transition is a weak open transition. Finally, Rule **WT3** allows any number of τ transitions before or after any weak open transition. This rule carefully composes predicates, effects, and actions of the holes. Indeed, predicate Pred_2 manipulates variables of s_1 that result from the first weak

open transition. Their values thus depend on the initial state but also on the effect (as a substitution function $Post_1$) of the first weak open transition. In the same manner, $Pred_3$ must be applied the substitution defined by the composition $Post_2 \odot Post_1$. Similarly, effects on variables must be applied to obtain the global effect of the composed weak open transition, to observable actions of the holes, and to the global action of the weak open transition.

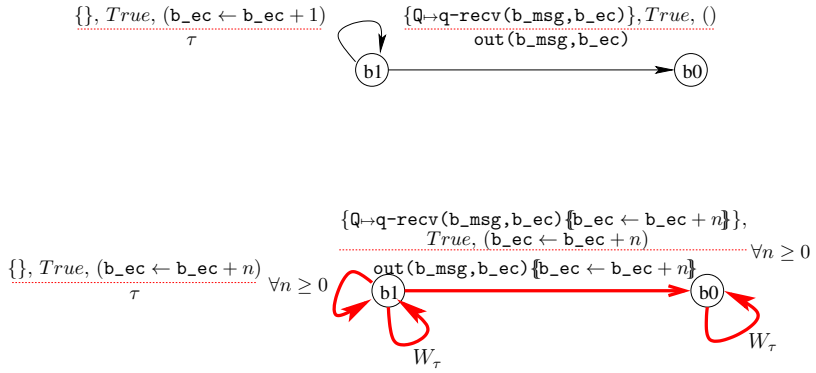


Figure 7: Construction of an example of weak open transition

Example 4 (A weak open-transition). Figure 7 shows the construction of one of the weak transitions of the open automaton of *SimpleProtocolSpec*. On the top we show the subset of the original open automaton (from Figure 4) considered here, and at the bottom the generated weak transition. For readability, we abbreviate the weak open transitions encoded by $\frac{\{\}, True, ()}{s \stackrel{\tau}{\Rightarrow} s'}$ as W_τ. The weak open transition shown here is the transition delivering the result of the algorithm to hole Q by applying rules: **WT1**,**WT2**, and **WT3**. First rule **WT1** adds a WT_τ loop on each state. Rule **WT2** transforms each 2 OTs into WOTs. Then consider application of Rule **WT3** on a sequence of 3 WOTs. $\frac{\{\}, True, (\texttt{b\_ec} \leftarrow \texttt{b\_ec} + 1)}{\texttt{b1} \stackrel{\tau}{\Rightarrow} \texttt{b1}}$; $\frac{\{\}, True, (\texttt{b\_ec} \leftarrow \texttt{b\_ec} + 1)}{\texttt{b1} \stackrel{\tau}{\Rightarrow} \texttt{b1}}$; $\frac{\{\}, True, ()}{\texttt{b1} \stackrel{\tau}{\Rightarrow} \texttt{b1}}$. The result will be: $\frac{\{\}, True, (\texttt{b\_ec} \leftarrow \texttt{b\_ec} + 2)}{\texttt{b1} \stackrel{\tau}{\Rightarrow} \texttt{b1}}$. We can iterate this construction an arbitrary number of times, getting for any natural number n a weak open transition: $\frac{\emptyset, True, (\texttt{ec} \leftarrow \texttt{ec} + n)}{\texttt{b1} \stackrel{\tau}{\Rightarrow} \texttt{b1}} \forall n \geq 0$. Finally, applying again **WT3**, and using the central open transition having `out(b_msg,b_ec)` as α, we get the resulting weak open transition between b1 and b0 (as shown in Figure 7). Applying the substitutions finally yields the weak transitions family WS_7 in Figure 8.

Example 5 (Weak open automata). Figures 8 and 9 respectively show the

weak automata of *SimpleProtocolSpec* and *SimpleProtocolImpl*. We encode weak open transitions by WS on the specification model and by WI on the implementation model.

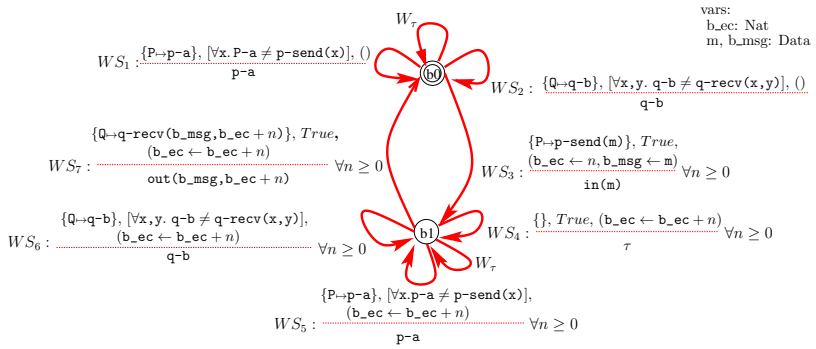


Figure 8: Weak Open Automaton of *SimpleProtocolSpec*

For readability, we only give names to the weak open transitions of *SimpleProtocolImpl* in Figure 9; we detail some of these transitions below and the full list is included in the extended version [3] . Let us point out that the weak OT loops (WI_1,WI_2 and W_τ) on state 000 are also present in all other states, we did not repeat them. Additionally, many WOTs are similar, and numbered accordingly as 3, 3a, 3b, 3c and 8, 8a, 8b, 8c respectively: they only differ by their respective source or target states; the "variant" WOTs appear in blue in Figure 9.

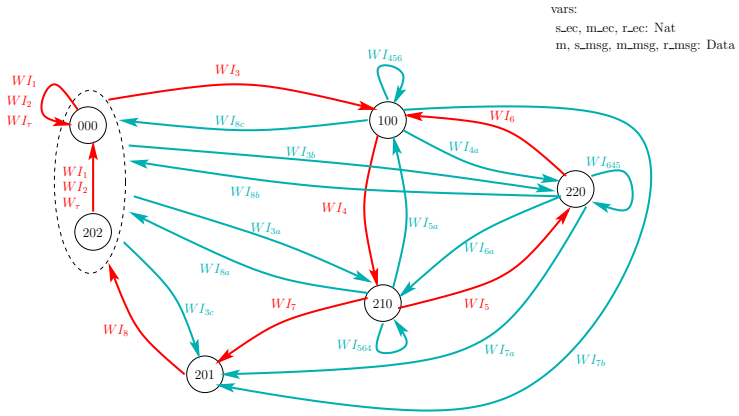


Figure 9: Weak Open Automaton of *SimpleProtocolImpl*

Now let us give some details about the construction of the weak automaton of the *SimpleProtocolImpl* pNet, obtained by application of the weak rules as explained above. We concentrate on weak open transitions WI_3 and WI_4. Let us denote as $post_n$ the effect (as a substitution function) of the strong open transitions SI_n from Figure 5:

$post_3 = (\texttt{s\_msg} \leftarrow \texttt{m}, \texttt{s\_ec} \leftarrow 0)$
$post_4 = (\texttt{m\_msg} \leftarrow \texttt{s\_msg}, \texttt{m\_ec} \leftarrow \texttt{s\_ec})$
$post_5 = ()$
$post_6 = (\texttt{s\_ec} \leftarrow \texttt{s\_ec+1})$

Then the effect of one single $100 \xrightarrow{OT_4} 210 \xrightarrow{OT_5} 220 \xrightarrow{OT_6} 100$ loop is[11]:

$$post_{456} = post_6 \odot post_5 \odot post_4 = (\texttt{s\_ec} \leftarrow \texttt{s\_ec} + 1)$$

So if we denote $post_{456*}$ any iteration of this loop, we get $post_{456*} = (\texttt{s\_ec} \leftarrow \texttt{s\_ec} + n)$ for any $n \geq 0$, and the *Post* of the weak OT WI_3 is:
$Post_3 = post_{456*} \odot post_3 = (\texttt{s\_msg} \leftarrow m, \texttt{s\_ec} \leftarrow n), \forall n \geq 0$ and *Post* of WI_{3a} is:
$post_4 \odot post_{456*} \odot post_3 = (\texttt{m\_msg} \leftarrow m, \texttt{m\_ec} \leftarrow n), \forall n \geq 0$.

We can now show some of the weak OTs of Figure 9 (the full table is included in the extended version [3]). As we have seen above, the effect of rule WT_3 when a silent action have an effect on the variable *ec* will generate an infinite family of WOTs, depending on the number of iterations through the loops. We denote these families using a "meta-variable" n, ranging over `Nat`.

$$WI_1 = \frac{\{\texttt{P} \mapsto \texttt{p-a}\}, [\forall \texttt{x}.\texttt{p-a} \neq \texttt{p-send(x)}], ()}{s \xRightarrow{\texttt{p-a}} s} \text{ (for any } s \in S)$$

$$\forall n \geq 0.\ WI_3(n) = \frac{\{\texttt{P} \mapsto \texttt{p-send(m)}\},\ True,\ (\texttt{s\_msg} \leftarrow \texttt{m}, \texttt{s\_ec} \leftarrow n)}{000 \xRightarrow{\texttt{in(m)}} 100}$$

$$\forall n \geq 0.\ WI_4(n) = \frac{\{\},\ True,\ (\texttt{m\_msg} \leftarrow \texttt{s\_msg}, \texttt{m\_ec} \leftarrow \texttt{s\_ec} + n, \texttt{s\_ec} \leftarrow \texttt{s\_ec} + n)}{100 \xRightarrow{\tau} 210}$$

$$\forall n \geq 0.\ WI_{456}(n) = \frac{\{\},\ True,\ (\texttt{s\_ec} \leftarrow \texttt{s\_ec} + n)}{100 \xRightarrow{\tau} 100}$$

The *Post* of the weak OT WI_{6a} is:

$$\begin{aligned} Post_{6a} &= post_4 \odot post_{456*} \odot post_6 \\ &= (\texttt{m\_msg} \leftarrow \texttt{s\_msg}, \texttt{m\_ec} \leftarrow \texttt{s\_ec}) \odot (\texttt{s\_ec} \leftarrow \texttt{s\_ec} + n) \odot (\texttt{s\_ec} \leftarrow \texttt{s\_ec} + 1) \\ &= (\texttt{m\_msg} \leftarrow \texttt{s\_msg}, \texttt{m\_ec} \leftarrow \texttt{s\_ec} + 1 + n, \texttt{s\_ec} \leftarrow \texttt{s\_ec} + 1 + n) \end{aligned}$$

So we get:

$$\forall n \geq 0.\ WI_{6a}(n) = \frac{\{\},\ True,\ (\texttt{m\_ec} \leftarrow \texttt{s\_ec} + 1 + n, \texttt{s\_ec} \leftarrow \texttt{s\_ec} + 1 + n)}{220 \xRightarrow{\tau} 210}$$

[11] when showing the result of *Posts* composition, we will omit the identity substitution functions introduced by the $\odot$ definition in page 7

5.3. Composition properties: composition of weak open transitions

We now have two different semantics for open pNets: a strong semantics, defined as an open automaton, and as a weak semantics, defined as a weak open automaton. Like the open automaton, the weak open automaton features valuable composition properties. We can exhibit a composition property and a decomposition property that relate open pNet composition with their semantics, defined as weak open automata. These are however technically more complex than the ones for open automata because each hole performs a set of actions, and thus a composed transition is the composition of one transition of the top-level pNet and a sequence of transitions of the sub-pNet that fills its hole. Composition and decomposition properties can be found as Lemma 6, Lemma 7, and Lemma 8 in [3].

5.4. Weak FH-bisimulation

For defining a bisimulation relation between weak open automata, two options are possible. One option is that we define a simulation similar to the strong simulation but based on weak open automata, this would look like the FH-simulation but would need to be adapted to weak open transitions. Alternatively, we could define directly and classically a weak FH-simulation as a relation between two open automata, relating the open transitions of the first one with the transitions of the weak open automaton derived from the second one.

The definition below specifies how a set of weak open transitions can simulate an open transition, and under which condition; this is used to relate, by weak FH-bisimulation, two open automata by reasoning on the weak open automata that can be derived from the strong ones.

Definition 14 (Weak FH-bisimulation).
Let $A_1 = \langle\!\langle J, \mathcal{S}_1, s_0, V_1, \mathcal{T}_1 \rangle\!\rangle$ and $A_2 = \langle\!\langle J, \mathcal{S}_2, t_0, V_2, \mathcal{T}_2 \rangle\!\rangle$ be open automata with disjoint sets of variables. Let $\langle\!\langle J, \mathcal{S}_1, s_0, V_1, \mathcal{WT}_1 \rangle\!\rangle$ and $\langle\!\langle J, \mathcal{S}_2, t_0, V_2, \mathcal{WT}_2 \rangle\!\rangle$ be the weak open automata derived from A_1 and A_2 respectively. Let $\mathcal{R}$ a relation over $\mathcal{S}_1$ and $\mathcal{S}_2$, as in Definition 7.

Then $\mathcal{R}$ is a weak FH-bisimulation iff for any states $s \in \mathcal{S}_1$ and $t \in \mathcal{S}_2$ such that $(s, t|Pred_{s,t}) \in \mathcal{R}$, we have the following:

- For any open transition OT in $\mathcal{T}_1$:

$$\frac{\beta_j^{j \in J'}, Pred_{OT}, Post_{OT}}{s \xrightarrow{\alpha} s'}$$

  there exists an indexed set of weak open transitions $WOT_x^{x \in X} \subseteq \mathcal{WT}_2$:

$$\frac{\gamma_{jx}^{j \in J_x}, Pred_{OT_x}, Post_{OT_x}}{t \xRightarrow{\alpha_x} t_x}$$

such that $\forall x.\{j \in J' | \beta_j \neq \tau\} = J_x, (s', t_x | Pred_{s',t_x}) \in \mathcal{R}$; and

$$Pred_{s,t} \wedge Pred_{OT} \implies \bigvee_{x \in X} \left(\forall j \in J_x . (\beta_j)^{\nabla} = \gamma_{jx} \wedge Pred_{OT_x} \wedge \alpha = \alpha_x \wedge Pred_{s',t_x} \{\!\{ Post_{OT} \uplus Post_{OT_x} \}\!\} \right)$$

- and symmetrically any open transition from t in $\mathcal{T}_2$ can be covered by a set of weak transitions from s in $\mathcal{WT}_1$.

Two open automata are *weak FH-bisimilar* if there exists a weak FH-bisimulation relation that relates their initial states. This relation is called *weak FH-bisimilarity*. Two pNets are weak FH-bisimilar if their associated open automata are weakly bisimilar.

Compared to strong bisimulation, except the obvious use of weak open transitions to simulate an open transition, the condition on predicate is slightly changed concerning actions of the holes. Indeed only the visible actions of the holes must be compared and they form a list of actions, but of length at most one.

Our first important result is that weak FH-bisimilarity is an equivalence in the same way as strong FH-bisimilarity.

Theorem 6 (Weak FH-bisimilarity is an equivalence). *Weak FH-bisimilarity is reflexive, symmetric and transitive.*

The proof is detailed in [3], it follows a similar pattern as the proof that strong FH-bisimilarity is an equivalence, but technical details are different, and in practice we rely on a variant of the definition of weak FH-bisimilarity; this equivalent version simulates a *weak* open transition with a set of weak open transition. The careful use of the best definition of weak FH-bisimilarity makes the proof similar to the strong FH-bisimilarity case.

Proving bisimulation in practice

In practice, we are dealing with finite representations of the (infinite) open automata. In [31], we defined a slightly modified definition of the "coverage" proof obligation, in the case of strong FH-bisimulation. This modification is required to manage in a finite way all possible instantiations of an OT. In the case of weak FH-bisimulation, the proof obligation from Definition 14 becomes:

$$\forall fv_{OT}. \left\{ Pred_{s,t} \wedge Pred_{OT} \implies \bigvee_{x \in X} \left[\exists fv_{OT_x} . \left(\forall j \in J_x . (\beta_j)^{\nabla} = \gamma_{jx} \wedge Pred_{OT_x} \wedge \alpha = \alpha_x \wedge Pred_{s',t_x} \{\!\{ Post_{OT} \uplus Post_{OT_x} \}\!\} \right) \right] \right\}$$

where fv_{OT} denotes the set of free variables of all expressions in OT.

5.5. Weak FH-bisimilation for open pNets

Before defining a weak open automaton for the semantics of open pNets, it is necessary to state under which condition a pNet is unable to observe silent actions of its holes. In the setting of pNets this can simply be expressed as a condition on the synchronisation vectors. Precisely, the set of synchronisation vectors must contain vectors that let silent actions go through the pNet, i.e. synchronisation vectors where one hole does a τ transition, and the global visible action is a τ. Additionally, no other synchronisation vector must be able to react on a silent action from a hole, i.e. if a synchronisation vector observes a τ from a hole it cannot synchronise it with another action nor emit an action that is not τ. This is formalised as follows:

Definition 15 (Non-observability of silent actions for pNets). A pNet $\langle\!\langle P_i^{i\in I}, Sort_j^{j\in J}, \overline{SV}\rangle\!\rangle$ *cannot observe silent actions* if it verifies: $\forall i \in I \uplus J.\, (i \mapsto \tau) \to \tau[True] \in \overline{SV}$ and

$$\forall \left((\alpha_i)^{i\in I'} \to \alpha'[e_b] \in \overline{SV} \right), \forall i \in I' \cap J.\, \alpha_i = \tau \implies \alpha' = \tau \ \wedge \ I' = \{i\}$$

With this definition, it is easy to check that the open automaton that gives the semantics of such an open pNet cannot observe silent actions in the sense of Definition 11.

Property 1 (Non-observability of silent actions). *The semantics of a pNet, as provided in Definition 9, that cannot observe silent actions is an open automaton that cannot observe silent actions.*

Under this condition, it is safe to define the weak open automaton that provides a weak semantics to a given pNet. This is simply obtained by applying Definition 13 to generate a weak open automaton from the open automaton that is the strong semantics of the open pNet, as provided by Definition 9.

Definition 16 (Semantics of pNets as a weak open automaton). Let A be the open automaton expressing the semantics of an open pNet P; let $\langle\!\langle J, \mathcal{S}, s_0, V, \mathcal{WT}\rangle\!\rangle$ be the weak open automaton derived from A; we call this weak open automaton the weak semantics of the pNet P. Then, we denote $P \models WOT$ whenever $WOT \in \mathcal{WT}$.

From the definition of the weak open automata of pNets, we can now study the properties of weak bisimulation concerning open pNets.

5.6. Properties of weak bisimulation for open pNets

When silent actions cannot be observed, weak FH-bisimilarity is a congruence for open pNets: if P and Q are weakly bisimilar to P' and Q' then the composition of P and Q is weakly bisimilar to the composition of P' and Q', where composition is the hole replacement operator: $P[Q]_j$ and $P'[Q']_j$ are weak FH-bisimilar. This can be shown by proving the two following theorems.

The detailed proof of these theorem can be found in [3]. The proof strongly relies on the fact that weak FH-bisimulation is an equivalence, but also on the composition properties for open automata.

Theorem 7 (Congruence for weak FH-bisimilarity). *Consider an open pNet P that cannot observe silent actions, of the form $P = \langle\!\langle P_i^{i\in I}, Sort_j^{j\in J}, \overline{SV}\rangle\!\rangle$. Let $j_0 \in J$ be a hole. Let Q and Q' be two weak FH-bisimilar pNets such that*[12] $\mathrm{Sort}(Q) = \mathrm{Sort}(Q') \subseteq Sort_{j_0}$. *Then $P[Q]_{j_0}$ and $P[Q']_{j_0}$ are weak FH-bisimilar.*

Theorem 8 (Context equivalence for weak FH-bisimilarity). *Consider two open pNets $P = \langle\!\langle P_i^{i\in I}, Sort_j^{j\in J}, \overline{SV}\rangle\!\rangle$ and $P' = \langle\!\langle {P'}_i^{i\in I}, Sort_j^{j\in J}, \overline{SV'}\rangle\!\rangle$ that are weak FH-bisimilar (recall they must have the same holes to be FH-bisimilar) and that cannot observe silent actions. Let $j_0 \in J$ be a hole, and Q be a pNet such that* $\mathrm{Sort}(Q) \subseteq Sort_{j_0}$. *Then $P[Q]_{j_0}$ and $P'[Q]_{j_0}$ are weak FH-bisimilar.*

Finally, the previous theorems can be composed to state a general theorem about composability and weak FH-bisimilarity.

Theorem 9 (Composability of weak FH-bisimilarity). *Consider two weak FH-bisimilar pNets with an arbitrary number of holes, such that the two pNets cannot observe silent actions. When replacing, inside those two original pNets, a subset of the holes by weak FH-bisimilar pNets, we obtain two weak FH-bisimilar pNets.*

Example 6 (CCS Choice). Consider the + operator of CCS, shown in Example 1. The pNet does not satisfy Definition 15. Indeed, if a or b is τ then the + operator can observe the τ transition. It is well-known that weak bisimularity is not a congruence in CCS, this corresponds to the fact that the + operator can observe the τ transitions. Thus, even if we can define a weak FH-bisimilarity for CCS with + it does not verify the necessary requirements for being a congruence.

On the other side, the parallel operator defined similarly *satisfies* Definition 15, and indeed bisimilarity is a congruence for the parallel operator in CCS.

Running example

In Section 5 we have shown the full saturated weak automaton for both *SimpleProtocolSpec* and *SimpleProtocolImpl*. We will show here how we can check if some given relation between these two automata is a weak FH-bisimulation.

Preliminary remarks:

- Both pNets trivially verify the "non-observability" condition: the vectors having τ as an action of a sub-net are of the form "$< -, \tau, - >\!\!\rightarrow \tau$".

[12] Note that $\mathrm{Sort}(Q) = \mathrm{Sort}(Q')$ is ensured by weak FH-bisimilarity.

- We must take care of variable name conflicts: in our example, the variables of the 2 systems already have different names, but the action parameters occurring in the transitions (`m`, `msg`, `ec`) are the same, that is not correct. In the tools, this is managed by the static semantic layer; in the example, we rename the only conflicting variables m into $m1$ for *SimpleProtocolSpec*, and $m2$ for *SimpleProtocolImpl*.

Now consider the relation $\mathcal{R}$ defined by the following triples:

SimpleProtocolSpec states	*SimpleProtocolImpl* states	Predicate
b0	000	True
b0	202	True
b1	100	`b_msg = s_msg` $\wedge$ `b_ec = s_ec`
b1	210	`b_msg = m_msg` $\wedge$ `b_ec = m_ec`
b1	220	`b_msg = s_msg` $\wedge$ `b_ec = s_ec`
b1	201	`b_msg = r_msg` $\wedge$ `b_ec = r_ec`

Checking that $\mathcal{R}$ is a weak FH-bisimulation means checking, for each of these triples, that each (strong) OT of one the states corresponds to a set of WOTs of the other, using the conditions from Definition 14. We give here one example: consider the second triple from the table, and transition SS_3 from state `b0`. Its easy to guess that it will correspond to $WI_3(0)$ of state 202 (and equivalently state 000, see Figure 9):

$$SS_3 = \frac{\{\texttt{P}\mapsto\texttt{p-send(m1)}\}, True, (\texttt{b\_msg} \leftarrow \texttt{m1}, \texttt{b\_ec} \leftarrow 0)}{\text{b0} \xrightarrow{\texttt{in(m1)}} \text{b1}}$$

$$WI_3(0) = \frac{\{\texttt{P}\mapsto\texttt{p-send(m2)}\}, True, (\texttt{s\_msg} \leftarrow \texttt{m2}, \texttt{s\_ec} \leftarrow 0)}{000 \xRightarrow{\texttt{in(m2)}} 100}$$

Let us check formally the conditions:

- Their sets of active (non-silent) holes is the same: $J' = J_x = \{\texttt{P}\}$.
- Triple $(\text{b1}, 100, \texttt{b\_msg = s\_msg} \wedge \texttt{b\_ec = s\_ec})$ is in $\mathcal{R}$.
- The verification condition
  $\forall fv_{OT}.\{Pred \wedge Pred_{OT}$
  $\Longrightarrow \bigvee_{x\in X} [\exists fv_{OT_x}.(\forall j \in J_x.(\beta_j)^{\nabla} = \gamma_{jx} \wedge Pred_{OT_x} \wedge \alpha = \alpha_x \wedge Pred_{s',t_x}\{\!\{Post_{OT} \uplus Post_{OT_x}\}\!\})]\}$

  Gives us:

  $\forall \texttt{m1}.\{True \wedge True \implies \exists \texttt{m2}.$
  $(\texttt{[p-send(m1)]} = \texttt{[p-send(m2)]} \wedge True \wedge \texttt{in(m1)} = \texttt{in(m2)} \wedge$
  $(\texttt{b\_msg = s\_msg} \wedge \texttt{b\_ec = s\_ec})\{\!\{(\texttt{b\_msg} \leftarrow \texttt{m1}, \texttt{b\_ec} \leftarrow 0) \uplus (\texttt{s\_msg} \leftarrow \texttt{m2}, \texttt{s\_ec} \leftarrow 0)\}\!\})\}$

  That is reduced to:

  $\forall \texttt{m1}.\exists \texttt{m2}.(\texttt{p-send(m1)} = \texttt{p-send(m2)} \wedge \texttt{in(m1)} = \texttt{in(m2)} \wedge \texttt{m1} = \texttt{m2} \wedge 0 = 0)$

  That is a tautology.

6. Related Works

To the best of our knowledge, there are not many research works on Weak Bisimulation Equivalences between such complicate system models (open, symbolic, data-aware, with loops and assignments). We give a brief overview of other related publications, focussing first on Open and Compositional approaches, then on Symbolic Bisimulation for data-sensitive systems.

Open and compositional systems

In [36, 35], the authors investigate several methodologies for the compositional verification of software systems, with the aim to verify reconfigurable component systems. To improve scaling and compositionality, the authors decompose the verification problem that is to be resolved by a SMT (satisfiability modulo theory) solver into independent sub-problems on independent sets of variables. These works clearly highlight the interest of incremental and compositional verification in a very general setting. In our own work on open pNets, adding more structure to the composition model, we show how to enforce a compositional proof system that is more versatile than independent sets of variables as the composition is structured and allows arbitrary synchronisations between sub-entities. Our theory has also been encoded into an SMT solver and it would be interesting to investigate how the examples of evolving systems studied by Johnson et al. could be encoded into pNet and verified by our framework. However, the models of Johnson et al. are quite different from ours, in particular they are much less structured, and translating them is clearly outside the scope of this article.

In previous work [20], we also have shown how (closed) pNet models could be used to encode and verify finite instances of reconfigurable component systems.

Methodologies for reasoning about abstract semantics of open systems can be found in [5, 6, 18], authors introduce behavioural equivalences for open systems from various symbolic approaches. Working in the setting of process calculi, some close relations exist with the work of the authors of [5, 6], where both approaches are based on some kinds of labelled transition systems. The distinguishing feature of their approach is that the transitions systems are labelled with logical formulae that provides an abstract characterization of the structure that a hole must possess and of the actions it can perform in order to allow a transition to fire. Logical formulae are suitable formalisms that capture the general class of components that can act as the placeholders of the system during its evolution. In our approach we purposely leave the algebra of action terms undefined but the only operation we allow on action of holes is the comparison with other actions. Defining properly the interaction between a logical formulae in the action and the logics of the pNet composition seems very difficult. mCRL2 [22] is another effective model for specifying and proving properties of concurrent systems. mCRL2 has an established tool-suite and share similarities with pNets. However, pNets feature hierarchical composition with more structure than mCRL2 that composes processes with a parallel operator. Synchronisation of processes is expressed very differently; it is difficult

to precisely compare multi-actions of mCRL2 with synchronisation vectors of pNets but synchronisation vector ogf pNets enforce a synchronisation based on the structure while in mCRL2 synchronisation is specified in a versatile, flexible, but less structured way.

In the same vein as context systems [38], pNets is a formalism for modular and possibly incomplete description of concurrent systems. The two formalisms are however different as the theory of contexts relies on a form of rewrite rules, while pNets rely on parametric automata to express the system behaviour. pNets have similar features as context systems [38] and static constructs [33]. Indeed all these approaches allow for modular and possibly incomplete description and structural composition of systems. The main originality of pNets compared to these other compositional approaches is the parameterised nature of the specification, which enables reasoning on value-passing systems but also on rich synchronisations that depend on the value of parameters.

Decomposition techniques

Quotienting of process algebras [38] and decomposition techniques for mCRL2 [39] share similarities with our approach; they propose to overcome the state-space-explosion problem by decomposing formulas to be verified according to the process composition. The decomposed problem must be equivalent to the original one. However these techniques are expressed in a very different setting from ours and it is difficult to precisely relate them to the more structural and parameterised point of view we adopt here. We could try to apply such automatic decomposition techniques to open pNets, but deriving a decomposition for systems synchronised in a very parameterised way like we do requires further investigations. Both parallel composition [38] and mCRL2 [39] feature a concrete verification setting where decomposition is useful, while open automata provide a more general setting that could be used to represent both frameworks and hopefully generalise process decomposition results of [38, 39].

Logical and semantics approaches

Among the approaches for modelling open systems, one can cite [8] that uses transition conditions depending on an external environment, and introduce bisimulation relations based on this approach. The approach of [8] is highly based on logics and their bisimulation theory is richer than ours in this aspect, while our theory is highly structural and focuses on relation between structure and equivalence. Also, we see composition as a structural operation putting systems together, and do not focus on the modelling of an unknown outside world. Overall we believe that the two approaches are complementary but comparing precisely the two different bisimulation theories is not trivial.

There is also a clear relation with the seminal works on rule formats for Structured Operational Semantics, e.g. De Simone format, GSOS, and conditional rules with or without negative premises [16, 10, 24, 47]. The Open pNets model provides a way to define operators similar to these rules formats, but with quite different aim and approach. A formal comparison would be interesting, though not trivial. What we can say easily is that: the pNet format

syntactically encompasses De Simone, GSOS, and conditional premises rules. Then our compositionality result is more powerful than their classical results, but this is not a surprise, as we rely on a (sufficient) syntactic hypothesis on a particular system, rather than the general rules defining an operator. Last, we intentionally do not accept negative premises, that would be more to put into practice in our implementation. This extension could be studied in future work.

Symbolic and data-sensitive systems

As mentioned in the *Introduction*, we were substantially inspired by the works of Lin et al. [34, 26, 40]. They developed the theory of symbolic transition graphs (STG), and the associated symbolic (early and late, strong and weak) bisimulations. Moreover, they studied STGs with assignments as a model for message-passing processes. Our work extends those contributions in several ways: first our models are compositional, and our bisimulations come with effective conditions for being preserved by pNet composition (i.e. congruent), even for the weak version. This result is more general than the bisimulation congruences for value-passing CCS in [34]. Then our settings for management of data types are much less restrictive, thanks to our use of satisfiability engines, while Lin's algorithms were limited to data-independent systems.

In a similar way, [1] presents a notion of "data-aware" bisimulations on data graphs, in which computation of such bisimulations is studied based on XPath logical language extended with tests for data equality.

Research related to the keyword "Symbolic Bisimulation" refer to two very different domains, namely BDD-like techniques for modelling and computing finite-state bisimulations, that are not related to our topic; and symbolic semantics for data-dependant or high-order systems, that are very close in spirit to our approach. In this last area, we can mention Calder's work [15], that defines a symbolic semantic for full Lotos, with a symbolic bisimulation over it; Borgstrom et al., Liu et al, Delaune et al. and Buscemi et al. providing symbolic semantics and equivalence for different variants of pi calculus respectively [12, 17, 41, 14]; and more recently Feng et al. provide a symbolic bisimulation for quantum processes [19]. All the above works are based on models definitely different from ours, and none of them allows system to be as much parameterised as open pNets; this additional expressiveness is due to the open and symbolic nature of our constructs.

7. Conclusion and discussion

pNets (Parameterised Networks of Automata) is a formalism adapted to the representation of the behaviour of parallel or distributed systems. One strength of pNets is their parameterised nature, making them suitable for to the representation of systems of arbitrary size, and making the modelling of parameterised systems possible. Parameters are also crucial to reason about interaction protocols that can address one entity inside an indexed set of processes. pNets have been successfully used to represent behavioural specification

of parallel and distributed components and verify their correctness [2, 29]. VCE is the specification and verification platform that uses pNets as an intermediate representation. In this platform we have developed tool support for computing the symbolic semantics in term of open automata; this is presented in [45, 46], together with a case-study based on the on-board control software of satellites. In [9] we present how to encode reactive systems from the BIP specification language and check their temporal properties using VCE. In [31, 32] we describe our strong bisimulation algorithms, with illustration on the equivalence of different encodings of operators.

Open pNets are pNets with holes; they are adapted to represent processes parameterised by the behaviour of other processes, like composition operators or interaction protocols that synchronise the actions of processes that can be provided afterwards. Open pNets are hierarchical composition of automata with holes and parameters. We defined here a semantics for open pNets and a complete bisimulation theory for them. The semantics of open pNets relies on the definition of open automata that are automata with holes and parameters, but no hierarchy. Open automata are a flattened view of the pNet; their behaviour is expressed as open transitions that allow for a more semantic interpretation of process parameters (holes) than pNets. In the end, open automata are labelled transition systems with parameters and holes, a notion that is useful to define semantics, but makes less sense for the high level modelling of a system, compared to pNets. Open automata is the formalism that makes it possible to define FH-bisimilarity.

This article defines a strong and a weak bisimulation relation that are adapted to parameterised systems and hierarchical composition. FH-bisimulation handles pNet parameters in the sense that two states might be or not in relation depending on the value of parameters. Strong FH-bisimilarity is compositional in the sense that it is maintained when composing processes. We also identified a simple and realistic condition on the semantics of non-observable actions that allows weak FH-bisimilarity to be also compositional. Overall we believe that this article paved the way for a solid theoretical foundation for compositional verification of parallel and distributed systems.

The pNets formalism supports the refinement checking at the automaton level through a simulation, with symbolic evaluation of guards and transitions. The definition of simulation on open automata should be stronger than a classical simulation since it matches a transition with a family of transitions. Such a relation should be able to check the refinement by taking into account state duplication, transition removal, guard strengthening, variable modification. Additionally, composition of pNets gives the possibility to either add new holes to a system or fill holes. A useful simulation relation should thus support the comparison of automata that do not have the same number of holes. Designing such a simulation relation is a non-trivial extension that we leave for future work.

We are currently looking at further properties of FH-bisimulation, but also the relations with existing equivalences on both closed and open systems. In particular, our model being significantly different from those considered in [34], it would be interesting to compare our “FH” family of bisimulations with the

hierarchy of symbolic bisimulations from those authors. We also plan to apply open pNets to the study of complex composition operators in a symbolic way, for example in the area of parallel skeletons, or distributed algorithms.

Recently we published preliminary work on methods for checking weak FH-bisimulation [48]. The challenges here, in the context of our symbolic systems, are not so much algorithmic complexity, as was the case with classical weak bisimulation on finite models, but decidability and termination. The naive approach, using an explicit construction of the weak transition, may in itself introduce non-termination, so we prefer a direct implementation of the weak bisimulation definition, without constructing the weak automata beforehand, but searching *on demand* to construct the required weak transitions. We illustrate this approach on a simple error-correcting transport protocol case-study. Beside, we explore in [49] more pragmatic approaches using weak bisimulation preserving (pattern-based) reduction rules.

References

[1] Sergio Abriola, Pablo Barceló, Diego Figueira, and Santiago Figueira. Bisimulations on data graphs. *Journal of Artificial Intelligence Research*, 61:171–213, 2018.

[2] Rabéa Ameur-Boulifa, Ludovic Henrio, Oleksandra Kulankhina, Eric Madelaine, and Alexandra Savu. Behavioural semantics for asynchronous components. *Journal of Logical and Algebraic Methods in Programming*, 89:1–40, 2017.

[3] Rabéa Ameur-Boulifa, Ludovic Henrio, and Eric Madelaine. Compositional equivalences based on open pnets. *CoRR*, abs/2007.10770, 2020.

[4] André Arnold. Synchronised behaviours of processes and rational relations. *Acta Informatica*, 17:21–29, 1982.

[5] Paolo Baldan, Andrea Bracciali, and Roberto Bruni. Bisimulation by unification. In *AMAST 2002 - Algebraic Methodology and Software Technology, 9th International Conference*, volume 2422 of *Lecture Notes in Computer Science*, pages 254–270. Springer, 2002.

[6] Paolo Baldan, Andrea Bracciali, and Roberto Bruni. A semantic framework for open processes. *Theoretical Computer Science*, 389(3):446–483, 2007.

[7] Tomás Barros, Rabéa Ameur-Boulifa, Antonio Cansado, Ludovic Henrio, and Eric Madelaine. Behavioural models for distributed fractal components. *Annales des Télécommunications*, 64(1-2):25–43, 2009.

[8] Harsh Beohar, Barbara König, Sebastian Küpper, and Alexandra Silva. Conditional transition systems with upgrades. *Science of Computer Programming*, 186:102320, 2020.

[9] Simon Bliudze, Ludovic Henrio, and Eric Madelaine. Verification of concurrent design patterns with data. In Hanne Riis Nielson and Emilio Tuosto, editors, *COORDINATION 2019 - 21st International Conference on Coordination Models and Languages*, volume LNCS-11533, pages 161–181. Springer International Publishing, 2019.

[10] Bard Bloom, Sorin Istrail, and Albert R. Meyer. Bisimulation can't be traced. *Journal of the ACM*, 42(1):232–268, 1995.

[11] Tommaso Bolognesi and Ed Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks*, 14(1):25–59, 1987.

[12] Johannes Borgström, Sébastien Briais, and Uwe Nestmann. Symbolic Bisimulation in the Spi Calculus. In *CONCUR 2004 - Concurrency Theory, 15th International Conference*, volume 3170 of *Lecture Notes in Computer Science*, pages 161–176. Springer, 2004.

[13] Rabéa Ameur Boulifa, Raluca Halalai, Ludovic Henrio, and Eric Madelaine. Verifying safety of fault-tolerant distributed components. In *FACS 2011 - 8th International Symposium on Formal Aspects of Component Software*, Lecture Notes in Computer Science. Springer, 2011.

[14] Maria Grazia Buscemi and Ugo Montanari. Open Bisimulation for the Concurrent Constraint Pi-Calculus. In *ESOP 2008 - Programming Languages and Systems, 17th European Symposium on Programming*, volume 4960 of *Lecture Notes in Computer Science*, pages 254–268. Springer, 2008.

[15] Muffy Calder and Carron Shankland. A symbolic semantics and bisimulation for full LOTOS. In *FORTE 2001 - 21st International Conference on Formal Techniques for Networked and Distributed Systems*, pages 185–200. Springer, 2001.

[16] Robert De Simone. Higher-level synchronising devices in MEIJE-SCCS. *Theoretical Computer Science*, 37:245–267, 1985.

[17] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Symbolic Bisimulation for the Applied Pi Calculus. In *FSTTCS 2007 - Foundations of Software Technology and Theoretical Computer Science, 27th International Conference*, volume 4855 of *Lecture Notes in Computer Science*, pages 133–145. Springer, 2007.

[18] Jérémy Dubut. Bisimilarity of Diagrams. In *RAMiCS 2020 - Relational and Algebraic Methods in Computer Science, 18th International Conference*, volume 12062 of *Lecture Notes in Computer Science*, pages 65–81. Springer, 2020.

[19] Yuan Feng, Yuxin Deng, and Mingsheng Ying. Symbolic bisimulation for quantum processes. *ACM Transactions on Computational Logic (TOCL)*, 15(2):14, 2014.

[20] Nuno Gaspar, Ludovic Henrio, and Eric Madelaine. Formally reasoning on a reconfigurable component-based system - A case study for the industrial world. In *FACS 2013 - Formal Aspects of Component Software, 10th International Symposium,*, volume 8348 of *Lecture Notes in Computer Science*, pages 137–156. Springer, 2013.

[21] Jan Friso Groote. Transition system specifications with negative premises. *Theoretical Computer Science*, 118(2):263–299, 1993.

[22] Jan Friso Groote, Jeroen J. A. Keiren, Bas Luttik, Erik P. de Vink, and Tim A. C. Willemse. Modelling and analysing software in mcrl2. In Farhad Arbab and Sung-Shik Jongmans, editors, *Formal Aspects of Component Software*, pages 25–48, Cham, 2020. Springer International Publishing.

[23] Jan Friso Groote, Mohammad Reza Mousavi, and Michel A. Reniers. A hierarchy of SOS rule formats. *Electronic Notes in Theoretical Computer Science*, 156(1):3–25, 2006. Proceedings of the Second Workshop on Structural Operational Semantics.

[24] Jan Friso Groote and Frits Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100(2):202–260, 1992.

[25] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming.*, 8(3):231–274, 1987.

[26] Matthew Hennessy and Huimin Lin. Symbolic bisimulations. *Theoretical Computer Science*, 138(2):353–389, 1995.

[27] Matthew Hennessy and Humin Lin. Symbolic bisimulations. *Theoretical Computer Science*, 138(2):353–389, 1995. Meeting on the mathematical foundation of programing semantics.

[28] Matthew Hennessy and Julian Rathke. Bisimulations for a calculus of broadcasting systems. *Theoretical Computer Science*, 200(1-2):225–260, 1998.

[29] Ludovic Henrio, Oleksandra Kulankhina, Siqi Li, and Eric Madelaine. Integrated environment for verifying and running distributed components. In *FASE 2016 - 19th International Conference on Fundamental Approaches to Software Engineering*, pages 66–83. Springer Berlin Heidelberg, 2016.

[30] Ludovic Henrio, Eric Madelaine, and Min Zhang. A Theory for the Composition of Concurrent Processes. In *FORTE 2016 - 36th International Conference on Formal Techniques for Distributed Objects, Components, and Systems*, volume 9688, pages 175 – 194. Springer, 2016.

[31] Zechen Hou and Eric Madelaine. Symbolic Bisimulation for Open and Parameterized Systems. In *PEPM 2020 - Workshop on Partial Evaluation and Program Manipulation*. ACM SIGPLAN, 2020.

[32] Zechen Hou, Eric Madelaine, Jing Liu, and Yuxin Deng. Symbolic Bisimulation for Open and Parameterized Systems - Extended version. Research Report RR-9304, Inria & Université Cote d'Azur, CNRS, I3S, Sophia Antipolis, France ; East China Normal University (Shanghai), November 2019.

[33] Hans Hüttel and Kim Guldstrand Larsen. The use of static constructs in A modal process logic. In *Logic at Botik '89, Symposium on Logical Foundations of Computer Science*, volume 363 of *Lecture Notes in Computer Science*, pages 163–180. Springer, 1989.

[34] Anna Ingólfsdóttir and Huimin Lin. A symbolic approach to value-passing processes. In *Handbook of Process Algebra*, pages 427–478. North-Holland/Elsevier, 2001.

[35] Kenneth Johnson and Radu Calinescu. Efficient Re-resolution of SMT Specifications for Evolving Software Architectures. In *QoSA 2014 - 10th International ACM Sigsoft Conference on Quality of Software Architectures*, pages 93–102. ACM, 2014.

[36] Kenneth Johnson, Radu Calinescu, and Shinji Kikuchi. An incremental verification framework for component-based software systems. In *CBSE 2013 - 16th International ACM Sigsoft Symposium on Component-based Software Engineering*, pages 33–42. ACM, 2013.

[37] Kim G. Larsen. A context dependent equivalence between processes. *Theoretical Computer Science*, 49:184–215, 1987.

[38] Kim Guldstrand Larsen and Liu Xinxin. Compositionality through an operational semantics of contexts. *Journal of Logic and Computation*, 1(6):761–795, 1991.

[39] Maurice Laveaux and Tim A. C. Willemse. Decomposing monolithic processes in a process algebra with multi-actions. In *ICE*, volume 347 of *EPTCS*, pages 57–76, 2021.

[40] Huimin Lin. Symbolic transition graph with assignment. In *CONCUR'96 - Concurrency Theory, 7th International Conference*, volume 1119, pages 50–65. Springer Berlin Heidelberg, 1996.

[41] Jia Liu and Huimin Lin. A Complete Symbolic Bisimulation for Full Applied Pi Calculus. In *SOFSEM 2010 - 36th Conference on Current Trends in Theory and Practice of Computer Science*, volume 5901, pages 552–563. Springer, 2010.

[42] Robin Milner. *A Calculus of Communicating Systems.* Springer-Verlag, Berlin, Heidelberg, 1982.

[43] Robin Milner. *Communication and Concurrency.* Int. Series in Computer Science. Prentice-Hall, Englewood Cliffs, New Jersey, 1989. SU Fisher Research 511/24.

[44] Gordon D. Plotkin. A structural approach to operational semantics. *Journal of Logic and Algebraic Programming*, 60-61:17–139, 2004.

[45] Xudong Qin, Simon Bliudze, Eric Madelaine, and Min Zhang. Using SMT engine to generate symbolic automata. In *AVOCS 2018 - 18th International Workshop on Automated Verification of Critical Systems*, volume 076. Electronic Communications of the EASST, 2018.

[46] Xudong Qin, Simon Bliudze, Eric Madelaine, and Min Zhang. Using SMT engine to generate Symbolic Automata -Extended version. Research Report RR-9177, Inria & Université Cote d'Azur, CNRS, I3S, Sophia Antipolis, France ; inria, June 2018.

[47] Robert Jan Van Glabbeek. The meaning of negative premises in transition system specifications II. *The Journal of Logic and Algebraic Programming*, 60-61:229–258, 2004.

[48] Biyang Wang, Eric Madelaine, and Min Zhang. New symbolic model and equivalences checking for open automata. In *SMC 2021 - IEEE International Conference on Systems, Man, and Cybernetics*, pages 2360–2367. IEEE, 2021.

[49] Biyang Wang, Eric Madelaine, and Min Zhang. Symbolic Weak Equivalences: Extension, Algorithms, and Minimization - Extended version. Research Report RR-9389, Inria, Université Cote d'Azur, CNRS, I3S, Sophia Antipolis, France; East China Normal University (Shanghai), 2021.

Chapter 5

Refinement of Open Systems

5.1 Summary

As introduced in the previous chapter, the open automata models, which express the semantics of open pNets, are themselves convenient to model parallel systems that are parameterised. They express value-passing communication and have parallel composition as a basic operation. But they also offer the possibility to talk about unknown processes, and to reason without their specification. An open automaton is a classical labelled transition system (LTS) with variables and holes. Similarly to modal LTSs (e.g. [26, 78, 83]), which add information on transitions in order to distinguish allowed and required behaviours, open automata use labelled transitions between states to model behaviours. But unlike modal LTSs, they distinguish internal and environmental actions, which makes them a suitable semantic model for modelling reactive systems [100], i.e. systems which continuously interact with their environment, such as process controllers.

The notion of holes enables a form of compositional verification approach, since once an appropriate partial specification has been developed for a component of a system, one must only verify an implementation with respect to this specification – the remainder of the system is irrelevant. Indeed, holes enable the composition of automata: an automaton with a hole is an operator that takes another automaton as parameter and reacts to the actions it emits; the composed automaton is a more precise automaton where the behaviour of one "process parameter". For their part, variables make automata symbolic and allow them to encode infinite state systems.

Intuitively, the composition operation of open automata can be viewed as a specific way to connect automata and to fill holes. The holes, which are endowed with a signature, can be filled with compatible open automata. The open automaton which fills a hole may itself carry several holes, and thus create several other holes on the automaton resulting from the composition. Therefore, although the composition operation allows to specify more, it does not necessarily reduce the number of holes.

Example

To illustrate the expressiveness and the composability of open automata, we use as example a specification of the traffic light system that controls traffic at an intersection. The open automaton modelling this system is illustrated in Figure 5.1[1]. This automata shows a model of an incomplete specification of the system, the timer is not specified. It accepts an unimplemented control circuit in the hole which gives the timings and an also unimplemented counter in the hole to count external tick actions. As the automaton shows, it has three states remembering which coloured light is on (Red, Yellow or Green). It includes two holes: a controller (ctl) and a counter (cnt) depicting together the behaviour of the timer. The color switches when the counter and the controller components agree that the time is over. The new time limit can be set by the counter component and the exposed action to the environment is an unobservable action τ.

[1] Note that we have slightly modified the notations form the previous chapter. The predicate is noted $\top$ instead of True and the assignments in braces instead of parentheses.

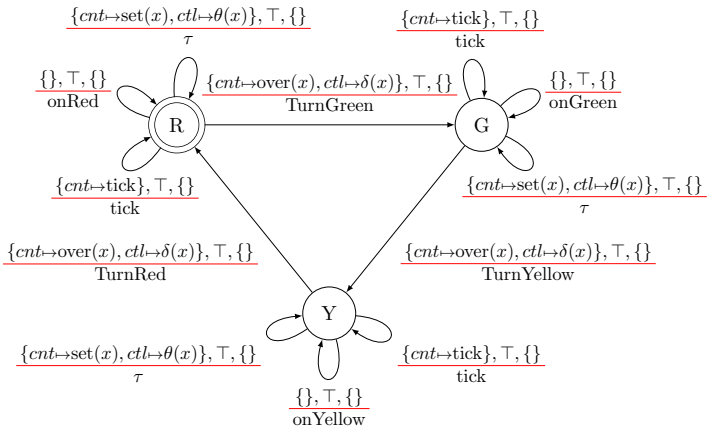


Figure 5.1: The specification of a Traffic Light system

Recall that the particular form of the open transitions encodes the informations about the context, which that are required to execute it. For instance, the following open transition of the traffic light automaton: $\frac{\{cnt{\mapsto}\text{set}(x), ctl{\mapsto}\theta(x)\}, \top, \{\}}{R \xrightarrow{\tau} R}$ expresses that the automaton executes unconditionally (the predicate is true $\top$) an unobservable action when the controller *ctl* and the counter *cnt* execute their actions $\theta(x)$ and set(x) respectively; and the execution has no effect on the system variables (the set of assignments is empty $\{\}$).

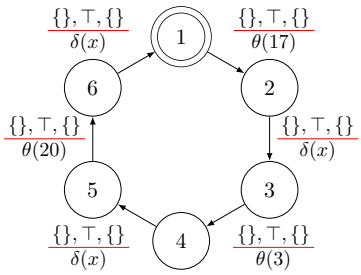
Examples of specifications, that can be used to complete the model of traffic light system and fill its holes, are provided as open automata, they are shown in Figure 5.2. On the left (a), the controller component designed to be connected in the hole *ctl*. Its role is to decide the duration before switching the lights. We control the time interval for each light by setting them by prior knowledge: 17s for the first duration, 3s for the second, and 20s for the third. On the right (b), the tick counter component designed to be connected in the hole *cnt*.

In Figure 5.3 is shown the result of the composition of the main system, namely the traffic light system, with the example given of a counter. The result of the composition is also an open automaton. In this case, the result of the composition is an open automaton with only one hole because the filling automaton has no hole. In general, the resulting automaton contains the number of holes of the encompassing automaton, plus those of the filling automaton minus one (the hole that is filled by the composition).

In Figure 5.4 is shown the result of the composition of the three automata. Each state of the result of the composition consists of a state of traffic light system together with a state of controller component and one of counter component. The composed automaton takes over the same steps as the traffic light automaton but it also includes new steps, indicating the change of states for the setting of timer. Its τ transitions involve both the traffic light automaton and the holes automata, they correspond to a joint step of sending time thresholds of the controller, the time setting of the counter. For instance, the τ open transition starting from $R1S$ is the result of the composition of three open transitions: the composition of the open transition of the traffic light automaton $\frac{\{cnt{\mapsto}\text{set}(x), ctl{\mapsto}\theta(x)\}, \top, \{\}}{R \xrightarrow{\tau} R}$, the open transition of the controller $\frac{\{\}, \top, \{\}}{1 \xrightarrow{\theta(17)} 2}$, and the open transition of the counter $\frac{\{\}, \top, \{t \leftarrow x, c \leftarrow 0\}}{S \xrightarrow{\text{set}(x)} C}$.

It is important to emphasize, as one can notice, that not all open transitions are composable. For instance, the open transition $\frac{\{cnt{\mapsto}\text{set}(x), ctl{\mapsto}\theta(x)\}, \top, \{\}}{R \xrightarrow{\tau} R}$ of traffic light system, cannot be composed with the controller transition $\frac{\{\}, \top, \{\}}{2 \xrightarrow{\delta(x)} 3}$. The second does not meet the expectations of the first. Indeed, the first open transition expresses that the traffic light system is waiting for the controller to execute action $\theta(x)$. On its side, the controller by the considered open transition transition offers the action $\delta(x)$, so the composition cannot be performed. In the technical definition (presented in the paper below) and the tool, the composition will produce an open transition whose guard is unsatisfiable, it will then be discarded.

Through this example, we have shown that the semantics of partially defined systems (partial processes) can be conveniently expressed via open labelled transition systems whose states are terms over a certain



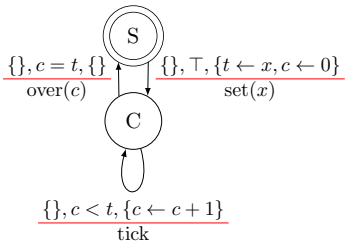


Figure 5.2: (a) An example of controller component (b) An example of counter component

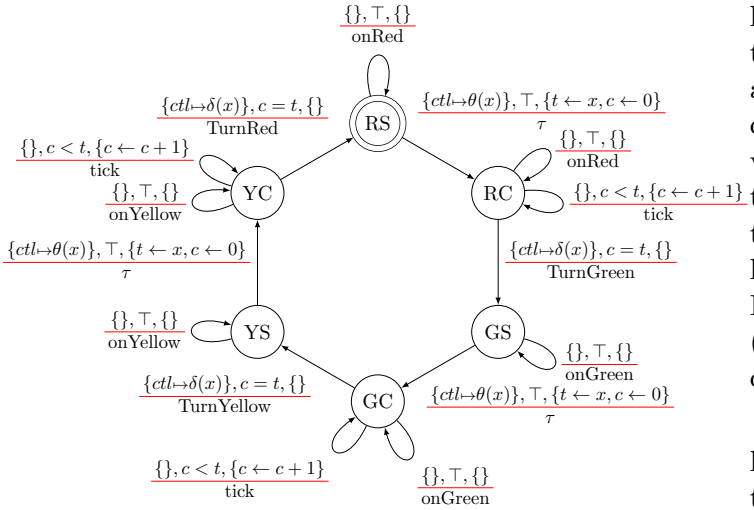


Figure 5.3: The partially instantiated of the Traffic Lights system

algebra and whose labels describe some abstract behavioural information. This way of composing open automata, by adding information (and details) to the enclosing automaton from the filling open automata is comparable to the notion of refinement which also consists in adding implementation details, which should ideally imply compatibility.

The key feature of the refinement techniques is that they enable incremental reasoning. Indeed, they provide a step-wise refinement process for developing systems by starting with a high-level specification of what the system is required to do. The simplified version of the system is then refined into a corresponding concrete version by gradually adding details and functionalities.

It appears clearly that the open automata provides a foundation for the analysis of systems whose behaviour interacts with the environment. It also emerges that this formalism supports incremental specification and verification methodologies through refinement and composition. We believe powerful tools could be developed using this formalism. Tools that can support the vertical and the horizontal dimension of compositional verification in an uniform and compatible way, helping to improve the capabilities of existing tools. For instance, for the analysis of reactive applications (e.g. [45]) or for the adaptation and integration of open components (e.g. [89, 101]).

The paper included in this chapter reports the preliminary results of our research work that investigate the potential of open automata for the refinement approach and the incremental verification techniques. In support of this idea of incremental verification, we define a refinement relation for open automata that has the following characteristics:

- Classical simulation characterisation but also an additional criterion ensuring that refinement does not introduce deadlocks when following a trace from the simulated automaton.

- Good properties in regards to composition: we proved that filling the same hole with the same automaton preserves the refinement relation.

- Ability to take into account both composition and transitivity: this is a major issue because composition changes the set of holes of the open automaton and refinement takes into account the actions of the holes.

More specifically, on the basis of the defined refinement relation over open automata, we have proved the

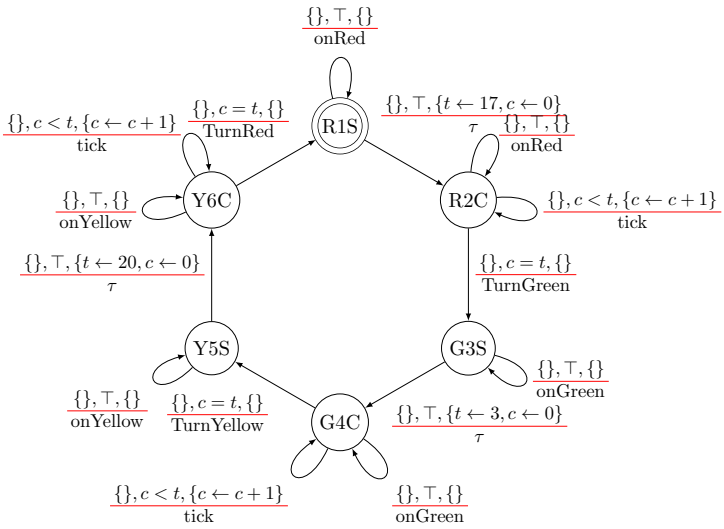


Figure 5.4: The full Traffic Lights system

transitivity and compositionality properties. These properties imply the independent implementability, which is the core features of the incremental component-based approach. More details and technical definitions about this relation are given in the enclosed paper. The refinement relation that we discuss in this paper is called *extension* relation in the literature (e.g. [58]) . This was because in this kind of refinement, the specification is a partial specification and the implementation (concretisation) extends the specification to comply with other constraints, not mentioned in the partial specification. This relation is the complement of reduction relation, which on the contrary is used to reduce nondeterminism in the specification. The notion of extension was introduced in [36] to deal with partial specifications in LOTOS [30]. And it has been used in other work on the context of incremental modelling approach (e.g. [88, 102]).

The principle of this refinement is that it enables the implementation to exhibit more behaviour than the specification, provided that the new traces are built only on the new actions, i.e., the filling automaton can introduce completely different new actions. Meaning that the implementation deadlocks less often than the specification for the traces that they have in common, since the implementation cannot refuse more than the specification for those traces.

5.2 Paper Accepted to SEFM

Refinements for Open Automata

Rabéa Ameur-Boulifa[1][0000−0002−2471−8012], Quentin Corradi[2][0000−0003−4218−3987], Ludovic Henrio[2][0000−0001−7137−3523], and Eric Madelaine[3][0000−0002−5552−5993]

[1] LTCI, Télécom Paris, Institut Polytechnique de Paris, France
first.last@telecom-paris.fr
[2] Université Lyon, EnsL, UCBL, CNRS, Inria, LIP, France
first.last@ens-lyon.fr
[3] INRIA Sophia Antipolis Méditérannée, UCA, France
first.last@inria.fr

Abstract. Establishing equivalence and refinement relations between programs is an important mean for verifying their correctness. By establishing that the behaviours of a modified program simulate those of the source one, simulation relations formalise the desired relationship between a specification and an implementation, two equivalent implementations, or a program and its optimised implementation. This article discusses a notion of simulation between *open automata*, which are symbolic behavioural models for communicating systems. Open automata may have *holes* modelling elements of their context, and can be composed by instantiation of the holes. This allows for a compositional approach for verification of their behaviour.
We define a simulation between open automata that may or may not have the same holes, and show under which conditions these refinements are preserved by composition of open automata.

Keywords: Labelled transition systems · Simulation · Composition.

1 Introduction

Compositional design is a highly convenient approach for specifying and verifying large systems. Automata are often used as the basic formalism for this approach, but most automata definitions allow only the specifications of finite closed systems. These systems can be verified efficiently, but programming often consists in writing systems that should be interfaced with others, and with potentially unbound behaviours. We investigate in our works the reasoning on open symbolic systems, with a strong focus on compositionality of properties. More precisely, we say that a system is open if it contains a "hole" to be filled by another system. Open systems are typically composition operators [16] or componentised systems where some of the components are yet to be provided [6]. This form of composition is more complex to handle than top-level interaction usually found in process algebra, as the behaviour of each entity in the system is parameterised both by classical symbolic variables and by process variables.

Symbolic systems and their bisimulation raises additional challenges [15,16]. Reasoning on a symbolic automaton allows one to represent an infinite system in a finite manner, but then the state of the system is not only characterised by an automaton state but also by the value of the different variables representing the system. In parameterised systems, it is necessary to guard state transitions depending on the system state and on the input values. This is why in previous works and in this article, it we extend the classical form of bisimulation relation: in a symbolic setting a bisimulation relation relates classically states of two systems but it is additionally parameterised by a formula that must be verified by the state variables. This has been introduced in details in previous works [6] and will be recalled briefly in Section 2.2. We have shown in previous works that open symbolic systems are particularly convenient to model process algebra operators and open component systems with infinite behaviour [6,16].

The refinement concept plays an important role in software engineering. In addition to helping to cope with the complexity of requirements and design, refinement provides a foundation for ensuring system correctness. The correctness of a system can be established by proving, that a system refines its specification with the idea that some properties of the specification are preserved in the refined system. Refinement entails that one system can be considered as a more precise version of another one that is considered to be the specification. The refined model features all the specified behaviours with more concrete details. From a formal point of view, refinement is a mathematical relations between a specification and its implementation, with trace inclusion or simulation being frequently used relations [21,19].

In this article, we design a simulation theory for open symbolic systems. We build a very generic theory that should allow us to reason on simulation-based verification for most concurrent systems, as our base theory merely relies on automata parameterised by both variables and processes. As we shall see, our composition of automata is also very generic to account for any interaction mechanism found in concurrent systems. While our contribution is theoretical, it establishes the foundations for to the verification of any compositionally designed system, like component systems, algorithmic skeletons.

Open automata (that we abbreviate OA) were defined as a way to provide a semantics for open parameterised hierarchical labelled transition systems (abbreviated LTS). They were proposed as a theoretical foundation for parametrised automata used in verification tools and called *pNets*. An OA [16] is similar to a classical automaton but with variables and holes. Variables make automata symbolic and allow them to encode infinite-state systems. Holes enable the composition of automata: an automaton with a hole is an operator that takes another automaton as parameter and reacts to the actions it emits; the composed automaton is an automaton where the behaviour of one "process parameter" of the main automaton has been provided. Due to their generic nature, the notion of OA model is quite abstract but we already illustrated previously how to derive OAs for process algebra operators [16] or for component systems [6,5].

In previous works [6,23] a bisimulation relation was defined for OA and open parameterised hierarchical LTSs. It exhibited good properties concerning bisimulation, but refinement relations were not studied. In this article we go further to define a theory of simulation for OA. The simulation relation we introduce in the paper is based on the notion of simulation, in a similar way to that defined in classical automata theory [20,8]. It possesses the common behaviour-preserving property: all the behaviour of the abstract specification must be followed by its (complex) implementation but additional behaviours may exist. However we also ensure that a whole scenario, made of several steps, of the specification can also be simulated by the refined system, which is slightly richer than the traditional simulation relation and allows us to obtain a compositionality result.

Our contribution in this paper is the definition of a simulation relation for OA that has the following characteristics:

- Classical simulation characterisation but also an additional criteria ensuring that simulation does not introduce deadlocks when following a trace from the simulated automaton.
- Good properties relatively to composition: we prove that composition preserves the simulation relation.
- Ability to take into account both composition and transitivity: this is a challenge because composition changes the set of holes of the OA and simulation takes into account the actions of the holes.

The simulation relation is introduced in two steps. First we define a simulation that relates two automata with the same holes, which allows us to focus on the automaton aspect. Second we introduce a relation that relates two automata with different sets of holes, which allows us to take into account the open nature of OA, and to deal with composition. Properties of the simulation are stated and proven on the second, more general version of the relation, thus also being valid for the first simpler simulation relation.

This paper is organized as follows. Section 2 recalls the definition of OA and defines their composition. We then define a simulation relation for OA, first only considering two automata with the same set of holes in Section 3 and generalize it to automata with a different set of holes in Section 4. Section 5 is dedicated to formalize and prove basic properties of the simulation defined, including the proof that simulation is a preorder and has nice composability properties. In Section 6 we review related works, and Section 7 concludes the paper.

2 Open Automata and their Composition

This section presents our notations and the principles of automata. Except for minor changes in the notations, compared to previous works [6] the only new contribution is the definition of a composition operator for OA.

2.1 Preliminaries and notations

Countable families of values (equivalent to maps) will be noted $x_i^{i\in I}$, $\{i \mapsto x_i \mid i \in I\}$, or $\{i \leftarrow x_i \mid i \in I\}$, depending on what is more convenient (e.g. $i \leftarrow x_i$ is used

for maps that are used as substitution). Statements like $\exists c_j^{j\in J}$ defines both J and the mapping $j \mapsto c_j$. The disjoint union on sets is noted $\uplus$. Disjoint union is also used on maps. There are several ways of ensuring a union is disjoint, we will indifferently either suppose sets are disjoint or rename conflicting objects (useful for variables). In a formula, a quantifier followed by a finite set will be used as a shorthand for the quantification on every variable in the set: $\forall\{a_1,\ldots,a_n\},\exists\{b_1,\ldots,b_m\},P$ means $\forall a_1,\ldots,\forall a_n,\exists b_1,\ldots,\exists b_m,P$.

Our expression algebra E is the disjoint union of terms, actions, and formulas $E = \mathcal{T} \uplus \mathcal{A} \uplus \mathcal{F}$. $\mathcal{T}$ and $\mathcal{A}$ are term algebras. The set of formulas $\mathcal{F}$ contain at least first order formulas and equality[4] over $\mathcal{T}$ and $\mathcal{A}$. For $e \in E$, $vars(e)$ is the set of variables in e that are not bound by a binder. An expression is closed if $vars(e) = \varnothing$. The set $\mathcal{P}$ denotes values which is a subset of closed terms. $\mathcal{F}_V$ is the set of formulas f that only uses variables in V, i.e., the formulas such that $vars(f) \subseteq V$. The parallel substitution of variables in e by a map $\psi : V \to \mathcal{T}$ is denoted $e\{\!\{\psi\}\!\}$.

We suppose given a satisfiability relation on closed formulas, denoted $\models f$. We will use two variants of the satisfiability relation:

- The satisfiability of a formula $f \in \mathcal{F}$ under some valuation $\sigma : V \to \mathcal{P}$ is defined as follows: $\sigma \models f \iff \vdash \exists vars(f\{\!\{\sigma\}\!\}), f\{\!\{\sigma\}\!\}$
- The satisfiability of a formula $f \in \mathcal{F}$ with some variable set V as context is defined as follows: $V \vdash f \iff \vdash \forall V, \exists(vars(f) \setminus V), f$

2.2 Open Automata (OA)

OA are labelled transition systems with variables that can be used to compose other automata: they are made of transitions that are dependent on the actions of "holes", a composition operation consists in filling a hole with another automaton to obtain a more complex automaton. The variables makes the OA symbolic, and the holes allow for a partial definition of the behaviour.

Definition 1 (Open transition, Open automaton). *An* open automaton *is a tuple $\langle S, s_0, V, \sigma_0, J, T\rangle$ with S a set of states, $s_0 \in S$ the initial state, V the finite set of variable names, $\sigma_0 : V \to \mathcal{P}$ the initial valuation of variables, J the set of hole names and T the set of open transitions.*

An open transition *is a structure* $s \xrightarrow{\alpha} s'$ *with* $\beta_j^{j\in J'}, g, \psi$ *made of several composing entities, equivalent to a tuple. In an open transition $s, s' \in S$ are the source and target states, $\alpha \in \mathcal{A}$ is the resulting action that can be observed from the outside, $J' \subseteq J$ are the holes involved in the transition, $g \in \mathcal{F}$ is the guard that may constraint the transition, and $\psi : V \to \mathcal{T}$ are the variable assignments that have an effect on the state of the automaton. Each $\beta_j \in \mathcal{A}$ is an action of the holes j, To be well-formed, an open transition should use only variables of the automaton*

[4] Equality does not need to be only syntactic.

and variables appearing in the involved actions, formally:

$$vars(g) \subseteq vars(\alpha) \cup \bigcup_{j \in J'} vars(\beta_j) \cup V$$
$$\forall v \in V.\, vars(\psi(v)) \subseteq vars(\alpha) \cup \bigcup_{j \in J'} vars(\beta_j) \cup V$$

A pair consisting of a state and a valuation is called a *configuration* (of the automaton). We use two operators to access pieces of information of the OA.

Definition 2 (Out-transition, Transition variables). *Let $\langle S, s_0, V, \sigma_0, J, T\rangle$ be an automaton and let r be a state in S. $\mathrm{OT}_T(r) \subset T$ are the transition outgoing from state r[5]. The local variables of a transition $vars(t)$ are all variables appearing in transition t except the variables of the automaton. Outgoing transitions and variables are formally defined as follows.*

$$\mathrm{OT}_T(r) = \left\{ \begin{array}{c} \beta_j^{j\in J'}, g, \psi \\ s \xrightarrow{\alpha} s' \end{array} \in T \;\middle|\; s = r \right\}$$

$$vars\left(\begin{array}{c} \beta_j^{j\in J'}, g, \psi \\ s \xrightarrow{\alpha} s' \end{array} \right) = \left(vars(\alpha) \cup \bigcup_{j \in J'} vars(\beta_j) \right) \setminus V$$

Example 1 (`prod-cons`). As a running example, we consider a classical producer-consumer pair interacting through FIFO buffer, named `prod-cons`. Fig. 1 reflects the overall structure of the system involving a producer process, a consumer process and an orchestrator that coordinates their activities.

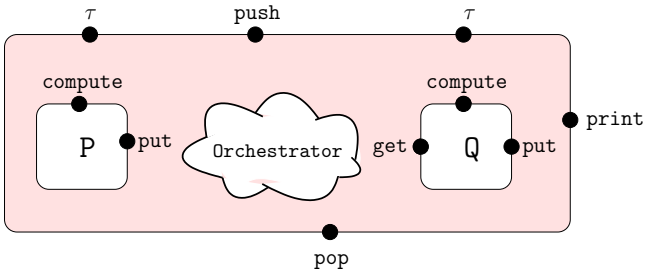


Fig. 1. Structure of the example. Each box corresponds to a process whose ports are the actions it can perform. The actions observable by the environment are `push`, which indicates the enqueuing of an element, `pop` which indicates the dequeuing, and `print` which indicates the production of results.

The OA modelling the behaviour of such a system using an unbounded circular/ring buffer is depicted in Fig. 2. The automaton has a single state with

[5] When the set T is clear from the context, it will be omitted and we will use $\mathrm{OT}(r)$

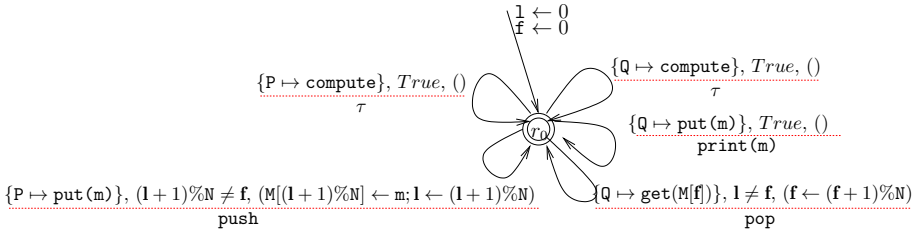


Fig. 2. OA for the `prod-cons` system using FIFO circular buffer.

two holes: `P` and `Q` that are the two interacting processes. `l` (as last) indicates the next available position for enqueuing an element and `f` (as first) is the position that contains the next element to be dequeued. The buffer reacts to a push from `P` and enqueues it. Similarly, whenever `Q` pops an element, it dequeues it. Additionally, whenever `Q` produces an item, it is exposed as an external observable `print` action. When any process do its internal computation, it is exposed externally as unobservable action τ.

The example uses several kinds of data. Variable `m` holds a message (we can leave the message type abstract here). We additionally use arrays of messages with a syntax of the form `M[l]` for array accesses; `M` is an array of `N` elements, from 0 to `N` − 1. Finally we use addition and modulo operation (%) on integers. □

Open automata composition. OA are partially specified automata, the partiality arises from the holes. A hole can be seen as a port in which we can plug an OA. The plugging operation is called composition. The composition of OA was already implicitly defined by the means of composition on pNets in previous work [16]. We provide here a (new) direct definition of composition for OA.

Definition 3 (Composition of OA). *Let $A_c = \langle S_c, s_{0c}, V_c, \sigma_{0c}, J_c, T_c\rangle$ be an OA and k one of its holes, $k \in J_c$. Let $A_p = \langle S_p, s_{0p}, V_p, \sigma_{0p}, J_p, T_p\rangle$ be another OA, the composition $A_c[A_p/k]$ that fills the hole k of the context OA A_c with the parameter OA A_p is defined as follows:*

$$A_c[A_p/k] ::= \langle S_c \times S_p, (s_{0c}, s_{0p}), V_c \uplus V_p, \sigma_{0c} \uplus \sigma_{0p}, J_p \uplus J_c \setminus \{k\}, T\rangle$$

with

$$T = \left\{ \frac{\beta_j^{j\in J'_p \uplus J'_c}, g_c \wedge g_p \wedge \alpha_p = \beta_k, \psi_c \uplus \psi_p}{(s_c, s_p) \xrightarrow{\alpha_c} (s'_c, s'_p)} \;\middle|\; \frac{\beta_j^{j\in J'_c \uplus \{k\}}, g_c, \psi_c}{s_c \xrightarrow{\alpha_c} s'_c} \in T_c, \frac{\beta_j^{j\in J'_p}, g_p, \psi_p}{s_p \xrightarrow{\alpha_p} s'_p} \in T_p \right\}$$
$$\cup \left\{ \frac{\beta_j^{j\in J'_c}, g_c, \psi_c}{(s_c, s_p) \xrightarrow{\alpha_c} (s'_c, s_p)} \;\middle|\; \frac{\beta_j^{j\in J'_c}, g_c, \psi_c}{s_c \xrightarrow{\alpha_c} s'_c} \in T_c, k \notin J'_c, s_p \in S_p \right\}$$

The first OA decides when the second can evolve by involving its hole in a transition: the action emitted when A_p makes a transition is synchronised with

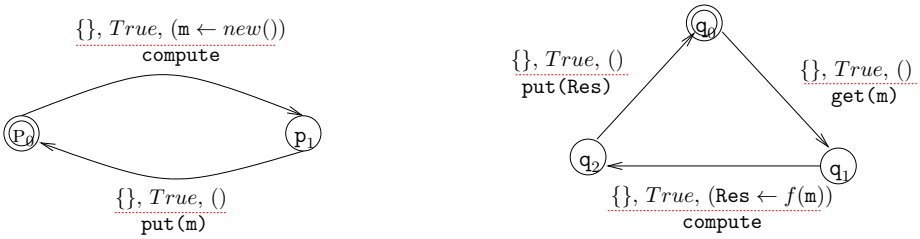


Fig. 3. (Left) A `producer`. It produces one item at a time and pushes it. (Right) A `consumer`. It pops an item, does some work and pushes the result.

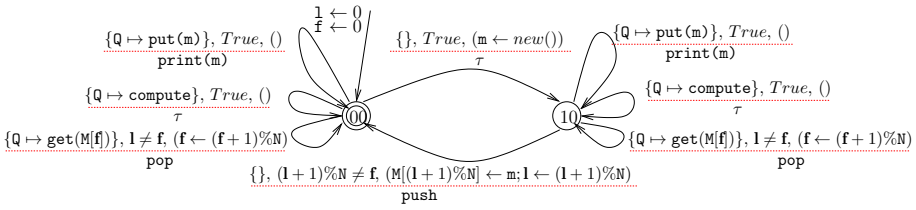


Fig. 4. OA for filling the hole `P` in `prod-cons`: `prod-cons`[`P`/`producer`].

the action of the hole k in transitions of A_c. The condition $\alpha_p = \beta_k$ ensures that the action emitted by the automaton A_p filling the hole is the one expected in the hole k of the open automaton A_c.

Example 2. Fig. 3 shows a producer automaton and a consumer automaton that can be used to fill the holes `P` and `Q` of `prod-cons` defined in Example 1.

The OA on Fig. 4 is the composition of the system in Fig. 2 and the producer in Fig. 3 (left). The composition consists of two states (the product of the states of both automata). The transitions from one state to another come from the synchronisation of the transitions of the encompassing automaton with those of the producer filling the hole `P`, this is why there is no more action from hole `P` in the composed automaton. Only elements related to the hole `P` are changed and in particular, transitions involving `Q` remain unchanged. □

2.3 Relations between Open Automata

Establishing semantic equivalences and simulation relations between different OA requires to compare their states. For this purpose, we suppose that the variables of the two OA are disjoint (a renaming of variables may have to be applied before comparing OA states).

Definition 4 (Relation on open automata configurations). *Suppose V_1 and V_2 are disjoint. A relation on configurations of two OA $\langle S_1, s_{01}, V_1, \sigma_{01}, J_1, T_1\rangle$ and $\langle S_2, s_{02}, V_2, \sigma_{02}, J_2, T_2\rangle$ is a function $\mathcal{R} : S_1 \times S_2 \to \mathcal{F}_{V_1 \uplus V_2}$.*

The idea is that two states are related depending on the satisfiability of the expression relying their variables, i.e., if the variables of the OA verify a certain formula. In other words, to each pair of states is attached a boolean formula that may refer to the variables of each of the two OA, stating whether the two states are related or not. Additionally, we say that the relation $\mathcal{R}$ *relates the initial states* of the automata if: $\sigma_{01} \uplus \sigma_{02} \vdash \mathcal{R}(s_{01}, s_{02})$. We illustrate such a relation over automata with bisimulation relation below.

2.4 A Bisimulation for Open Automata

Bisimulation between OA was defined in [6]. We show below the principles of this bisimulation. We first recall the usual definition of bisimulation. Bisimulation can be defined as follows for standard transition systems:

Definition 5 (Classical Bisimulation). *A bisimulation is a relation $\mathcal{R}$ such that if $s\ \mathcal{R}\ t$ then:*

$$\forall l\ s',\ s \xrightarrow{l} s' \implies \exists t'.\ s'\ \mathcal{R}\ t' \wedge t \xrightarrow{l} t'$$

and conversely

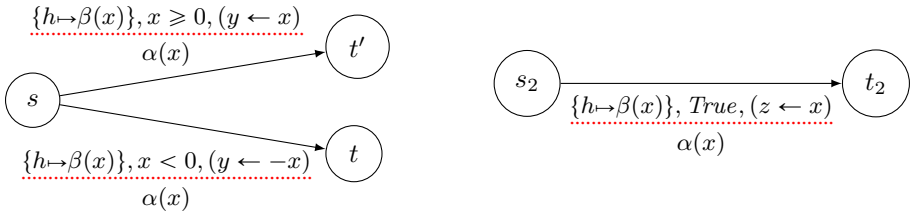
$$\forall l\ t',\ t \xrightarrow{l} t' \implies \exists s'.\ s'\ \mathcal{R}\ t' \wedge s \xrightarrow{l} s'$$

i.e.

$$\begin{array}{ccc} s & \mathcal{R} & t \\ l\downarrow & & \downarrow l \\ s' & \mathcal{R} & t' \end{array}$$

s and t are bisimilar, written $s \sim t$ iff there is a bisimulation relation $\mathcal{R}$ such that $s\ \mathcal{R}\ t$. If only the first one of the two implications above is verified, we say that s simulates t and denote it $s \leqslant t$.

A bisimulation relation relates pair of states and ensures that any behaviour of one automaton can be performed by the other one while staying in relation. We informally explain here the symbolic nature of the bisimulation for OA and the related complexity of its definition. The notion of symbolic bisimulation, as it was introduced in [15], is aimed at computing bisimulation of value-passing systems, i.e. systems made of processes exchanging data with their environment and between processes, where data are values from a possibly infinite domain. The presence of holes in fact raises no strong difficulty but the variables must be handled carefully. Consider the two following simple OA:



We should be able to consider these two OA as bisimilar. Both can input any $\beta(x)$ input on their hole and stores the value of x, emitting $\alpha(x)$ along the transition. The difference is the way x is stored. We can then define a configuration relation $\mathcal{R}$ such that $\mathcal{R}(s, s_2)$ is true and $\mathcal{R}(t, t_2)$ holds when $z \geqslant 0$ and

$y = z$, while $\mathcal{R}(t', t_2)$ holds when $z < 0$ and $y = -z$. This illustrates relation on configurations, but also shows that bisimulation on OA is more complex than in the classical case. Indeed, we need two transitions on the left OA to simulate a single one on the right OA. We should check that these two transitions cover all the cases accepted by the right hand side OA, and of course that destination states are in relation. Formally, FH-bisimulation is defined as follows [6]:

Definition 6 (Strong FH-bisimulation).
Suppose $\langle S_1, s_{01}, V_1, \sigma_{01}, J_1, T_1\rangle$ *and* $\langle S_2, s_{02}, V_2, \sigma_{02}, J_2, T_2\rangle$ *are OA with identical holes of the same sort, with disjoint sets of variables* ($V_1 \cap V_2 = \varnothing$).

Then $\mathcal{R}$*, a relation on configurations of OA, is an FH-bisimulation if and only if for any states* $s \in S_1$ *and* $t \in S_2$*, we have the following:*

- *For any open transition* OT *in* T_1*:* $\dfrac{\beta_j^{j\in J'}, g_{OT}, \psi_{OT}}{s \xrightarrow{\alpha} s'}$ *there exists an indexed set of open transitions* $OT_x^{x\in X} \subseteq T_2$*:* $\dfrac{\beta_{jx}^{j\in J_x}, g_{OT_x}, \psi_{OT_x}}{t \xrightarrow{\alpha_x} t_x}$ *such that the following holds*

$$\mathcal{R}(s,t) \wedge g_{OT} \implies \bigvee_{x\in X} \left(\forall j.\beta_j = \beta_{jx} \wedge g_{OT_x} \wedge \alpha = \alpha_x \wedge \mathcal{R}\left(s', t_x\right) \left\{\!\!\left\{\psi_{OT} \uplus \psi_{OT_x}\right\}\!\!\right\}\right)$$

- *and symmetrically any open transition from* t *in* T_2 *can be covered by a set of transitions from* s *in* T_1*.*

Two automata are bisimilar if there exists a strong FH-bisimulation $\mathcal{R}$ *that relates their initial states.*

Note that this definition matches an open transition t_1 to a family of covering open transitions $t_{2x}^{x\in X}$. Intuitively, this means that for every pair of related states (s_1, s_2) of the two automata, and for every transition of the first automaton from s_1, there is a set of matching transitions of the second automaton from s_2 such that the produced action match, the actions of the same holes and the successors are related after variable update. Technically, the following sections do not rely on the definition of strong bisimulation on OA, but they follow the same principles and in particular the same way to faithfully simulate an open transition by a set of other open transitions.

2.5 Reachability

We finally define a new predicate abstracting state reachability for OA, it allows us to reason on reachable states in an automaton. It can be seen as an abstraction of the reachable states under the form of a predicate that must stay verified along the execution of the OA.

Definition 7 (Reachability). *For any OA* $A = \langle S, s_0, V, \sigma_0, J, T\rangle$*, a reachability predicate* $\checkmark_A : S \to \mathcal{F}_V$ *is any predicate on states that is valid on initial state, and preserved across transitions:*

$$\sigma_0 \vdash \checkmark_A(s_0) \quad \wedge \quad \forall t = \frac{\beta_j^{j\in J'}, g, \psi}{s \xrightarrow{\alpha} s'} \in T, vars(t) \vdash \big(\checkmark_A(s) \wedge g \implies \checkmark_A(s')\{\!\{\psi\}\!\}\big)$$

Reachability takes into account all paths, and can over-approximate the reachable configurations. From an automation point of view, finding the most precise reachability predicate for a given automaton is not decidable because of the symbolic nature of OA, but only an over-approximation is necessary.

3 Simulation for Automata with the Same Holes

Similarly to FH-bisimulation [6] we are interested in finding simulation relations between configurations of two OA that contain variables and holes. When dealing with open systems it is common to define simulation in terms of a simulation relation. We rely on a classical notion of simulation and perform the same extension as in [6], i.e., we start from a simulation relation and add holes and symbolic. The idea is to consider two configurations related by a relation; if one state can do a transition, then the other can also make this transition. Like for bisimulation, a simulation relation characterises when two states are related, and this characterisation is expressed as a predicate on the variables of the two automata. Simulation defines conditions on a relation $\mathcal{R}$ such that $\mathcal{R}(s_1, s_2)$ is a predicate (possibly involving variables of the automata) that is true when the state s_1 of A_1 simulates the state s_2 of A_2.

However here we want to build a simulation relation that also guarantees that no deadlock is introduced when refining the automaton. This property is quite frequent in simulation relation, and referred to as *lack of new deadlocks* [19] or *complete simulation* [22]. The notion of deadlock should however be specialised to our OA. Indeed, it is not very useful to check the existence of a transition, instead it makes more sense to use the guards to check if a transition can be taken. We thus define a deadlock reduction criterion based on how the outgoing transitions are guarded. As such, a simulation does not introduce deadlocks if in the conditions where no transition is possible in the refined automaton, no transition were already possible in the more general one. More formally, for any pair of states s_1 and s_2 we introduce a criterion of the form:

$$\forall (s_1, s_2) \in S_1 \times S_2,$$
$$V_1 \uplus V_2 \vdash \Big(\mathcal{R}(s_1, s_2) \wedge \neg\Big(\bigvee_{t_1 \in \mathrm{OT}(s_1)} guard(t_1)\Big) \implies \neg\Big(\bigvee_{t_2 \in \mathrm{OT}(s_2)} guard(t_2)\Big)\Big)$$

Which can be rewritten as:

$$\forall (s_1, s_2) \in S_1 \times S_2, V_1 \uplus V_2 \vdash \Big(\mathcal{R}(s_1, s_2) \implies \Big(\bigvee_{t_1 \in \mathrm{OT}(s_1)} guard(t_1)\Big) \vee \neg\Big(\bigvee_{t_2 \in \mathrm{OT}(s_2)} guard(t_2)\Big)\Big)$$

Both statements being equivalent, as each of them may reveal more intuitive than the other in different situations, we use them interchangeably. We can now state the definition of simulation between OA that have the same set of holes.

Definition 8 (Hole-equal simulation). *Consider two OA with identical set of holes:* $A_1 = \langle S_1, s_{01}, V_1, \sigma_{01}, J, T_1\rangle$ *and* $A_2 = \langle S_2, s_{02}, V_2, \sigma_{02}, J, T_2\rangle$*, the relation on configurations* $\mathcal{R} : S_1 \times S_2 \to \mathcal{F}_{V_1 \uplus V_2}$ *is a hole-equal simulation from* A_1 *to* A_2 *if the following conditions hold :*

(1) $\sigma_{01} \uplus \sigma_{02} \vdash \mathcal{R}(s_{01}, s_{02})$

(2) $\forall (s_1, s_2) \in S_1 \times S_2$,

$$\forall t_1 = \frac{\beta_{1j}^{j\in J_1'}, g_1, \psi_1}{s_1 \xrightarrow{\alpha_1} s_1'} \in \mathrm{OT}(s_1). \quad \exists \left(t_{2x} = \frac{\beta_{2xj}^{j\in J_{2x}'}, g_{2x}, \psi_{2x}}{s_2 \xrightarrow{\alpha_{2x}} s_{2x}'} \right)^{x\in X} \in \mathrm{OT}(s_2).$$

$$\big(\forall x \in X, J_{2x}' = J_1'\big) \wedge$$

$$V_1 \uplus V_2 \uplus vars(t_1) \vdash \mathcal{R}(s_1, s_2) \wedge g_1 \implies \bigvee_{x\in X} \begin{pmatrix} \alpha_{2x} = \alpha_1 \wedge \bigwedge_{j\in J_1'} \beta_{2xj} = \beta_{1j} \wedge \\ g_{2x} \wedge \mathcal{R}\big(s_1', s_{2x}'\big) \{\!\!\{ \psi_{2x} \uplus \psi_1 \}\!\!\} \end{pmatrix}$$

(3) Deadlock reduction:

$$\forall (s_1, s_2) \in S_1 \times S_2, V_1 \uplus V_2 \vdash \Big(\mathcal{R}(s_1, s_2) \implies \Big(\bigvee_{t_1 \in \mathrm{OT}(s_1)} guard(t_1) \Big) \vee \neg \Big(\bigvee_{t_2 \in \mathrm{OT}(s_2)} guard(t_2) \Big) \Big)$$

If there is a hole-equal simulation from A_1 *to* A_2*, then we say that* A_2 *simulates* A_1*; we denote it* $A_2 \leqslant A_1$*.*

The first and second conditions coincide with the natural way to prove inductively that an automaton simulates another by starting with the initial state. The third condition ensures that simulation prevents the introduction of deadlocks. Similarly to bisimulation, the second condition states that, for any transition of the simulating automaton A_1, it corresponds to a transition of the automaton A_2 that does the same thing and ends up in a similar state. However a family is needed in A_2 because of the symbolic nature of transitions, and because depending on the values of the variables, t_1 may correspond to different transitions in A_2. Our definition captures a simple simulation for OA with the same holes that is more expressive than a strict simulation since it matches a transition with a family of transitions. For example, with such a relation we are able to check the simulation between two OA that differ by duplicated states, removed duplicated transitions, reinforced guards, different variables, etc. We will show in Section 5 that this simulation relation has good properties in terms of transitivity, compositionality, and reflexivity.

Example 3. To illustrate the simulation of OA, we consider a variation on the `prod-cons` example. Namely, we suppose that the two processes `P` and `Q` communicate through a one-place buffer. Fig. 5 shows the OA modelling this simpler version of the system, that we refer to as `simprod-cons`. We can easily

check that this automaton simulates the one of Fig. 2. Indeed, one can see that $\mathcal{R} = \{(r_0, s_0) \mapsto \texttt{l} = \texttt{f}, (r_0, s_1) \mapsto \texttt{f} = \texttt{l} + 1\%\texttt{N}\}$ is a simulation relation. It follows that `simprod-cons` $\leqslant$ `prod-cons`. □

The simulation relation defined above is insufficient in the setting of composition which is the main advantage of the OA-based approach. Indeed, it should be possible to refine an automaton by filling its hole, providing a concrete view of a part of the application that was not specified originally. More generally, it should be possible to relate automata that do not have the same holes because composition is a crucial part of system specification. However, filling holes can result in a system with more or less holes than the original system because the plugged subsystem can contain itself many holes. Next section defines a more powerful simulation relation able to reason on automata with different sets of holes.

4 A Simulation Relation that Takes Holes into Account

This section extends the preceding relation to automata where the set of holes is not the same. This is particularly useful to state whether the automaton after composition is a simulation of the original automaton or not. Indeed, when composing the set of holes changes. Being able to compare automata with only some of their holes in common seems useful in general.

One major challenge in the extension of simulation to different sets of holes is to maintain a form of transitivity while being able to take into account the actions of some of the holes. A naive definition of simulation would ensure that only the holes that are identical in the two OA are taken into account in the simulation. Unfortunately, considering all the common holes does not ensure transitivity of the simulation for the following reason. If A_1 simulates A_2 and A_2 simulates A_3, and one hole j appears in A_3 and in A_1 but not in A_2 then we have no guarantee on the way A_1 and A_3 take the actions of this hole into account, thus a simulation between and A_1 and A_3 would require conditions involving actions of the hole j which cannot be ensured. The way we solve this issue is to remember in the simulation relation which holes have been compared. This makes the relation parameterised by a subset of the set of holes that belong to the two automata that we want to take into account. This way, in the example

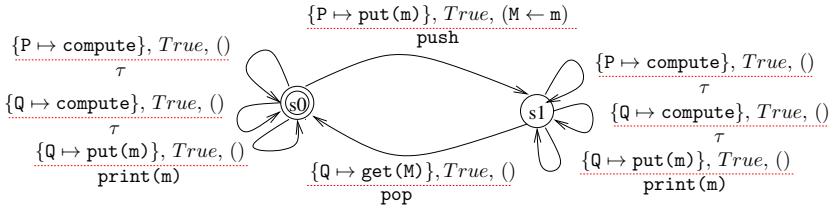


Fig. 5. The `simprod-cons` OA: the system using one-place buffer.

above, we would have no guarantee on actions the hole j by transitivity but can state a simulation relation with guarantees on the actions of the other holes.

In the following definition we add a parameter H which is the set of holes tracked by the simulation relation and adapt the definition by ignoring actions of the holes that are not in H.

Consequently, there is no guarantee related to the actions of the holes outside H. We provide compositionality properties when plugging an automaton inside a hole in H but cannot state anything when plugging an automaton outside H. The principle is that any property concerning holes that are not in H should be proven specifically for the considered automaton or the considered composition of automata.

Definition 9 (Hole-tracking simulation). *For two OA* $A_1 = \langle S_1, s_{01}, V_1, \sigma_{01}, J_1, T_1\rangle$ *and* $A_2 = \langle S_2, s_{02}, V_2, \sigma_{02}, J_2, T_2\rangle$*,* A_1 *is a simulation of* A_2 *tracking holes* H*, noted* $A_1 \leqslant_H A_2$*, with* $H \subseteq J_1 \cap J_2$*, if there is a relation on configurations* $\mathcal{R} : (S_1 \times S_2) \to \mathcal{F}_{V_1 \uplus V_2}$ *such that*[6]*:*

(1) $\sigma_{01} \uplus \sigma_{02} \vdash \mathcal{R}(s_{01}, s_{02})$

(2) $\forall (s_1, s_2) \in S_1 \times S_2$,

$$\forall \frac{\beta_{1j}^{j\in J_1'}, g_1, \psi_1}{s_1 \xrightarrow{\alpha_1} s_1'} \in \mathrm{OT}(s_1), \exists \left(\frac{\beta_{2xj}^{j\in J_{2x}'}, g_{2x}, \psi_{2x}}{s_2 \xrightarrow{\alpha_{2x}} s_{2x}'} \in \mathrm{OT}(s_2) \right)^{x\in X},$$
$$\big(\forall x \in X, J_{2x}' \cap H = J_1' \cap H\big) \wedge$$
$$V_1 \uplus V_2 \uplus \mathit{vars}(t_1) \vdash$$
$$\left(\mathcal{R}(s_1, s_2) \wedge g_1 \implies \bigvee_{x\in X} \left(\begin{array}{c} \alpha_1 = \alpha_{2x} \wedge \bigwedge\limits_{j\in J_1'\cap H} \beta_{1j} = \beta_{2xj} \wedge \\ g_{2x} \wedge \mathcal{R}\big(s_1', s_{2x}'\big)\{\!\{\psi_1 \uplus \psi_{2x}\}\!\} \end{array} \right) \right)$$

(3) Deadlock reduction:

$$\forall (s_1, s_2) \in S_1 \times S_2, V_1 \uplus V_2 \vdash \left(\mathcal{R}(s_1, s_2) \implies \Big(\bigvee_{t_1 \in \mathrm{OT}(s_1)} \mathit{guard}(t_1) \Big) \vee \neg \Big(\bigvee_{t_2 \in \mathrm{OT}(s_2)} \mathit{guard}(t_2) \Big) \right)$$

Note that every action of the holes outside H is unconstrained according to the simulation relation.

Property 1 (Relating simulations). Hole-equal simulation is a particular case of hole-tracking simulation when $J_1 = J_2 = H$.

In particular, if an OA has no hole, the two definitions are equivalent and result in a "symbolic simulation", if additionally there is no variable in the OA, this corresponds to classical simulation.

Example 4. Consider the automata of Examples 1 and 3. As we saw above, `simprod-cons` $\leqslant$ `prod-cons`, therefore `prod-cons` $\leqslant_{\{\mathtt{P},\mathtt{Q}\}}$ `simprod-cons`.

[6] Note that the definition below is identical to the hole equal simulation except $\cap H$ is added in a few places.

Property 2 (Tracked holes). By construction, if an automaton is the simulation of another one, it is also a simulation by tracking less holes.

$$A_1 \leqslant_H A_2 \wedge H' \subseteq H \implies A_1 \leqslant_{H'} A_2$$

Now that we have a simulation relation that takes both variable parameters and process parameters into account, we would like to ensure that it has properties one would expect for a simulation relation.

5 Properties of our Simulation Relations

Before reasoning on the properties of simulation, we need to introduce one additional notion that characterises when the composition of two automata does not introduce new blocked transitions.

5.1 Non-blocking Composition

Unfortunately, the deadlock reduction property in the definition of simulation is not compositional: the composition operator can itself introduce a deadlock. In other words, when filling the hole of two related automata with a third one, even if there is a deadlock reduction between the two original automata, there might not be a deadlock reduction in the composed ones. The same problem may arise when two related automata are composed in the same hole of a third one.

This creates a conflict between deadlock reduction and the properties involving composition. We call *non-blocking composition* a composition that can safely be used to compose OA that are involved in a deadlock reducing relation.

Definition 10 (Non-blocking composition). *Consider two OA:* $A_1 = \langle S_1, s_{01}, V_1, \sigma_{01}, J_1, T_1\rangle$ *and* $A_2 = \langle S_2, s_{02}, V_2, \sigma_{02}, J_2, T_2\rangle$. *Let* A *be the OA resulting from the composition* $A = A_1[A_2/k] = \langle S, s_0, V, \sigma_0, J, T\rangle$. *The composition* $A_1[A_2/k]$ *is non-blocking if* A *has a reachability predicate such that, for each reachable configuration, if there is a possible transition in* A_1 *then there is a possible transition in* A*:*

$$\forall s = (s_1, s_2) \in S, V \uplus \biguplus_{t \in \mathrm{OT}(s_1)} vars(t) \vdash \left(\checkmark_A(s) \wedge \bigvee_{t \in \mathrm{OT}(s_1)} guard(t) \implies \bigvee_{t \in \mathrm{OT}(s)} guard(t) \right)$$

Like in the definition of simulation (Definition 8) we use guards to ensure that the transition can occur. In general, one would not want to only consider non-blocking composition as it may reveal a bit restrictive, but it is the best necessary condition that we could identify for compositionality of simulation. It will be used to prove composition theorems given below. In absence of non-blocking composition, simulation may also be checked specifically for a given composed automaton.

5.2 Properties

We now state the properties of our simulation, their formal proofs can be found in the appendices. We express these properties in terms of hole-tracking simulation because, thanks to Property 1 all the properties of hole-tracking simulation are also valid for hole-equal simulation. The first crucial theorem of simulation is that it is a preorder on the set of OA. This latter enables stepwise refinement.

Theorem 1 (Simulation is a preorder). *Hole-tracking simulation is reflexive and transitive: it is a preorder on the set of OA.*

Proof sketch. The relation $\leqslant_H$ is reflexive, $A \leqslant_H A$. This is shown by considering the relation $\mathcal{R}$ such that $\mathcal{R}(s_1, s_2) \triangleq s_1 = s_2 \wedge \bigwedge_{v \in vars(s_1)} v = v$ we can prove the conditions for Definition 9. Appendix A gives the proof of transitivity. It is done classically by identifying the relation between A_1 and A_3 that is a simulation. What is less classical is the definition of this relation because it is a boolean formula. For each couple of states s_1 and s_3 of A_1 and A_3 we build a formula that defines the simulation. To do this, we take the disjunction of formulas relating s_1 and s_3, and passing by all states s_2 of A_2. More precisely, we define a relation of the following form:

$$\mathcal{R}_{13}(s_1, s_3) = \bigvee_{s_2 \in S_2} (\mathcal{R}_{12}(s_1, s_2) \wedge \mathcal{R}_{23}(s_2, s_3))$$

We then prove that this relation is a simulation, according to Definition 9. □

The next theorem states that if two automata are in simulation relation and the same automaton is placed in the same hole of the two automata, then the simulation is preserved. This is the first step toward proving that simulation is compositional in the sense that it is sufficient to prove simulation for the composed automata separately to obtain a simulation relation.

Theorem 2 (Context refinement). *Let A_1, A_2 and A_3 be three OA with $A_1 \leqslant_H A_2$. Let J_3 be the set of holes of A_3 and suppose that $k \in H$. Suppose additionally that $A_1[A_3/k]$ is non-blocking. We have:*

$$A_1[A_3/k] \leqslant_{J_3 \uplus H \setminus \{k\}} A_2[A_3/k]$$

Proof sketch. The proof relies on a simulation relation that we consider is the one that makes A_1 and A_2 similar, complemented with identity of configurations for A_3. Then, by construction, all transitions of the composed automaton $A_1[A_3/k]$ are specified by open transitions of A_1. For the transitions that do not involve hole k, the transition of $A_1[A_3/k]$ is the same and simulation between A_1 and A_2 allows us to conclude directly. If the hole k is involved the considered relation implies that valuations in A_3 are equal (i.e., the value for each variable are the same in both valuations), after a transition we should obtain "equal" valuations because post-conditions are deterministic. The requirement "$A_1[A_3/k]$ is non-blocking" ensures the deadlock reduction property holds. More precisely, if $A_1[A_3/k]$ is stuck, then A_1 is stuck, and thus $A_1[A_2/k]$ is also stuck. □

Example 5. Consider again the `prod-cons` and `simprod-cons` automata given in the examples above. Since `prod-cons` $\leqslant_{\{\texttt{P},\texttt{Q}\}}$ `simprod-cons`, then according to Theorem 2, `prod-cons[producer/P]` $\leqslant_{\{\texttt{Q}\}}$ `simprod-cons[producer/P]`. The automaton of `prod-cons[producer/P]` is shown in Fig. 4. The automaton resulting from the composition of `simprod-cons` and `producer` is bigger and not shown here. □

Theorem 3 (Congruence). *Let A_1, A_2 and A_3 be three OA with $A_2 \leqslant_H A_3$. Let J_1 be the set of holes of A_1 and suppose that $k \in J_1$. Suppose additionally that the composition $A_1[A_2/k]$ is non-blocking. We have:*

$$A_1[A_2/k] \leqslant_{J_1 \uplus H \setminus \{k\}} A_1[A_3/k]$$

Consequently, as the simulation is transitive we can compose the previous theorems and state the following:

Theorem 4 (Composability). *Let A_1, A_2, A_3 and A_4 be four OA with $A_1 \leqslant_H A_2$ and $A_3 \leqslant_{H'} A_4$. Suppose that $k \in H$. We have:*

$$A_1[A_3/k] \leqslant_{H \uplus H' \setminus \{k\}} A_2[A_4/k]$$

Example 6. As an example of the use of this theorem, if we design a refined version of the producer process of Example 2 called `Refproducer`. According to Theorem 4, we have `prod-cons[producer/P]` $\leqslant_{\{\texttt{Q}\}}$ `simprod-cons[Refproducer/P]`.

Note that the substitution operation can be extended to a multiple substitution that fills several holes at the same time, and the theorems can be adapted accordingly.

6 Related Work

The origins of refinement are in the approach of programming that aims to provide solid foundations for building correct programs [12]. Many work contributed to the development of elaborated notions of refinement in various area (e.g. [7,1,10,8]). In the context of process algebra, refinement between processes can be defined in terms of simulations relation (e.g. ([18,21]). However, the concept of simulations presented so far has focused on the refinement of systems that are inherently closed, i.e., systems which are bounded and without environment,

The simulation ensures the preservation of safety properties as deadlock-freeness and, more generally, all linear temporal logic properties [1,19]. The difference between the existing refinement principles have been studied in [13], for example the authors explain in what sense failure semantics is different from (bi)simulation in the compared systems and properties ensured. In this paper we particularly focus on the compositionality of simulation-based refinement.

There are not a lot of works that study refinement for open systems. Defining refinement of open systems as trace inclusion is addressed as a notion of subtyping in type theory (e.g. [14,9]). The definition of refinement is based on a

connection between session types and communicating automata theories – a notion of session automata based on Communicating Finite-State Machines, that are used for modelling processes communicating through FIFO channels. The refinement of open systems is also defined in terms of alternating simulation [4,3]. Alternating simulation is originating from the game theory [2], it allows the study of relation between individual components by viewing them as alternating transition systems. In particular, a refinement of game-based automata expresses that the refined component can offer more services (input actions) and fewer service demands (output actions). However, the composition of such automata may lead to illegal states, where one automaton issues an output that is not acceptable as input in the other one. The theory of alternating simulation provides an optimistic approach to compute compatibility between automata based on the fact that each automaton expects the other to provide legal inputs, i.e., two components can be composed if there is an environment where they can work together. Our approach has some commonalities with the above mentioned simulation [3]: both are process-oriented approaches even if they are not based on the same notion of simulation, and both include in the model how to compose and interact with processes that are accepted as parameters. Nevertheless, they differ in that our approach focuses on the compositional properties of the simulation, and not on the fact that entities can be composed.

Previous works on OA focused on equivalence relations compatible with composition. In [17], a computable bisimulation is introduced, while in [6] a weak version of the bisimulation is introduced. In this paper we tackle the refinement relation in the form of simulation, as is the case for the corresponding relations on labelled transition systems [8]. Unlike the standard simulation we deal with symbolic and open models. In [24], the authors exploit transition systems to reason about the systems that are partially specified by using variables, making the state space potentially infinite.

Some work target component-based refinement with the concern of preserving deadlock freedom (e.g. [11,19]). These works are not concerned with the theory of open symbolic systems, and therefore do not focus on the same modularity as we do, in particular we provide preservation of refinement by composition.

7 Conclusion

In this article we investigated the notion of refinement for a symbolic and open model: open automata. OA are convenient for compositional software verification. Indeed, OA model parallel systems that are parameterised both by the use of variables and by the possibility to compose automata. The formalism supports compositional specification through the simulation paradigm. In this paper, we introduce a refinement relation between open automata. It relies on a simulation relation between the two automata; it specifies that the refined process must follow the behaviour of the simulated one. We finally showed that simulation is a preorder that is preserved by composition, both when filling a hole and when placing automata in comparable contexts.

References

1. Abrial, J.R.: The B-book - Assigning programs to meanings. Cambridge University Press (1996)
2. de Alfaro, L.: Game models for open systems. In: Dershowitz, N. (ed.) Verification: Theory and Practice, Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday. Lecture Notes in Computer Science, vol. 2772, pp. 269–289. Springer (2003). https://doi.org/10.1007/978-3-540-39910-0_12
3. de Alfaro, L., Henzinger, T.A.: Interface automata. In: Tjoa, A.M., Gruhn, V. (eds.) Proceedings of the 8th European Software Engineering Conference held jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering 2001, Vienna, Austria, September 10-14, 2001. pp. 109–120. ACM (2001). https://doi.org/10.1145/503209.503226
4. Alur, R., Henzinger, T.A., Kupferman, O., Vardi, M.Y.: Alternating refinement relations. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR'98 Concurrency Theory. pp. 163–178. Springer Berlin Heidelberg, Berlin, Heidelberg (1998)
5. Ameur-Boulifa, R., Henrio, L., Kulankhina, O., Madelaine, E., Savu, A.: Behavioural semantics for asynchronous components. Journal of Logical and Algebraic Methods in Programming **89**, 1–40 (2017). https://doi.org/https://doi.org/10.1016/j.jlamp.2017.02.003, https://www.sciencedirect.com/science/article/pii/S2352220817300287
6. Ameur-Boulifa, R., Henrio, L., Madelaine, E.: Compositional equivalences based on Open pNets. Journal of Logical and Algebraic Methods in Programming **131**, 100842 (2023). https://doi.org/https://doi.org/10.1016/j.jlamp.2022.100842, https://www.sciencedirect.com/science/article/pii/S2352220822000955
7. Back, R., Sere, K.: Stepwise refinement of parallel algorithms. Science of Computer Programming **13**(2), 133–180 (1990). https://doi.org/https://doi.org/10.1016/0167-6423(90)90069-P, https://www.sciencedirect.com/science/article/pii/016764239090069P
8. Bellegarde, F., Julliand, J., Kouchnarenko, O.: Ready-simulation is not ready to express a modular refinement relation. In: Maibaum, T. (ed.) Fundamental Approaches to Software Engineering. pp. 266–283. Springer Berlin Heidelberg, Berlin, Heidelberg (2000)
9. Bravetti, M., Zavattaro, G.: Asynchronous session subtyping as communicating automata refinement. Softw. Syst. Model. **20**(2), 311–333 (2021). https://doi.org/10.1007/s10270-020-00838-x
10. Butler, M.J., Grundy, J., Långbacka, T., Ruksenas, R., von Wright, J.: The refinement calculator: Proof support for program refinement. In: Groves, L., Reeves, S. (eds.) Proc. Conf. Formal Methods Pacific'97, Springer Series in Discrete Mathematics and Theoretical Computer Science (01/01/97). pp. 40–61 (1997), https://eprints.soton.ac.uk/250550/
11. Dihego, J., Sampaio, A., Oliveira, M.: A refinement checking based strategy for component-based systems evolution. Journal of Systems and Software **167**, 110598 (2020). https://doi.org/https://doi.org/10.1016/j.jss.2020.110598
12. Dijkstra, E.W.: A Discipline of Programming. Prentice-Hall (1976)
13. Eshuis, R., Fokkinga, M.M.: Comparing refinements for failure and bisimulation semantics. Fundam. Inf. **52**(4), 297–321 (Apr 2002)
14. Gay, S.J., Hole, M.: Subtyping for session types in the pi calculus. Acta Informatica **42**(2-3), 191–225 (2005). https://doi.org/10.1007/s00236-005-0177-z

15. Hennessy, M., Lin, H.: Symbolic bisimulations. Theoretical Computer Science **138**(2), 353–389 (1995). https://doi.org/https://doi.org/10.1016/0304-3975(94)00172-F, https://www.sciencedirect.com/science/article/pii/030439759400172F, meeting on the mathematical foundation of programing semantics
16. Henrio, L., Madelaine, E., Zhang, M.: A theory for the composition of concurrent processes. In: Albert, E., Lanese, I. (eds.) Formal Techniques for Distributed Objects, Components, and Systems. pp. 175–194. Springer International Publishing, Cham (2016)
17. Hou, Z., Madelaine, E.: Symbolic bisimulation for open and parameterized systems. In: Proceedings of the 2020 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation. pp. 14–26. PEPM 2020, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3372884.3373161
18. Jifeng, H.: Process simulation and refinement. Form. Asp. Comput. **1**(1), 229–241 (mar 1989). https://doi.org/10.1007/BF01887207, https://doi.org/10.1007/BF01887207
19. Kouchnarenko, O., Lanoix, A.: How to verify and exploit a refinement of component-based systems. In: Virbitskaite, I., Voronkov, A. (eds.) Perspectives of Systems Informatics. pp. 297–309. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
20. Milner, R.: Communication and Concurrency. Prentice-Hall, Inc., USA (1989)
21. Milner, R.: A Calculus of Communicating Systems, Lecture Notes in Computer Science, vol. 92. Springer (1980). https://doi.org/10.1007/3-540-10235-3
22. Sangiorgi, D.: Introduction to Bisimulation and Coinduction. Cambridge University Press (2012), https://hal.inria.fr/hal-00907026
23. Wang, B., Madelaine, E., Zhang, M.: Symbolic Weak Equivalences: Extension, Algorithms, and Minimization - Extended version. Research Report RR-9389, Inria & Université Cote d'Azur, CNRS, I3S, Sophia Antipolis, France ; East China Normal University (Shanghai) (Jan 2021), https://hal.inria.fr/hal-03126313
24. Zhang, L., Meng, Q., Lo, K.: Compositional abstraction refinement for component-based systems. Journal of Applied Mathematics **2014**, 1–12 (2014). https://doi.org/10.1155/2014/703098

A Proof of Transitivity for Refinement (Theorem 1)

If $A_1 \leqslant_H A_2$ and $A_2 \leqslant_{H'} A_3$, then $A_1 \leqslant_{H \cap H'} A_3$.

Proof. If $A_1 \leqslant_H A_2$ then there is $\mathcal{R}_{12}$ a relation between states of A_1 and of A_2; If $A_2 \leqslant_{H'} A_3$ then there is $\mathcal{R}_{23}$ a relation between states of A_2 and of A_3. We build a relation between states of A_1 and of A_3 as follows: for each pair of states s_1, s_3, for each state s_2 such that $\mathcal{R}_{12}$ relates s_1 and s_2, and $\mathcal{R}_{23}$ relates s_2 and s_3. Let $\mathcal{R}_{13}$ be the relation:

$$\mathcal{R}_{13}(s_1, s_3) = \bigvee_{s_2 \in S_2} (\mathcal{R}_{12}(s_1, s_2) \wedge \mathcal{R}_{23}(s_2, s_3))$$

We will show that $A_1 \leqslant_{H \cap H'} A_3$ by exhibiting $\mathcal{R}_{13}$ as a hole-tracking simulation of A_1 by A_3.

We have to prove that the relation $\mathcal{R}_{13}$ satisfies the three conditions of the definition of a refinement of OA.

1. Firstly, we have to $\mathcal{R}_{13}$ satisfies initial configurations:

   $$\sigma_{01} \uplus \sigma_{03} \vdash \mathcal{R}_{13}(s_{01}, s_{03})$$

   By knowing that substitutions only have an effect on the variables of the OA they belong to, they also produce terms containing only variables of the OA they belong to. We have:

   $$\begin{aligned} (\sigma_{01} \uplus \sigma_{02} \vdash \mathcal{R}_{12}(s_{01}, s_{02})) \wedge (\sigma_{02} \uplus \sigma_{03} \vdash \mathcal{R}_{23}(s_{02}, s_{03})) &\implies \\ \mathcal{R}_{12}(s_{01}, s_{02})\{\!\{\sigma_{01} \uplus \sigma_{02}\}\!\} \wedge \mathcal{R}_{23}(s_{02}, s_{03})\{\!\{\sigma_{02} \uplus \sigma_{03}\}\!\} &\implies \\ \mathcal{R}_{12}(s_{01}, s_{02})\{\!\{\sigma_{01} \uplus \sigma_{02} \uplus \sigma_{03}\}\!\} \wedge \mathcal{R}_{23}(s_{02}, s_{03})\{\!\{\sigma_{01} \uplus \sigma_{02} \uplus \sigma_{03}\}\!\} &\implies \\ \underbrace{\mathcal{R}_{12}(s_{01}, s_{02}) \wedge \mathcal{R}_{23}(s_{02}, s_{03})}_{\implies \mathcal{R}_{13}(s_{01}, s_{03})} \{\!\{\sigma_{01} \uplus \sigma_{02} \uplus \sigma_{03}\}\!\} & \end{aligned}$$

   Since σ_{02} has no effect on variables of s_{01} and s_{03} thus we get the expected result.
2. Secondly, we need to prove that for any open transition t_1 in T_1 originating from s_1:

   $$\frac{\beta_{1j}^{j \in J'_1}, g_1, \psi_1}{s_1 \xrightarrow{\alpha_1} s'_1} \in \mathrm{OT}(s_1)$$

   there exists an indexed family of OTs originating from s_3:

   $$V_1 \uplus V_3 \uplus \mathit{vars}(t_1) \vdash$$
   $$\left(R_{13}(s_1, s_3) \wedge g_1 \implies \bigvee_{z \in Z} \left(\begin{array}{c} \alpha_1 = \alpha_{3z} \wedge \bigwedge\limits_{j \in J'_{3z} \cap (H \cap H')} \beta_{1j} = \beta_{3jz} \wedge \\ g_{3z} \wedge R_{12}\big(s'_1, s'_{3z}\big)\{\!\{\psi_1 \uplus \psi_{3z}\}\!\} \end{array} \right) \right)$$

   The proof of this simulation step follows the same principles as the one given in [6] for the proof of transitivity of bisimulation[7]. It relies on the fact that

[7] In particular, the bisimulation relation exhibited for proving transitivity is the same.

one open transition of A_1 is simulated by a family of open transitions of A_2, and one open transition of A_2 is simulated by a family of open transitions of A_3. Transitivity leads to a doubly indexed family of simulating open transition.

3. Lastly, we have to prove the satisfaction of the deadlock reduction condition. To build $\mathcal{R}_{13}$ we need to rely on the disjunction of all possible paths to relate s_1 and s_3, which leads to

$$\mathcal{R}_{13}(s_1, s_3) = \bigvee_{p \in P} (\mathcal{R}_{12}(s_1, s_{2p}) \wedge \mathcal{R}_{23}(s_{2p}, s_3))$$

Consider any $(s_1, s_3) \in \mathcal{R}_{13}$ there is a set of states $(s_{2p})^{p \in P}$ of A_2 relating s_1 and s_3.
First, according to the relation between A_1 and A_2 OA, for all $(s_1, s_{2p}) \in S_1 \times S_2$ we have:

$$V_1 \uplus V_2 \vdash \mathcal{R}_{12}(s_1, s_{2p}) \implies \bigvee_{t_1 \in \mathrm{OT}(s_1)} guard(t_1) \vee \neg(\bigvee_{t_{2p} \in \mathrm{OT}(s_{2p})} guard(t_{2p}))$$

Second, according to the relation between A_2 and A_3 OA, for all $(s_{2p}, s_3) \in S_2 \times S_3$ we have:

$$V_2 \uplus V_3 \vdash \mathcal{R}_{23}(s_{2p}, s_3) \implies \bigvee_{t_1 \in \mathrm{OT}(s_{2p})} guard(t_{2p}) \vee \neg(\bigvee_{t_3 \in \mathrm{OT}(s_3)} guard(t_3))$$

With the conjunction of both, we get:

$$V_1 \uplus V_2 \uplus V_3 \vdash \mathcal{R}_{12}(s_1, s_{2p}) \wedge \mathcal{R}_{23}(s_{2p}, s_3) \implies \bigvee_{t_1 \in \mathrm{OT}(s_1)} guard(t_1) \vee \neg(\bigvee_{t_3 \in \mathrm{OT}(s_3)} guard(t_3))$$

This is valid for all $s_{2p} \in (s_{2p})^{p \in P}$, then we have:

$$V_1 \uplus V_2 \uplus V_3 \vdash \underbrace{\bigvee_{p \in P} (R_{12}(s_1, s_{2p}) \wedge R_{23}(s_{2p}, s_3))}_{R_{13}(s_1, s_3)} \implies (\bigvee_{t_1 \in \mathrm{OT}(s_1)} guard(t_1)) \vee \neg(\bigvee_{t_3 \in \mathrm{OT}(s_3)} guard(t_3))$$

By removing the A_2 variables that have not effect on A_2 et A_3, we get the desired result:

$$V_1 \uplus V_3 \vdash \mathcal{R}_{13}(s_1, s_3) \implies (\bigvee_{t_1 \in \mathrm{OT}(s_1)} guard(t_1)) \vee \neg(\bigvee_{t_3 \in \mathrm{OT}(s_3)} guard(t_3))$$

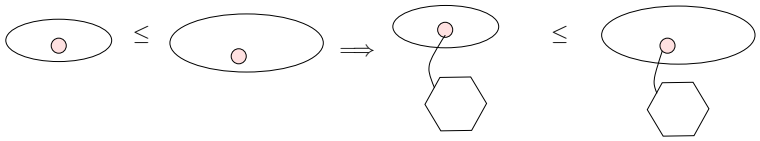
□

B Proof of Context Refinement (Theorem 2)

Suppose that $A_1 \leqslant_H A_2$, $k \in H$ and that $A_1[A_3/k]$ is non-blocking. We have:

$$A_1[A_3/k] \leqslant_{J_3 \uplus H \setminus \{k\}} A_2[A_3/k]$$

Pictorially, the theorem states the following result:



Proof. Let us denote by A_{13} (resp. A_{23}) the OA resulting from $A_1[A_3/k]$ (resp. $A_2[A_3/k]$), to prove the theorem it is sufficient to prove that there exists a relation between states of the two OA that satisfies the conditions of Definition 9. We denote $A_1 = \langle S_1, s_{01}, J_1, V_1, \sigma_{01}, T_1 \rangle$, $A_2 = \langle S_2, s_{02}, J_2, V_2, \sigma_{02}, T_2 \rangle$ and $A_3 = \langle S_3, s_{03}, J_3, V_3, \sigma_{03}, T_3 \rangle$. The proof requires to rename the variables of one instance of the two A_3 automata to avoid clashes in variable names (this is required by the definition of refinement). In practice we will use superscripts ^{1} and ^{2} to distinguish elements of the two instances of A_3.

Let $\mathcal{R}$ be the refinement relation relating states of A_1 and A_2. Let us denote with t^1 and t^2 the elements of A_1 and A_2 respectively. Consider any two states $s_{13} = \left(s_1, s_3^1\right)$ and $s_{23} = \left(s_2, s_3^2\right)$ (s_3^1 and s_3^2 are the same with renaming). We define a relation $\mathcal{R}'$ relating states of s_{13} and s_{23} as follows:

$$\mathcal{R}'(s_{13}, s_{23}) = \mathcal{R}(s_1, s_2) \wedge \checkmark_{A_{13}}(s_{13}) \wedge \bigwedge_{v_3 \in V_3} v_3^1 = v_3^2 \wedge s_3^1 = s_3^2$$

We want to prove that $(\mathcal{R}', H \uplus J_3 \setminus \{k\})$ is a hole-tracking simulation of A_{13} and A_{23}. In the following we denote $H' = H \cup J_3 \setminus \{k\}$.

1. First, we have to prove the relation for initial states:

   $$\sigma_{013} \uplus \sigma_{023} \vdash \mathcal{R}'(s_{013}, s_{023})$$

   with $\sigma_{013} = \sigma_{01} \uplus \sigma_{03}^1$, $\sigma_{023} = \sigma_{02} \uplus \sigma_{03}^2$, $s_{013} = (s_{01}, s_{03}^1)$, and $s_{023} = (s_{02}, s_{02}^2)$.
   By using the fact that $\mathcal{R}$ relates initial configurations of A_1 and A_2, we have:

   $$(\sigma_{01} \uplus \sigma_{02} \vdash \mathcal{R}(s_{01}, s_{02}))$$

   Considering that initial valuations σ_{03}^1 and σ_{03}^2 associate the same values to the "same" variables modulo renaming, so the following holds:
   $\left(\bigwedge_{v_3 \in V_3} v_3^1 = v_3^2 \wedge s_3^1 = s_3^2 \right) \{\!\{\sigma_{03}^1 \uplus \sigma_{03}^2\}\!\}$.

Additionally, because the domains of the substitution function are disjoint, the substitution function has an effect only on the related elements, we get:

$$
\begin{aligned}
&(\sigma_{01} \uplus \sigma_{02} \vdash \mathcal{R}(s_{01}, s_{02})) \\
\Longrightarrow\ &\mathcal{R}(s_{01}, s_{02})\{\!\!\{\sigma_{01} \uplus \sigma_{02}\}\!\!\} \wedge \left(\bigwedge_{v_3 \in V_3} v_3^1 = v_3^2 \wedge s_3^1 = s_3^2\right)\{\!\!\{\sigma_{03}^1 \uplus \sigma_{03}^2\}\!\!\} \\
\Longrightarrow\ &\left(\mathcal{R}(s_{01}, s_{02}) \wedge \bigwedge_{v_3 \in V_3} v_3^1 = v_3^2 \wedge s_3^1 = s_3^2\right)\{\!\!\{\sigma_{01} \uplus \sigma_{02} \uplus \sigma_{03}^1 \uplus \sigma_{03}^2 \uplus \sigma_{013}\}\!\!\} \\
\Longrightarrow\ &\sigma_{013} \uplus \sigma_{023} \vdash \mathcal{R}'(s_{013}, s_{023})
\end{aligned}
$$

The last step comes from the additional fact that $\checkmark_{A_{13}}(s_{013})$.

2. Second, we need to prove for any open transition t_{13} in T_{13} originating from s_{13}:

$$\frac{\beta_{13j}^{j \in J'_{13}}, g_{13}, \psi_{13}}{s_{13} \xrightarrow{\alpha_{13}} s'_{13}} \in \mathrm{OT}(s_{13})$$

there exists an indexed family t_{23x} of OTs originating from s_{23} that simulate it:

$$\left(\frac{\beta_{23xj}^{j \in J'_{23x}}, g_{23x}, \psi_{23x}}{s_{23} \xrightarrow{\alpha_{23x}} s'_{23x}} \in \mathrm{OT}(s_{23})\right)^{x \in X}$$

such that $\left(\forall x \in X, J'_{23x} \cap H' = J'_{13} \cap H'\right)$ and

$V_{13} \uplus V_{23} \uplus \mathit{vars}(t_{13}) \vdash$

$$\mathcal{R}'(s_{13}, s_{23}) \wedge g_{13} \implies \bigvee_{x \in X} \begin{pmatrix} \alpha_{13} = \alpha_{23x} \wedge \bigwedge_{j \in J'_{23x} \cap H'} \beta_{13j} = \beta_{23xj} \wedge \\ g_{23x} \wedge \mathcal{R}'\left(s'_{13}, s'_{23x}\right)\{\!\!\{\psi_{13} \uplus \psi_{23x}\}\!\!\} \end{pmatrix}$$

Recall that by definition of composition and OA refinement we have:

$$
\begin{aligned}
&V_{13} = V_1 \uplus V_3^1 \text{ and } V_{23} = V_2 \uplus V_3^2 \\
&H' \subseteq J_3 \uplus (J_1 \cap J_2) \setminus \{k\} = (J_3 \uplus J_1 \setminus \{k\}) \cap (J_3 \uplus J_2 \setminus \{k\})
\end{aligned}
$$

First of all, we have by hypothesis $A_1 \leqslant_H A_2$, then for any open transition t_1 in T_1 originating from s_1:

$$\frac{\beta_{1j}^{j \in J'_1}, g_1, \psi_1}{s_1 \xrightarrow{\alpha_1} s'_1} \in \mathrm{OT}(s_1)$$

there exists an indexed family of OTs originating from s_2:

$$\left(\begin{array}{c}\beta_{2xj}^{j\in J'_{2x}}, g_{2x}, \psi_{2x} \\ \hline s_2 \xrightarrow{\alpha_{2x}} s'_{2x}\end{array} \in \mathrm{OT}(s_2)\right)^{x\in X}$$

such that $\forall x \in X, J'_{2x} \cap H = J'_1 \cap H$ and

$$V_1 \uplus V_2 \uplus vars(t_1) \vdash$$
$$\mathcal{R}(s_1, s_2) \wedge g_1 \implies \bigvee_{x\in X} \left(\begin{array}{c}\alpha_1 = \alpha_{2x} \wedge \bigwedge_{j\in J'_{2x}\cap H} \beta_{1j} = \beta_{2xj} \wedge \\ g_{2x} \wedge \mathcal{R}\big(s'_1, s'_{2x}\big)\{\!\{\psi_1 \uplus \psi_{2x}\}\!\}\end{array}\right) \qquad (*)$$

Consider any transition t_{13} in A_{13}. Based on the definition of composition t_{13} can be obtained from two different cases, we will consider the two cases separately.

First case: Both automata perform a transition. The transition t_{13} is obtained by the composition of transitions $t_1 = \dfrac{\beta_{1j}^{j\in J'_1}, g_1, \psi_1}{s_1 \xrightarrow{\alpha_1} s'_1} \in \mathrm{OT}(s_1)$ and

$$t_3^1 = \frac{(\beta_{3j}^1)^{j\in J_3^{1\prime}}, g_3^1, \psi_3^1}{s_3^1 \xrightarrow{\alpha_3^1} s_3^{1\prime}} \in \mathrm{OT}\big(s_3^1\big) \qquad \text{when } k \in J'_1$$

The result is:

$$t_{13} = \frac{\beta_{1j}^{j\in J'_1\setminus\{k\}} \uplus (\beta_{3j}^1)^{j\in J_3^{1\prime}}, g_1 \wedge g_3^1 \wedge \alpha_3^1 = \beta_{1k}, \psi_1 \uplus \psi_3^1}{\big(s_1, s_3^1\big) \xrightarrow{\alpha_1} \big(s'_1, s_3^{1\prime}\big)} \qquad \text{where } k \in J'_1$$

We then obtain a family of OTs by the simulation of A_1 by A_2 (as stated above). By hypothesis we have $k \in H$, so in the case where $k \in J'_1$, we deduce that $k \in J'_{2x}$ we can then build a family of OTs $t_{23x}^{x\in X}$ with the same transitions of A_3 (up to renaming) as those used to build t_{13}.

$$t_{23x} = \left(\frac{\beta_{2xj}^{j\in J'_{2x}\setminus\{k\}} \uplus (\beta_{3j}^2)^{j\in J_3^{2\prime}}, g_{2x} \wedge g_3^2 \wedge \alpha_3^2 = \beta_{2xk}, \psi_{2x} \uplus \psi_3^2}{\big(s_2, s_3^2\big) \xrightarrow{\alpha_{2x}} \big(s'_{2x}, s_3^{2\prime}\big)}\right)^{x\in X}$$

Recall that in this case $k \in J'_1$, so $\forall x \in X$ we have

$$\begin{aligned}
J'_{23x} \cap H' &= ((J'_{2x} \setminus \{k\}) \uplus J_3^{2\prime}) \cap (H \uplus J_3 \setminus \{k\}) \\
&= ((J'_{2x} \cap (H \uplus J_3)) \uplus (J_3^{2\prime} \cap (H \uplus J_3))) \setminus \{k\} \\
&= ((J'_{2x} \cap H) \uplus (J_3^{2\prime} \cap J_3)) \setminus \{k\} \\
&\qquad\qquad \text{since } J_3 \cap J'_{2x} = \varnothing \text{ and } H \cap J_3^{2\prime} = \varnothing \\
&= ((J'_{2x} \cap H) \uplus J_3^{2\prime}) \setminus \{k\} \text{ since } J_3^{2\prime} \subseteq J_3 \\
&= ((J'_1 \cap H) \uplus J_3^{1\prime}) \setminus \{k\} \quad \text{since } J_3^{1\prime} = J_3^{2\prime} \text{ and } J'_1 \cap H = J'_{2x} \cap H \\
&= ((J'_1 \cap H) \uplus (J_3^{1\prime} \cap J_3)) \setminus \{k\} \text{ since } J_3^{1\prime} \subseteq J_3 \\
&= (J'_1 \cap (J_3 \uplus H)) \uplus ((J_3^{1\prime} \cap (J_3 \uplus H)) \setminus \{k\} \\
&\qquad\qquad \text{since } J_3 \cap J'_1 = \varnothing \text{ and } H \cap J_3^{1\prime} = \varnothing \\
&= \big((J'_1 \uplus J_3^{1\prime}) \setminus \{k\}\big) \cap ((J_3 \uplus H) \setminus \{k\}) \\
&= J'_{13} \cap H'
\end{aligned}$$

In this case the composition gives:

$$g_{13} \Leftrightarrow g_1 \wedge g_3^1 \wedge \alpha_3^1 = \beta_{1k} \text{ and } g_{23x} \Leftrightarrow g_{2x} \wedge g_3^2 \wedge \alpha_3^2 = \beta_{2xk}$$

As $k \in H$ we have $\beta_{1k} = \beta_{2xk}$ then we deduce:

$$g_3^1 \wedge \alpha_3^1 = \beta_{1k} \Leftrightarrow g_3^2 \wedge \alpha_3^2 = \beta_{2xk}$$

The proof of the rest is based on the following facts:

(a) By construction of t_{13} and t_{23x} we have $\alpha_{13} = \alpha_1$ and $\alpha_{23x} = \alpha_{2x}$. So we deduce: $\alpha_1 = \alpha_{2x} \Rightarrow \alpha_{13} = \alpha_{23x}$.

(b) By composition we have also:
$\beta_{13j}^{j \in J'_{13}} = \beta_{1j}^{j \in J'_1 \setminus \{k\}} \uplus (\beta_{3j}^1)^{j \in J_3^{1\prime}}$ and $\beta_{23xj}^{j \in J'_{23}} = \beta_{2xj}^{j \in J'_{2x} \setminus \{k\}} \uplus (\beta_{3j}^2)^{j \in J_3^{2\prime}}$
Therefore, we have for all $j \in J'_{13}$ (recall that $J'_{13} = J'_{23}$):

$$\beta_{13j} = \beta_{23xj} \Rightarrow (j \in J'_1 \wedge \beta_{1j} = \beta_{2xj}) \vee (j \in J_3^{1\prime} \wedge \beta_{3j}^1 = \beta_{3j}^2)$$

(c) Considering β_{3j}^1 and β_{3j}^2 are the same (up to renaming) we have:
$V_3^1 \uplus V_3^2 \uplus vars(t_{13}) \vdash \bigwedge_{v_3 \in V_3} v_3^1 = v_3^2 \implies \bigwedge_{j \in J_3^{2\prime}} \beta_{3j}^1 = \beta_{3j}^2$
Then we compose by disjunction with the following hypothesis (part of formula $(*)$).

$$V_1 \uplus V_2 \uplus vars(t_1) \vdash \bigwedge_{j \in J'_{2x} \cap H} \beta_{1j} = \beta_{2xj}$$

$$V_1 \uplus V_2 \uplus V_3^1 \uplus V_3^2 \uplus vars(t_{13}) \vdash \bigwedge_{v_3 \in V_3} v_3^1 = v_3^2 \implies \bigwedge_{j \in J'_{2x} \cap H} \beta_{1j} = \beta_{2xj} \vee \bigwedge_{j \in J_3^{2\prime}} \beta_{3j}^1 = \beta_{3j}^2$$

$$\Rightarrow V_{13} \uplus V_{23} \uplus vars(t_{13}) \vdash \bigwedge_{v_3 \in V_3} v_3^1 = v_3^2 \implies \bigwedge_{j \in (J'_{2x} \cap H) \uplus J_3^{2\prime}} \beta_{13j} = \beta_{23xj}$$

$$\Rightarrow V_{13} \uplus V_{23} \uplus vars(t_{13}) \vdash \bigwedge_{v_3 \in V_3} v_3^1 = v_3^2 \implies \bigwedge_{j \in ((J'_{2x} \cap H) \uplus J_3^{2\prime}) \setminus \{k\}} \beta_{13j} = \beta_{23xj}$$

$$\Rightarrow V_{13} \uplus V_{23} \uplus vars(t_{13}) \vdash \bigwedge_{v_3 \in V_3} v_3^1 = v_3^2 \implies \bigwedge_{j \in (J'_{23x} \cap H')} \beta_{13j} = \beta_{23xj}$$

We can extend the valuation context of the variables to cover the variables of the transitions t_3 and the variables in V_3^1 and V_3^2 in the formula $(*)$. By using the statements resulting from the cases (a), (b) and (c), we get:

$$V_{13} \uplus V_{23} \uplus vars(t_{13}) \vdash$$
$$\mathcal{R}(s_1, s_2) \wedge \big(g_1 \wedge g_3^1 \wedge \alpha_3^1 = \beta_{1k}\big) \wedge \bigwedge_{v_3 \in V_3} v_3^1 = v_3^2 \implies$$
$$\bigvee_{x \in X} \begin{pmatrix} \alpha_{13} = \alpha_{23x} \wedge \bigwedge_{j \in (J'_{23x} \cap H')} \beta_{13j} = \beta_{23xj} \wedge \\ g_{2x} \wedge \mathcal{R}\big(s'_1, s'_{2x}\big) \{\!\{\psi_1 \uplus \psi_{2x}\}\!\} \end{pmatrix} \wedge g_3^2 \wedge \alpha_3^2 = \beta_{2xk}$$

That can be re-written as follows:

$$V_{13} \uplus V_{23} \uplus vars(t_{13}) \vdash$$
$$\mathcal{R}(s_1, s_2) \wedge g_{13} \wedge \bigwedge_{v_3 \in V_3} v_3^1 = v_3^2 \implies \bigvee_{x \in X} \begin{pmatrix} \alpha_{13} = \alpha_{23x} \wedge \bigwedge_{j \in J'_{23x} \cap H'} \beta_{13j} = \beta_{23xj} \wedge \\ g_{23x} \wedge \mathcal{R}\big(s'_1, s'_{2x}\big) \{\!\{\psi_1 \uplus \psi_{2x}\}\!\} \end{pmatrix}$$

Moreover, we have for any transition t_3 in A_3 relating s_3 and s'_3 the following:

$$V_{13} \uplus V_{23} \uplus vars(t_{13}) \vdash$$
$$\bigwedge_{v_3 \in V_3} v_3^1 = v_3^2 \wedge s_3^1 = s_3^2 \implies \bigwedge_{v_3 \in V_3} \psi_{13}(v_3^1) = \psi_{13}(v_3^2) \wedge s_3^{1\prime} = s_3^{2\prime}$$

Furthermore, according to the Definition 7 (reachability applied to the composed automaton $A_1[A_3/k]$) we have, for all $t_{13} \in T_{13}$:

$$vars(t_{13}) \vdash \big(\checkmark_{A_{13}}(s_{13}) \wedge g_{13} \implies \checkmark_{A_{13}}\big(s'_{13}\big) \{\!\{\psi_{13}\}\!\}\big)$$

Thus, we get:

$$
\begin{array}{l}
V_{13} \uplus V_{23} \uplus \mathit{vars}(t_1) \cap \mathit{vars}(t_{13}) \vdash \\
\quad \mathcal{R}(s_1, s_2) \wedge g_{13} \wedge \checkmark_{A_{13}}(s_{13}) \wedge \bigwedge_{v_3 \in V_3} v_3^1 = v_3^2 \wedge s_3^1 = s_3^2 \implies \\
\bigvee_{x \in X} \left(\begin{array}{l}
\alpha_{13} = \alpha_{23x} \wedge \bigwedge_{j \in J'_{23x} \cap H'} \beta_{13j} = \beta_{23xj} \wedge g_{23x} \wedge \mathcal{R}\big(s'_1, s'_{2x}\big)\{\!\{\psi_1 \uplus \psi_{2x}\}\!\} \\
\wedge \checkmark_{A_{13}}\big(s'_{13}\big)\{\!\{\psi_{13}\}\!\} \wedge \bigwedge_{v_3 \in V_3} \psi_{13}(v_3^1) = \psi_{13}(v_3^2) \wedge s_3^{1\prime} = s_3^{2\prime}
\end{array} \right)
\end{array}
$$

From the two previous formulas, we get:

$$
\begin{array}{l}
V_{13} \uplus V_{23} \uplus \mathit{vars}(t_{13}) \vdash \\
\mathcal{R}(s_1, s_2) \wedge g_{13} \wedge \checkmark_{A_{13}}(s_{13}) \wedge \bigwedge_{v_3 \in V_3} v_3^1 = v_3^2 \wedge s_3^1 = s_3^2 \implies \\
\quad \bigvee_{x \in X} \left(\begin{array}{l}
\alpha_{13} = \alpha_{23x} \wedge \bigwedge_{j \in J'_{23x} \cap H'} \beta_{13j} = \beta_{23xj} \wedge g_{23x} \wedge \checkmark_{A_{13}}\big(s'_{13}\big)\{\!\{\psi_{13}\}\!\} \wedge \\
\mathcal{R}\big(s'_1, s'_{2x}\big)\{\!\{\psi_1 \uplus \psi_{2x}\}\!\} \wedge \bigwedge_{v_3 \in V_3} \psi_{13}(v_3^1) = \psi_{13}(v_3^2) \wedge s_3^{1\prime} = s_3^{2\prime}
\end{array} \right)
\end{array}
$$

Because of the independence of the substitution domains, we simplify and get the expected formula:

$$
\begin{array}{l}
V_{13} \uplus V_{23} \uplus \mathit{vars}(t_{13}) \vdash \\
\quad \mathcal{R}'(s_{13}, s_{23}) \wedge g_{13} \implies \bigvee_{x \in X} \left(\begin{array}{l}
\alpha_{13} = \alpha_{23x} \wedge \bigwedge_{j \in J'_{23x} \cap H'} \beta_{13j} = \beta_{23xj} \wedge \\
g_{23x} \wedge \mathcal{R}'\big(s'_{13}, s'_{23x}\big)\{\!\{\psi_{13} \uplus \psi_{23x}\}\!\}
\end{array} \right)
\end{array}
$$

Second case: Only the encompassing automaton performs a transition. t_{13} is obtained by the transition $t_1 = \dfrac{\beta_{1j}^{j \in J'_1}, g_1, \psi_1}{s_1 \xrightarrow{\alpha_1} s'_1}$ alone with the state s_3^1 unchanged, if $k \notin J'_1$

$$
t_{13} = \frac{\beta_{1j}^{j \in J'_1}, g_1, \psi_1}{\big(s_1, s_3^1\big) \xrightarrow{\alpha_1} \big(s'_1, s_3^1\big)}
$$

As stated above, from the simulation of A_1 by A_2 we get a family of OTs $t_{2x}^{x \in X}$. The composition of this family of OTs with the same transitions of A_3 (up to renaming) as those used to build t_{13} produces a family of OTs $t_{23x}^{x \in X}$ in the form:

$$
t_{23x} = \left(\frac{\beta_{2xj}^{j \in J'_{2x}}, g_{2x}, \psi_{2x}}{\big(s_2, s_3^2\big) \xrightarrow{\alpha_{2x}} \big(s'_{2x}, s_3^2\big)} \right)^{x \in X}
$$

By hypothesis we have $k \in H$, since $k \notin J'_1$, we deduce $k \notin J'_{2x}$.
$\forall x \in X$ we have:

$$\begin{aligned} J'_{23x} \cap H' &= J'_{2x} \cap (J_3 \uplus H \setminus \{k\}) \\ &= \big(J'_{2x} \cap H\big) \text{ since } J'_{2x} \cap J_3 = \varnothing \wedge k \notin J'_{2x} \\ &= \big(J'_1 \cap H\big) \text{ since } J'_1 \cap H = J'_{2x} \cap H \\ &= \big(J'_{13} \cap H'\big) \text{ since } J_3 \cap J'_1 = \varnothing \wedge k \notin J'_1 \end{aligned}$$

The proof of the rest of the formula follows the same steps as the previous case the only argument that change is that by composition we obtain: g_{13} is the same as g_1 and g_{23x} is the same as g_{2x} and the actions of α_{13}, resp. α_{23x}, is the same as α_1, resp. α_{2x}. Similarly β_{13j} and β_{23xj} that are the same as β_{1j} and β_{2xj} respectively. We also apply the reachability definition but only on the automaton A_1

3. Lastly, we have to prove the satisfaction of the deadlock reduction condition, i.e., for all $(s_{13}, s_{23}) \in S_{13} \times S_{23}$

$$V_{13} \uplus V_{23} \vdash \mathcal{R}(s_{13}, s_{23}) \wedge \neg\Big(\bigvee_{t_{13} \in \mathrm{OT}(s_{13})} guard(t_{13})\Big) \implies \neg\Big(\bigvee_{t_{23} \in \mathrm{OT}(s_{23})} guard(t_{23})\Big)$$

By hypothesis the composition $A_1[A_3/k]$ is non-blocking, then according to the Definition 10 we have for all state s_{13}:

$$V_{13} \uplus \biguplus_{t_1 \in \mathrm{OT}(s_1)} vars(t_1) \vdash \left(\checkmark_{A_{13}(s_{13})} \wedge \bigvee_{t_1 \in \mathrm{OT}(s_1)} guard(t_1)\right) \implies \bigvee_{t_{13} \in \mathrm{OT}(s_{13})} guard(t_{13})$$

From this we can infer:

$$V_{13} \uplus \biguplus_{t_1 \in \mathrm{OT}(s_1)} vars(t_1) \vdash \neg\Big(\bigvee_{t_{13} \in \mathrm{OT}(s_{13})} guard(t_{13})\Big) \implies \neg\Big(\bigvee_{t_1 \in \mathrm{OT}(s_1)} guard(t_1)\Big) \vee \neg\checkmark_{A_{13}(s_{13})}$$

Furthermore, from the second hypothesis stating that $A_1 \leqslant_H A_2$, we have for all $(s_1, s_2) \in S_1 \times S_2$

$$V_1 \uplus V_2 \vdash \mathcal{R}(s_1, s_2) \wedge \neg\Big(\bigvee_{t_1 \in \mathrm{OT}(s_1)} guard(t_1)\Big) \implies \neg\Big(\bigvee_{t_2 \in \mathrm{OT}(s_2)} guard(t_2)\Big)$$

From the two previous formula can deduce:

$$\begin{aligned} &V_{13} \uplus V_2 \uplus \biguplus_{t_1 \in \mathrm{OT}(s_1)} vars(t_1) \vdash \\ &\quad R(s_1, s_2) \wedge \neg\Big(\bigvee_{t_{13} \in \mathrm{OT}(s_{13})} guard(t_{13})\Big) \implies \neg\Big(\bigvee_{t_2 \in \mathrm{OT}(s_2)} guard(t_2)\Big) \vee \neg\checkmark_{A_{13}}(s_{13}) \end{aligned}$$

By definition of the composition of open automata, each guard of automaton A_{23} is of the form $guard(t_{23}) = guard(t_2) \wedge t$, thus we have:

$$V_{23} \quad \vdash \quad \neg(\bigvee_{t_1 \in \mathrm{OT}(s_2)} guard(t_2)) \qquad \Longrightarrow \qquad \neg(\bigvee_{t_{23} \in \mathrm{OT}(s_{23})} guard(t_{23}))$$

From the two previous formula and by introducing the following tautology $\left(\bigwedge_{v_3 \in V_3} v_3^1 = v_3^2 \wedge s_3^1 = s_3^2\right)$, we get:

$$V_{13} \uplus V_{23} \vdash \underbrace{\mathcal{R}(s_1, s_2) \wedge \checkmark_{A_{13}}(s_{13}) \wedge \left(\bigwedge_{v_3 \in V_3} v_3^1 = v_3^2 \wedge s_3^1 = s_3^2\right)}_{\mathcal{R}'(s_{13}, s_{23})} \wedge \neg \bigvee_{t_{13} \in \mathrm{OT}(s_{13})} guard(t_{13})$$

$$\Longrightarrow \left(\neg \bigvee_{t_{23} \in \mathrm{OT}(s_{23})} guard(t_{23}) \vee \neg \checkmark_{A_{13}}(s_{13})\right) \wedge \checkmark_{A_{13}}(s_{13}) \Longrightarrow \neg \bigvee_{t_{23} \in \mathrm{OT}(s_{23})} guard(t_{23})$$

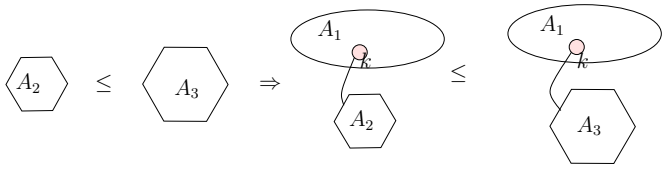
which gives the expected result. □

C Proof of Congruence (Theorem 3)

Suppose $A_2 \leqslant_H A_3$ and $k \in H$. and the composition $A_1[A_2/k]$ is non-blocking. We have:

$$A_1[A_2/k] \leqslant_{J_1 \uplus H \setminus \{k\}} A_1[A_3/k]$$

Pictorially, the theorem states the following result:



Proof. Let us denote by A_{12} (resp. A_{13}) the OA resulting from $A_1[A_2/k]$ (resp. $A_1[A_3/k]$), to prove the theorem it is sufficient to prove that there exists a relation between states of the two OA that satisfies the conditions of the Definition 9.

We denote $A_1 = \langle S_1, s_{01}, J_1, V_1, \sigma_{01}, T_1 \rangle$ and
$A_2 = \langle S_2, s_{02}, J_2, V_2, \sigma_{02}, T_2 \rangle$ and $A_3 = \langle S_3, s_{03}, J_3, V_3, \sigma_{03}, T_3 \rangle$.

Let R be the refinement relation relating states of A_2 and A_3. Let us denote with t^2 and t^3 the elements of A_2 and A_3 respectively. Consider any two states $s_{12} = \left(s_1^1, s_2\right)$ and $s_{13} = \left(s_1^2, s_3\right)$ (s_1^1 and s_1^2 are the same with renaming). We define a relation $\mathcal{R}'$ relating states of s_{12} and s_{13} as follows:

$$\mathcal{R}'(s_{12}, s_{13}) = \mathcal{R}(s_2, s_3) \wedge \checkmark_{A_{12}}(s_{12}) \wedge \bigwedge_{v_1 \in V_1} v_1^1 = v_1^2 \wedge s_1^1 = s_1^2$$

Let us denote $H' = H \cup J_1 \setminus \{k\}$. We want to prove that $(\mathcal{R}', H')$ is a hole-tracking simulation of A_{12} and A_{13}. As for the previous proof we deal with cases:

1. First, we have to prove the relation for initial states:

$$\sigma_{012} \uplus \sigma_{013} \vdash \mathcal{R}'(s_{012}, s_{013})$$

with $\sigma_{012} = \sigma^1_{01} \uplus \sigma_{03}$, $\sigma_{013} = \sigma^2_{01} \uplus \sigma_{03}$, $s_{012} = (s^1_{01}, s_{02})$, and $s_{013} = (s^2_{01}, s_{03})$.
The proof of this point is similar to that of Theorem 2.
2. Second, we need to prove for any OT t_{12} in T_{12} originating from s_{12}

$$\frac{\overset{\cdots\cdots}{\beta_{12j}^{j\in J'_{12}}, g_{12}, \psi_{12}}}{s_{12} \xrightarrow{\alpha_{12}} s'_{12}} \in \mathrm{OT}(s_{12})$$

there exists an indexed family t_{13x} of OTs originating from s_{13} that simulates it:

$$\left(\frac{\overset{\cdots\cdots}{\beta_{13xj}^{j\in J'_{13x}}, g_{13x}, \psi_{13x}}}{s_{13} \xrightarrow{\alpha_{13x}} s'_{13x}} \in \mathrm{OT}(s_{13})\right)^{x\in X}$$

such that $(\forall x \in X, J'_{13x} \cap H' = J'_{12} \cap H')$ and

$$V_{12} \uplus V_{13} \uplus \mathit{vars}(t_{12}) \vdash$$
$$\mathcal{R}'(s_{12}, s_{13}) \wedge g_{12} \implies \bigvee_{x\in X} \begin{pmatrix} \alpha_{12} = \alpha_{13x} \wedge \bigwedge\limits_{j\in J'_{13x}\cap H'} \beta_{12j} = \beta_{13xj} \wedge \\ g_{13x} \wedge \mathcal{R}'\big(s'_{12}, s'_{13x}\big)\{\!\{\psi_{12} \uplus \psi_{13x}\}\!\} \end{pmatrix}$$

By definition of composition and OA refinement we have:

$V_{12} = V^1_1 \uplus V_3$ and $V_{13} = V^2_1 \uplus V_3$
$H' = H \uplus J_1 \setminus \{k\} \subseteq J_1 \uplus (J_2 \cap J_3) \setminus \{k\} = \big(J^1_1 \uplus J_2 \setminus \{k\}\big) \cap \big(J^2_1 \uplus J_3 \setminus \{k\}\big)$
$= J_{12} \cap J_{13}$

We have by hypothesis $A_2 \leqslant_H A_3$, then for any open transition t_2 in T_2 originating from s_2:

$$\frac{\overset{\cdots\cdots}{\beta_{2j}^{j\in J'_2}, g_2, \psi_2}}{s_2 \xrightarrow{\alpha_2} s'_2} \in \mathrm{OT}(s_2)$$

there exists an indexed family of OTs originating from s_3:

$$\left(\frac{\overset{\cdots\cdots}{\beta_{3xj}^{j\in J'_{3x}}, g_{3x}, \psi_{3x}}}{s_3 \xrightarrow{\alpha_{3x}} s'_{3x}} \in \mathrm{OT}(s_3)\right)^{x\in X}$$

such that $\forall x \in X, J'_{3x} \cap H = J'_2 \cap H$ and

$$V_2 \uplus V_3 \uplus vars(t_2) \vdash$$
$$\mathcal{R}(s_2, s_3) \wedge g_2 \implies \bigvee_{x \in X} \begin{pmatrix} \alpha_2 = \alpha_{3x} \wedge \bigwedge_{j \in J'_{3x} \cap H} \beta_{2j} = \beta_{3xj} \wedge \\ g_{3x} \wedge \mathcal{R}\big(s'_2, s'_{3x}\big) \{\!\{ \psi_2 \uplus \psi_{3x} \}\!\} \end{pmatrix} \qquad (*)$$

Consider any transition t_{12} in A_{12}. Based on the definition of the composition t_{12} can be obtained from two different cases, we will consider the two cases separately.

First case: Both automata perform a transition. The transition t_{12} is obtained by the composition of transition $t_2 = \dfrac{\beta_{2j}^{j \in J'_2}, g_2, \psi_2}{s_2 \xrightarrow{\alpha_2} s'_2}$ and a transition

$$t_1^1 = \frac{(\beta_{1j}^1)^{j \in J_1^{1\prime}}, g_1^1, \psi_1^1}{s_1^1 \xrightarrow{\alpha_1^1} s_1^{1\prime}} \qquad \text{when } k \in J_1^{1\prime}$$

The result:

$$t_{12} = \frac{\beta_{1j}^{1\ j \in J_1^{1\prime} \setminus \{k\}} \uplus (\beta_{2j})^{j \in J'_2}, g_1^1 \wedge g_2 \wedge \alpha_1^1 = \beta_{2k}, \psi_1^1 \uplus \psi_2}{\big(s_1^1, s_2\big) \xrightarrow{\alpha_1^1} \big(s_1^{1\prime}, s'_2\big)} \qquad \text{where } k \in J_1^{1\prime}$$

We then obtain a family of OTs by the simulation of A_2 by A_3 (as stated above). By hypothesis we have $k \in H$, so in the case where $k \in J_1^{1\prime}$, we deduce that $k \in J'_{3x}$ we can then build a family of OTs $t_{13x}^{x \in X}$ with the same transition, up to renaming, as the one used to build t_{12} (i.e., t_1^1), where $s_1^{2\prime}$ is the same as $s_1^{1\prime}$ up to renaming.

$$t_{13x} = \left(\frac{(\beta_{1j}^2)^{j \in J_1^{2\prime}} \uplus \beta_{3xj}^{j \in J'_{3x} \setminus \{k\}}, g_{3x} \wedge g_1^2 \wedge \alpha_1^2 = \beta_{3xk}, \psi_{3x} \uplus \psi_1^2}{\big(s_1^2, s_3\big) \xrightarrow{\alpha_1^2} \big(s_1^{2\prime}, s'_{3x}\big)} \right)^{x \in X}$$

Recall that in this case $k \in J_1^{1'}$, so $\forall x \in X$ we have:

$$\begin{aligned}
J'_{13x} \cap H' &= ((J'_{3x} \setminus \{k\}) \uplus J_1^{2'}) \cap (H \uplus J_1 \setminus \{k\}) \\
&= ((J'_{3x} \cap (H \uplus J_1)) \uplus (J_1^{2'} \cap (H \uplus J_1))) \setminus \{k\} \\
&= ((J'_{3x} \cap H) \uplus (J_1^{2'} \cap J_1)) \setminus \{k\} \\
&\qquad\qquad \text{since } J_1 \cap J'_{3x} = \varnothing \text{ and } H \cap J_1^{2'} = \varnothing \\
&= ((J'_{3x} \cap H) \uplus J_1^{2'}) \setminus \{k\} \text{ since } J_1^{2'} \subseteq J_1 \\
&= ((J'_2 \cap H) \uplus J_1^{1'}) \setminus \{k\} \\
&\qquad\qquad \text{since } J_1^{1'} = J_1^{2'} \text{ and } J'_2 \cap H = J'_{3x} \cap H \\
&= ((J'_2 \cap H) \uplus (J_1^{1'} \cap J_1)) \setminus \{k\} \text{ since } J_1^{1'} \subseteq J_1 \\
&= (J'_2 \cap (J_1 \uplus H)) \uplus ((J_1^{1'} \cap (J_1 \uplus H)) \setminus \{k\} \\
&\qquad\qquad \text{since } J_1 \cap J'_2 = \varnothing \text{ and } H \cap J_1^{1'} = \varnothing \\
&= \big((J'_2 \uplus J_1^{1'}) \setminus \{k\}\big) \cap ((J_1 \uplus H) \setminus \{k\}) \\
&= J'_{12} \cap H'
\end{aligned}$$

In this case the composition gives:

$$g_{12} \Leftrightarrow g_1^1 \wedge g_2 \wedge \alpha_1^1 = \beta_{2k} \text{ and } g_{13x} \Leftrightarrow g_1^2 \wedge g_{3x} \wedge \alpha_1^2 = \beta_{3xk}$$

As $k \in H$ we have $\alpha_1^1 = \alpha_1^2$ then we deduce:

$$g_1^1 \wedge \alpha_1^1 = \beta_{2k} \Leftrightarrow g_1^2 \wedge \alpha_1^2 = \beta_{3xk}$$

The proof of the rest is based on the following facts:

(a) By construction of t_{12} and t_{13x} we have $\alpha_{12} = \alpha_1^1$ and $\alpha_{13x} = \alpha_1^2$. Since α_1^1 and α_1^2 are the same (up to renaming) we deduce: $\alpha_{12} = \alpha_{13x}$.

(b) By composition we have also:
$\beta_{12j}^{j \in J'_{12}} = (\beta_{1j}^1)^{j \in J_1^{1'} \setminus \{k\}} \uplus (\beta_{2j})^{j \in J'_2}$ and $\beta_{13xj}^{j \in J'_{13}} = (\beta_{1j}^2)^{j \in J_1^{2'} \setminus \{k\}} \uplus \beta_{3xj}^{j \in J'_{3x} \setminus \{k\}}$
Therefore, we have for all $j \in J'_{12}$ (recall that $J'_{12} = J'_{13}$):

$$\beta_{12j} = \beta_{13xj} \Rightarrow (j \in J_1^{1'} \wedge \beta_{1j}^1 = \beta_{1j}^2) \vee (j \in J'_2 \wedge \beta_{2j} = \beta_{3xj})$$

(c) Considering β_{1j}^1 and β_{1j}^2 are the same (up to renaming) we have:
$V_1^1 \uplus V_1^2 \uplus vars(t_{12}) \vdash \bigwedge_{v_1 \in V_1} v_1^1 = v_1^2 \implies \bigwedge_{j \in J_1^{2'}} \beta_{1j}^1 = \beta_{1j}^2$
We compose by disjunction with the following hypothesis (part of formula $(*)$).

$$V_2 \uplus V_3 \uplus vars(t_2) \vdash \bigwedge_{j \in J'_{3x} \cap H} \beta_{2j} = \beta_{3xj}$$

$$\begin{aligned}
V_2 \uplus V_3 \uplus V_1^1 \uplus V_1^2 \uplus vars(t_{12}) \vdash \bigwedge_{v_1 \in V_1} v_1^1 = v_1^2 \implies & \bigwedge_{j \in J'_{3x} \cap H} \beta_{2j} = \beta_{3xj} \vee \bigwedge_{j \in J_1^{2\prime}} \beta_{1j}^1 = \beta_{1j}^2 \\
\Rightarrow V_{12} \uplus V_{13} \uplus vars(t_{12}) \vdash \bigwedge_{v_1 \in V_1} v_1^1 = v_1^2 \implies & \bigwedge_{j \in (J'_{3x} \cap H) \uplus J_1^{2\prime}} \beta_{12j} = \beta_{13xj} \\
\Rightarrow V_{12} \uplus V_{13} \uplus vars(t_{12}) \vdash \bigwedge_{v_1 \in V_1} v_1^1 = v_1^2 \implies & \bigwedge_{j \in ((J'_{3x} \cap H) \uplus J_1^{2\prime}) \setminus \{k\}} \beta_{12j} = \beta_{13xj} \\
\Rightarrow V_{12} \uplus V_{12} \uplus vars(t_{12}) \vdash \bigwedge_{v_1 \in V_1} v_1^1 = v_1^2 \implies & \bigwedge_{j \in (J'_{13x} \cap H')} \beta_{12j} = \beta_{13xj}
\end{aligned}$$

We can extend the valuation context of the variables to cover the variables of the transitions t_1 and variables V_1^1 and V_1^2 in the formula $(*)$. By using the statements resulting from the cases (a), (b) and (c), we get:

$$\begin{aligned}
& V_{13} \uplus V_{13} \uplus vars(t_{12}) \vdash \\
& \mathcal{R}(s_2, s_3) \wedge \big(g_2 \wedge g_1^1 \wedge \alpha_1^1 = \beta_{2k}\big) \wedge \bigwedge_{v_1 \in V_1} v_1^1 = v_1^2 \implies \\
& \qquad \bigvee_{x \in X} \begin{pmatrix} \alpha_{12} = \alpha_{13x} \wedge \bigwedge\limits_{j \in (J'_{13x} \cap H')} \beta_{12j} = \beta_{13xj} \wedge \\ g_{3x} \wedge \mathcal{R}\big(s'_2, s'_{3x}\big) \{\!\!\{ \psi_2 \uplus \psi_{3x} \}\!\!\} \end{pmatrix} \wedge g_1^2 \wedge \alpha_1^2 = \beta_{3xk}
\end{aligned}$$

That can be re-written as follows:

$$\begin{aligned}
& V_{12} \uplus V_{13} \uplus vars(t_{12}) \vdash \\
& \mathcal{R}(s_2, s_3) \wedge g_{12} \wedge \bigwedge_{v_1 \in V_1} v_1^1 = v_1^2 \implies \bigvee_{x \in X} \begin{pmatrix} \alpha_{12} = \alpha_{13x} \wedge \bigwedge\limits_{j \in J'_{13x} \cap H'} \beta_{12j} = \beta_{13xj} \wedge \\ g_{13x} \wedge \mathcal{R}\big(s'_2, s'_{3x}\big) \{\!\!\{ \psi_2 \uplus \psi_{3x} \}\!\!\} \end{pmatrix}
\end{aligned}$$

Moreover, we have for any transition t_1 in A_1 relating s_1^1 and $s_1^{1\prime}$ the following:

$$\begin{aligned}
& V_{12} \uplus V_{13} \uplus vars(t_{12}) \vdash \\
& \qquad \bigwedge_{v_1 \in V_1} v_1^1 = v_1^2 \wedge s_1^1 = s_1^2 \implies \bigwedge_{v_1 \in V_1} \psi_{12}(v_1^1) = \psi_{13x}(v_1^2) \wedge s_1^{1\prime} = s_1^{2\prime}
\end{aligned}$$

From the two previous formula, we get:

$$V_{12} \uplus V_{13} \uplus vars(t_{12}) \vdash$$
$$R(s_2, s_3) \wedge g_{12} \wedge \bigwedge_{v_1 \in V_1} v_1^1 = v_1^2 \wedge s_1^1 = s_1^2 \implies$$
$$\bigvee_{x \in X} \left(\begin{array}{l} \alpha_{12} = \alpha_{13x} \wedge \bigwedge_{j \in J'_{13x} \cap H'} \beta_{12j} = \beta_{13xj} \wedge g_{13x} \wedge \\ R\big(s'_2, s'_{3x}\big)\{\!\{\psi_2 \uplus \psi_{3x}\}\!\} \wedge \bigwedge_{v_1 \in V_1} \psi_{12}(v_1^1) = \psi_{13x}(v_1^2) \wedge s_1^{1\prime} = s_1^{2\prime} \end{array} \right)$$

Furthermore, according to the Definition 7 (reachability, applied to the composed automaton $A_1[A_2/k]$) we have, for all $t_{12} \in T_{12}$:

$$vars(t_{12}) \vdash \big(\checkmark_{A_{12}}(s_{12}) \wedge g_{12} \implies \checkmark_{A_{12}}\big(s'_{12}\big)\{\!\{\psi_{12}\}\!\}\big)$$

Thus, we get:

$$V_{12} \uplus V_{13} \uplus vars(t_{12}) \vdash$$
$$\mathcal{R}(s_2, s_3) \wedge g_{12} \wedge \checkmark_{A_{12}}(s_{12}) \wedge \bigwedge_{v_1 \in V_1} v_1^1 = v_1^2 \wedge s_1^1 = s_1^2 \implies$$
$$\bigvee_{x \in X} \left(\begin{array}{l} \alpha_{12} = \alpha_{13x} \wedge \bigwedge_{j \in J'_{13x} \cap H'} \beta_{12j} = \beta_{13xj} \wedge g_{13x} \wedge \\ \mathcal{R}\big(s'_2, s'_{3x}\big)\{\!\{\psi_2 \uplus \psi_{3x}\}\!\} \wedge \checkmark_{A_{12}}\big(s'_{12}\big)\{\!\{\psi_{12}\}\!\} \\ \wedge \bigwedge_{v_1 \in V_1} \psi_{12}(v_1^1) = \psi_{13x}(v_1^2) \wedge s_1^{1\prime} = s_1^{2\prime} \end{array} \right)$$

Because of the independence of the substitution domains, we simplify and get the expected formula:

$$V_{12} \uplus V_{13} \uplus vars(t_{12}) \vdash \mathcal{R}'(s_{12}, s_{13}) \wedge g_{12} \implies$$
$$\bigvee_{x \in X} \left(\alpha_{12} = \alpha_{13x} \wedge \bigwedge_{j \in J'_{13x} \cap H'} \beta_{12j} = \beta_{13xj} \wedge g_{13x} \wedge \mathcal{R}'\big(s'_{12}, s'_{13x}\big)\{\!\{\psi_{12} \uplus \psi_{13x}\}\!\} \right)$$

Second case: Only the encompassing automaton performs a transition. t_{12} is obtained by the transition $t_1^1 = \dfrac{(\beta_{1j}^1)^{j \in J_1^{1\prime}}, g_1^1, \psi_1^1}{s_1^1 \xrightarrow{\alpha_1^1} s_1^{1\prime}}$ alone with the state s_2 unchanged, if $k \notin J_1^{1\prime}$

$$t_{12} = \frac{(\beta_{1j}^1)^{j \in J_1^{1\prime}}, g_1^1, \psi_1^1}{\big(s_1^1, s_2\big) \xrightarrow{\alpha_1^1} \big(s_1^{1\prime}, s_2\big)}$$

The hole is not involved we can define t_{13} with the same transition t_1 of A_1 (where elements of t_1 are the same as above modulo renaming of variables):

$$t_{13} = \frac{(\beta_{1j}^2)^{j\in J_1^{2'}}, g_1^2, \psi_1^2}{\left(s_1^1, s_3\right) \xrightarrow{\alpha_1^2} \left(s_1^{2'}, s_3\right)}$$

We thus take $(t_{13x})^{x\in X} = \{t_{13}\}$. $\forall x \in X$ we have $J'_{13x} = J_1^{2'}$ and trivially:

$$\begin{aligned} J'_{13x} \cap H' &= J_1^{2'} \cap (J_1 \uplus H \setminus \{k\}) \\ &= J_1^{1'} \cap (J_1 \uplus H \setminus \{k\}) \text{ since } J_1^{2'} = J_1^{1'} \\ &= \left(J'_{12} \cap H'\right) \end{aligned}$$

Moreover, we have by definition

$$\mathcal{R}'(s_{12}, s_{13}) = \mathcal{R}(s_2, s_3) \wedge \checkmark_{A_{12}}(s_{12}) \wedge \bigwedge_{v_1 \in V_1} v_1^1 = v_1^2 \wedge s_1^1 = s_1^2$$

Thus, because transitions t_1^1 and t_1^2 are the same:

$$\mathcal{R}'(s_{12}, s_{13}) \implies \underbrace{\mathcal{R}(s_2, s_3) \wedge \checkmark_{A_{12}}(s_{12}) \wedge \bigwedge_{v_1 \in V_1} v_1^1 = v_1^2 \wedge s_1^{1'} = s_1^{2'}}_{\mathcal{R}'(s'_{12}, s'_{13})}$$

Moreover, we have the following:

$$V_{12} \uplus V_{13} \uplus vars(t_{12}) \vdash \bigwedge_{v_1 \in V_1} v_1^1 = v_1^2 \wedge s_1^1 = s_1^2 \implies \bigwedge_{v_1 \in V_1} \psi_{12}(v_1^1) = \psi_{13}(v_1^2) \wedge s_1^{1'} = s_1^{2'}$$

Furthermore, according to the Definition 7 (reachability, applied to the composed automaton A_1) we have, for all $t_{12} \in T_{12}$:

$$vars(t_{12}) \vdash \left(\checkmark_{A_{12}}(s_{12}) \wedge g_{12} \implies \checkmark_{A_{12}}\left(s'_{12}\right)\{\!\{\psi_{12}\}\!\}\right)$$

Because α_{12} and α_{13} are the same, and also β_{12} and β_{13} are the same (modulo renaming of variables), we deduce from the above the expected formula (instantiated with X a singleton):

$$V_{12} \uplus V_{13} \uplus vars(t_{12}) \vdash \mathcal{R}'(s_{12}, s_{13}) \wedge g_{12} \implies \left(\alpha_{12} = \alpha_{13} \wedge \bigwedge_{j \in J'_{13} \cap H'} \beta_{12j} = \beta_{13j} \wedge g_{13} \wedge \mathcal{R}'\left(s'_{12}, s'_{13}\right)\{\!\{\psi_{12} \uplus \psi_{13}\}\!\}\right)$$

3. Lastly, we have to prove the satisfaction of the deadlock reduction condition, i.e., for all $(s_{12}, s_{13}) \in S_{12} \times S_{13}$

$$V_{12} \uplus V_{13} \vdash \mathcal{R}'(s_{12}, s_{13}) \wedge \neg\left(\bigvee_{t_{12} \in \mathrm{OT}(s_{12})} guard(t_{12})\right) \implies \neg\left(\bigvee_{t_{13} \in \mathrm{OT}(s_{13})} guard(t_{13})\right)$$

Let's start with the hypothesis of the theorem stating that $A_2 \leqslant_H A_3$, thus we have for all $(s_2, s_3) \in S_2 \times S_3$

$$V_2 \uplus V_3 \vdash \mathcal{R}(s_2, s_3) \wedge \neg(\bigvee_{t_2 \in \mathrm{OT}(s_2)} guard(t_2)) \implies \neg(\bigvee_{t_3 \in \mathrm{OT}(s_3)} guard(t_3))$$

By hypothesis the composition $A_1[A_2/k]$ is non-blocking, then according to the Definition 10 we have for all state s_{12}:

$$V_{12} \uplus \biguplus_{t_1 \in \mathrm{OT}(s_1)} vars(t_1) \vdash \left(\checkmark_{A_{12}(s_{12})} \wedge \bigvee_{t_1 \in \mathrm{OT}(s_1)} guard(t_1) \right) \implies \bigvee_{t_{12} \in \mathrm{OT}(s_{12})} guard(t_{12})$$

Which implies

$$V_{12} \uplus \biguplus_{t_1 \in \mathrm{OT}(s_1)} vars(t_1) \vdash \neg \bigvee_{t_{12} \in \mathrm{OT}(s_{12})} guard(t_{12}) \implies \left(\neg \checkmark_{A_{12}(s_{12})} \vee \neg \bigvee_{t_1 \in \mathrm{OT}(s_1)} guard(t_1) \right)$$

Additionally, each transition t_{13} of $\mathrm{OT}(s_{13})$ is of the form $g_1 \wedge g_3 \wedge g$ with g_1 a guard of a transition t_1 of $\mathrm{OT}(s_1)$. Thus we have

$$\neg \bigvee_{t_1 \in \mathrm{OT}(s_1)} guard(t_1) \implies \neg(\bigvee_{t_{13} \in \mathrm{OT}(s_{13})} guard(t_{13}))$$

Thus we have, as $\mathcal{R}$ contains the reachability of state s_{12}:

$$V_{12} \uplus V_{13} \vdash \mathcal{R}'(s_{12}, s_{13}) \wedge \neg(\bigvee_{t_{12} \in \mathrm{OT}(s_{12})} guard(t_{12})) \implies$$
$$\left(\checkmark_{A_{12}(s_{12})} \wedge \left(\neg \checkmark_{A_{12}(s_{12})} \vee \neg(\bigvee_{t_{13} \in \mathrm{OT}(s_{13})} guard(t_{13})) \right) \implies \neg(\bigvee_{t_{13} \in \mathrm{OT}(s_{13})} guard(t_{13})) \right)$$

□

Conclusion

The work reported in this manuscript mirrors some of my research activities over the last few years. It follows a work that I have started since my Phd thesis, which continues to raise further exploration. Our long term goal is to provide a theoretical and practical framework for the design and development of reliable distributed and embedded systems.

While many approaches and techniques have been proposed to ease the reasoning about large scale distributed systems, and solutions have been effectively implemented to overcome the current limitations of verification techniques, it remains a real challenge to combine the results of such specialized techniques and to foresee their impact on open systems. Each technique exploits the characteristics of the formalism on which it is based. Some of them, use symbolic models for explicit data manipulation, to overcome the finiteness limit constraints, others allow the compositional description of systems to enable compositional verification methods. To the best of our knowledge, there is no work that combines all these techniques together. It could be because most of the languages on which they are based are not rich enough to cover all aspects, which is especially the key strength of the open pNets (and open automata) formalism.

A further important dimension of the open pNets (and open automata) model, it provides a rigorous methodology for the design of concurrent and distributed systems; it supports the vertical and the horizontal dimension of systems development through refinement and composition. However, there are very few approaches that combine both (e.g. [69]).

The work discussed in this document provides theoretical foundations for a fully tool-equipped approach and relies on the proposed bisimulation engines [111] and SMT Solving algorithms [53]. Practically, the advocated framework is based on an approach combining symbolic operational semantic and bisimulation equivalences with deductive reasoning on the data part, and in practice combining bisimulation algorithms with SMT solvers to get automatic procedures proving behavioural properties of these open systems.

Work still needs to be done in that field, we envisage several directions for extending our proposal, both theoretically and practically. Some future research that can be considered in the short term:

- Despite the rich language of pNets, simply expressing parametrized topologies (vectors, rows, rings, matrices,...) that are very useful in real applications is still very difficult. Expressing them requires to concretely extend the pNets model, by introducing a notion of topological parameters and spatial structures. Doing so requires enriching the automata description formalism, but also the synchronization mechanism.

- The composition operation of open automata is a kind of refinement that allows for an extension of the functionality. However, this refinement is not safe with respect to trace safety, because it is possible to introduce new traces in the implementation (necessarily because the intended use is that the specification model is a partial specification that is completed). However, the ability to complete the specification and preserve behaviours is a property that is sometimes required in some applications (e.g. [96]). There is a need to investigate under what conditions (structural or syntactic restrictions) the preservation can be guaranteed.

- The development of practical tools that supports equivalence checking of open pNets (open automata) models for verifying real-life concurrent systems. In fact, bisimulation algorithms have already been

implemented but they require further improvement. The challenges for the implementation, in the context of our symbolic systems, are not so much algorithmic complexity, as was the case with classical weak bisimulation on finite models, but decidability and termination. The naive approach, using an explicit construction of the weak transition, may in itself introduce non-termination, so the adopted solution was a direct implementation of the weak bisimulation definition [111], without constructing the weak automata beforehand, but searching on demand to construct the required weak transitions. It may be more effective to explore other more pragmatic approaches of weak bisimulation over weak automata.

Bibliography

[1] Martín Abadi and Leslie Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, 1991.

[2] Martín Abadi and Leslie Lamport. Conjoining specifications. *ACM Transactions on Programming Languages and Systems*, 17(3):507–535, 1995.

[3] Jean-Raymond Abrial. *The B-book - assigning programs to meanings*. Cambridge University Press, 1996.

[4] Marco Aldinucci, Carlo Bertolli, Sonia Campa, Massimo Coppola, Marco Vanneschi, and Ccorrado Zoccolo. Autonomic Grid Components: the GCM Proposal and Self-optimising ASSIST Components. In *Joint Workshop on HPC Grid programming Environments and COmponents and Component and Framework Technology in High-Performance and Scientific Computing at HPDC'15*, June 2006.

[5] Rajeev Alur and Thomas A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(1):7–48, 1999.

[6] Rajeev Alur, Thomas A. Henzinger, Orna Kupferman, and Moshe Y. Vardi. Alternating refinement relations. In Davide Sangiorgi and Robert de Simone, editors, *CONCUR: Concurrency Theory*, pages 163–178. Springer Berlin Heidelberg, 1998.

[7] Rabéa Ameur-Boulifa, Ana R. Cavalli, and Stephane Maag. Verifying complex software control systems from test objectives: Application to the ETCS system. In Marten van Sinderen and Leszek A. Maciaszek, editors, *Proceedings of the 14th International Conference on Software Technologies, ICSOFT*, pages 397–406. SciTePress, 2019.

[8] Rabéa Ameur-Boulifa, Raluca Halalai, Ludovic Henrio, and Eric Madelaine. Verifying safety of fault-tolerant distributed components. In *International Symposium on Formal Aspects of Component Software*, Lecture Notes in Computer Science. Springer, September 2011.

[9] Rabéa Ameur-Boulifa, Raluca Halalai, Ludovic Henrio, and Eric Madelaine. Verifying safety of fault-tolerant distributed components. In Farhad Arbab and Peter Csaba Ölveczky, editors, *Formal Aspects of Component Software*, pages 278–295. Springer Berlin Heidelberg, 2012.

[10] Rabéa Ameur-Boulifa, Ludovic Henrio, Oleksandra Kulankhina, Eric Madelaine, and Alexandra Savu. Behavioural semantics for asynchronous components. *Journal of Logical and Algebraic Methods in Programming*, 89:1–40, 2017.

[11] Rabéa Ameur-Boulifa, Ludovic Henrio, and Eric Madelaine. Compositional equivalences based on open pNets. *Journal of Logical and Algebraic Methods in Programming*, 131:100842, 2023.

[12] Nina Amla, E. Allen Emerson, Kedar S. Namjoshi, and Richard J. Trefler. Assume-guarantee based compositional reasoning for synchronous timing diagrams. TACAS 2001, pages 465–479, Berlin, Heidelberg, 2001. Springer-Verlag.

[13] Henrik Reif Andersen. Partial model checking. *Proceedings of Tenth Annual IEEE Symposium on Logic in Computer Science*, pages 398–407, 1995.

[14] André Arnold. Synchronised behaviours of processes and rational relations. *Acta Informatica*, 17:21–29, 1982.

[15] Isabelle Attali, Tomás Barros, and Eric Madelaine. Parameterized specification and verification of the chilean electronic invoices system. In *XXIV International Conference of the Chilean Computer Science Society*, pages 14–25. IEEE Computer Society, 2004.

[16] Didier Austry and Gérard Boudol. Algebre de processus et synchronisation. Research Report RR-0187, INRIA, 1983.

[17] Ralph-Johan Back and Joakim von Wright. *Refinement Calculus - A Systematic Introduction*. Graduate Texts in Computer Science. Springer, 1998.

[18] J.C.M. Baeten. A brief history of process algebra. *Theoretical Computer Science*, 335(2):131–146, 2005. Process Algebra.

[19] T. Barros, A. Cansado, E. Madelaine, and M. Rivera. Model checking distributed components : The Vercors platform. In *3rd workshop on Formal Aspects of Component Systems (FACS'06)*, volume 182, pages 3–16, Prague, Czech Republic, Sep 2007. ENTCS.

[20] Tomás Barros. *Spécification et vérification formelles des systèmes de composants répartis. (Formal specification and verification of distributed component systems)*. PhD thesis, University of Nice Sophia Antipolis, France, 2005.

[21] Tomás Barros, Rabéa Ameur-Boulifa, Antonio Cansado, Ludovic Henrio, and Eric Madelaine. Behavioural models for distributed Fractal components. *Annales des Télécommunications*, 64(1-2):25–43, 2009.

[22] Tomás Barros, Rabéa Boulifa, Antonio Cansado, Ludovic Henrio, and Eric Madelaine. Behavioural models for distributed fractal components (extended version). Research Report RR-6491, INRIA, 04 2008.

[23] Tomás Barros, Rabéa Boulifa, and Eric Madelaine. Parameterized models for distributed Java objects. In *Formal Techniques for Networked and Distributed Systems FORTE 2004*, volume LNCS 3235, pages 43–60, Madrid, September 2004. Spinger Verlag.

[24] F. Baude, D. Caromel, N. Maillard, and E.N. Mathias. Hierarchical mpi-like programming using gcm components as implementation support. Selected and presented at the CoreGRID workshop: Grid Systems, Tools and Environments, Grids@work II conference, Dec. 2006.

[25] Françoise Baude, Denis Caromel, Cédric Dalmasso, Marco Danelutto, Vladimir Getov, Ludovic Henrio, and Christian Pérez. GCM: A Grid Extension to Fractal for Autonomous Distributed Components. *Annals of Telecommunications*, 64(1):5–24, 2009.

[26] Sebastian S. Bauer, Philip Mayer, Andreas Schroeder, and Rolf Hennicker. On Weak Modal Compatibility, Refinement, and the MIO Workbench. In Javier Esparza and Rupak Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 175–189. Springer Berlin Heidelberg, 2010.

[27] Françoise Bellegarde, Célina Charlet, and Olga Kouchnarenko. *How to Compute the Refinement Relation for Parameterized Systems*, pages 181–209. Kluwer Academic Publishers, USA, 2004.

[28] Françoise Bellegarde, Jacques Julliand, and Olga Kouchnarenko. Ready-simulation is not ready to express a modular refinement relation. In Tom Maibaum, editor, *Fundamental Approaches to Software Engineering*, pages 266–283, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[29] Bernard Berthomieu, Jean-Paul Bodeveix, Patrick Farail, M Filali, Hubert Garavel, Pierre Gaufillet, Frederic Lang, and François Vernadat. Fiacre: an Intermediate Language for Model Verification in the Topcased Environment. In *4th European Congress ERTS Embedded Real Time Software*, January 2008.

[30] Tommaso Bolognesi and Ed Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks and ISDN Systems*, 14(1):25–59, 1987.

[31] Johannes Borgström, Sébastien Briais, and Uwe Nestmann. Symbolic Bisimulation in the Spi Calculus. In *CONCUR - Concurrency Theory, 15th International Conference*, volume 3170 of *LNCS*, pages 161–176. Springer, 2004.

[32] R. Boulifa and E. Madelaine. Model generation for distributed Java programs. In E. Astesiano N. Guelfi and G. Reggio, editors, *Workshop on scientiFic engIneering of Distributed Java applIcations*, Luxembourg, nov 2003. Springer-Verlag, LNCS 2952.

[33] Rabéa Boulifa. *Génération de modèles comportementaux des applications réparties*. PhD thesis, University of Nice - Sophia Antipolis – UFR Sciences, December 2004.

[34] Rabéa Boulifa, Ludovic Henrio, and Eric Madelaine. Behavioural models for group communications. In *WCSI-10: International Workshop on Component and Service Interoperability*, number 37 in EPTCS, pages 42–56, 2010.

[35] Mario Bravetti and Gianluigi Zavattaro. Asynchronous session subtyping as communicating automata refinement. *Software and Systems Modeling*, 20(2):311–333, 2021.

[36] Ed Brinksma, Giuseppe Scollo, and Chris Steenbergen. *Lotos Specifications, Their Implementations and Their Tests*, pages 468–479. IEEE Computer Society Press, Washington, DC, USA, 1995.

[37] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A Theory of Communicating Sequential Processes. *J. ACM*, 31(3):560 – 599, Jun 1984.

[38] Eric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Quéma, and Jean-Bernard Stefani. The FRACTAL component model and its support in Java. *Software Practice and Experience*, 36(11-12):1257–1284, 2006.

[39] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C–35(8):677–691, 1986.

[40] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[41] Maria Grazia Buscemi and Ugo Montanari. Open Bisimulation for the Concurrent Constraint Pi-Calculus. In *ESOP 2008 - Programming Languages and Systems, 17th European Symposium on Programming*, volume 4960 of *Lecture Notes in Computer Science*, pages 254–268. Springer, 2008.

[42] Muffy Calder and Carron Shankland. A symbolic semantics and bisimulation for full lotos. In *International Conference on Formal Techniques for Networked and Distributed Systems*, pages 185–200. Springer, 2001.

[43] Antonio Cansado, Denis Caromel, Ludovic Henrio, Eric Madelaine, Marcela Rivera, and Emil Salageanu. *A Specification Language for Distributed Components Implemented in GCM/ProActive*, pages 418–448. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[44] Antonio Cansado and Eric Madelaine. Specification and verification for grid component-based applications: from models to tools. In Frank S. de Boer, Marcello M. Bonsangue, and Eric Madelaine, editors, *FMCO 2008*, number 5751 in LNCS. Springer-Verlag, 2009.

[45] Sarah Chabane, Rabéa Ameur-Boulifa, and Mohamed Mezghiche. Rethinking of I/O-Automata Composition. In *Forum on specification & Design Languages (FDL)*, Verona, Italy, September 2017.

[46] Shing-Chi. Cheung and Jeff Kramer. Compositional reachability analysis of finite-state distributed systems with user-specified constraints. In *Proceedings of the 3rd ACM SIGSOFT Symposium on Foundations of Software Engineering*, SIGSOFT'95, pages 140–150, New York, NY,

USA, 1995. Association for Computing Machinery.

[47] Shing Chi Cheung and Jeff Kramer. Context constraints for compositional reachability analysis. *ACM Transactions on Software Engineering and Methodology*, 5(4):334–377, 1996.

[48] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In E. Allen Emerson and Aravinda Prasad Sistla, editors, *Computer Aided Verification*, pages 154–169, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[49] Rance Cleaveland and Oleg Sokolsky. CHAPTER 6 - Equivalence and Preorder Checking for Finite-State Systems. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, pages 391–424. Elsevier Science, Amsterdam, 2001.

[50] CoreGRID Programming Model Virtual Institute. Programming models for the single gcm component: a survey, Sep. 2006. Deliverable D.PM.06.

[51] Luca de Alfaro. Game Models for Open Systems. In Nachum Dershowitz, editor, *Verification: Theory and Practice*, volume 2772 of *Lecture Notes in Computer Science*, pages 269–289. Springer, 2003.

[52] Luca de Alfaro and Thomas A. Henzinger. Interface automata. In A Min Tjoa and Volker Gruhn, editors, *Proceedings of the 8th European Software Engineering Conference held jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering 2001, Vienna, Austria, September 10-14, 2001*, pages 109–120. ACM, 2001.

[53] Leonardo Mendonça de Moura and Nikolaj S. Bjørner. Z3: an efficient SMT solver. In *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.

[54] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Symbolic Bisimulation for the Applied Pi Calculus. In *FSTTCS 2007 - Foundations of Software Technology and Theoretical Computer Science, 27th International Conference*, volume 4855 of *LNCS*, pages 133–145. Springer, 2007.

[55] José Dihego, Augusto Sampaio, and Marcel Oliveira. A refinement checking based strategy for component-based systems evolution. *Journal of Systems and Software*, 167:110598, 2020.

[56] Edsger W. Dijkstra. *A discipline of programming*. Series in automatic computation. Prentice-Hall, 1976.

[57] E. Allen Emerson and Edmund M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In Jaco De Bakker and Jan Van Leeuwen, editors, *Automata, Languages and Programming*, pages 169–181. Springer Berlin Heidelberg, 1980.

[58] Rik Eshuis and Maarten M. Fokkinga. Comparing Refinements for Failure and Bisimulation Semantics. *Fundamenta Informaticae*, 52(4):297–321, 2002.

[59] Yuan Feng, Yuxin Deng, and Mingsheng Ying. Symbolic bisimulation for quantum processes. *ACM Transactions on Computational Logic (TOCL)*, 15(2):14, 2014.

[60] Hubert Garavel. Revisiting sequential composition in process calculi. *Journal of Logical and Algebraic Methods in Programming*, 84(6):742–762, 2015. "Special Issue on Open Problems in Concurrency Theory".

[61] Hubert Garavel, Frédéric Lang, and Radu Mateescu. An overview of CADP 2001. *European Association for Software Science and Technology Newsletter*, 4:13–24, aug 2002.

[62] Hubert Garavel, Frédéric Lang, and Radu Mateescu. Compositional verification of asynchronous concurrent systems using cadp. *Acta Informatica*, 52(4–5):337–392, jun 2015.

[63] Hubert Garavel, Frédéric Lang, and Laurent Mounier. Compositional Verification in Action.

In Falk Howar and Jiří Barnat, editors, *Formal Methods for Industrial Critical Systems*, pages 189–210, Cham, 2018. Springer International Publishing.

[64] Nuno Gaspar, Ludovic Henrio, and Eric Madelaine. Formally reasoning on a reconfigurable component-based system - A case study for the industrial world. In José Luiz Fiadeiro, Zhiming Liu, and Jinyun Xue, editors, *Formal Aspects of Component Software - 10th International Symposium, FACS 2013*, volume 8348 of *LNCS*, pages 137–156. Springer, 2013.

[65] Simon J. Gay and Malcolm Hole. Subtyping for session types in the pi calculus. *Acta Informatica*, 42(2-3):191–225, 2005.

[66] Dimitra Giannakopoulou, Kedar S. Namjoshi, and Corina S. Păsăreanu. *Compositional Reasoning*, pages 345–383. Springer International Publishing, Cham, 2018.

[67] Susanne Graf, Bernhard Steffen, and Gerald Lüttgen. Compositional minimisation of finite state systems using interface specifications. *Formal Aspects of Computing*, 8(5):607–616, sep 1996.

[68] Jan Friso Groote, Jeroen J. A. Keiren, Bas Luttik, Erik P. de Vink, and Tim A. C. Willemse. Modelling and Analysing Software in mCRL2. In Farhad Arbab and Sung-Shik Jongmans, editors, *Formal Aspects of Component Software*, pages 25–48, Cham, 2020. Springer International Publishing.

[69] Serge Haddad, Rolf Hennicker, and Mikael H. Møller. Specification of Asynchronous Component Systems with Modal I/O-Petri Nets. In *8th Symposium on Trustworthy Global Computing (TGC)*, volume LNCS 8358, pages 219–234, Buenos Aires, Argentina, 2014. Springer.

[70] D. Harel and A. Pnueli. On the development of reactive systems. In Krzysztof R. Apt, editor, *Logics and Models of Concurrent Systems*, pages 477–498, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg.

[71] David Harel. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[72] Matthew Hennessy and Huimin Lin. Symbolic bisimulations. *Theoretical Computer Science*, 138(2):353–389, 1995.

[73] Ludovic Henrio, Oleksandra Kulankhina, Siqi Li, and Eric Madelaine. *Integrated Environment for Verifying and Running Distributed Components*, volume 9633 of *Fundamental Approaches to Software Engineering*, pages 66–83. Eindhoven, Netherlands, Apr 2016.

[74] Ludovic Henrio, Eric Madelaine, and Min Madelaine. pnets: An expressive model for parameterised networks of processes. In Masoud Daneshtalab, Marco Aldinucci, Ville Leppänen, Johan Lilius, and Mats Brorsson, editors, *23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP*, pages 492–496. IEEE Computer Society, 2015.

[75] Ludovic Henrio, Eric Madelaine, and Min Zhang. A Theory for the Composition of Concurrent Processes. In Elvira Albert and Ivan Lanese, editors, *36th International Conference on Formal Techniques for Distributed Objects, Components, and Systems (FORTE)*, volume LNCS-9688, pages 175–194, Heraklion, Greece, June 2016.

[76] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, 1978.

[77] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International Series in Computer Science. Prentice Hall, 1985.

[78] Hans Hüttel and Kim Guldstrand Larsen. The use of static constructs in A modal process logic. In *Logic at Botik '89, Symposium on Logical Foundations of Computer Science*, volume 363 of *Lecture Notes in Computer Science*, pages 163–180. Springer, 1989.

[79] Anna Ingólfsdóttir and Huimin Lin. A symbolic approach to value-passing processes. In Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors, *Handbook of Process Algebra*, pages 427–478. North-Holland/Elsevier, 2001.

[80] Olga Kouchnarenko and Arnaud Lanoix. How to verify and exploit a refinement of component-based systems. In Irina Virbitskaite and Andrei Voronkov, editors, *Perspectives of Systems Informatics*, pages 297–309. Springer Berlin Heidelberg, 2007.

[81] Jean-Pierre Krimm and Laurent Mounier. Compositional state space generation from lotos programs. In Ed Brinksma, editor, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 239–258, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[82] Kåre J. Kristoffersen, François Laroussinie, Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. A compositional proof of a real-time mutual exclusion protocol. In Michel Bidoit and Max Dauchet, editors, *TAPSOFT: Theory and Practice of Software Development, 7th International*, volume 1214 of *LNCS*, pages 565–579. Springer, 1997.

[83] Kim Guldstrand Larsen and Bent Thomsen. A modal process logic. *[1988] Proceedings. Third Annual Information Symposium on Logic in Computer Science*, pages 203–210, 1988.

[84] Kim Guldstrand Larsen and Liu Xinxin. Compositionality through an operational semantics of contexts. *Journal of Logic and Computation*, 1(6):761–795, 1991.

[85] Huimin Lin. Symbolic transition graph with assignment. In Ugo Montanari and Vladimiro Sassone, editors, *CONCUR: Concurrency Theory*, pages 50–65. Springer Berlin Heidelberg, 1996.

[86] Jørn Lind-Nielsen, Henrik Reif Andersen, Gerd Behrmann, Henrik Hulgaard, Kåre J. Kristoffersen, and Kim Guldstrand Larsen. Verification of large state/event systems using compositionality and dependency analysis. In Bernhard Steffen, editor, *Tools and Algorithms for Construction and Analysis of Systems, 4th International Conference, TACAS*, volume 1384 of *Lecture Notes in Computer Science*, pages 201–216. Springer, 1998.

[87] Jia Liu and Huimin Lin. A Complete Symbolic Bisimulation for Full Applied Pi Calculus. In *SOFSEM - 36th Conference on Current Trends in Theory and Practice of Computer Science*, volume 5901, pages 552–563. Springer, 2010.

[88] Hong-Viet Luong, Thomas Lambolais, and Anne-Lise Courbis. Implementation of the Conformance Relation for Incremental Development of Behavioural Models. In Krzysztof Czarnecki, Ileana Ober, Jean-Michel Bruel, Axel Uhl, and Markus Völter, editors, *Model Driven Engineering Languages and Systems, 11th International Conference, MoDELS*, volume 5301 of *LNCS*, pages 356–370. Springer, 2008.

[89] Radu Mateescu, Pascal Poizat, and Gwen Salaün. Behavioral Adaptation of Component Compositions based on Process Algebra Encodings. In *Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 385–388, Atlanta, United States, 2007.

[90] K. L. McMillan. A compositional rule for hardware design refinement. In Orna Grumberg, editor, *Computer Aided Verification*, pages 24–35. Springer Berlin Heidelberg, 1997.

[91] R. Milner. *Communication and Concurrency*. Int. Series in Computer Science. Prentice-Hall, Englewood Cliffs, New Jersey, 1989. SU Fisher Research 511/24.

[92] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.

[93] Robin Milner. *A Calculus of Communicating Systems*. Springer-Verlag, Berlin, Heidelberg, 1982.

[94] Robin Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25(3):267–310, 1983.

[95] Robin Milner. *Communication and Concurrency*. Int. Series in Computer Science. Prentice-Hall, Englewood Cliffs, New Jersey, 1989. SU Fisher Research 511/24.

[96] Hocine Mokrani, Rabéa Ameur-Boulifa, and Emmanuelle Encrenaz-Tiphène. Assisting refinement in system-on-chip design. In *FDL*, pages 1–6. IEEE, 2013.

[97] Matthieu Morel. *Composants pour la grille. (Components for grid computing)*. PhD thesis, University of Nice Sophia Antipolis, France, 2006.

[98] OMG. Unified Modeling Language (UML), 2011.

[99] David Park. Concurrency and automata on infinite sequences. In Peter Deussen, editor, *Theoretical Computer Science*, pages 167–183, Berlin, Heidelberg, 1981. Springer Berlin Heidelberg.

[100] A. Pnueli. *Applications of temporal logic to the specification and verification of reactive systems: A survey of current trends*, pages 510–584. pringer Berlin Heidelberg, Berlin, Heidelberg, 1986.

[101] Pascal Poizat and Gwen Salaün. Adaptation of open component-based systems. In Marcello M. Bonsangue and Einar Broch Johnsen, editors, *Formal Methods for Open Object-Based Distributed Systems*, pages 141–156. Springer Berlin Heidelberg, 2007.

[102] Christian Prehofer. Behavioral refinement and compatibility of statechart extensions. *Electronic Notes in Theoretical Computer Science*, 295:65–78, 2013.

[103] Xudong Qin, Simon Bliudze, Eric Madelaine, Zechen Hou, Yuxin Deng, and Min Zhang. SMT-based generation of symbolic automata. *Acta Informatica*, 57:627–656, May 2020.

[104] Xudong Qin, Simon Bliudze, Eric Madelaine, and Min Zhang. Using SMT engine to generate symbolic automata. In *18th International Workshop on Automated Verification of Critical Systems (AVOCS)*. Electronic Communications of the EASST, 2018.

[105] Davide Sangiorgi. On the origins of bisimulation and coinduction. *ACM Transactions on Programming Languages and Systems*, 31(4), May 2009.

[106] Joseph Sifakis. A unified approach for studying the properties of transition systems. *Theoretical Computer Science*, 18(3):227–258, 1982.

[107] Rishabh Singh, Dimitra Giannakopoulou, and Corina Păsăreanu. Learning component interfaces with may and must abstractions. In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *Computer Aided Verification*, pages 527–542. Springer Berlin Heidelberg, 2010.

[108] Kuo-Chung Tai and P.V. Koppol. An incremental approach to reachability analysis of distributed programs. In *Proceedings of 1993 IEEE 7th International Workshop on Software Specification and Design*, pages 141–150, 1993.

[109] Rob J. van Glabbeek. The Linear Time - Branching Time Spectrum I. In Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors, *Handbook of Process Algebra*, pages 3–99. North-Holland/Elsevier, 2001.

[110] Moshe Y. Vardi. Verification of open systems. In S. Ramesh and G. Sivakumar, editors, *Foundations of Software Technology and Theoretical Computer Science*, pages 250–266. Springer Berlin Heidelberg, 1997.

[111] Biyang Wang, Eric Madelaine, and Min Zhang. Symbolic Weak Equivalences: Extension, Algorithms, and Minimization - Extended version. Research Report RR-9389, Inria, Université Cote d'Azur, CNRS, I3S, Sophia Antipolis, France ; East China Normal University (Shanghai), January 2021.

[112] Glynn Winskel. On the Compositional Checking of Validity (Extended Abstract). In *Proceedings on Theories of Concurrency: Unification and Extension*, CONCUR, pages 481–501. Springer-Verlag, 1990.

[113] Fei Xie, Guowu Yang, and Xiaoyu Song. Compositional reasoning for hardware/software coverification. In Susanne Graf and Wenhui Zhang, editors, *Automated Technology for Verification and Analysis*, pages 154–169. Springer Berlin Heidelberg, 2006.

[114] Lianyi Zhang, Qingdi Meng, and Kueiming Lo. Compositional abstraction refinement for component-based systems. *Journal of Applied Mathematics*, 2014:1–12, 2014.