# Descriptions formelles : Comprendre, Corriger, Implanter, Résutiliser. Application au contrôle d'accès

Mathieu Jaume

▶ **To cite this version:**

Mathieu Jaume. Descriptions formelles : Comprendre, Corriger, Implanter, Résutiliser. Application au contrôle d'accès. Informatique [cs]. Pierre et Marie Curie, Paris VI, 2008. tel-04273170

**HAL Id: tel-04273170**
**https://hal.science/tel-04273170**

Submitted on 7 Nov 2023

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Public Domain

UNIVERSITÉ PIERRE ET MARIE CURIE – PARIS 6

HABILITATION À DIRIGER DES RECHERCHES

Spécialité : INFORMATIQUE

présentée par :

Mathieu JAUME

# Descriptions formelles : Comprendre, Corriger, Implanter, Réutiliser

# Application au contrôle d'accès

Soutenue le 06 Novembre 2008 devant le jury composé de :

*Rapporteurs*

| | | |
|---|---|---|
| Hélène KIRCHNER | Directrice de Recherche | INRIA Bordeaux, France |
| César A. MUÑOZ | Senior Scientist | National Institute of Aerospace, USA |
| Luca VIGANÒ | Professeur | Università di Verona, Italie |

*Examinateurs*

| | | |
|---|---|---|
| Irène GUESSARIAN | Professeur | Université Paris 6, France |
| Thérèse HARDIN | Professeur | Université Paris 6, France |
| Ludovic MÉ | Professeur | SUPELEC Rennes, France |
| Isabelle SIMPLOT-RYL | Professeur | Université Lille 1, France |

*A la mémoire de Ivan Evrard.*

# Remerciements

Hélène Kirchner, César A. Muñoz et Luca Viganò ont accepté de rapporter ce document. Je leur exprime ici toute ma gratitude pour leur lecture attentive de ce document, pour les remarques qu'ils m'ont faites pour l'améliorer ainsi que pour les discussions que nous avons eues.

Depuis 2001, j'ai la chance immense de travailler aux côtés de Thérèse Hardin. Son enthousiasme et sa gentillesse m'ont accompagné ces dernières années, durant lesquelles j'ai pu profiter de sa patience, de sa disponibilité, de son soutien, de son optimisme ainsi que de ses connaissances scientifiques dans de nombreux domaines. Mais je voudrais surtout remercier Thérèse pour le plaisir que j'ai à travailler dans l'équipe SPI qu'elle dirige.

Le premier cours de sémantique que j'ai suivi lorsque j'étais étudiant était celui que donnait Irène Guessarian. Je lui dois donc beaucoup pour m'avoir fait découvrir ce domaine. Irène a ensuite présidé mon jury de thèse et m'a accueilli dans son équipe pédagogique lors de mon arrivée à l'UPMC. Irène est depuis une collègue loyale et généreuse et a accepté de présider le jury de cette HDR. J'en suis très heureux.

J'ai rencontré Ludovic Mé lors de la soutenance de thèse de Charles Morisset. Son intérêt pour la sécurité et les méthodes formelles nous ont alors conduit à travailler ensemble sur la détection de flots. Je tiens bien sûr à remercier Ludovic pour sa participation à ce jury, mais aussi pour les échanges fructueux que nous avons eus.

J'adresse aussi de chaleureux remerciements à Isabelle Simplot-Ryl pour avoir consacré du temps à la lecture de ce document et je suis très honoré de sa participation au jury de cette HDR.

A l'issue de ma thèse, j'ai commencé à travailler avec Catherine Dubois. Depuis, nos chemins se croisent régulièrement et nous conduisent à collaborer sur des sujets très variés. Je tiens ici à remercier Catherine pour avoir accompagné mes activités de recherche ainsi que pour sa gentillesse et son soutien.

L'encadrement de doctorants est enrichissant à plus d'un titre et je tiens à remercier Charles Morisset (qui a soutenu sa thèse en 2007), ainsi que Lionel Habib et Liên Tran (actuellement en thèse). J'ai beaucoup appris en tentant de répondre à leurs nombreuses questions, toujours pertinentes, et les remercie pour leur confiance et leurs contributions.

L'ensemble des travaux présentés ici est bien sûr le fruit de collaborations : Véronique Delebarre, Catherine Dubois, Thérèse Hardin, Ludovic Mé, Virgile Prevosto, Valérie Viet Triem Tong ont contribué au travail présenté dans ce document et je les remercie vivement de m'avoir permis de partager avec eux mes activités de recherche.

Je tiens aussi à remercier Philippe Ayrault, Julien Blond, Matthieu Carlier, David Delahaye, Damien Doligez, Emmanuel Gureghian, Eric Jaeger, Ivan Noyer, François Pessaux, Renaud Rioboo, Véronique Viguié-Donzeau-Gouge, Pierre Weis, très proches collaborateurs de l'équipe SPI, qui par leur aide, leurs encouragements et leurs remarques m'ont permis de mener mes recherches dans un environnement stimulant. J'associe à ces remerciements l'ensemble des participants des actions Modulogic et ANR SSURF, ainsi que du PlanPluriFormation Logiciels Sûrs.

Mon goût pour les études coïncide avec ma rencontre avec Alain David. Il ne s'agit bien sûr pas d'une simple coïncidence, et je tiens à remercier Alain pour l'originalité de ses points de vue et pour son soutien depuis de très nombreuses années maintenant.

Kevin Adoua a toujours su m'accompagner dans mes démarches administratives. Je le remercie pour son efficacité et sa gentillesse.

Je souhaite aussi remercier mes parents pour m'avoir les premiers appris à apprendre et pour leurs encouragements. Enfin, je remercie Areski, Caroline, James, Luc, Mark, Myriam, Olivier, Philippe et Sylvie qui avec beaucoup de délicatesse, de chaleur, de rires et de sourires me rendent la vie si douce et si joyeuse.

# Table des matières

# Introduction

### Des détails qui n'en sont pas

Comme dans toutes les autres disciplines scientifiques, le besoin de recourir à des modèles et formalismes mathématiques se fait ressentir en informatique, pour mieux comprendre et analyser les problèmes traités par cette science. C'est de ce besoin que viennent les méthodes dites "formelles". Dans [4], P. Amey définit la "chose" formelle comme une "chose" soutenue par une rigueur mathématique. Effectivement, les méthodes formelles peuvent être définies comme des méthodes soutenues par une rigueur mathématique, et peut-être plus exactement, comme des méthodes s'appuyant fortement sur une (ou des) branche(s) des mathématiques. Leur utilité première est leur imperméabilité à toute ambiguïté. Cela est très apprécié lorsqu'il s'agit de spécifier (décrire ce qui est requis d'un système) ou encore lorsqu'il s'agit de développer (architecture, code). De plus, elles permettent très naturellement de démontrer le respect de certaines propriétés et donc, d'augmenter la confiance dans le système, que ce soit sur la cohérence de sa spécification ou sur la bonne adéquation de son développement à cette spécification. Lorsqu'il s'agit de systèmes logiciels critiques, ces propriétés peuvent être vitales.

Le besoin de confiance est déjà fortement ressenti pour les "outils de base" de l'informatique comme les algorithmes classiques. Plus qu'une simple vérification, la spécification formelle de tels outils et la formalisation des preuves de leurs propriétés permettent bien sûr d'accroître la fiabilité de leur implémentation. Mais elles permettent souvent de mettre en lumière certains points délicats et donc sources d'erreurs ou de confusions, que le "monde informel" ignore volontairement par souci de "lisibilité". Ces points délicats sont souvent injustement qualifiés de "détails". Selon le dictionnaire de l'Académie Française, le mot "détail" peut désigner un élément accessoire, un fait secondaire sur lequel on passe légèrement, un point de peu d'importance ou encore un élément négligeable. La pratique des méthodes formelles conduit donc à se pencher sur des détails qui de fait n'en sont pas ... et si la présentation des travaux issus de cette pratique est rarement spectaculaire, les problèmes qu'elle permettrait d'éviter peuvent parfois l'être :

- En 1996, le vol inaugural du lanceur européen Ariane 5 "crash" quelques secondes après son décollage. L'explosion, due à une erreur logicielle (non respect d'une contrainte de domaine) a coûté environ un demi milliard de dollars.
- En Novembre 2005, un bogue bloque toutes les côtations durant une journée à la bourse de Tokyo.
- En Octobre 2007, un canon anti-aérien Oerlikon de l'armée Sud-Africaine s'est retourné, tout en tirant, au hasard, faisant 9 morts et 14 blessés. Le logiciel est en cause dans cet incident.

- Le 13 Novembre 2007 au soir, 10000 sites web hébergés par Ovh, premier hébergeur français, connaissent des difficultés. En effet, Ovh subit un "crash" majeur sur un des 16 serveurs de stockage. Les sites ont été inaccessibles (erreur 404) une vingtaine d'heures environ et les données ajoutées durant les 7 derniers mois dans les CMS par les utilisateurs ont été définitivement perdues.

Cette liste n'est évidemment pas exhaustive et une étude menée par le *National Institute of Standards and Technology* évalue à 60 milliards de dollars par an le coût des bogues informatiques. Rappelons qu'il est souvent considéré que les éditeurs de logiciels consacrent environ 80% de leurs dépenses à identifier et à corriger des erreurs de logiciel, mais cela ne suffit pas à empêcher la multiplication des bogues.

Les méthodes formelles reposent sur l'utilisation d'outils mathématiques pour spécifier et valider les systèmes informatiques. Le recours à ces outils est justifié par le besoin de rigueur et de précision dans la description du comportement de systèmes complexes, et par la nécessité de disposer de mécanismes d'abstraction afin de juguler cette complexité. Les travaux présentés dans ce document illustrent la mise en œuvre de méthodes formelles et des outils logiciels associés pour étudier des objets variés. Pour certains d'entre eux, la littérature fournit une description mathématique et il peut sembler inutile d'exprimer cette description à l'aide d'un langage formel logique. Pourtant, une lecture minutieuse de ces descriptions, nécessaire pour obtenir un développement formel, révèle l'existence de plusieurs définitions non équivalentes pour un même objet. Les différences qui apparaissent entre ces définitions sont alors parfois sources de confusions ou d'erreurs dans les preuves mais aussi dans les énoncés de résultats garantissant des propriétés sur l'objet formalisé. L'exemple le plus typique est celui de la notion de substitution. Cette opération permet de remplacer certaines variables par un terme au sein d'une expression et peut sembler à première vue simple à spécifier. Cependant, la spécification formelle de cette opération nécessite de préciser certains "détails" : s'agit-il d'une fonction totale ? d'une fonction partielle ? quelles sont les propriétés de séparation des variables apparaissant dans le domaine et dans l'image de cette fonction ? Des réponses différentes à ces questions conduisent à des définitions non équivalentes. En rassemblant des résultats établis à partir de définitions non équivalentes pour une même notion, il devient alors possible d'établir des résultats erronnés si l'on omet ces "détails". Formaliser permet donc non seulement de préciser l'objet formalisé mais aussi de corriger certaines erreurs. Bien sûr, une difficulté supplémentaire apparait lorsque l'objet à formaliser est décrit en langage naturel. Plusieurs interprétations de cette description sont généralement possibles et il s'agit de lever les ambiguïtés et d'expliciter les hypothèses implicites. Dans ce cas, le travail de formalisation permet non seulement de corriger certaines confusions mais permet aussi d'avoir une compréhension plus fine de l'objet par sa formalisation.

Si la mise en œuvre des méthodes formelles permet de garantir les propriétés essentielles d'un algorithme, celles des mécanismes d'exécution d'un programme ou encore des propriétés de sécurité d'un système d'information, cette mise en œuvre a un coût certain : temps de développement, connaissances requises, technicité du développement ... Aussi, il convient de chercher à réduire ce coût non seulement en étudiant les techniques de réutilisation de développements formels, en outillant ces méthodes, mais aussi en concevant des cadres génériques dédiés aux domaines considérés et permettant de factoriser le travail de développement. Une partie du chapitre 1 est consacrée à cette problématique et

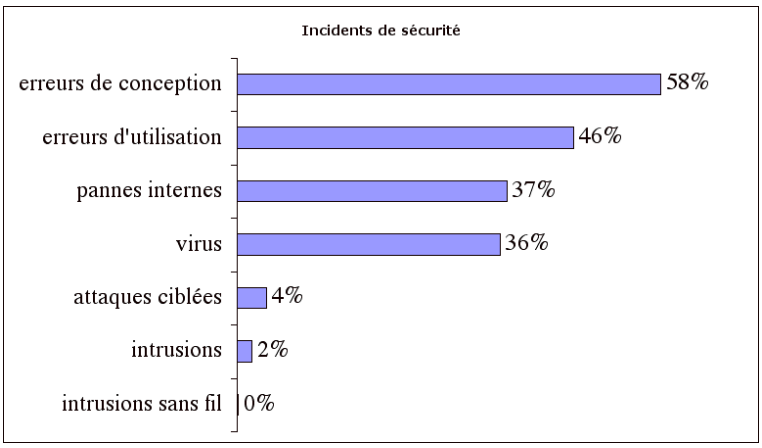


FIG. 1 – Incidents de sécurité (Source : enquête 2005 du CLUSIF)

le chapitre 2 est dédié à la construction *ex-nihilo* d'un nouveau cadre générique dédié à un domaine particulier, celui du contrôle d'accès, domaine sur lequel repose, en partie, la sécurité des systèmes d'information.

Selon Vinton Cerf, co-fondateur du protocole TCP/IP, entre 100 et 150 millions des 600 millions d'ordinateurs connectés à Internet sont actuellement sous le contrôle direct de pirates[1]. Ces ordinateurs sont utilisés à l'insu de leur propriétaire légitime afin d'effectuer toutes sortes d'actions illégales, comme l'envoi de "spam" ou des attaques contre des serveurs. De fait, la maîtrise de l'information est devenue aujourd'hui un enjeu primordial pour les états, pour les entreprises mais aussi pour les citoyens. En 2005, le CLUSIF (Club de la Sécurité de l'Information Français) fournissait une classification d'incidents de sécurité et une estimation quantifiée des causes de ces incidents. Les résultats de cette étude, présentés dans la figure 1, montrent que 58% des incidents de sécurité sont dûs à des erreurs de conception. Les domaines liés à la sûreté (nucléaire, ferroviaire, aérospatiale, etc) ont mis en place un arsenal de mesures et techniques pour la spécification, la conception et l'implantation des parties logicielles d'un système, permettant de déjouer les menaces causées par l'usure des composants, les perturbations de transmission des signaux, etc. Dans les domaines liés à la sécurité, de tels procédés, destinés à déjouer les menaces relatives aux attaques informatiques, ne sont mis que lentement en place et de manière encore trop parcellaire. Les demandes de tels outils sont d'autant plus pertinentes que l'intégration des communications réseaux dans les grands systèmes informatiques abolit toute notion de barrière physique de protection. Dans le domaine du logiciel, les Critères Communs [22] fournissent une méthodologie permettant d'atteindre des hauts niveaux de sûreté. Les hauts niveaux de sûreté des Critères Communs (EAL – *Evaluation Assurance Level* – 5 à 7) requièrent l'utilisation de méthodes formelles dans les étapes de spécification et de conception du logiciel.

La sécurité d'un système d'information repose sur plusieurs mécanismes aux objectifs différents :

[1] Article de la BBC du 25/01/07.

- Prévenir : Empêcher les attaques (moniteur de référence, mécanismes d'isolation, de confinement, etc).
- Résister : Limiter la portée des attaques (pare-feu, "chroot", cloisonnement, etc).
- Détecter : Surveiller le système pour identifier les attaques (systèmes de détection d'intrusion, analyse de "logs", etc).
- Réparer : Remettre en état le système après les attaques (sauvegarde, récupération d'informations, etc).

Dans ce document nous abordons principalement les aspects liés à la prévention et plus particulièrement au contrôle d'accès. En effet, le contrôle des flots d'information dans les réseaux et dans les systèmes d'information est un élément clé dans la construction de la sécurité d'un système informatique. Ce contrôle nécessite de développer des mécanismes permettant de filtrer les accès afin de ne laisser passer que ceux autorisés, et donc de définir une politique de sécurité, c'est-à-dire la caractérisation des accès permis. Le programme chargé de mettre en application cette politique, le moniteur de référence, est souvent considéré comme l'une des clés de voûte de la sécurité d'un système. Sa conception et son développement doivent être menés de manière à garantir sa fiabilité et sa sûreté. En effet, toute faille au sein de ce programme pourrait entraîner des violations de la politique de sécurité. L'emploi des méthodes formelles dans le développement d'un moniteur de référence permet de garantir que certaines propriétés sont toujours respectées.

Ce document synthétise les activités de recherche que j'ai effectuées dans les domaines évoqués dans cette introduction. Le chapitre 1 présente un ensemble de formalisations effectuées avec les systèmes Coq et Focal, ainsi qu'un ensemble de techniques pour permettre une plus grande réutilisabilité des développements formels obtenus. Le chapitre 2 présente les développements effectués dans le domaine du contrôle d'accès. Il s'agit d'un travail démarré en 2004 en collaboration avec C. Morisset et depuis 2007, avec L. Habib. On peut également lire ce chapitre 2 comme un exemple de formalisation à base de mathématiques classiques et de logique d'un domaine de connaissance donné, formalisation dont on suivra les évolutions, rendues nécessaires pour refléter plus exactement les caractéristiques du domaine étudié, au fur et à mesure de leur explicitation. Il est intéressant de noter comment, un certain niveau de formalisation étant atteint, des questions non formulables jusque là deviennent explicites. Y répondre conduit à revoir la formalisation mise en place pour faire émerger les concepts restés enfouis dans la connaissance des experts du domaine. Du fait de la limitation imposée sur le nombre de pages, plusieurs annexes permettent d'éclairer certaines parties du document : l'annexe A présente rapidement l'atelier Focal, l'annexe B explicite les formalisations réalisées dans le domaine de la programmation logique, l'annexe C décrit sur des exemples concrets les techniques mises en œuvre pour faciliter la réutilisation de développements formels et l'annexe D complète le chapitre 2 par un ensemble de définitions, de propriétés et d'exemples.

# Chapitre 1

# Développements formels

Les travaux présentés dans ce document ont en commun l'utilisation de méthodes formelles. Ces travaux portent sur des objets variés mais partagent un même objectif : obtenir une description formelle de ces objets à un niveau de détail proche de celui d'une implantation afin de mieux les comprendre, d'en garantir les propriétés fondamentales et éventuellement d'en construire une implantation certifiée. Formaliser un algorithme, la sémantique d'un langage, ou encore certains aspects liés à la sécurité d'un système d'information permet d'en donner une description au travers de définitions et de spécifications non ambiguës et explicites. Cette description s'accompagne d'un ensemble de propriétés dont les preuves reposent sur les définitions. Le niveau de détail visé lors de ce travail de formalisation doit permettre de garantir ces propriétés, grâce à la mécanisation des définitions et des raisonnements à l'aide de systèmes d'aide à la preuve.

La première partie de ce chapitre (section 1.1) décrit un ensemble de développements formels réalisés en essayant de dégager les apports obtenus par le travail de formalisation. Certains de ces travaux sont développés en annexe B. Les formalisations que j'ai réalisées portent essentiellement sur la logique (unification, sémantique de la programmation logique, formalismes logiques) durant la période 1997-2003, et depuis 2004, sur le contrôle d'accès. L'ensemble des travaux présentés dans la section 1.1 a permis d'aborder la problématique de la réutilisation de développements formels et de l'ingénierie de la preuve. En effet, si les systèmes d'aide à la preuve connaissent une utilisation croissante, trop souvent encore, écrire une preuve formelle nécessite de spécifier à partir de "presque rien" (i.e. *"from scratch"*) les objets considérés avant de construire les preuves des propriétés que l'on souhaite obtenir.

Formaliser une théorie, c'est décrire dans un langage formel les objets de cette théorie, exprimer, dans ce même langage, les propriétés que ces objets peuvent vérifier et prouver certaines d'entre elles, souvent à l'aide d'un langage de tactiques. Il s'agit donc d'un développement de logiciel et la problématique de la réutilisation de développements formels est naturellement proche de celle de la réutilisation de programmes. On retrouve donc les notions classiques de modularité, de généricité, d'abstraction ... qui sont des critères importants de réutilisabilité. Une approche par réutilisation pose cependant quelques problèmes. En effet, rares sont les formalisations directement réutilisables : on peut distinguer trois classes de contexte de réutilisation :

– La formalisation existante porte sur un "problème isomorphe" et il reste à "transposer"

les propriétés formelles du "domaine existant" (ou d'un sous-ensemble de ce domaine) au domaine considéré.
- La preuve existante est "presque identique" et il suffit de l'adapter.
- Les définitions et les preuves formelles existantes sont génériques (i.e. paramétrées) et il suffit d' "instancier" l'existant pour obtenir directement la formalisation souhaitée (il s'agit là de la situation idéale).

Dans la section 1.2, nous illustrons chacune de ces situations à l'aide d'exemples concrets et esquissons les méthodes mises en œuvre pour réutiliser effectivement des développements formels existants. Une présentation plus détaillée de ces travaux est donnée en annexe C.

## 1.1 Formalisations

Nous présentons dans cette section les formalisations effectuées en indiquant ce que l'activité de formalisation a permis de comprendre et/ou de corriger.

### 1.1.1 Unification et SLD-résolution

Les résultats fondamentaux de la programmation logique sont en général exposés en omettant les détails techniques relatifs aux mécanismes de renommage mis en œuvre lors de l'exécution de programmes logiques. Pourtant, ces mécanismes jouent un rôle important et sont sources d'erreur dans certains énoncés et preuves. Ces détails ne peuvent pas être ignorés si l'on souhaite effectivement implanter un mécanisme d'exécution des programmes logiques. La spécification de la manipulation des identificateurs, et donc de la sémantique opérationnelle d'un langage, doit être suffisamment explicite pour l'implanteur d'un interpréteur ou d'un compilateur pour ce langage. Afin d'expliciter ces mécanismes, j'ai donc formalisé, dans le système Coq, les propriétés essentielles de la SLD-résolution, mécanisme de base de la programmation logique.

Le langage de programmation logique étudié (correspondant au noyau logique de Prolog) a été représenté dans le langage du Calcul des Constructions. Deux plongements de nature différente ont été réalisés. Dans un premier temps, un "plongement profond" (*deep embedding*) a été effectué [59, 61] : il s'agit ici de représenter la syntaxe abstraite d'un langage, le langage "plongé" (langage de la programmation logique), par un type d'un autre langage, le langage "de plongement" (le langage du Calcul des Constructions) puis de définir la sémantique du langage plongé dans le langage de plongement. Cette approche permet d'analyser finement le langage "plongé". Dans un deuxième temps, un "plongement superficiel" (*shallow embedding*) a été effectué [62, 64, 63, 65] : il s'agit ici de représenter un programme du langage "plongé" par un programme du "langage de plongement". Cette approche permet d'analyser les programmes "plongés" mais ne permet pas de prouver dans le langage de plongement les propriétés du langage plongé.

A partir d'une formalisation de l'unification au premier ordre, pour obtenir un plongement profond du langage de programmation logique, j'ai formalisé tout d'abord la sémantique opérationnelle des programmes logiques, définie par la SLD-résolution et basée sur le mécanisme d'unification. Cela m'a conduit à expliciter totalement et *a priori* les conditions de renommage des variables mises en jeu lors d'une SLD-dérivation et cela m'a

permis de détecter certaines erreurs et confusions dans la littérature. Deux lemmes classiques de généralisation et de commutation ont pu être prouvés dans le système Coq. Le premier permet d'établir que si l'exécution d'un programme logique à partir d'une requête $R$ "réussit", alors l'exécution de ce programme à partir d'une requête dont $R$ est une instance "réussit" aussi. Le deuxième permet d'établir que le non-déterminisme dans le choix de l'atome sélectionné à chaque étape de résolution relève d'un non-déterminisme par indifférence. Enfin, la sémantique déclarative des programmes a été décrite dans ce même cadre et les preuves des deux résultats fondamentaux de validité et de complétude de la SLD-résolution ont pu être mécanisées. Ces deux preuves ont été obtenues en suivant l'approche "traditionnelle" introduite par M.H. van Emden et R.A. Kowalski dans [124]. Cette approche utilise les notions d'interprétations de Herbrand et de points fixes d'un opérateur continu associé à tout programme et correspond à celle que l'on peut trouver dans la plupart des ouvrages de base consacrés à la programmation logique. Bien sûr, d'autres approches plus récentes ont été développées pour établir ces résultats, mais notre objectif n'était pas d'écrire *une* preuve formalisée de ces résultats, mais *la* preuve traditionnelle de ces résultats. Les sections B.1 et B.2.1 décrivent plus amplement l'ensemble de ce travail.

Alors que les résultats classiques de la programmation logique concernent la sémantique des calculs finis, certains programmes logiques peuvent donner lieu à des exécutions infinies. Pour de tels programmes, ces résultats ne s'appliquent pas et il n'existe pas une unique "sémantique classique" mais plusieurs approches non équivalentes. Dans la plupart d'entre elles, la dénotation d'un programme logique $P$ est obtenue en considérant le plus grand point fixe d'un opérateur associé à $P$. Hélas, lorsque l'univers du discours contient des éléments infinis, l'utilisation d'un plus grand point fixe conduit à définir une sémantique valide mais non complète (i.e., certains éléments apparaissant dans la dénotation d'un programme $P$ ne sont pas "constructibles" par une exécution de $P$). Les approches existantes sont exclusivement dédiées à la caractérisation d'objets infinis. L'approche que nous avons développée correspond à une démarche "opposée" : l'univers du discours considéré ne contient pas d'éléments infinis et seules les exécutions ne construisant aucun terme infini sont considérées. Il s'agit là d'une restriction qui permet de définir une sémantique complète en terme de plus grand point fixe d'un opérateur associé au programme considéré.

Comme point de départ pour cette étude des exécutions infinies, les preuves SLD, finies ou infinies, engendrées pas les exécutions d'un programme logique ont été comparées à celles que l'on peut obtenir, par induction ou par co-induction, à partir des clauses d'un programme logique vues comme des règles d'inférence (les dérivations infinies ont été interprétées selon l'isomorphisme de Curry-Howard en identifiant ces dérivations infinies à des $\lambda$-termes d'un type co-inductif). Cette lecture sémantique permet de considérer un programme, non plus comme une théorie du premier ordre, mais comme un ensemble de définitions (co-)inductives. Dans le cas fini, la correspondance entre preuves et SLD réfutations est complète. Dans le cas infini, certains objets non calculables (par une SLD dérivation) sont toutefois prouvables (par co-induction), la correspondance est donc incomplète. Un plongement superficiel de la programmation logique dans le Calcul des Constructions permet non seulement d'expliquer ce phénomène d'incomplétude mais aussi de définir une sémantique valide et complète pour une classe d'exécutions infinies. Ce travail est décrit dans la section B.2.2.

### 1.1.2 Modèles de contrôles d'accès

Nous présentons à présent les formalisations réalisées dans le domaine de la sécurité informatique, et plus précisément dans le domaine du contrôle d'accès. Différentes politiques de sécurité peuvent être utilisées pour définir un mécanisme de contrôle d'accès. Toutefois, rien ne sert de mettre en place une politique de sécurité pour gérer un système si les programmes chargés de garantir le bon fonctionnement de cette politique ne sont pas fiables. Cette section rend compte de manière informelle de différentes expériences réalisées depuis 2004 en collaboration avec E.Gureghian, T.Hardin et C.Morisset, synthétisées dans [70], afin d'obtenir des développements formels de politiques de contrôle d'accès. Comme nous le verrons, ces développements nous ont conduit à introduire un "cadre sémantique", présenté dans le chapitre 2, dans lequel il est possible de spécifier par étapes successives, d'implanter et de comparer des politiques de contrôle d'accès.

Les travaux présentés dans cette section ont pour objectif de certifier des programmes en charge de la sécurité dans un système d'information. Plus précisément, il s'agit de garantir qu'un moniteur de référence chargé du contrôle des accès dans un système maintient une politique de contrôle d'accès donnée. Cette propriété est bien sûr cruciale pour la plupart des systèmes d'information. Aux difficultés déjà évoquées dans la section 1.1.1 s'ajoutent avec cet exemple celles liées au passage de l'informel au formel. En effet, bon nombre de politiques sont exprimées de manière informelle dans la littérature. Pour illustrer cela, considérons l'exemple classique de la politique de la Muraille de Chine, introduite par D.F.C. Brewer et M.J. Nash [19], pour résoudre les problèmes de conflit d'intérêt dans le monde des consultants. Chaque objet du système appartient à un ensemble de données d'une compagnie, et chaque compagnie appartient à une unique classe de conflit d'intérêt. Ces classes de conflit correspondent à des milieux professionnels distincts, comme par exemple les banques ou les compagnies pétrolières. Selon cette politique, un consultant peut travailler en même temps pour une banque et une compagnie pétrolière, mais ne peut pas le faire pour deux banques ou deux compagnies pétrolières. Il existe de plus une classe de conflit spéciale qui ne contient qu'une seule compagnie et qui contient les informations "sanitisées", c'est-à-dire celles qui peuvent être lues par tout le monde sans provoquer de conflit d'intérêt. La politique est énoncée comme la combinaison de deux propriétés de sécurité. Ces deux propriétés sont volontairement répétées en anglais, afin de permettre au lecteur de mieux percevoir le niveau de formalisation de la description originale de cette politique. La première est la règle dite de sécurité-simple :

> [19] *Access is only granted if the object requested : (a) is in the same company dataset as an object already accessed by that subject, i.e. within the Wall, or (b) belongs to an entirely different conflict of interest class.*

La deuxième est celle de sécurité-$\star$ :

> [19] *Write access is only permitted if (a) access is permitted by the simple security rule, and (b) no object can be read which is in a different company dataset to the one for which write access is requested and contains unsanitized information.*

Quel est le statut exact d'un objet qualifié de "*object can be read*" ? Est-ce un objet que le sujet est en train de lire ? ou bien un objet qu'il lira dans l'avenir ? cela signifie-t-il qu'il a les droits pour le faire ? Dans ce dernier cas, cela signifie qu'un sujet voulant écrire

dans un objet doit révoquer tous les droits qu'il a sur des objets appartenant à une autre classe de conflit. D'autre part, dans l'article original [19], il n'est mentionné nulle part que les accès peuvent être relâchés. Cela signifie-t-il qu'il n'est pas possible de le faire, et qu'un accès est éternel ? Dans ce cas, le modèle de la Muraille de Chine est très restrictif, puisqu'une fois qu'un sujet a écrit dans un objet, il ne peut plus avoir d'accès en lecture ou en écriture à un objet d'une classe de conflit différente. Pour formaliser ces deux propriétés de sécurité, il faut faire des choix quant à l'interprétation de ces propriétés. Le danger provient de ces choix qui permettent plusieurs formalisations non équivalentes de cette politique. Enfin, l'étape de formalisation permet de distinguer clairement d'une part les propriétés de sécurité souhaitées et d'autre part le moniteur de référence chargé de les faire respecter. En effet, la présentation informelle de la politique de la Muraille de Chine peut s'apparenter à une description algorithmique de la fonction d'autorisation des accès. Cela peut amener à considérer immédiatement le "comment", donc l'implantation, au lieu de réfléchir d'abord au "quoi", c'est-à-dire à la spécification de la politique de sécurité.

### Formalisation du modèle de Bell & LaPadula dans le système Coq

Nous présentons ici brièvement une formalisation du modèle de Bell et LaPadula [80, 12] dans le système Coq à partir de laquelle un programme certifié implantant un moniteur de référence pour cette politique a été extrait. Ce travail est décrit dans [47]. Il a été effectué en collaboration avec T.Hardin et E.Gureghian et a servi de guide à l'élaboration d'un projet industriel dans la société employant E.Gureghian.

**Politique de Bell & LaPadula** La politique de Bell et LaPadula est habituellement décrite par une machine à états [80, 12]. Elle dépend d'un ensemble $\mathcal{S}$ de sujets, d'un ensemble $\mathcal{O}$ d'objets, d'un ensemble $\mathcal{A}$ de modes d'accès et d'un treillis fini $(\mathcal{L}, \preceq, \curlyvee, \curlywedge)$ de niveaux de sécurité. Un état du système est décrit par un quadruplet $(m, D, f_s, f_o)$ où $m$ est l'ensemble des accès courants effectués dans le système, $D$ est l'ensemble des droits d'accès et $f_s : \mathcal{S} \to \mathcal{L}$ (resp. $f_o : \mathcal{O} \to \mathcal{L}$) est une fonction associant un niveau de sécurité aux sujets (resp. aux objets). Les éléments de $m$ et de $D$ sont des accès représentés par des triplets de la forme $(s, o, a)$ : $(s, o, a) \in m$ signifie qu'un sujet $s$ accède à un objet $o$ selon le mode d'accès $a$ tandis que $(s, o, a) \in D$ signifie qu'un sujet $s$ dispose des droits (discrétionnaires) pour accéder à un objet $o$ selon le mode d'accès $a$. Les trois propriétés de sécurité définies dans la politique de Bell et LaPadula sont les suivantes.

- Propriété DAC (*Discretionnary Access Control*) : La propriété DAC exprime simplement que tout accès courant est conforme aux droits d'accès. Plus formellement, cette propriété s'écrit :
$$m \subseteq D$$
- Propriétés MAC et MAC$^\star$ (*Mandatory Access Control*) :
  - La propriété MAC connue sous le nom de "*no read-up property*" exprime qu'un sujet ne peut accéder en lecture à un objet que si son niveau de sécurité est supérieur à celui de l'objet accédé. Plus formellement, cette propriété s'écrit :
$$(s, o, \mathsf{read}) \in m \Rightarrow f_s(s) \succeq f_o(o) \tag{1.1}$$
  - La propriété MAC$^\star$ connue sous le nom de "*no write-down property*" permet d'éviter qu'un sujet "malicieux" recopie de l'information sensible à un niveau de sécurité

inférieur. Plus formellement, cette propriété s'écrit :

$$((s, o_1, \mathsf{read}) \in m \wedge (s, o_2, \mathsf{write}) \in m) \Rightarrow f_o(o_1) \preceq f_o(o_2) \tag{1.2}$$

**Certification d'un moniteur de référence** Nous esquissons ici les grandes lignes de la formalisation du modèle de Bell et LaPadula dans le système Coq. Une description détaillée de ce développement est présentée dans [47]. Ce développement est paramétré par $\mathcal{S}$, $\mathcal{O}$, $\mathcal{A}$ et un treillis $(\mathcal{L}, \preceq, \curlyvee, \curlywedge)$. Ce treillis est en fait obtenu à partir de deux paramètres : un ensemble $\mathcal{K}$ de "domaines" (connu sous le nom de "*needs-to-know*"), comme par exemple {nucléaire, médical, ... }, et un ensemble $\mathcal{C}$ de classifications, comme par exemple {Top-secret, secret, public, ... }, muni d'une relation d'ordre total. $\mathcal{L}$ est alors défini comme le treillis produit $\mathcal{C} \times T_k$ où $T_k = (\wp(\mathcal{K}), \subseteq, \cup, \cap)$ est le treillis des parties de $\mathcal{K}$.

A partir de la définition de l'ensemble $\Sigma$ des états, les fonctions de transition sont définies comme des fonctions de $\mathcal{R} \times \Sigma$ dans $\mathcal{D} \times \Sigma$ où $\mathcal{R}$ est un ensemble de requêtes et $\mathcal{D} = \{\mathsf{yes}, \mathsf{no}\}$ contient les réponses possibles[1]. Une fonction de transition $\tau$ correspond à la mise en application d'une politique de sécurité. Elle sera qualifiée de fonction "sûre" si et seulement si elle préserve les propriétés de sécurité, c'est-à-dire si et seulement si elle transforme chaque état vérifiant les propriétés DAC, MAC et MAC$^\star$ en états vérifiant encore ces propriétés.

Le développement réalisé consiste en une implantation dans Coq de la fonction de transition $\tau_{BLP}$ introduite par Bell et LaPadula ainsi que de la preuve mécanisée du célèbre "*Basic Security Theorem*" [12] affirmant que $\tau_{BLP}$ est "sûre". Le mécanisme d'extraction de Coq a permis, à partir de cette preuve, d'obtenir un programme certifié qui implante $\tau_{BLP}$.

Ce travail de mécanisation de la preuve a non seulement fourni une implantation avec un niveau de confiance élevé mais a aussi permis de reconnaître, parmi les hypothèses sur la fonction de transition, celles qui sont nécessaires au maintien de la politique de sécurité. Cela permet donc d'envisager des variantes de la fonction $\tau_{BLP}$ qui préservent les propriétés de sécurité.

Toutefois, si elle remplit ses objectifs en termes de garantie de correction, une telle approche peut conduire à des développements difficilement réutilisables. En effet, la moindre modification tant sur la spécification de la politique que sur son implantation conduit à modifier une preuve de plus d'un millier de lignes. Il s'agit là d'un exercice particulièrement chronophage. Pour faciliter la réutilisation, il m'est apparu nécessaire d'élaborer un cadre formel caractérisant les éléments communs aux politiques de contrôle d'accès et permettant d'en dériver progressivement la spécification complète de la politique choisie, puis différentes implantations. Il est également préférable d'utiliser un environnement de développement qui facilite l'implantation de ces différents traits. Même si le système Coq dispose à présent de mécanismes permettant de conduire des développements formels de manière modulaire, nous avons choisi d'utiliser l'atelier Focal, mieux adapté car offrant des traits objets[2]. La section qui suit présente une expérience selon cette approche.

[1] On peut facilement considérer un ensemble fini de réponses.

[2] De plus, l'existence de la bibliothèque de calcul formel [112] permet d'envisager aisément l'implantation dans Focal d'un cadre générique pour le contrôle de d'accès, un tel cadre faisant appel à de nombreuses structures algébriques classiques (treillis, algèbres de Boole, ...).

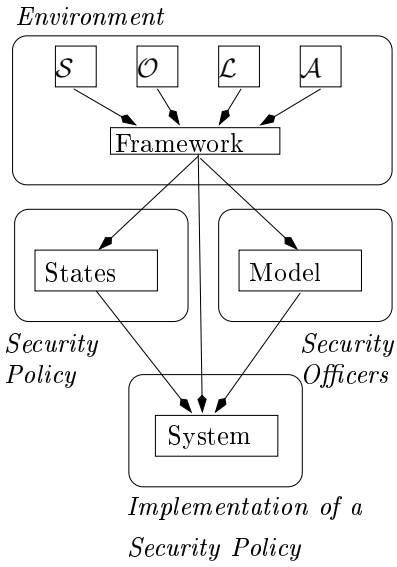


FIG. 1.1 – Algèbre des modèles de sécurité de McLean

**Implantation de l'algèbre des modèles de sécurité de McLean**

Afin de faciliter la réutilisation des développements formels de modèles de contrôle d'accès, nous avons choisi de définir, dans l'environnement de développement Focal [110, 38], l' "algèbre des modèles de sécurité" introduite par McLean [89]. Ce cadre générique pour le contrôle d'accès peut ensuite être instancié afin d'en dériver une implantation de la politique de Bell et LaPadula, qui peut à son tour être utilisée pour gérer les accès à une base de données relationnelle. Nous décrivons ici l'architecture de l'ensemble de ces dévelopements qui sont détaillés dans [93, 67, 68, 14, 15]. Ce travail a été réalisé en collaboration avec C.Morisset.

**Algèbre des modèles de sécurité** Pour factoriser le travail de formalisation et de développement, il nous fallait trouver un cadre abstrait suffisamment générique pour considérer chacune des politiques envisagées comme une instance de ce cadre. A notre connaissance, le seul cadre générique existant était l'"algèbre des modèles de sécurité", introduite par J. McLean en 1988 [89], qui permet la description d'une politique de contrôle d'accès à trois niveaux de spécification différents (figure 1.1) : les frameworks, les modèles et les systèmes.

Le framework décrit l'environnement : il est paramétré par $\mathcal{S}$, $\mathcal{O}$, $\mathcal{A}$ et un treillis $\mathcal{L}$ et spécifie quels sont les ensembles de sujets qui pourront conjointement demander à accéder à un objet ou demander à changer le niveau de sécurité d'un sujet ou d'un objet (sans toutefois spécifier comment sera traitée cette demande). J.Mc Lean introduit donc la notion d'accès conjoints : certaines opérations ne peuvent être autorisées que si elles sont demandées par un certain groupe de sujets[3]. Dans le cas le plus général, les ensembles de sujets qui pourront conjointement soumettre une requête sont des parties non vides quel-

[3] L'exemple le plus classique vient du monde militaire, où pour lancer un missile, il faut que des personnes habilitées appuient sur un bouton en même temps.

conques de $\mathcal{S}$ (toute opération est initiée par au moins un sujet) mais il peut exister des contraintes sur ces ensembles. Par exemple, certaines politiques imposent que si un groupe de sujets peut conjointement soumettre une requête, alors tout sous-ensemble de ce groupe peut également soumettre cette requête. Dans le cas le plus courant, sans accès conjoints, les seuls ensembles autorisés à soumettre une requête sont les singletons construits à partir de $\mathcal{S}$. Les différentes possibilités pour les ensembles de sujets permettent de construire une hiérarchie de frameworks permettant de factoriser les spécifications.

La notion de modèle a été introduite par J.McLean afin de pallier aux problèmes posés par des systèmes de contrôle d'accès ne fixant aucune règle régissant le changement de niveau de sécurité d'un sujet ou d'un objet. La notion de modèle est paramétrée par celle de framework. Un modèle spécifie quels sont les ensembles de sujets qui seront effectivement autorisés à modifier les niveaux de sécurité. Comme pour les frameworks, les différentes propriétés vérifiées par ces ensembles de sujets permettent de construire une hiérarchie de modèles. Certaines relations et opérations sur les modèles sont définies et permettent d'implanter l'ensemble des modèles comme une instance de treillis distributif.

Les états décrivent les informations relatives aux différents niveaux de sécurité associés aux sujets et aux objets ainsi que les accès courants. Une fois la notion d'état définie, on spécifie par un prédicat quels sont les états sûrs, c'est-à-dire quelle est la politique de sécurité appliquée. C'est à partir de cette notion d'état et de modèle qu'est définie la notion de système. Celle-ci décrit la fonction de transition et les propriétés qu'elle doit vérifier. Cette fonction de transition entre états décrit les changements d'états produits lors des requêtes d'accès émises par les sujets.

Ces trois notions ont été mécanisées dans l'atelier Focal puis utilisées pour traiter le cas particulier de la politique de Bell et LaPadula.

**Détection de flots d'information non autorisés** Ici encore, l'étape de formalisation et de mécanisation de la formalisation avec Focal a permis de corriger la spécification initiale. En effet, ce travail nous a conduit à identifier un problème dans la définition de l'algèbre des modèles de sécurité. Dans [89], McLean illustre l'utilisation du cadre qu'il introduit en spécifiant en son sein une version enrichie de la politique de Bell et LaPadula, essentiellement en considérant la notion d'accès conjoints. La propriété de sécurité $\mathsf{MAC}^\star$ définie en (1.2) est alors (re)définie comme suit :

> [89] *a state is* $\star$*-secure if for any subjects* $S_1, S_2$ *and objects* $o_1, o_2$*, if* $(S_1, o_1, \mathsf{read}) \in m$ *and* $(S_2, o_2, \mathsf{write}) \in m$ *and the classification of* $o_1$ *dominates that of* $o_2$*, then* $S_1 \cap S_2 = \emptyset$

Ici, $m$ est l'ensemble des accès courants, $S_1$ et $S_2$ sont des ensembles de sujets et un accès est un triplet $(S, o, a)$ exprimant que les sujets présents dans l'ensemble $S$ accèdent conjointement à l'objet $o$ selon le mode $a$. La formalisation de cet énoncé permet d'obtenir la spécification suivante (figure 1.2) :

$$((S_1, o_1, \mathsf{read}) \in m \land (S_2, o_2, \mathsf{write}) \in m \land f_o(o_2) \prec f_o(o_1)) \Rightarrow S_1 \cap S_2 = \emptyset$$

Par contraposition, cette spécification est équivalente à :

$$((S_1, o_1, \mathsf{read}) \in m \land (S_2, o_2, \mathsf{write}) \in m \land S_1 \cap S_2 \neq \emptyset) \Rightarrow \neg(f_o(o_2) \prec f_o(o_1)) \qquad (1.3)$$

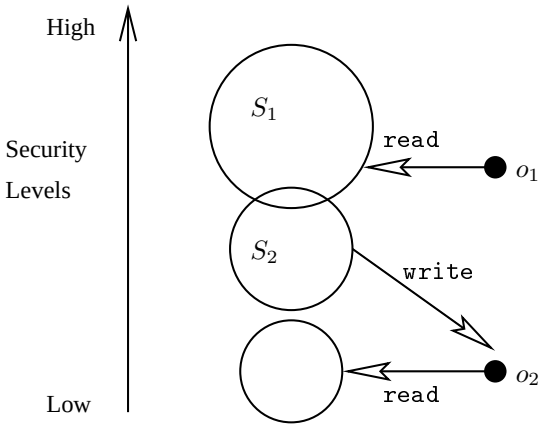


FIG. 1.2 – Violation de la propriété MAC*

Or, si on se limite au cas où $S_1$ et $S_2$ sont réduits à des singletons, c'est-à-dire si on ne considère pas les accès conjoints, la propriété (1.2) implique la propriété (1.3), mais la réciproque est en général fausse : (1.3) n'implique (1.2) que si l'ordre sur les niveaux de sécurité est total. En général cet ordre est partiel (puisqu'il définit une structure de treillis) et la propriété MAC* n'est alors pas vérifiée (bien que (1.3) soit satisfaite) comme le montre l'exemple suivant (figure 1.3). En respectant la propriété (1.3), un sujet $s_1$ peut accéder simultanément en lecture à un objet de niveau de sécurité $l_1$ et en écriture à un objet de niveau de sécurité $l_2$ tels que $l_1$ et $l_2$ ne soient pas comparables. Un sujet $s_2$ peut alors lire ce dernier objet de niveau $l_2$ et écrire simultanément dans un objet de niveau $l_3$ tel que $l_3$ et $l_2$ ne soient pas comparables mais tel que $l_3$ soit inférieur à $l_1$. Il y a alors une "fuite d'information vers le bas" puisqu'il devient ainsi possible de recopier les informations d'un objet de niveau $l_1$ dans un objet de niveau $l_3 \preceq l_1$.

L'activité de formalisation permet de mettre en lumière de tels problèmes. Ils peuvent être découverts en essayant de prouver des "énoncés faux" (par exemple en essayant de prouver l'équivalence entre (1.2) et (1.3) lorsque $S_1$ et $S_2$ sont des singletons), mais ils peuvent parfois aussi être découverts en utilisant des méthodes permettant de tester une propriété, par exemple une propriété assurant qu'il n'existe pas de séquence d'accès permettant de copier de l'information de niveau élevé dans des objets de niveaux inférieurs. Pour ce faire, il est possible d'utiliser, dans certaines conditions, un outil de test intégré à l'atelier Focal développé par M. Carlier et C. Dubois [20]. Citons aussi les travaux de C. Morisset et A. Santana de Oliveira [50, 94, 93, 28] qui décrivent un algorithme permettant de détecter la "fuite d'information" correspondant à l'exemple présenté dans cette section à partir d'une spécification de politique par un système de réécriture. La description de politiques de sécurité *via* un système de réécriture a été étudiée par A. Santana de Oliveira dans [28].

**Instanciation du modèle et application** En définissant, c'est-à-dire en instanciant, chacune des entités spécifiées dans le cadre générique introduit par J.McLean et implanté avec l'atelier Focal, nous avons défini la politique de Bell et LaPadula et avons fourni

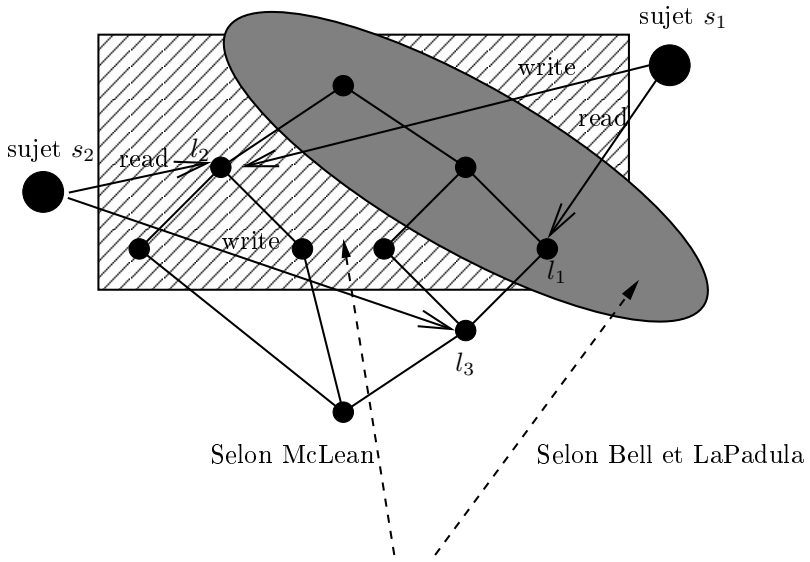


FIG. 1.3 – Violation de la politique de sécurité

une implantation de la fonction de transition $\tau_{BLP}$ évoquée dans la section 1.1.2. Notons que, cette implantation reste encore relativement générique. Par exemple, elle ne spécifie ni les sujets, ni les objets. En effet, un programme qui implante une certaine politique de contrôle d'accès est *a priori* indépendant du système sur lequel il doit s'appliquer (bases de données, systèmes de gestion de fichiers, ...). Il possède des paramètres destinés à prendre en compte l'environnement concret dans lequel il va s'exécuter. Nous allons maintenant illustrer succintement l'intégration de tels programmes dans un système "concret", une base de donnée relationnelle. Il s'agit ici d'une mise à l'épreuve du terrain de la méthode toute entière. Ce travail a été effectué en collaboration avec C.Morisset (doctorant de 2004 à 2007) et J.Blond (doctorant).

La base de données (MySQL) contient à la fois les données des utilisateurs, et les données relatives à la sécurité (droits d'accès, accès courants, niveaux de classification, "*needs-to-know*"). Les paramètres du système de Bell et LaPadula obtenu dans le paragraphe précédent sont instanciés par des objets stockés dans la partie "sécurité" de la base de données et les états de ce système sont construits à partir de ces données. Deux tables particulières concernant les sujets et les objets sont présentes dans la zone dédiée aux données relatives à la sécurité de la base de données. La table sujets contient pour chaque sujet son identifiant unique, son login, son mot de passe, ainsi que son niveau de sécurité. De même, la table objets contient pour chaque objet son identifiant unique, le nom de la table à laquelle il correspond ainsi que son niveau de sécurité. Ainsi, les objets du sytème d'information sont les tables de la base de données. Une granularité plus fine pour la notion d'objet pourrait être envisagée mais soulèverait des problèmes connus dans le domaine des bases de données pour lesquels des solutions existent mais sortent du cadre de notre

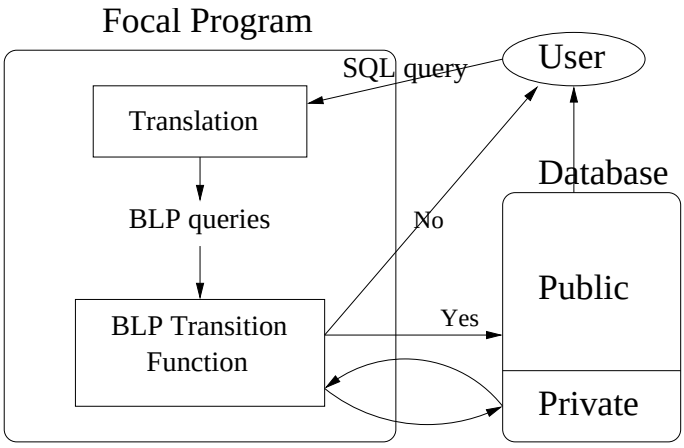


FIG. 1.4 – Contrôle d'accès pour un SGBD

prototype[4].

L'architecture de l'implantation se décompose en trois parties : la politique de sécurité, le gestionnaire de base de données et le moniteur de référence. La partie concernant la politique de sécurité est entièrement définie en Focal et repose sur le modèle de Bell et LaPadula. L'accès au gestionnaire de base de données s'effectue grâce à une interface définie en Focal proposant des fonctions de connexion/déconnexion à la base de données, ainsi que des fonctions permettant l'exécution d'une requête et la récupération du résultat. Ces fonctions reposent sur la librairie *ocaml-mysql* [1], et sont donc spécifiques au gestionnaire de base de données MySQL. La figure 1.4 illustre le traitement des requêtes effectué par l'application obtenue. Etant donnée une requête SQL soumise par un sujet (authentifié), le programme d'analyse de requêtes SQL que nous avons développé fait appel au système de Bell et LaPadula et, selon la réponse obtenue, traite ou refuse l'exécution de la requête soumise. Pour cela, nous avons défini pour chaque requête SQL, un ensemble de requêtes pour le système de Bell et LaPadula, donnant en quelque sorte une sémantique en termes d'accès aux requêtes SQL. On remarquera que les requêtes SQL soumises par l'utilisateur utilisent la syntaxe standard de SQL, rendant ainsi transparente pour l'utilisateur la mise en œuvre de notre application, qui peut être vue comme un filtre entre l'utilisateur et le système de gestion de la base de données. Ce développement est décrit en détail dans [93, 14, 15].

**Nécessité de définir un cadre plus générique** S'il permet de caractériser certains aspects communs à différentes politiques de contrôle d'accès, le cadre introduit par McLean n'est pas encore suffisamment riche. Il est certes est bien adapté à la définition de politiques basées sur un ensemble de niveaux de sécurité associés à certaines entités du système mais ne permet pas de définir des politiques comme la Muraille de Chine (dans le contexte de la Muraille de Chine nous ne disposons pas d'un treillis de niveaux de sécurité mais d'un

[4]Par exemple, considérer les tuples d'une base de données comme des objets conduit à un phénomène de polyinstanciation [116] lorsque deux tuples ont la même clé primaire, mais des informations de niveaux de sécurité différents ([13] propose des solutions à ce problème).

ensemble de classes de conflits) ou encore des politiques à base de rôles (dans le contexte d'une politique à base de rôles, nous ne disposons pas d'un treillis de niveaux de sécurité mais d'un ensemble de rôles muni d'un ordre partiel). Le cadre de McLean se révèle donc à la fois trop contraignant et trop peu expressif (en termes de propriétés de comparaison de modèles et de simulation d'implantations permettant une plus grande réutilisabilité). Nous avons alors été amenés à concevoir (en 2005) un cadre sémantique pour le contrôle d'accès permettant de répondre à nos objectifs. Il ne s'agit pas de la définition d'un langage mais plutôt d'une spécification de style mathématique de ce qui constitue une politique de contrôle d'accès. Ce cadre et son utilisation sont présentés dans le chapitre 2. Il a été implanté avec l'atelier Focal et instancié pour obtenir une implantation des principales politiques que l'on peut rencontrer dans la littérature : [18] décrit une implantation de politique discrétionnaire à base de matrice de droits d'accès [79, 51] développée en 2007, [6] décrit les implantations obtenues en 2008 d'une politique *à la* Unix intégrant la notion de groupes d'utilisateurs, d'une politique à base de "tickets" [83], et d'une politique discrétionnaire intégrant un mécanisme de délégation, [93] décrit une implantation de la politique de Bell et LaPadula [80, 12] développée en 2006 et [48] décrit une implantation obtenue en 2007 d'une politique à base de rôles (RBAC96) [42, 117]. L'ensemble de ces développements a été effectué lors de stages que j'ai encadrés.

## 1.2 Réutilisation de développements formels

Dans cette section nous commentons les travaux portant sur la réutilisation de développements formels. Nous avons expérimenté deux approches distinctes. La première relève de l'ingénierie de la preuve et consiste essentiellement à décrire comment exploiter une preuve existante pour obtenir une preuve. Elle est présentée dans la section 1.2.1. La deuxième approche repose sur le constat qu'un développement formel est plus facilement réutilisable s'il a été conduit de manière modulaire et générique. Elle est présentée dans la section 1.2.2.

### 1.2.1 Ingénierie de la preuve

Tandis que les spécifications formelles des définitions des objets et des propriétés sont généralement assez facilement réutilisables, la réutilisation des preuves est par contre plus délicate. Nous allons voir au travers de trois exemples les problèmes posés par la réutilisation de preuves formelles et les solutions possibles. La réutilisation est ici abordée dans divers contextes, la preuve à réutiliser pouvant avoir été obtenue au sein d'un outil de formalisation différent de celui utilisé pour la preuve souhaitée. La manipulation (parcours, recherche, modification, transformation ...) d'une preuve est difficile et peu outillée (citons toutefois [40, 105, 16]) si l'on reste au niveau du langage de tactiques. L'accès à la structure d'une preuve, au niveau de l'utilisateur, est rarement aisé et nécessite une connaissance des représentations internes des preuves, donc du code source de l'outil de développement formel utilisé. On peut envisager de travailler directement sur le terme de preuve mais, au moins en théorie de types, la complexité et la taille d'une preuve ne permettent pas d'envisager l'écriture de la preuve elle-même qui ne figure donc pas dans le code source du développement (les preuves y sont exprimées *via* une suite de tactiques).

### Définition d'isomorphismes

Souvent, il existe plusieurs manières de représenter un même objet. Le choix de la représentation est en général dicté par le traitement que l'on souhaite effectuer sur ces objets. Lorsque le développement formel existant porte sur un problème isomorphe au problème à traiter, c'est à dire lorsque le problème à traiter porte sur des objets obtenus par changement de représentation, la réutilisation consiste à définir des fonctions permettant de passer d'une représentation à une autre et à montrer la préservation des propriétés considérées. Les techniques de changement de représentation ont déjà été étudiées dans [92, 11, 86]. J'ai également considéré ce problème en reprenant les travaux de J.Rouyer [115] sur les quasi-termes, structure de données plus générale que celle des termes. J'ai construit une traduction des quasi-termes vers les termes (fonction partielle) et j'ai ainsi transposé les propriétés de l'unification des quasi-termes à l'unification des termes. L'annexe B.1 détaille le traitement de l'unification au premier ordre et le passage des quasi-termes aux termes fait l'objet de l'annexe C.1.1 et de [60].

Nous avons donc utilisé le mécanisme de réutilisation d'une preuve formelle suivant : considérer la preuve existante comme une boîte noire et construire des "passerelles" – des ponts formels – entre le domaine sur lequel porte la preuve existante et le domaine sur lequel porte la preuve cherchée. Il reste ensuite à prouver que les propriétés souhaitées sont bien conservées par les transformations définies.

### Outil externe de transformation de preuve

Nous présentons maintenant une approche différente pour laquelle la preuve à réutiliser n'est pas considérée comme une boîte noire mais comme un objet sur lequel des transformations vont être effectuées.

Le problème est de réutiliser une preuve établie dans le système Coq pour obtenir une preuve dans l'atelier Focal. Bien sûr, la proximité entre Coq et Focal facilite cette réutilisation. L'environnement Focal, présenté en annexe A, propose un langage d'expression de propriétés qui peuvent être prouvées soit directement à l'aide de Coq, soit à l'aide de l'outil Zenon. Dans les deux cas, un terme de preuve Coq s'appuyant sur la structure particulière des objets spécifiés dans l'environnement Focal est construit par le compilateur. Nous avons construit un outil permettant de transformer une preuve établie avec Coq en une preuve réutilisable dans l'environnement Focal. Cet outil permet d'une part de généraliser une preuve en la paramétrant par une relation d'équivalence et un ensemble de propriétés que doit vérifier cette relation (il s'agit essentiellement de propriétés de congruence) et d'autre part de la réutiliser dans Focal. Cette approche a été appliquée pour réutiliser dans l'environnement Focal des preuves établies sur les groupes avec le système Coq. Elle est présentée plus complètement ainsi que l'exemple dans l'annexe C.1.2 et dans [37]. Ce travail a été réalisé en collaboration avec C.Dubois et J.Grandguillot.

### Preuves dans une architecture hiérarchique

Comme nous l'avons déjà mentionné, la problématique de la réutilisation de preuves formelles est proche de celle de la réutilisation de programmes. Toutefois, une difficulté méthodologique supplémentaire apparaît lorsque les preuves sont développées dans une

architecture hiérarchique, comme c'est le cas avec l'atelier Focal dont le langage sous-jacent est muni de traits objets. Les preuves peuvent alors être faites à différents niveaux dans la hiérarchie, ce qui introduit de la souplesse dans les développements. Mais se pose alors la question de la place des preuves dans l'architecture du développement. Doit-on faire les preuves "le plus tôt" possible (c'est à dire dès que l'on dispose des spécifications et des définitions nécessaires), comme c'est l'usage en mathématiques ? Doit-on au contraire faire les preuves "le plus tard" possible (c'est à dire aux feuilles du graphe d'héritage) ? Dans le premier cas, le mécanisme de redéfinition risque d'invalider les preuves des propriétés qui reposent sur la définition de l'objet redéfini, tandis que dans le deuxième cas, les preuves risquent d'être dupliquées. Il n'y a donc pas de réponses définitives à ces questions. Nous avons dégagé des critères permettant de répondre à ces questions et esquissé une méthodologie de développement de preuve dans un contexte hiérarchique. Pour chaque opération, il s'agit principalement de procéder en trois temps : déclarer, spécifier, puis implanter. Ce travail est décrit dans [109]. Il est de plus rappelé dans l'annexe C.1.3. Il a été réalisé en collaboration avec V. Prevosto.

### 1.2.2 Définition de cadres génériques

La situation idéale pour la réutilisation de développements formels correspond au cas où le développement à réutiliser a été obtenu par instanciation d'un cadre générique. Dans ce cas, il suffit d'instancier ce cadre de manière différente afin d'obtenir le développement souhaité. Ainsi, on devrait s'astreindre, au cours de tout travail de formalisation, à la recherche systématique des abstractions possibles afin de déterminer si un cadre générique utile peut être dégagé (dans le même esprit que le typage de fonctions permet parfois de reconnaître le caractère polymorphe de certaines fonctions). Il s'agit ici de décrire comment la définition d'un cadre abstrait générique permet d'obtenir plusieurs développements, dédiés à des cahiers des charges différents, qui peuvent être obtenus par instanciation du cadre générique. Bien sûr, la réutilisation d'un ensemble de spécifications formelles génériques nécessite de respecter le cadre défini par ces spécifications. Ce cadre peut parfois paraître trop riche ou trop général (et, dans ce cas on préfere le qualifier de "trop compliqué") pour l'objet à formaliser. Toutefois, conduire un développement au sein d'un cadre générique bien conçu permet d'une part de factoriser le travail de développement et de le rendre réutilisable, et d'autre part, de rendre possible la comparaison des notions formaliséss dans ce cadre.

#### Spécification de formalismes logiques

L'informatique consomme aujourd'hui beaucoup de formalismes logiques différents. Tous ces formalismes partagent en fait des notions exprimables de manière générique. Le cadre unifié que nous avons adopté et implanté dans le système Coq est une variante de la notion de logiques générales introduite par J. Meseguer [91, 44]. Chacune des logiques est vue comme une instance de logique générale, ses propriétés s'exprimant alors dans ce cadre. L'implantation réalisée avec Coq décrit les notions de syntaxe (formules, morphismes), de sémantique (institutions) et de preuve (déduction). La notion de système formel y est aussi décrite afin de faciliter la définition des aspects "preuve" d'une logique. Le développement obtenu a été utilisé pour implanter la logique des propositions et la logique équationnelle.

Ce travail est décrit dans l'annexe C.2 et dans [66]. Il a été réalisé en collaboration avec C.Dubois.

**Spécification de modèles de contrôle d'accès**

Dans la section 1.1.2, nous avons présenté la formalisation d'un cadre générique permettant la définition de modèles de contrôle d'accès. Ce cadre, introduit par J.McLean [89], permet de considérer les politiques basées sur un ensemble de niveaux de sécurité partiellement ordonné. Il est donc particulièrement bien adapté pour décrire des (variantes de) modèles *à la* Bell et LaPadula. Toutefois, il s'est révélé trop contraignant puisqu'il ne permet pas de prendre en compte des politiques exprimées à partir de paramètres de sécurité quelconques (comme c'est le cas avec les modèles à base de rôles et le modèle de la Muraille de Chine). De plus, comme nous l'avons dit, conduire une formalisation au sein d'un cadre générique permet de comparer les objets formalisés. Dans le contexte du contrôle d'accès, cette possibilité peut fournir des critères de choix entre plusieurs politiques de sécurité pour une application donnée. Enfin, comparer des modèles peut conduire à des schémas de traduction d'un modèle vers un autre et fonder ainsi des techniques de réutilisation.

Après l'étude de l'algèbre des modèles de sécurité de J.McLean, nous avons entrepris, en 2006, la définition d'un cadre générique permettant la spécification, l'implantation et la comparaison de modèles de contrôle d'accès. Ce cadre fournit une caractérisation des éléments communs aux politiques de contrôle d'accès et permet d'en dériver différentes implantations. Il fournit plusieurs niveaux de spécification. Nous donnons ici rapidement les grandes articulations de ce cadre, qui sera présenté en détail dans le chapitre 2. Tout d'abord, ce cadre permet de décrire comment est conçu le système d'information considéré, c'est à dire quels sont les paramètres de sécurité mis en jeu et quelles sont les informations qui décrivent l'état du système. La politique de contrôle d'accès peut alors être spécifiée en caractérisant le sous-ensemble des états du système qui sont considérés comme sûrs, c'est à dire qui respectent la politique de sécurité. Ensuite, les requêtes permettant de modifier l'état du système peuvent être spécifiées non seulement par la syntaxe mais aussi par la sémantique du langage de requêtes (quelles sont les modifications attendues sur les états lors de l'exécution d'une requête). Ces deux notions (politique et langage de requêtes) permettent d'introduire la notion de modèle de contrôle d'accès. Ensuite, le cadre défini sert à décrire les implantations possibles d'un modèle par la donnée d'un ensemble d'états initiaux et d'une fonction de transition entre états. Bien sûr, la fonction de transition considérée doit satisfaire des propriétés de correction vis à vis de la politique considérée et vis à vis de la sémantique des requêtes. Puisqu'il existe plusieurs implantations possibles, plus ou moins restrictives, d'un même modèle, une relation de préordre sur les implantations peut être introduite. Enfin, le cadre sémantique proposé fournit également une relation de comparaison entre modèles de contrôle d'accès qui repose sur la notion de simulation d'implantations. La conception de ce cadre ne correspond pas à la définition d'un langage mais plutôt à une spécification abstraite de ce qui constitue une politique de contrôle d'accès. Ce cadre, décrit en détail dans le chapitre 2, a été utilisé avec succès pour spécifier, implanter et comparer les principaux modèles existant dans la littérature.

# Chapitre 2

# Contrôler le contrôle d'accès

La confiance dans un produit passe généralement par un certain nombre d'avis de personnes reconnnues comme experts. Ces experts se basent sur des normes pour certifier que le produit en question est conforme aux exigences issues de la demande de confiance. Dans le domaine du logiciel, les Critères Communs [22] (recueil de normes définies par des agences gouvernementales) fournissent une méthodologie permettant d'atteindre des hauts niveaux de sûreté. Ils définissent à la fois un cadre de travail pour la conception et la réalisation de logiciels et une référence pour les utilisateurs de ces logiciels. Les hauts niveaux de sûreté des Critères Communs (EAL – *Evaluation Assurance Level* – 5 à 7) requièrent l'utilisation de méthodes formelles dans les étapes de spécification et de conception du logiciel implantant les fonctions de sécurité afin de justifier la confiance accordée à ces fonctions.

Dans ce chapitre, nous utilisons une démarche formelle pour étudier certaines des propriétés classiques de sécurité des systèmes d'information. Selon les Critères Communs, un système est vu comme une installation donnée de technologies de l'information, avec un objectif et un environnement opérationnel particuliers et une politique de sécurité est un ensemble de règles qui précisent comment gérer, protéger ou distribuer les informations ou ressources du système. Nous nous intéressons ici plus particulièrement aux politiques de contrôle d'accès dont l'objectif est de régir et gérer les accès effectués selon certains modes (lecture, écriture, etc.) par des sujets, les entités actives (processus, programmes, utilisateurs, etc.) sur des objets, les entités passives (données, fichiers, programmes, etc). Mettre en place un mécanisme de contrôle d'accès consiste dans un premier temps à définir la politique de contrôle d'accès à proprement parler, c'est-à-dire spécifier les accès autorisés et ceux interdits. Dans un deuxième temps, il faut définir un moniteur de référence, c'est-à-dire un programme chargé de mettre en œuvre la politique de contrôle d'accès au sein du système. Toujours selon les Critères Communs, un moniteur de référence doit posséder les trois caractéristiques suivantes :

- des sujets non sûrs ne peuvent pas interférer avec son fonctionnement, i.e. il est à l'épreuve d'une intrusion physique,
- des sujets non sûrs ne peuvent pas court-circuiter les contrôles qu'il effectue, i.e. il est systématiquement appelé,
- il est suffisamment simple pour être analysé et pour comprendre son comportement, i.e. sa conception est simple.

Ces trois caractéristiques, introduites dans [5], sont connues sous l'acronyme *NEAT*, pour "*Non-bypassable*" (il n'est pas possible d'éviter les fonctions de sécurité), "*Evaluable*" (les fonctions de sécurité sont suffisamment simples pour être mathématiquement vérifiées et évaluées), "*Always Invoked*" (les fonctions de sécurité sont systématiquement appelées) et "*Tamperproof*" (les fonctions de sécurité ne peuvent pas être altérées). Cet acronyme est défini dans le cadre de *MILS*[1] (*Multiple Independent Levels of Security*), une approche de développement de systèmes sécurisés. L'utilisation de méthodes formelles pour la conception d'un moniteur de référence facilite son évaluation et sa vérification, puisque la correction du programme vis-à-vis de sa spécification peut être énoncée et prouvée de manière formelle.

De nombreux modèles de contrôle d'accès existent dans la littérature, comme le modèle de Bell de LaPadula [12], le modèle HRU [51], le modèle de la Muraille de Chine [19], celui à base de rôles [42], sans oublier un ensemble d'extensions de ce dernier, comme le modèle à base d'organisations [72], à base de coalition [24], à base d'équipes [121], etc. Chacun de ces modèles a été conçu pour répondre à un besoin de sécurité précis dans un contexte précis. Néanmoins, certains de ces modèles ne sont pas exprimés de manière formelle, ce qui peut conduire à une mauvaise compréhension, une mauvaise utilisation, voire à une mauvaise implantation de ces modèles. De plus, ces différents modèles ne sont pas tous exprimés dans un même formalisme, et il est ainsi difficile de comparer deux modèles entre eux. Dans ce chapitre, nous présentons donc un cadre formel uniforme permettant la spécification, l'implantation, la comparaison et l'analyse de flots d'information de ces différents modèles de contrôle d'accès.

La suite de ce chapitre est organisée comme suit :

- La section 2.1 présente le cadre sémantique permettant de spécifier et d'implanter les modèles de contrôle d'accès. Ce travail a été réalisé en collaboration avec C. Morisset.
- Dans la section 2.2, nous introduisons des critères à partir desquels nous pouvons définir des relations de comparaison entre implantations d'un modèle mais aussi entre modèles. Ce travail a été réalisé en collaboration avec C. Morisset et L. Habib.
- Dans la section 2.3, nous montrons comment il est possible d'analyser les flots d'information engendrés par les exécutions d'un moniteur de référence. Ce travail a été réalisé en collaboration avec V. Viet Triem Tong et L. Mé.

Ce travail étend les résultats décrits dans [69] et rend compte, en partie, des travaux présentés dans [70, 93, 48, 49, 71]. Les résultats présentés dans ce chapitre sont détaillés et illustrés dans la section D figurant en annexe.

## 2.1 Un cadre sémantique pour la définition de modèles de contrôle d'accès

Nous introduisons ici un cadre permettant non seulement de spécifier et d'implanter des modèles de contrôle d'accès mais aussi de les comparer et d'analyser les flots d'information qu'ils autorisent. Il s'agit d'identifier les "ingrédients" communs aux modèles de contrôle d'accès, d'exprimer les propriétés génériques qu'ils vérifient et de comprendre le rôle de ces

[1] `http ://www.ois.com/MILS/`

"ingrédients" dans une implantation. Ce cadre fournit une spécification abstraite de ce qui constitue un modèle. Il s'agit donc d'un *cadre sémantique pour le contrôle d'accès.*

### 2.1.1 Politiques de contrôle d'accès

Une politique de contrôle d'accès permet de caractériser les états d'un système et de spécifier ce qu'est un état sûr en fonction d'informations de sécurité associées aux entités du système.

**Entités** Les entités du système peuvent être réparties dans deux ensembles : l'ensemble $\mathcal{S}$ des sujets, également appelés *entités actives*, qui correspondent aux entités qui effectuent les actions dans le système, et l'ensemble $\mathcal{O}$ des objets, également appelés *entités passives*, qui subissent les actions. Les sujets et les objets sont généralement considérés comme des entités atomiques. Ces deux ensembles ne sont pas nécessairement disjoints : par exemple, un processus peut à la fois être un sujet et ainsi effectuer des opérations, et un objet, dans le cas où un autre processus tente de l'arrêter.

**Accès** Nous désignons par $\mathcal{A}$ l'ensemble des modes d'accès qui caractérisent les différents types d'accès effectués par les sujets sur les objets. Cet ensemble contient généralement `read`, `write`, `append`, etc. Une approche classique consiste à représenter un accès par un triplet $(s, o, a)$, signifiant que le sujet $s$ accède à l'objet $o$ selon le mode d'accès $a$. Néanmoins, d'autres approches existent (on peut par exemple regrouper les modes d'accès, ou encore considérer les accès conjoints de sujets sur des objets). Afin de pouvoir prendre en compte ces différentes situations, nous nous limitons à dénoter par $\mathbb{A}$ l'ensemble de tous les accès, sans introduire une véritable définition qui serait trop précise à ce stade.

**Politiques de contrôle d'accès** Les systèmes de contrôle d'accès sont ici modélisés sous la forme de machines à états. Un état représente le système à un instant donné et contient au moins une description de l'ensemble des *accès courants*, c'est-à-dire de tous les accès qui ont été acceptés et qui n'ont pas encore été relâchés. Ces accès sont donc supposés être effectués simultanément dans le système. L'ensemble des états est noté $\Sigma$ et l'ensemble des accès courants d'un état $\sigma$ est noté $\Lambda(\sigma)$ ($\Lambda$ est donc une fonction de $\Sigma$ dans $\wp(\mathbb{A})$). Une politique de contrôle d'accès permet de spécifier un sous-ensemble de $\Sigma$, contenant les états sûrs, c'est-à-dire les états qui vérifient la politique. Afin de déterminer si un état est sûr, les entités du système sont associées à des informations de sécurité :

- le paramètre de sécurité, noté $\rho$, décrit les informations de sécurité *statiques* de la politique, c'est-à-dire les informations de sécurité qui ne changeront pas durant la vie du système,
- les fonctions de sécurité décrivent les informations de sécurité *dynamiques* de la politique, c'est-à-dire les informations de sécurité susceptibles d'évoluer durant la vie du système (étant donné un état $\sigma$, on note $\Upsilon(\sigma)$ les fonctions de sécurité associées à l'état $\sigma$).

A partir des informations dynamiques décrites par un état (ensemble des accès courants et fonctions de sécurité) et des informations statiques du système (paramètre de sécurité), les états sûrs sont caractérisés par un prédicat $\Omega$ sur les états. On note $\Sigma_{|\Omega}$ l'ensemble

$\{\sigma \in \Sigma \mid \Omega(\sigma)\}$ des états sûrs. Toutes ces notions nous permettent de définir une politique de contrôle d'accès comme suit.

**Définition 2.1** *Une politique de contrôle d'accès $\mathbb{P}[\rho]$, basée sur un paramètre de sécurité $\rho$, est définie par un quintuplet $\mathbb{P}[\rho] = (\mathcal{S}, \mathcal{O}, \mathcal{A}, \Sigma, \Omega)$.*

A ce niveau de spécification, il est possible de caractériser certaines propriétés que peuvent vérifier les politiques de contrôle d'accès. Dans la suite nous utiliserons les deux propriétés suivantes :

- Une politique est dite *compacte* si lorsque l'on supprime des accès de l'ensemble des accès courants d'un état sûr, l'état reste sûr :

$$\forall \sigma_1 \in \Sigma \quad \Omega(\sigma_1) \Rightarrow (\forall \sigma_2 \in \Sigma \ (\Lambda(\sigma_2) \subseteq \Lambda(\sigma_1) \wedge \Upsilon(\sigma_1) = \Upsilon(\sigma_2)) \Rightarrow \Omega(\sigma_2))$$

  La plupart des politiques classiques (HRU, Bell et LaPadula, Muraille de Chine, RBAC, ...) sont compactes. Toutefois, il peut être utile de définir des politiques non compactes dans certains contextes. Considérons par exemple une politique de contrôle des accès aux ressources d'un système qui stipule qu'un utilisateur est autorisé à utiliser des ressources d'une équipe à laquelle il n'appartient pas, mais seulement dans le cas où toutes les ressources de son équipe sont déjà accédées. Une telle politique n'est pas compacte puisque lorsqu'un utilisateur accède à une ressource d'une équipe à laquelle il n'appartient pas, retirer un accès sur une ressource de son équipe conduit à un état non sûr.
- Une politique est dite *libre* si lorsqu'un accès est autorisé pour un état sûr du système, il est autorisé pour tous les états sûrs[2] :

$$\begin{array}{l} \forall \sigma \in \Sigma \ \forall s \in \mathcal{S} \ \forall o \in \mathcal{O} \ \forall a \in \mathcal{A} \\ ((\exists \sigma' \in \Sigma \ (\Omega(\sigma') \wedge (s,o,a) \in \Lambda(\sigma'))) \wedge \Omega(\sigma)) \Rightarrow \Omega(\sigma \oplus (s,o,a)) \end{array}$$

  Par exemple, HRU définit une politique libre puisque le prédicat caractérisant les états sûrs est défini en examinant chacun des accès courants indépendamment des autres accès en cours. En revanche, la politique de Bell et LaPadula n'est pas libre puisqu'il est possible qu'un sujet soit autorisé à écrire dans un objet lorsque ses accès en lecture vérifient certaines conditions et ne soit plus autorisé à écrire dans cet objet lorsque ses accès en lecture ne vérifient plus ces conditions.

Enfin, étant donné un état $\sigma$, nous introduisons l'ensemble $\mathcal{W}(\sigma)$ des ensembles d'accès qui peuvent être ajoutés à l'ensemble des accès courants de l'état $\sigma$, sans changer les fonctions de sécurité, de manière à ce que l'état ainsi obtenu reste sûr :

$$\mathcal{W}(\sigma) = \left\{ \begin{array}{l} A \in \wp(\mathbb{A}) \mid \forall \sigma' \in \Sigma \\ (\Upsilon(\sigma') = \Upsilon(\sigma) \wedge \Lambda(\sigma') = \Lambda(\sigma) \cup A) \Rightarrow \Omega(\sigma') \end{array} \right\}$$

Les ensembles d'accès présents dans $\mathcal{W}(\sigma)$ sont appelés les accès potentiels de $\sigma$. Cette définition introduit une "sémantique" sur les états et nous permettra par la suite de définir

[2] $\sigma \oplus (s,o,a)$ (resp. $\sigma \ominus (s,o,a)$) dénote un état tel que :

$$\begin{array}{l} \Lambda(\sigma \oplus (s,o,a)) = \Lambda(\sigma) \cup \{(s,o,a)\} \text{ et } \Upsilon(\sigma \oplus (s,o,a)) = \Upsilon(\sigma) \\ (\text{resp.}\Lambda(\sigma \oplus (s,o,a)) = \Lambda(\sigma) \backslash \{(s,o,a)\} \text{ et } \Upsilon(\sigma \oplus (s,o,a)) = \Upsilon(\sigma)) \end{array}$$

une relation d'équivalence entre les états de systèmes basés sur des politiques différentes. Nous introduisons également la fonction $\mathcal{W}_\emptyset : \Sigma \rightarrow \wp(\wp(\mathbb{A}))$, qui étant donné un état, retourne tous les ensembles d'accès "compatibles" avec les fonctions de sécurité de cet état. Etant donné un état $\sigma$, $\mathcal{W}_\emptyset(\sigma) = \mathcal{W}(\sigma')$ avec $\Lambda(\sigma') = \emptyset$ et $\Upsilon(\sigma') = \Upsilon(\sigma)$. Ces définitions permettent d'établir plusieurs propriétés "techniques" utiles lors de la comparaison de politiques de contrôle d'accès. On peut montrer par exemple que si un état $\sigma$ est tel que $\mathcal{W}(\sigma) = \emptyset$, alors il n'est pas sûr (i.e. $\Omega(\sigma)$ est faux). La réciproque n'est cependant pas vraie dans le cas général mais l'est pour les politiques compactes.

Plusieurs exemples de politiques classiques de contrôle d'accès sont formalisés en annexe D.1.1.

### 2.1.2 Modèles de contrôle d'accès

Nous introduisons à présent la notion de modèle de contrôle d'accès, qui permet de spécifier comment passer d'un état du système à un autre. Pour cela, nous introduisons tout d'abord la notion de langage de requêtes.

#### Requêtes

Une requête est soumise par un sujet afin de faire évoluer le système, soit en ajoutant ou en enlevant un accès, soit en changeant les informations de sécurité dynamiques du système. Dans ce dernier cas, on parle de *requêtes administratives*. La plupart des modèles de contrôle d'accès considèrent au moins les droits d'accès read (lecture) et write (écriture). On peut généralement considérer les requêtes suivantes :

$$\mathcal{R}^{acc} = \{\langle +, s, o, a\rangle, \langle -, s, o, a\rangle\} \tag{2.1}$$

où $\langle +, s, o, a\rangle$ (resp. $\langle -, s, o, a\rangle$) correspond à la demande du sujet $s$ d'accéder (resp. de relâcher l'accès) à l'objet $o$ selon le mode d'accès $a$. La notion de relâchement d'accès n'est pas nécessairement présente dans tous les modèles de contrôle d'accès. En effet, elle sous-entend la notion (implicite) de persistance des accès, c'est-à-dire qu'une fois qu'ils sont effectués, ces accès restent en mémoire.

Etant donné un ensemble de requêtes $\mathcal{R}$, il est nécessaire d'introduire une sémantique pour le langage de requêtes afin de pouvoir caractériser les modifications sur les états engendrées par l'application de ces requêtes lors d'une transition effectuée par le moniteur de référence qui implante la politique. Nous introduisons ici deux procédés différents pour définir cette sémantique. Ils seront utilisés pour comparer des modèles de contrôle d'accès selon deux techniques différentes.

Le procédé le plus classique pour définir une sémantique du langage des requêtes consiste à introduire une "sémantique de transitions" *via* une relation $[\![\mathcal{R}]\!]^+_\Sigma \subseteq \Sigma \times \mathcal{R} \times \Sigma$. Avec une telle approche, $(\sigma_1, R, \sigma_2) \in [\![\mathcal{R}]\!]^+_\Sigma$ permet de spécifier les propriétés d'un état $\sigma_2$ lorsqu'il a été obtenu par application d'une requête $R$ sur un état $\sigma_1$. Par exemple, si l'on considère l'ensemble $\mathcal{R}^{acc}$ introduit plus haut, la relation $[\![\mathcal{R}^{acc}]\!]^+_\Sigma$ peut être définie comme suit :

$$(\sigma_1, \langle +, s, o, a\rangle, \sigma_2) \in [\![\mathcal{R}^{acc}]\!]^+_\Sigma \Leftrightarrow (\Lambda(\sigma_2) = \{(s, o, a)\} \cup \Lambda(\sigma_1) \wedge \Upsilon(\sigma_1) = \Upsilon(\sigma_2))$$
$$(\sigma_1, \langle -, s, o, a\rangle, \sigma_2) \in [\![\mathcal{R}^{acc}]\!]^+_\Sigma \Leftrightarrow (\Lambda(\sigma_2) = \Lambda(\sigma_1) \backslash \{(s, o, a)\} \wedge \Upsilon(\sigma_1) = \Upsilon(\sigma_2))$$

Une autre approche, plus faible, peut être utilisée pour définir la sémantique des requêtes. Cette approche nous permettra d'obtenir des résultats facilitant la comparaison de modèles de contrôle d'accès. Elle consiste en la définition d'une relation $[\![\mathcal{R}]\!]_\Sigma \subseteq \mathcal{R} \times \Sigma$ telle que $(R, \sigma)$ appartient à $[\![\mathcal{R}]\!]_\Sigma$, si l'état $\sigma$ a pu être obtenu en appliquant avec succès la requête $R$ (on ne tient pas compte de l'état de départ). Par exemple, une sémantique faible possible pour l'ensemble $\mathcal{R}^{acc}$ est :

$$(\langle +, s, o, a\rangle, \sigma) \in [\![\mathcal{R}]\!]_\Sigma \Leftrightarrow (s, o, a) \in \Lambda(\sigma)$$
$$(\langle -, s, o, a\rangle, \sigma) \in [\![\mathcal{R}]\!]_\Sigma \Leftrightarrow (s, o, a) \notin \Lambda(\sigma)$$

Cette notion de sémantique est qualifiée de sémantique faible dans le sens où elle spécifie les propriétés que doit vérifier un état après application d'une requête, et non pas les changements effectués à partir d'un état. Néanmoins, ces changements sont en partie caractérisés par un partitionnement de l'ensemble des requêtes :

$$\mathcal{R} = \mathcal{R}^{\ominus} \cup \mathcal{R}^{\oplus} \cup \mathcal{R}^{\oslash}$$

qui permet de spécifier la variation des accès potentiels lors de l'application des requêtes. L'ensemble $\mathcal{R}^{\ominus}$ (resp. $\mathcal{R}^{\oplus}$) contient les requêtes d'élargissement (resp. rétrécissement) des accès potentiels et l'ensemble $\mathcal{R}^{\oslash}$ contient les autres requêtes. Autrement dit, si l'on passe d'un état $\sigma_1$ à un état $\sigma_2$ en appliquant avec succès une requête $R \in \mathcal{R}^{\oplus}$ (resp. $R \in \mathcal{R}^{\ominus}$), alors l'ensemble des ensembles d'accès que l'on peut ajouter à $\sigma_2$ (resp. $\sigma_1$), sans modifier les fonctions de sécurité, est inclus dans celui des ensembles d'accès que l'on peut ajouter à $\sigma_1$ (resp. $\sigma_2$) sans modifier les fonctions de sécurité. Par exemple, pour la politique de la Muraille de Chine, la requête $\langle +, s, o, a\rangle$ appartient à l'ensemble $\mathcal{R}^{\oplus}$ étant donné qu'ajouter un accès ne permet pas d'ajouter un autre accès précédemment interdit et empêche l'ajout éventuel d'autres accès. A l'inverse, la requête $\langle -, s, o, a\rangle$ appartient à $\mathcal{R}^{\ominus}$ étant donné que le relâchement d'un accès n'empêche pas d'ajouter un accès précédemment autorisé et peut autoriser l'ajout de certains accès interdits auparavant.

Bien sûr, les deux procédés de définition d'une sémantique du langage de requêtes n'ont pas le même pouvoir d'expression, mais peuvent, dans certains cas, être reliés.

### Modèles et Implantations

La notion de modèle est définie comme suit.

**Définition 2.2** *Un modèle de contrôle d'accès $\mathbb{M}[\rho]$ est défini par une paire $\mathbb{M}[\rho] = (\mathbb{P}[\rho], \mathcal{R})$ et par la donnée d'une sémantique pour $\mathcal{R}$.*

L'implantation d'un modèle de contrôle d'accès $\mathbb{M}[\rho]$ sous la forme d'une machine à états consiste à définir à la fois un ensemble d'*états initiaux* $\Sigma_I$ et une *fonction de transition* $\tau : \mathcal{R} \times \Sigma \to \mathcal{D} \times \Sigma$ (où $\mathcal{D} = \{\mathsf{yes}, \mathsf{no}\}$ est un ensemble de réponses) qui permet de passer d'un état à un autre en fonction d'une requête. Plusieurs propriétés de correction et de complétude sont définies sur les implantations. Elles sont détaillées en annexe D et concernent essentiellement la politique de contrôle d'accès appliquée (i.e. le prédicat $\Omega$) et le respect de la sémantique des requêtes. Nous notons $\mathbb{M}[\rho] \vdash (\tau, \Sigma_I)$ lorsque toutes les propriétés de correction d'une implantation $(\tau, \Sigma_I)$ d'un modèle $\mathbb{M}[\rho]$ sont satisfaites.

Plusieurs exemples de modèles classiques de contrôle d'accès accompagnés de leurs implantations sont formalisés en annexe D.1.2.

## 2.2 Comparaison de modèles de contrôle d'accès

Dans cette section nous présentons deux approches qui ont été utilisées pour comparer des modèles de contrôle d'accès. Intuitivement, un modèle de contrôle d'accès $\mathbb{M}_1[\rho_1]$ est plus restrictif qu'un modèle $\mathbb{M}_2[\rho_2]$ si et seulement si chaque implantation correcte de $\mathbb{M}_1[\rho_1]$ peut être simulée par une implantation correcte de $\mathbb{M}_2[\rho_2]$. Evidemment, la principale difficulté pour comparer deux modèles provient du fait qu'il est nécessaire de considérer toutes les implantations d'un modèle, ce qui, dans la pratique, est difficilement réalisable. Toutefois, nous allons voir que sous certaines hypothèses, vérifiées dans tous les exemples concrets que nous avons envisagés, il est possible de limiter le nombre d'implantations à considérer lors de la comparaison.

La première approche, détaillée en annexe D.2.1, a été utilisée pour comparer les modèles de la Muraille de Chine, de Bell et LaPadula et à base de rôles (RBAC96). La deuxième approche a été utilisée pour comparer les modèles HRU, de Bell et LaPadula et RBAC96. Elle est détaillée en annexe D.2.2.

### 2.2.1 Comparaison de modèles : Accès

#### Définition d'un préordre partiel sur les modèles

La première approche pour comparer deux modèles nécessite que ces deux modèles partagent le même ensemble de requêtes sur lequel une sémantique faible est définie. Cette approche ne permet donc pas de prendre en compte les requêtes administratives qui sont propres à chaque modèle. Cette restriction permet de considérer la notion de simulation d'implantations, définie de manière classique comme suit.

Etant données deux fonctions de transition $\tau_1 : \mathcal{R} \times \Sigma_1 \rightarrow \mathcal{D} \times \Sigma_1$ et $\tau_2 : \mathcal{R} \times \Sigma_2 \rightarrow \mathcal{D} \times \Sigma_2$, $\tau_2$ simule $\tau_1$, ce que nous notons $\tau_1 \overset{\kappa_\Sigma}{\rightarrow} \tau_2$, ssi il existe une relation $\kappa_\Sigma \subseteq \Sigma_1 \times \Sigma_2$ telle que :

$$\begin{array}{rl} & \forall \sigma_1, \sigma_1' \in \Sigma_1 \ \forall \sigma_2 \in \Sigma_2 \ \forall R \in \mathcal{R} \ \forall a \in \mathcal{D} \\ & \quad ((\sigma_1, \sigma_2) \in \kappa_\Sigma \wedge \tau_1(R, \sigma_1) = (a, \sigma_1')) \\ \Rightarrow & \quad (\exists \sigma_2' \in \Sigma_2 \ (\sigma_1', \sigma_2') \in \kappa_\Sigma \wedge \tau_2(R, \sigma_2) = (a, \sigma_2')) \end{array}$$

Cette définition est étendue aux implantations : l'implantation $(\tau_2, \Sigma_I^2)$ simule l'implantation $(\tau_1, \Sigma_I^1)$, ce que nous notons $(\tau_1, \Sigma_I^1) \overset{\kappa_\Sigma}{\rightarrow} (\tau_2, \Sigma_I^2)$, ssi il existe une relation $\kappa_\Sigma \subseteq \Sigma_1 \times \Sigma_2$ telle que :

$$\tau_1 \overset{\kappa_\Sigma}{\rightarrow} \tau_2 \wedge \forall \sigma_1 \in \Sigma_I^1 \ \exists \sigma_2 \in \Sigma_I^2 \quad (\sigma_1, \sigma_2) \in \kappa_\Sigma$$

En fait, il est nécessaire de préciser la notion d'ordre entre modèles. En effet, la relation de simulation utilisée pour montrer qu'un modèle $\mathbb{M}_1[\rho_1]$ est plus restrictif qu'un modèle $\mathbb{M}_2[\rho_2]$ doit satisfaire de "bonnes propriétés". Par exemple, si on considère le produit cartésien $\Sigma_1 \times \Sigma_2$ comme une relation de simulation entre implantations, il devient facile d'établir que tout modèle est plus restrictif que tout autre modèle. Il faut donc contraindre la notion de relation de simulation. La définition formelle des propriétés requises pour les relations de simulation est donnée en annexe D.2.1. Nous écrirons ici $P_s(\kappa_\Sigma)$ pour exprimer que la relation $\kappa_\Sigma$ satisfait ces propriétés. Ces propriétés reposent sur une relation d'équivalence $\equiv_\iota$ (définie en annexe D.2.1) sur les états d'un même modèle qui permet

de caractériser les états "sémantiquement" équivalents[3]. De plus, nous imposons que les fonctions de transition $\tau$ considérées préservent cette relation, ce que nous notons $\equiv_\iota \vdash \tau$ (deux états équivalents sont transformés en états équivalents par $\tau$).

Ces définitions nous permettent d'introduire une relation de préordre partiel sur les modèles : un modèle $\mathbb{M}_1[\rho_1]$ est plus restrictif qu'un modèle $\mathbb{M}_2[\rho_2]$ ssi il existe une relation de simulation (satisfaisant de bonnes propriétés) qui permet de simuler toute implantation correcte de $\mathbb{M}_1[\rho_1]$ par une implantation correcte de $\mathbb{M}_2[\rho_2]$.

**Définition 2.3** *Etant donnés deux modèles de contrôle d'accès* $\mathbb{M}_1[\rho_1] = (\mathbb{P}_1[\rho_1], \mathcal{R})$ *et* $\mathbb{M}_2[\rho_2] = (\mathbb{P}_2[\rho_2], \mathcal{R})$ *où* $\mathcal{R}$ *est muni d'une sémantique faible,* $\mathbb{M}_1[\rho_1] \leqq \mathbb{M}_2[\rho_2]$ *ssi il existe une relation* $\kappa_\Sigma \subseteq \Sigma_1 \times \Sigma_2$ *telle que* $P_s(\kappa_\Sigma)$ *et telle que :*

$$\begin{array}{ll} & \forall \tau_1 : \mathcal{R} \times \Sigma_1 \to \mathcal{D} \times \Sigma_1 \ \forall \Sigma_I^1 \subseteq \Sigma_1 \\ & \mathbb{M}_1[\rho_1] \vdash (\tau_1, \Sigma_I^1) \wedge \equiv_{\iota_1} \vdash \tau_1 \\ \Rightarrow & \exists \tau_2 : \mathcal{R} \times \Sigma_2 \to \mathcal{D} \times \Sigma_2 \ \exists \Sigma_I^2 \subseteq \Sigma_2 \\ & \mathbb{M}_2[\rho_2] \vdash (\tau_2, \Sigma_I^2) \wedge \equiv_{\iota_2} \vdash \tau_2 \wedge (\tau_1, \Sigma_I^1) \overset{\kappa_\Sigma}{\preceq} (\tau_2, \Sigma_I^2) \end{array}$$

Nous présentons à présent quelques moyens possibles pour réduire le nombre d'implantations à considérer lors de la comparaison de deux modèles.

### Notion de modèle réduit

La définition du préordre qui vient d'être introduite fait intervenir une relation d'équivalence $\equiv_\iota$ sur les états d'un même modèle. Cette relation préserve à la fois le prédicat de sécurité $\Omega$ et la sémantique faible des requêtes :

$$\begin{array}{l} (\sigma_1 \equiv_\iota \sigma_2 \wedge \Omega(\sigma_1)) \Rightarrow \Omega(\sigma_2) \\ \forall R \in \mathcal{R} \ (\sigma_1 \equiv_\iota \sigma_2 \wedge (R, \sigma_1) \in [\![\mathcal{R}]\!]_\Sigma) \Rightarrow (R, \sigma_2) \in [\![\mathcal{R}]\!]_\Sigma \end{array}$$

Une première façon de réduire le nombre d'implantations à simuler lors de la comparaison de deux modèles consiste à considérer l'ensemble quotient $\Sigma_{/\equiv_\iota}$ ce qui conduit à définir la notion de modèle réduit :

- la réduction de la politique $\mathbb{P}[\rho]$ est la politique $\mathbb{P}^\sharp[\rho] = (\mathcal{S}, \mathcal{O}, \mathcal{A}, \hat{e}(\Sigma), \Omega)$,
- la réduction du modèle $\mathbb{M}[\rho]$ est le modèle $\mathbb{M}^\sharp[\rho] = (\mathbb{P}^\sharp[\rho], \mathcal{R})$,

où $e : \Sigma \to \Sigma$ est la fonction de projection associée à $\equiv_\iota$ permettant d'associer un élément canonique à chaque état, $\hat{e}(E) = \{e(\sigma) \mid \sigma \in E\}$ et $\mathcal{R}$ est muni de la relation de sémantique faible définie par $[\![\mathcal{R}]\!]_{\hat{e}(\Sigma)} = \{(R, \sigma) \in [\![\mathcal{R}]\!]_\Sigma \mid \sigma \in \hat{e}(\Sigma)\}$.

Ce procédé permet d'associer à toute implantation $I$ d'un modèle $\mathbb{M}[\rho]$ une implantation $I^\sharp$ du modèle réduit $\mathbb{M}^\sharp[\rho]$, les propriétés de correction des implantations étant conservées

[3] Considérons par exemple le cas où une fonction de sécurité d'un état permet d'obtenir l'heure courante, afin de pouvoir enregister dans des fichiers de "*logs*" chaque action effectuée ainsi que l'heure à laquelle elle a été effectuée. L'heure de l'accès est une information qui n'a aucun impact sur la politique de sécurité ou le modèle. Autrement dit, une requête est autorisée ou non indépendamment de l'heure. Dans ce cas, deux états contenant les mêmes accès et les mêmes informations de sécurité, mais deux heures différentes, peuvent être considérés comme équivalents. Nous présentons de manière plus formelle cette notion d'équivalence, notée $\equiv_\iota$, entre états en annexe D.

par cette transformation. Il devient alors possible de définir un préordre partiel $\leqq^\sharp$ sur les modèles réduits tel que :

$$\mathbb{M}_1[\rho_1] \leqq \mathbb{M}_2[\rho_2] \Leftrightarrow \mathbb{M}_1^\sharp[\rho_1] \leqq^\sharp \mathbb{M}_2^\sharp[\rho_2]$$

Ce résultat permet de comparer deux modèles entre eux en considérant leurs modèles réduits. Cette comparaison est *a priori* plus simple, car un modèle réduit possède moins d'implantations que le modèle à partir duquel il a été obtenu par réduction. La notion de réduction de modèles permet d'abstraire un modèle en regroupant les états équivalents selon une certaine sémantique définie par la relation d'équivalence $\equiv_\iota$ utilisée pour construire le modèle réduit. Notons cependant qu'en pratique, un modèle réduit n'a pas vocation à être effectivement construit (i.e. calculé par un algorithme). En effet, la notion de modèle réduit sert essentiellement à simplifier certaines preuves (en faisant abstraction d'informations "non discriminantes" au regard de la politique[4]). Comparer deux modèles de contrôle d'accès ne nécessite pas forcément de les réduire (on peut par exemple utiliser des résultats dont l'énoncé ne fait pas intervenir la notion de modèles réduits, bien que leurs preuves reposent fortement sur cette notion). Toutefois, la caractérisation du modèle réduit associé à un modèle permet de simplifier sa manipulation dans les raisonnements et d'en avoir une compréhension plus fine.

Quoi qu'il en soit, même en comparant des modèles réduits, le nombre d'implantations à simuler peut constituer un obstacle et nous introduisons dans la suite d'autres techniques permettant de réduire, sous certaines conditions, le nombre d'implantations à considérer.

### Préordre sur les implantations

Etant donné un modèle, plusieurs implantations correctes, plus ou moins restrictives, de ce modèle peuvent être envisagées. Ces implantations peuvent correspondre à différents modes de fonctionnement du système. Nous introduisons donc un préordre $\sqsubseteq$ sur les implantations d'un même modèle (ce préordre est défini en annexe D.2.1). Intuitivement, étant données deux implantations $I$ et $I'$, $I \sqsubseteq I'$ signifie que $I'$ permet de faire des "plus petits pas" que $I$. En d'autres termes, si $I$ permet d'atteindre un état $\sigma_1$ à partir d'un état $\sigma$, alors $\sigma_1$ est également atteignable par $I'$ à partir d'un état $\sigma_2$ qui est "plus proche" de $\sigma_1$ que $\sigma$. L'intérêt de ce préordre provient du fait qu'il permet de factoriser certaines propriétés. On montre en effet que :

- Si $I$ est une implantation correcte d'un modèle, alors toute implantation $I'$ telle que $I' \sqsubseteq I$ est correcte pour ce modèle.
- Si $I$ est une implantation simulable par une implantation d'un modèle $\mathbb{M}[\rho]$, alors toute implantation $I'$ telle que $I' \sqsubseteq I$ est simulable par une implantation de $\mathbb{M}[\rho]$.

Ainsi, pour prouver qu'un modèle est plus restrictif qu'un autre, il suffit de savoir simuler toute implantation $\sqsubseteq$-maximale du premier par une implantation du deuxième. Pour profiter de ce résultat en pratique, il faut cependant que toute implantation soit inférieure à une implantation maximale. Dans le cas où il existe une chaîne infinie croissante d'implantations, cette technique n'est pas applicable.

[4] Ce mécanisme de réduction revient donc à caractériser l'information non discriminante décrite par une représentation des états du système. C'est précisément cette information qui est ignorée dans le modèle réduit. Ce procédé est similaire à celui utilisé dans les techniques d'analyse statique à base d'interprétation abstraite.

**Propriétés sur les relations de simulation**

En pratique, comparer deux modèles $\mathbb{M}_1[\rho_1]$ et $\mathbb{M}_2[\rho_2]$ conduit souvent à définir un "plongement" de $\Sigma_1$ vers $\Sigma_2$ satisfaisant de "bonnes propriétés". On peut alors définir une relation de simulation $\kappa_\Sigma$ à partir de ce plongement. Sur les exemples concrets que nous avons envisagés, les relations obtenues satisfont de "bonnes propriétés". En effet, lors de la comparaison des modèles de la Muraille de Chine, de Bell & LaPadula et de RBAC, chaque relation $\kappa_\Sigma$ préserve à la fois les propriétés de sécurité définies par les prédicats $\Omega_1$ et $\Omega_2$, et la sémantique faible des requêtes. Lorsque la relation $\kappa_\Sigma$ satisfait ces propriétés, il est possible de montrer que $\mathbb{M}_1[\rho_1] \leqq \mathbb{M}_2[\rho_2]$. Une fois établi, ce résultat permet donc de comparer deux modèles sans avoir à considérer les implantations de ces modèles, il suffit d'étudier les propriétés de la relation de simulation.

En adoptant cette approche, nous avons prouvé que le modèle de la Muraille de Chine est strictement plus restrictif que le modèle de Bell et LaPadula, qui est lui même strictement plus restrictif que le modèle RBAC à base de rôles.

### 2.2.2 Comparaison de modèles : Administration

Le mécanisme de comparaison présenté dans la section 2.2.1 n'est utilisable que si les deux modèles considérés partagent le même ensemble de requêtes muni d'une sémantique faible. Il ne permet donc pas la prise en compte des requêtes administratives permettant de modifier les informations de sécurité du système. Dans cette section, nous étendons ce mécanisme de comparaison afin de pouvoir comparer deux modèles lorsqu'ils ne partagent pas le même ensemble de requêtes. Pour ce faire, le préordre sur les modèles que nous introduisons repose sur la définition d'une relation mettant en correspondance les (séquences de) requêtes "sémantiquement" équivalentes des deux modèles et sur la notion de simulation faible qui permet la simulation d'une transition par une séquence de transitions. En effet, selon le pouvoir d'expression des deux langages de requêtes considérés $\mathcal{R}_1$ et $\mathcal{R}_2$, il se peut que pour simuler l'effet d'une requête appartenant à $\mathcal{R}_1$, il soit nécessaire d'appliquer plusieurs requêtes de $\mathcal{R}_2$. Comme nous le verrons, c'est par exemple le cas pour la requête d'ajout d'un rôle autorisé pour un utilisateur dans le modèle RBAC, dont la simulation dans le modèle HRU nécessite d'ajouter plusieurs accès autorisés et donc d'appliquer plusieurs fois la requête du modèle HRU permettant d'autoriser un nouvel accès.

**Langage de requêtes**

L'ensemble $\mathcal{R}$ des requêtes d'un modèle de contrôle d'accès peut se décomposer en trois sous-ensembles :

$$\mathcal{R} = \mathcal{R}^{acc} \cup \mathcal{R}^{adm} \cup \{Success, Fail\}$$

où $\mathcal{R}^{acc}$ est l'ensemble défini en (2.1), $\mathcal{R}^{adm}$ contient les requêtes administratives du modèle et $Success$ et $Fail$ sont des requêtes ne produisant aucun effet mais qui seront utiles pour envisager la simulation faible d'implantations. La sémantique des requêtes de $\mathcal{R}$ est définie *via* une relation $[\![\mathcal{R}]\!]^+_\Sigma \subseteq \Sigma \times \mathcal{R} \times \Sigma$. La sémantique des requêtes appartenant à $\mathcal{R}^{acc}$ a déjà été introduite page 24, celle des requêtes appartenant à $\mathcal{R}^{adm}$ dépend du modèle considéré

et la sémantique des requêtes $Succes$ et $Fail$ est définie par :

$$(\sigma_1, Success, \sigma_2) \in [\![\mathcal{R}]\!]^+_\Sigma \Leftrightarrow (\Lambda(\sigma_1) = \Lambda(\sigma_2) \wedge \Upsilon(\sigma_1) = \Upsilon(\sigma_2))$$
$$(\sigma_1, Fail, \sigma_2) \in [\![\mathcal{R}]\!]^+_\Sigma \Leftrightarrow (\Lambda(\sigma_1) = \Lambda(\sigma_2) \wedge \Upsilon(\sigma_1) = \Upsilon(\sigma_2))$$

Toutes les fonctions de transition que nous considérons dans cette section doivent vérifier les propriétés suivantes : les requêtes $Success$ et $Fail$ ne modifient pas les états, la requête $Success$ est toujours acceptée, la requête $Fail$ est toujours refusée, et si une requête est refusée alors l'état n'est pas modifié.

$$\forall \sigma \in \Sigma \quad \tau(Success, \sigma) = (\mathsf{yes}, \sigma) \wedge \tau(Fail, \sigma) = (\mathsf{no}, \sigma)$$
$$\forall \sigma_1, \sigma_2 \in \Sigma \; \forall R \in \mathcal{R} \quad \tau(R, \sigma_1) = (\mathsf{no}, \sigma_2) \Rightarrow \sigma_1 = \sigma_2$$

**Définition d'un préordre partiel sur les modèles**

Prendre en compte les requêtes administratives lors de la comparaison de modèles amène à la notion de simulation faible d'implantations, définie comme suit. Etant données deux fonctions de transition $\tau_1 : \mathcal{R}_1 \times \Sigma_1 \to \mathcal{D} \times \Sigma_1$ et $\tau_2 : \mathcal{R}_2 \times \Sigma_2 \to \mathcal{D} \times \Sigma_2$, $\tau_2$ simule faiblement $\tau_1$, ce que nous notons $\tau_1 \overset{\kappa_\Sigma, \kappa_\mathcal{R}}{\preceq} \tau_2$, ssi il existe deux relations $\kappa_\Sigma \subseteq \Sigma_1 \times \Sigma_2$ et $\kappa_\mathcal{R} \subseteq \Sigma_1 \times \mathcal{R}_1 \times \mathcal{R}_2^\star$ telles que :

$$\forall \sigma_1, \sigma_1' \in \Sigma_1 \quad \forall \sigma_2^0 \in \Sigma_2 \quad \forall R_1 \in \mathcal{R}_1 \quad \forall d_1 \in \mathcal{D}$$
$$\left( \begin{array}{ll} & \tau_1(R_1, \sigma_1) = (d_1, \sigma_1') \\ \wedge & (\sigma_1, \sigma_2^0) \in \kappa_\Sigma \end{array} \right) \Rightarrow \left( \begin{array}{ll} & \exists \sigma_2^1, \sigma_2^2, \ldots, \sigma_2^{n-1}, \sigma_2^n \in \Sigma_2 \\ & \exists (R_2^1, \ldots, R_2^n) \in \mathcal{R}_2^\star \; \exists d_2^1, \ldots, d_2^n \in \mathcal{D} \\ & \tau_2(R_2^1, \sigma_2^0) = (d_2^1, \sigma_2^1) \\ \wedge & \cdots \wedge \tau_2(R_2^n, \sigma_2^{n-1}) = (d_2^n, \sigma_2^n) \\ \wedge & d_1 = d_2^1 = d_2^2 = \cdots = d_2^n \\ \wedge & (\sigma_1, R_1, (R_2^1, \ldots, R_2^n)) \in \kappa_\mathcal{R} \\ \wedge & (\sigma_1', \sigma_2^n) \in \kappa_\Sigma \end{array} \right)$$

Chaque transition de $\tau_1$ doit donc pouvoir être simulée par une séquence de transitions de $\tau_2$. Cette définition, illustrée sur la figure 2.1, est étendue aux implantations : l'implantation $(\tau_2, \Sigma_2^I)$ simule faiblement $(\tau_1, \Sigma_1^I)$, ce que nous notons $(\tau_1, \Sigma_1^I) \overset{\kappa_\Sigma, \kappa_\mathcal{R}}{\preceq} (\tau_2, \Sigma_2^I)$, ssi il existe deux relations $\kappa_\Sigma \subseteq \Sigma_1 \times \Sigma_2$ et $\kappa_\mathcal{R} \subseteq \Sigma_1 \times \mathcal{R}_1 \times \mathcal{R}_2^\star$ telles que :

$$\tau_1 \overset{\kappa_\Sigma, \kappa_\mathcal{R}}{\preceq} \tau_2 \wedge \forall \sigma_1 \in \Sigma_1^I \;\; \exists \sigma_2 \in \Sigma_2^I \quad (\sigma_1, \sigma_2) \in \kappa_\Sigma$$

Remarquons ici que la relation $\kappa_\mathcal{R}$ est une relation ternaire. Le jugement $(\sigma, R_1, (R_2^1, \cdots, R_2^n))$ exprime que la séquence de requêtes $(R_2^1, \cdots, R_2^n)$ simule la requête $R_1$ lorsque le système se trouve dans un état $\sigma$. En effet, la séquence de requêtes peut dépendre de l'état dans lequel se trouve le système. C'est par exemple le cas pour la requête d'ajout d'une permission $(o, a)$ à un rôle $r$ dans le modèle RBAC, dont la simulation dans le modèle HRU nécessite d'ajouter les accès $(s, o, a)$ aux accès autorisés pour chaque sujet $s$ ayant activé un rôle $r'$ supérieur à $r$. Ici, l'ensemble des rôles activés par un sujet est défini par une fonction de sécurité, qui dépend donc de l'état du système.

Ici encore, les relations $\kappa_\Sigma$ et $\kappa_\mathcal{R}$ utilisables pour montrer les propriétés de simulation qui interviennent dans la définition du préordre sur les modèles doivent satisfaire certaines conditions. Il s'agit essentiellement de garantir que l'équivalence "sémantique" entre états

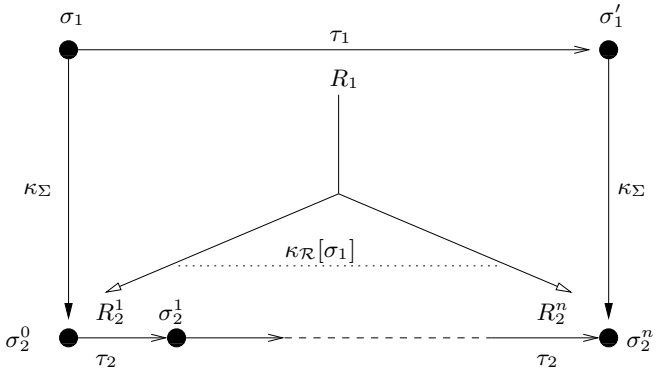


FIG. 2.1 – Simulation faible

induite par la relation $\kappa_\Sigma$ est préservée lors de la simulation faible. Plus formellement, nous imposons que $\kappa_\Sigma$ soit $\mathcal{W}$-préservante :

$$\forall \sigma_1 \in \Sigma_1 \quad \forall \sigma_2 \in \Sigma_2 \quad (\sigma_1, \sigma_2) \in \kappa_\Sigma \Rightarrow \mathcal{W}(\sigma_1) = \mathcal{W}(\sigma_2)$$

Ces définitions nous permettent d'introduire la relation de préordre partiel suivante sur les modèles de contrôle d'accès : un modèle $\mathbb{M}_1[\rho_1]$ est plus restrictif qu'un modèle $\mathbb{M}_2[\rho_2]$ ssi pour toute implantation correcte $I_1$ de $\mathbb{M}_1[\rho_1]$, il existe deux relations $\kappa_\Sigma$ et $\kappa_\mathcal{R}$ (satisfaisant de bonnes propriétés) qui permettent de simuler $I_1$ par une implantation correcte de $\mathbb{M}_2[\rho_2]$.

**Définition 2.4** *Soient* $\mathbb{M}_1[\rho_1] = (\mathbb{P}_1[\rho_1], \mathcal{R}_1)$ *et* $\mathbb{M}_2[\rho_2] = (\mathbb{P}_2[\rho_2], \mathcal{R}_2)$ *deux modèles.*

$$\mathbb{M}_1[\rho_1] \trianglelefteq \mathbb{M}_2[\rho_2]$$

$$\Leftrightarrow \left( \begin{array}{l} \forall \tau_1 : \mathcal{R}_1 \times \Sigma_1 \to \mathcal{D} \times \Sigma_1 \quad \forall \Sigma_1^I \subseteq \Sigma_1 \\ \\ \mathbb{M}_1[\rho_1] \vdash (\tau_1, \Sigma_1^I) \Rightarrow \left( \begin{array}{l} \exists \tau_2 : \mathcal{R}_2 \times \Sigma_2 \to \mathcal{D} \times \Sigma_2 \quad \exists \Sigma_2^I \subseteq \Sigma_2 \\ \exists \kappa_\Sigma \subseteq \Sigma_1 \times \Sigma_2 \; \exists \kappa_\mathcal{R} \subseteq \Sigma_1 \times \mathcal{R}_1 \times \mathcal{R}_2^\star \\ \quad \kappa_\Sigma \textit{ et } \kappa_\mathcal{R} \textit{ sont totales à gauche} \\ \wedge \; \kappa_\Sigma \textit{ est } \mathcal{W}\textit{-préservante} \\ \wedge \; \mathbb{M}_2[\rho_2] \vdash (\tau_2, \Sigma_2^I) \wedge (\tau_1, \Sigma_1^I) \overset{\kappa_\Sigma, \kappa_\mathcal{R}}{\rightharpoonup} (\tau_2, \Sigma_2^I) \end{array} \right) \end{array} \right)$$

Remarquons que la comparaison de deux modèles $\mathbb{M}_1[\rho_1]$ et $\mathbb{M}_2[\rho_2]$ selon la relation $\leqq$ nécessite la construction d'une relation de simulation qui permet de simuler toutes les implantations correctes de $\mathbb{M}_1[\rho_1]$ par des implantations correctes de $\mathbb{M}_2[\rho_2]$, alors que la comparaison de $\mathbb{M}_1[\rho_1]$ et $\mathbb{M}_2[\rho_2]$ selon la relation $\trianglelefteq$ nécessite la construction d'une relation de simulation pour chaque implantation correcte de $\mathbb{M}_1[\rho_1]$. Toutefois, en pratique, les relations $\kappa_\Sigma$ et $\kappa_\mathcal{R}$ sont définies indépendamment de l'implantation considérée et il suffit alors de restreindre $\kappa_\mathcal{R}$ en fonction de la fonction de transition à simuler.

### Propriétés sur les relations de simulation

Sur les exemples que nous avons considérés, les relations $\kappa_\Sigma$ et $\kappa_\mathcal{R}$ satisfont de "bonnes propriétés" : ces relations préservent à la fois les propriétés de sécurité définies par les

prédicats $\Omega_1$ et $\Omega_2$, et la sémantique des requêtes. Dans ce cas, il est possible de montrer que $\mathbb{M}_1[\rho_1] \trianglelefteq \mathbb{M}_2[\rho_2]$. Une fois établi, ce résultat permet donc de comparer deux modèles sans avoir à considerer les implantations de ces modèles, il suffit d'étudier les propriétés des relations $\kappa_\Sigma$ et $\kappa_\mathcal{R}$.

## 2.3 Analyse de flots d'information

Une politique de contrôle d'accès permet de spécifier quels sont les accès autorisés lorsque le système se trouve dans un certain état mais ne permet pas, tout du moins de manière explicite, de spécifier les flots d'information qui sont autorisés durant la vie du système. Ainsi, une fois qu'un sujet a pu accéder à un objet, il n'y a généralement aucun contrôle sur la propagation de l'information lue par le sujet. Par exemple, avec un modèle discrétionnaire, il est possible qu'un sujet $s_1$ lise un objet $o_1$ et recopie l'information lue dans un objet $o_2$ accessible en lecture par un sujet $s_2$ non autorisé à lire $o_1$. La politique de contrôle d'accès est ici respectée mais ne coïncide pas avec son interprétation en termes de flots d'information. Dans cette section nous étudions la cohérence entre ces deux lectures sémantiques des modèles de contrôle d'accès et introduisons les concepts utiles pour analyser les flots d'information engendrés par les exécutions des implantations de modèles de contrôle d'accès.

### 2.3.1 Flots et Politiques de Flots

Nous étudions ici les flots d'information qui se produisent entre les entités du système. Nous distinguons donc les flots entre objets, entre sujets et entre sujets et objets. Pour cela, nous intoduisons les notations suivantes :

- Nous notons $\hookrightarrow^{OO}$ le produit cartésien $\mathcal{O} \times \mathcal{O}$. Un élément $o_1 \hookrightarrow^{OO} o_2$ de $\mathcal{O} \times \mathcal{O}$ permet d'exprimer que le contenu d'un objet $o_1$ est propagé dans un objet $o_2$. Nous caractériserons par la suite plusieurs sous-ensembles de $\hookrightarrow^{OO}$ permettant de décrire des flots d'information entre objets dans différents contextes. Une politique de confinement est un sous-ensemble $\rightsquigarrow^{OO}$ de $\hookrightarrow^{OO}$.
- Nous notons $\hookrightarrow^{OS}$ le produit cartésien $\mathcal{O} \times \mathcal{S}$. Un élément $o \hookrightarrow^{OS} s$ de $\mathcal{O} \times \mathcal{S}$ permet d'exprimer qu'un sujet $s$ prend connaissance des informations contenues dans un objet $o$. Nous caractériserons par la suite plusieurs sous-ensembles de $\hookrightarrow^{OS}$ permettant de décrire des flots d'information des objets vers les sujets dans différents contextes. Une politique de confidentialité est un sous-ensemble $\rightsquigarrow^{OS}$ de $\hookrightarrow^{OS}$.
- Nous notons $\hookrightarrow^{SO}$ le produit cartésien $\mathcal{S} \times \mathcal{O}$. Un élément $s \hookrightarrow^{SO} o$ de $\mathcal{S} \times \mathcal{O}$ permet d'exprimer qu'un sujet $s$ propage les informations dont il dispose dans un objet $o$. Nous caractériserons par la suite plusieurs sous-ensembles de $\hookrightarrow^{SO}$ permettant de décrire des flots d'information des sujets vers les objets dans différents contextes. Une politique d'intégrité est un sous-ensemble $\rightsquigarrow^{SO}$ de $\hookrightarrow^{SO}$.
- Nous notons $\hookrightarrow^{SS}$ le produit cartésien $\mathcal{S} \times \mathcal{S}$. Un élément $s_1 \hookrightarrow^{SS} s_2$ de $\mathcal{S} \times \mathcal{S}$ permet d'exprimer qu'un sujet $s_1$ rend accessibles les informations dont il dispose à un sujet $s_2$. Nous caractériserons par la suite plusieurs sous-ensembles de $\hookrightarrow^{SS}$ permettant de décrire des flots d'information entre sujets dans différents contextes.

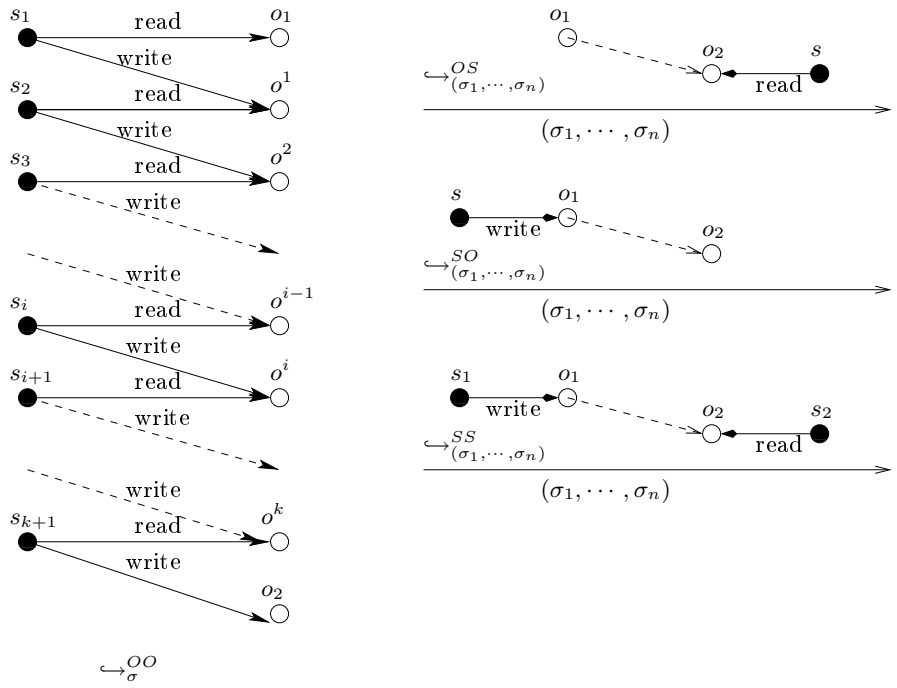


FIG. 2.2 – Flots d'information

**Flots d'information engendrés par une séquence d'états**

Les flots d'information créés durant la vie d'un système sont caractérisés à partir d'une séquence d'états décrivant les états successifs du système. A partir de l'ensemble des accès courants qui ont lieu lorsque le système se trouve dans un état $\sigma$, nous pouvons définir les flots d'information entre objets comme suit. Le contenu d'un objet $o_1$ est diffusé dans un objet $o_2$ s'il existe un ensemble de sujets dont les accès sur les objets du système permettent de recopier l'information de $o_1$ dans $o_2$, cette information pouvant transiter par des objets intermédiaires :

$$\hookrightarrow_\sigma^{OO} = \left\{ o_1 \hookrightarrow^{OO} o_2 \mid \left( \begin{array}{l} \exists s_1, \cdots, s_k, s_{k+1} \in \mathcal{S}\ \exists o^1, \cdots, o^k \in \mathcal{O} \\ \left\{ \begin{array}{l} (s_1, o_1, \mathsf{read}), (s_1, o^1, \mathsf{write}), \\ (s_2, o^1, \mathsf{read}), (s_2, o^2, \mathsf{write}), \\ \cdots, \\ (s_i, o^{i-1}, \mathsf{read}), (s_i, o^i, \mathsf{write}), \\ \cdots, \\ (s_{k+1}, o^k, \mathsf{read}), (s_{k+1}, o_2, \mathsf{write}) \end{array} \right\} \subseteq \Lambda(\sigma) \\ \vee o_1 = o_2 \end{array} \right) \right\}$$

Cette définition, illustrée sur la partie gauche de la figure 2.2, peut être étendue à un ensemble d'états $E \subseteq \Sigma$ : $\hookrightarrow_E^{OO} = \bigcup_{\sigma \in E} \hookrightarrow_\sigma^{OO}$.

Nous pouvons à présent définir les flots d'information qui ont lieu durant une séquence d'états $(\sigma_1, \cdots, \sigma_n)$ :

– Les flots entre objets sont obtenus par composition des flots engendrés par chacun des états apparaissant dans la séquence :

$$\hookrightarrow^{OO}_{(\sigma_1,\cdots,\sigma_n)} = \begin{cases} \hookrightarrow^{OO}_{\sigma_1} & \text{si } n = 1 \\ \hookrightarrow^{OO}_{\sigma_{k+1}} \circ \hookrightarrow^{OO}_{(\sigma_1,\cdots,\sigma_k)} & \text{si } n = k+1 \end{cases}$$

– Les flots d'un objet $o$ vers un sujet $s$ sont identifiés lorsqu'un objet $o'$ a reçu l'information contenue dans $o$ et que $s$ accède ensuite en lecture à $o'$ :

$$\hookrightarrow^{OS}_{(\sigma_1,\cdots,\sigma_n)} = \bigcup_{i=1}^{n} \left\{ o_2 \hookrightarrow^{OS} s \mid o_2 \hookrightarrow^{OO}_{(\sigma_1,\cdots,\sigma_i)} o_1 \wedge (s, o_1, \mathsf{read}) \in \Lambda(\sigma_i) \right\}$$

– Les flots d'un sujet $s$ vers un objet $o$ sont identifiés lorsque $s$ accède en écriture à un objet $o'$ dont le contenu est ensuite diffusé dans $o$ :

$$\hookrightarrow^{SO}_{(\sigma_1,\cdots,\sigma_n)} = \bigcup_{i=1}^{n} \left\{ s \hookrightarrow^{SO} o_2 \mid (s, o_1, \mathsf{write}) \in \Lambda(\sigma_i) \wedge o_1 \hookrightarrow^{OO}_{(\sigma_i,\sigma_{i+1},\cdots,\sigma_n)} o_2 \right\}$$

– Les flots d'un sujet $s_1$ vers un sujet $s_2$ sont identifiés lorsque $s_1$ écrit dans un objet dont le contenu est ensuite diffusé dans un objet qui est ensuite accédé en lecture par $s_2$ :

$$\hookrightarrow^{SS}_{(\sigma_1,\cdots,\sigma_n)} = \bigcup_{i=1}^{n} (\hookrightarrow^{OS}_{(\sigma_i,\cdots,\sigma_n)} \circ \hookrightarrow^{SO}_{(\sigma_1,\cdots,\sigma_i)})$$

Ces définitions, illustrées sur la partie droite de la figure 2.2, sont étendues aux ensembles d'états et aux ensembles de séquences d'états. Si $E \subseteq \Sigma$ est un ensemble d'états et $F \subseteq E^\star$ est un ensemble de séquences d'états, on définit :

$$\hookrightarrow^{X}_{Y} = \bigcup_{y \in Y} \hookrightarrow^{X}_{y}$$

où $X \in \{OO, OS, SO, SS\}$ et $Y \in \{E, F\}$.

Nous venons de caractériser formellement les flots d'information qui se produisent entre les entités du système durant la vie de ce système. Il est maintenant possible de comparer ces ensembles de flots avec les flots autorisés par une politique de confidentialité $\rightsquigarrow^{OS}$ et une politique d'intégrité $\rightsquigarrow^{SO}$. Un ensemble $X$, correspondant soit à un ensemble d'états ($X = E$), soit à un ensemble de séquences d'états ($X = F$), est :

– *correct* pour $\rightsquigarrow^{OS}$ (resp. pour $\rightsquigarrow^{SO}$) ssi $\hookrightarrow^{OS}_{X} \subseteq \rightsquigarrow^{OS}$ (resp. $\hookrightarrow^{SO}_{X} \subseteq \rightsquigarrow^{SO}$).
– *complet* pour $\rightsquigarrow^{OS}$ (resp. pour $\rightsquigarrow^{SO}$) ssi $\rightsquigarrow^{OS} \subseteq \hookrightarrow^{OS}_{X}$ (resp. $\rightsquigarrow^{SO} \subseteq \hookrightarrow^{SO}_{X}$).

La propriété de correction exprime que tous les flots engendrés par $X$ sont autorisés par la politique et la propriété de complétude exprime que tous les flots autorisés par la politique sont "réalisables" par $X$.

### Politiques de flots induites par une politique de contrôle d'accès

Une politique de contrôle d'accès $\mathbb{P}[\rho] = (\mathcal{S}, \mathcal{O}, \mathcal{A}, \Sigma, \Omega)$ peut être interprétée par une politique de confidentialité et/ou une politique d'intégrité. Exprimées en termes de flots d'information, ces politiques sont définies comme suit :

$$\rightsquigarrow^{OS}_{\mathbb{P}[\rho]} = \{ o \hookrightarrow^{OS} s \mid \exists \sigma \in \Sigma_{|\Omega} \ (s, o, \mathsf{read}) \in \Lambda(\sigma) \}$$
$$\rightsquigarrow^{SO}_{\mathbb{P}[\rho]} = \{ s \hookrightarrow^{SO} o \mid \exists \sigma \in \Sigma_{|\Omega} \ (s, o, \mathsf{write}) \in \Lambda(\sigma) \}$$

Ainsi, $o \rightsquigarrow^{OS}_{\mathbb{P}[\rho]} s$ signifie que pour un état sûr du système, le sujet $s$ accède à l'information contenue dans l'objet $o$ et $s \rightsquigarrow^{SO}_{\mathbb{P}[\rho]} o$ signifie que pour un état sûr du système, le sujet $s$ propage l'information dont il dispose dans l'objet $o$.

### 2.3.2 Mécanismes de détection de flots

Les définitions introduites ci-dessus permettent d'étudier la cohérence entre les flots engendrés par les exécutions des implantations d'un modèle de contrôle d'accès et les politiques de confidentialité et d'intégrité induites par la politique de contrôle d'accès. La section D.3 en annexe présente en détail une analyse de flots pour les modèles HRU et de Bell et LaPadula. Pour certains modèles, comme celui de HRU, certaines séquences d'états, respectant la politique de contrôle d'accès, engendrent des flots non autorisés par les politiques de confidentialité et d'intégrité induites par cette politique. Dans ce cas, il peut être utile de compléter le mécanisme de contrôle d'accès à l'aide d'un mécanisme de détection de flots. Nous donnons ici la spécification formelle d'un tel mécanisme. La section D.3 en annexe présente un exemple de mécanisme de détection de flots défini dans ce formalisme. Il s'agit d'une formalisation d'un système de détection d'intrusion, décrit dans [55, 54], dont nous illustrons l'utilisation pour le modèle HRU.

#### Détection de flots

Etant donné un ensemble de flots $\rightsquigarrow_{\mathbb{F}}$, détecter les flots appartenant à $\rightsquigarrow_{\mathbb{F}}$ lors de la vie d'un système consiste à observer les séquences d'états du système et à collecter des informations permettant d'identifier celles d'entre elles qui engendrent des flots recherchés. Chaque état $\sigma$ est donc associé à une information $\psi(\sigma)$ qui rend compte d'une partie du passé de cet état. C'est à partir de cette information que l'on peut définir un prédicat $\mho$ sur $\Sigma$ permettant de caractériser les états issus de séquences d'états engendrant un flot appartenant à $\rightsquigarrow_{\mathbb{F}}$. Ces états sont appelés des états d'alerte et on note $\Sigma_{|\mho}$ l'ensemble $\{\sigma \in \Sigma \mid \mho(\sigma)\}$. Etant donnés un ensemble $E$ d'états observables et un ensemble $F \subseteq E^\star$ de séquences d'états qui peuvent se produire, un mécanisme de détection de flots est défini par :

$$\mathbb{F}[\rho, E, F, \rightsquigarrow_{\mathbb{F}}] = (\Sigma, \mho)$$

Bien sûr, il faut s'assurer que le prédicat $\mho$ caractérise bien les états issus d'une séquence produisant un flot dans $\rightsquigarrow_{\mathbb{F}}$. Pour cela, nous introduisons les deux propriétés suivantes. $\mathbb{F}[\rho, E, F, \rightsquigarrow_{\mathbb{F}}] = (\Sigma, \mho)$ est :

- *correct* ssi tout état d'alerte est issu d'une séquence engendrant un flot dans $\rightsquigarrow_{\mathbb{F}}$ :

$$\forall(\sigma_1, \cdots, \sigma_n) \in F \quad \mho(\sigma_n) \Rightarrow \hookrightarrow^{X}_{(\sigma_1,\cdots,\sigma_n)} \cap \rightsquigarrow_{\mathbb{F}} \neq \emptyset$$

- *complet* ssi tout état issu d'une séquence engendrant un flot dans $\rightsquigarrow_{\mathbb{F}}$ est un état d'alerte :

$$\forall(\sigma_1, \cdots, \sigma_n) \in F \quad \hookrightarrow^{X}_{(\sigma_1,\cdots,\sigma_n)} \cap \rightsquigarrow_{\mathbb{F}} \neq \emptyset \Rightarrow \mho(\sigma_n)$$

où $X \in \{OO, OS, SO\}$.

**Detection de flots illégaux**

Un mécanisme de détection de flots $\mathbb{F}[\rho, E, F, \rightsquigarrow_{\mathbb{F}}]$ peut être utilisé pour s'assurer qu'une politique de flots $\rightsquigarrow$ est bien respectée. Dans ce cas, ce mécanisme est dit :

- correct pour $\rightsquigarrow$ ssi : $\rightsquigarrow_{\mathbb{F}} \cap \not\rightsquigarrow = \emptyset$
  Si $\mathbb{F}[\rho, E, F, \rightsquigarrow_{\mathbb{F}}]$ est de plus correct, alors tout état d'alerte est issu d'une séquence engendrant un flot qui ne respecte pas la politique $\rightsquigarrow$.
- complet pour $\rightsquigarrow$ ssi : $(\hookrightarrow_F^X \setminus \rightsquigarrow) = \not\rightsquigarrow_{\mathbb{F}}$
  Si $\mathbb{F}[\rho, E, F, \rightsquigarrow_{\mathbb{F}}]$ est de plus complet, alors tout état issu d'une séquence engendrant un flot ne respectant pas la politique $\rightsquigarrow$ est un état d'alerte.

Muni de ce formalisme, nous pouvons montrer que lorsque nous obtenons un état d'alerte, alors soit le mécanisme de détection de flots n'est pas correct, soit il n'est pas correct pour la politique de flots considérée, soit il est issu d'une séquence engendrant un flot non autorisé par la politique de flots.

**Proposition 2.1** *Soit* $\mathbb{F}[\rho, E, F, \rightsquigarrow_{\mathbb{F}}] = (\Sigma, \mho)$ *un mécanisme de détection de flots et* $\rightsquigarrow$ *une politique de flots. Si* $\hookrightarrow_F^X \subseteq \not\rightsquigarrow$ *(où* $X \in \{OO, OS, SO\}$*) et si* $\mathbb{F}[\rho, E, F, \rightsquigarrow_{\mathbb{F}}]$ *est correct et correct pour* $\rightsquigarrow$*, alors* $E \cap \Sigma_{|\mho} = \emptyset$.

## 2.4 Analyse de modèles de contrôle d'accès

Dans ce chapitre nous avons introduit un cadre formel permettant de définir *a priori* n'importe quel modèle de contrôle d'accès, et fournissant les outils nécessaires pour comparer deux modèles entre eux et analyser les flots d'information que les exécutions de leurs implantations engendrent (implicitement). Nous avons donc défini un cadre uniforme permettant la définition de modèles de contrôle d'accès : nous avons identifié les "ingrédients" communs aux politiques de contrôle d'accès, exprimé les propriétés génériques qu'ils vérifient, prouvé certaines et formalisé les politiques envisagées comme des instances du cadre générique. Ce cadre différencie la notion de politique de contrôle d'accès de celle de modèle de contrôle d'accès. Une politique est la spécification du "quoi", dans le sens où elle définit quelles sont les entités et quels sont les états (sûrs et non sûrs). Un modèle est la spécification du "comment", dans le sens où il décrit comment passer d'un état à un autre. Plusieurs techniques de comparaison ont été introduites. Tout d'abord, une relation de préordre a été définie sur les implantations d'un modèle. Cette relation correspond à une notion de restriction : intuitivement, l'implantation la plus petite d'un modèle est celle qui permet de faire le moins de choses. Conjointement à la notion de modèle, nous avons introduit la notion de modèle réduit, qui s'obtient en ignorant l'information "inutile", c'est-à-dire l'information non discriminante au regard de la politique de sécurité. Nous avons également proposé deux préordres sur les modèles permettant de comparer formellement deux modèles de contrôle d'accès. Ces préordres expriment une notion intuitive de "plongement" : un modèle est plus restrictif qu'un autre modèle si toute implantation du premier est simulable par une implantation du deuxième. Nous avons enfin établi des résultats permettant de comparer deux modèles sans avoir à considérer toutes leurs implantations.

Quelques travaux ont déjà eu lieu sur la comparaison de modèles de contrôle d'accès mais ils sont encore parcellaires : [122] envisage la comparaison de politiques de contrôle

d'accès en termes de puissance d'expression, [23] utilise des techniques de simulation pour comparer des modèles de contrôle d'accès discrétionnaire, [123] envisage la comparaison de politiques sous l'angle de la combinaison de politiques. Chacune de ces approches adopte un point de vue différent sur la notion de contrôle d'accès et il est difficile de les comparer directement. Notons toutefois que la plupart de ces approches abordent principalement la comparaison de modèles discrétionnaires et ne distinguent pas la sémantique des requêtes des fonctions de transition qui permettent d'appliquer ces requêtes comme nous le faisons. Elles sous-entendent donc que les politiques sont mises en œuvre de la manière la moins restrictive possible (au sens de la relation $\sqsubseteq$ introduite dans cet article). D'autre part, lorsqu'elles prennent en compte les requêtes administratives lors de la comparaison, ces approches ne permettent pas que la relation $\kappa_{\mathcal{R}}$ dépende de l'état du système et sont donc très restrictives. Quoi qu'il en soit, toutes ces approches portent sur le même objet et méritent d'être reconsidérées et étendues dans un cadre uniforme afin d'en étudier les liens et d'en dégager de nouvelles techniques de réutilisation. Cette étude permettrait donc d'outiller (ou d'enrichir) le cadre sémantique proposé ici pour prendre en compte ces approches.

Plusieurs mécanismes peuvent être envisagés pour garantir que les accès effectués dans un système d'information respectent la politique de sécurité souhaitée. Usuellement, on se ramène à :

- la mise en œuvre d'un moniteur de référence qui filtre les accès pour ne permettre que ceux qui ne violent pas la politique de sécurité,
- la mise en œuvre d'un système de détection d'intrusion qui repose sur l'observation des transferts d'information qui ont lieu dans le système et qui lève une alerte en cas de transfert "suspect".

Le premier moyen est une approche défensive puisqu'aucun accès non autorisé n'est possible, tandis que le second relève d'une approche préventive puisqu'elle se contente de lever un alerte en cas de doute. Généralement, ces deux mécanismes n'opèrent pas au même niveau : un moniteur de référence est habituellement défini et implanté dans un langage de haut niveau et peut être vu comme une application, tandis qu'un système de détection d'intrusion peut opérer au niveau du système d'exploitation en observant les transferts d'information engendrés lors de l'exécution d'un programme applicatif. Quoi qu'il en soit, ces deux approches ont pour vocation de garantir des propriétés de confidentialité et/ou d'intégrité sur les données d'un système d'information et partagent donc un ensemble de spécifications. Dans ce chapitre, nous avons étudié les liens qui existent entre ces deux mécanismes. Nous avons donc présenté une analyse des flots d'information associés à un modèle de contrôle d'accès. Il s'agit ici d'une interprétation de la notion de politique de contrôle d'accès différente de celle qui est implantée par un moniteur de référence. En effet, une politique de contrôle d'accès ne fournit généralement pas explicitement de propriétés sur la dissémination des informations dans le système et afin de garantir qu'une certaine politique de flots est respectée, il peut être utile d'adjoindre un mécanisme de détection de flots au moniteur de référence.

Une importante littérature sur les flots d'information existe. Dans [32], D.E. Denning introduit une notion de modèle de flots d'information et présente dans [33] un procédé de transformation de politiques de flots, satisfaisant certaines conditions, en une politique de flots ayant une structure de treillis (et pouvant donc être mise en œuvre par une politique

de contrôle d'accès *à la* Bell et LaPadula). Dans [34], D.E. Denning utilise ce même modèle pour définir un mécanisme de certification de programmes garantissant des propriétés sur les flots engendrés lors de l'exécution de programmes. Dans [43], S.N. Foley introduit un modèle de flots permettant de prendre en compte des politiques de flots non nécessairement transitives et illustre l'intérêt d'un tel modèle sur des exemples concrets. Les liens entre modèles de contrôle d'accès et modèles de contrôle de flots ont également été étudiés. Dans [90], J. McLean introduit une théorie des flots d'information afin de définir un modèle de sécurité à base de flots. Dans [99], S.L. Osborn décrit comment caractériser les flots possibles dans un système RBAC à partir des paramètres de sécurité de ce système. Plus récemment, dans [9] les auteurs formalisent la notion de DTE (*Domain Type Enforcement*) [10] afin de fournir un mécanisme de contrôle de flots qu'ils intègrent ensuite dans le modèle OrBAC (*Organization based access control model*) [73]. Ici encore, tous ces travaux portent sur des notions communes et il serait intéressant de les exprimer dans notre formalisme afin de pouvoir analyser leurs points communs et leurs différences, de comparer leur pouvoir d'expression et de comprendre comment une implantation de l'un de ces modèles peut être étendue pour inclure les particularités d'un autre modèle.

# Conclusion – Perspectives

Les travaux présentés dans ce document débutent tous par une description formelle du domaine étudié. A chaque fois, la formalisation entreprise a permis d'éclairer certains points obscurs.

- En obligeant à examiner de manière très approfondie les différents aspects et composants de l'objet formalisé. C'est le cas par exemple du domaine du contrôle d'accès.
- En suscitant l'expression de certaines hypothèses restées implicites et de certaines propriétés indispensables pour établir la preuve de propriétés souhaitées. Ce fût par exemple le cas lors de la formalisation des résultats classiques de la programmation logique.
- En permettant la détection et la correction de certaines erreurs. Cela a été le cas lors de la formalisation des propriétés de sécurité de la politique de Bell et LaPadula exprimées de manière informelle, ou encore lors de la mécanisation de la preuve de complétude de la SLD-résolution.

De plus, l'ensemble des formalisations effectuées a permis de dégager des méthodes permettant une réutilisation plus aisée de développements formels. Enfin, puisque l'activité de formalisation permet de mieux comprendre l'objet formalisé, il nous a semblé naturel d'adopter cette démarche dans certaines activités d'enseignement. L'article [30] présente une expérience d'enseignement de la sémantique en utilisant le système Coq et l'atelier Focal.

Le domaine principalement abordé dans ce document est celui de la sécurité, sur lequel nous travaillons depuis 2003 et plus particulièrement de la sécurité au niveau accès. Il s'agit ici de régir et de gérer les accès aux données d'un système d'information. De nombreux modèles de contrôle d'accès existent aujourd'hui, et nous avons introduit un cadre uniforme permettant de définir formellement ces modèles. Ce cadre fournit un guide méthodologique lors de la conception d'un modèle et peut aider un utilisateur à adopter une démarche rigoureuse lors de son développement. De plus, exprimer des modèles au sein d'un même cadre permet de les comparer et d'obtenir une compréhension plus fine des propriétés souhaitées pour le modèle en cours d'élaboration.

L'étude de la comparaison de modèles de contrôle d'accès est un premier pas vers l'étude de la composition de ces modèles. Cette problématique mérite aussi d'être développée puisqu'elle correspond à un problème concret très répandu dans les systèmes d'information. En effet, dans la plupart de ces systèmes, un sujet accède généralement à un objet en passant au travers de plusieurs filtres (par exemple un employé accède aux données du système d'information de son entreprise régi par une certaine politique, après avoir pénétré dans les locaux de cette entreprise, eux-mêmes régis par une autre politique de contrôle

d'accès). Il existe évidemment plusieurs procédés pour composer des politiques de sécurité mais il y a peu d'études sur les propriétés de ces compositions. Nous souhaitons donc étudier cette problématique afin de formaliser les concepts liés à la composition pour pouvoir exprimer les propriétés de sécurité que les mécanismes de composition envisagés peuvent garantir.

La démarche que nous avons adoptée pour aborder la problématique du contrôle d'accès a consisté à étudier plusieurs modèles afin d'en dégager les traits communs et de concevoir un cadre uniforme permettant non seulement de factoriser une partie du travail de développement lors de la conception d'un modèle, mais aussi d'établir des propriétés génériques sur les modèles de contrôle d'accès. Cette démarche peut être appliquée dans d'autres contextes. En collaboration avec V.Delebarre de la société SAFERIVER, j'encadre actuellement le travail de thèse de Liên Tran (débuté en Février 2008) sur la formalisation des propriétés relatives à l'intégrité d'exécution de composants de sécurité. Il s'agit ici non pas de s'assurer que la fonction d'un composant est assurée mais que ce composant fonctionne dans des conditions qui permettent d'avoir confiance dans les résultats qu'il produit. Plus précisément, nous tentons de formaliser les notions d'environnement et d'interfaces d'un composant ainsi que les propriétés que doit respecter ce composant lors de son exécution vis à vis de son environnement et de ses interfaces. Le composant à partir duquel nous travaillons est un composant cryptographique (assurant essentiellement des fonctions de chiffrement et de déchiffrement) dont le fonctionnement est régi par la norme FIPS140-3. Cette norme spécifie les diverses procédures d'initialisation et de chargement des clés ainsi qu'une hiérarchie de modes de fonctionnement du composant, chacun de ces modes garantissant un certain niveau de sécurité.

Les travaux que j'ai présentés sont fortement ancrés sur la logique et la sémantique des langages de programmation. Ces deux paradigmes sont des guides précieux et aussi des garde-fous efficaces pour construire une démarche d'analyse et de construction raisonnée des différents domaines, notions et outils constituant les centres d'interêt de la science informatique. Ils ne sont peut-être pas suffisamment connus ni reconnus. Ces travaux et les activités d'enseignement exercées conjointement peuvent aussi être lus comme une tentative de mieux faire connaître et apprécier ces paradigmes.

# Bibliographie


[1] Ocaml-mysql v. 1.0.4 : `http ://raevnos.pennmush.org/code/ocaml-mysql/`.

[2] M.A. Nait Abdallah. On the interpretation of infinite computations in logic programming. In J. Paredaens, editor, *11th International Colloquium on Automata, Languages and Programming, ICALP'84*, volume 172 of *Lecture Notes in Computer Science*, pages 358–370. Springer-Verlag, 1984.

[3] M.A. Nait Abdallah and M.H. van Emden. Top-down semantics of fair computations of logic programs. *Journal of Logic Programming*, 2(1) :67–76, 1985.

[4] P. Amey. Dear sir, yours faithfully : an everyday story of formality. In *Practical Elements of Safety, Proc. of the Twelfth Safety-critical Systems Symposium*. Spinger-Verlag, 2004.

[5] J. P. Anderson. Computer Security Technology Planning Study. Technical Report ESD-TR-73-51, USAF Electronic Systems Division, Hanscom Air Force Base, Bedford, Massachusetts, October 1972.

[6] F. Anseaume, J. Baron, P. Berthelin, M. Jacquel, and D. Pipon. Formalisation, spécification et implantation de politiques de contrôle d'accès avec l'atelier focal. Master's thesis, UPMC, Paris, France, 2008.

[7] K.R. Apt. Logic programming. In J. van Leewen, editor, *Handbook of Theoretical Computer Science*, volume B : Formal Models and Semantics, chapter 10, pages 493–574. The MIT Press, New York, 1990.

[8] K.R. Apt and M.H. Van Emden. Contributions to the theory of logic programming. *Journal of the ACM*, 29(3) :841–862, 1982.

[9] S. Ayed, N. Cuppens-Boulahia, and F. Cuppens. An integrated model for access control and information flow requirements. In I. Cervesato, editor, *ASIAN*, volume 4846 of *Lecture Notes in Computer Science*, pages 111–125. Springer, 2007.

[10] L. Badger, D.F. Sterne, D.L. Sherman, K.M. Walker, and S.A. Haghighat. Practical domain and type enforcement for UNIX. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 66–77, Oakland, CA, May 1995. IEEE Computer Society, Technical Committee on Security and Privacy, IEEE Computer Society Press.

[11] G. Barthe and O. Pons. Type isomorphisms and proof reuse in dependent type theory. In F. Honsell and M. Miculan, editors, *Proc. of 4th Int. Conf. on Found. of Software Science and Computation Structures, FoSSaCS'01*, volume 2030 of *Lecture Notes in Computer Science*, pages 57–71. Springer-Verlag, 2001.

[12] D. Bell and L. LaPadula. Secure Computer Systems : a Mathematical Model. Technical Report MTR-2547 (Vol. II), MITRE Corp., Bedford, MA, May 1973.

[13] E. Bertino and R.S. Sandhu. Database security-concepts, approaches, and challenges. *IEEE Trans. Dependable Sec. Comput*, 2(1), 2005.

[14] J. Blond and C. Morisset. Formalisation et implantation d'une politique de sécurité d'une base de données. In INRIA, editor, *17ème Journées Francophones des Langages Applicatifs, JFLA'2006*, pages 71–86, 2006.

[15] J. Blond and C. Morisset. Un moniteur de référence sûr d'une base de données. *Technique et Science Informatiques*, 26(9) :1091–1110, 2007.

[16] O. Boite. Proof reuse with extended inductive types. In K.Slind, A.Bunker, and G.Gopalakrishnan, editors, *TPHOLs*, volume 3223 of *Lecture Notes in Computer Science*, pages 50–65. Springer, 2004.

[17] S. Boulmé. *Spécification d'un environnement dédié à la programmation certifiée de bibliothèques de Calcul Formel.* PhD thesis, Université Paris 6, 2000.

[18] F. Brecht and A.D. Kadja. Implantation d'une politique de contrôle d'accès discrétionnaire avec focal. Master's thesis, UPMC, Paris, France, 2007.

[19] D. F. C. Brewer and M. J. Nash. The Chinese Wall Security Policy. In *Proc. IEEE Symposium on Security and Privacy*, pages 206–214, 1989.

[20] M. Carlier and C. Dubois. Functional testing in the Focal environment. In B.Beckert and R.Hähnle, editors, *Tests and Proofs, Second International Conference, TAP 2008, Prato, Italy, April 9-11, 2008. Proceedings*, volume 4966 of *Lecture Notes in Computer Science*, pages 84–98. Springer, 2008.

[21] Pierre Castéran. *An exercice on groups*, 1997. `http ://coq.inria.fr/contribs/groups.tar.gz`.

[22] CC. *Common Criteria for Information Technology Security Evaluation, v3.1*, 2006. `http ://www.commoncriteriaportal.org/`.

[23] A. Chander, J.C. Mitchell, and D. Dean. A state-transition model of trust management and access control. In *Proc. of the 14th IEEE Computer Security Foundations Workshop CSFW*, pages 27–43. IEEE Computer Society Press, 2001.

[24] E. Cohen, R. K. Thomas, W. Winsborough, and D. Shands. Models for coalition-based access control (CBAC). In *SACMAT '02 : Proceedings of the seventh ACM symposium on Access control models and technologies*, pages 97–106. ACM Press, 2002.

[25] A. Colmerauer, H. Kanoui, and M. Van Caneghem. Prolog, bases théoriques et développements actuels. *Technique et Science Informatiques*, 2(4), 1983.

[26] T. Coquand. Infinite objects in type theory. In H. Barendregt and T. Nipkow, editors, *1st International Workshop on Types for Proofs and Programs, TYPES'93*, volume 806 of *LNCS*, pages 62–78. Springer-Verlag, 1994.

[27] D. J. Howe. Importing mathematics from HOL into nuprl. In J. Von Wright, J. Grundy, and J. Harrison, editors, *Ninth International Conference on Theorem Proving in Higher Order Logics TPHOLs'96*, volume 1125 of *Lecture Notes in Computer Science*, pages 267–282. Springer Verlag, 1996.

[28] A. Santana de Oliveira. *Réécriture et Modularité pour les Politiques de Sécurité*. Phd thesis, Université Henri Poincaré, 2008.

[29] D. Delahaye, J.F. Étienne, and V. Donzeau-Gouge. Certifying airport security regulations using the Focal environment. In J. Misra, T. Nipkow, and E. Sekerinski, editors, *FM 2006 : Formal Methods, 14th International Symposium on Formal Methods, 2006, Proceedings*, volume 4085 of *Lecture Notes in Computer Science*, pages 48–63. Springer, 2006.

[30] D. Delahaye, M. Jaume, and V. Prevosto. Coq : un outil pour l'enseignement. *Technique et Science Informatiques*, 24(9) :1139–1160, 2005.

[31] E. Denney. A prototype proof translator from HOL to Coq. In M. Aagaard and J. Harrison, editors, *13th International Conference on Theorem Proving in Higher Order Logics TPHOLs'2000*, volume 1869 of *Lecture Notes in Computer Science*, pages 108–125. Springer-Verlag, 2000.

[32] D. E. Denning. A lattice model of information flow. *Communications of the ACM*, 19(5) :236–243, May 1976.

[33] D. E. Denning. On the derivation of lattice structured information flow policies. Technical Report TR-179, Dep. of Computer Science, Purdue U., W. Lafayette, Ind., 1976.

[34] D.E. Denning and P.J. Denning. Certification of programs for secure information flow. *Commun. ACM*, 20(7) :504–513, 1977.

[35] P. Deransart and J. Maluszynski. *A Grammatical View of Logic Programming*. The MIT Press, 1993.

[36] C. Dubois and V. Viguié Donzeau-Gouge. A step towards the mechanization of partial functions : domains as inductive predicates. In *Workshop on mechanization of partial functions, CADE 15*, July 1998.

[37] C. Dubois, J. Grandguillot, and M. Jaume. Réutilisation de preuves formelles : Une étude pour le système FoC. In INRIA, editor, *14ème Journées Francophones des Langages Applicatifs, JFLA'2003*, pages 63–75, 2003.

[38] C. Dubois, T. Hardin, and V. Viguié Donzeau Gouge. Building certified components within focal. In *Symposium on Trends in Functional Programming*, 2004.

[39] E. Eder. Properties of substitutions and unifications. *Journal of Symbolic Computation*, 1(1) :31–46, 1985.

[40] A. Felty and D. Howe. Generalization and reuse of tactic proofs. In *Fifth International Conference on Logic Programming and Automated Reasoning, LPAR'94*, volume 822 of *Lecture Notes in Computer Science*, pages 1–15. Springer Verlag, 1994.

[41] A. Felty and D. Howe. Hybrid interactive theorem proving using Nuprl in HOL. In W. McCune, editor, *14th International Conference on Automated Deduction, CADE'14*, volume 1249 of *LNAI*, pages 351–365. Springer Verlag, 1997.

[42] D. F. Ferraiolo and D. R. Kuhn. Role-based access control. In *Proceedings of the 15th National Computer Security Conference*, 1992.

[43] S.N. Foley. Secure information flow using security groups. In *IEEE Computer Security Foundations Workshop CSFW*, pages 62–72, 1990.

[44] J.A. Goguen and R.M. Burstall. Introducing Institutions. In E. Clarke and D. Kozen, editors, *Logics of Programs : Workshop, Carnegie Mellon University, June 1983*, volume 164 of *Lecture Notes in Computer Science*, pages 221–256. Springer-Verlag, 1984.

[45] W.G. Golson. Toward a declarative semantics for infinite objects in logic programming. *Journal of Logic Programming*, 5(2) :151–164, 1988.

[46] M.J.C. Gordon, R. Milner, and C. Wadsworth. *Edinburgh LCF : A mechanised logic of computation*. Springer-Verlag, 1979.

[47] E. Gureghian, Th. Hardin, and M. Jaume. A full formalisation of the Bell and La Padula security model. Research Report 2003-007, SPI-LIP6, University Paris 6, 2003.

[48] L. Habib. Formalisation, comparaison et implantation d'un modèle de contrôle d'accès à base de rôles. Master's thesis, UPMC, Paris, France, 2007.

[49] L. Habib, M. Jaume, and C. Morisset. A formal comparison of the Bell & LaPadula and RBAC models. In *Fourth International Conference on Information Assurance and Security (IAS'08)*. To appear, IEEE CS Press, 2008.

[50] T. Hardin, M. Jaume, and C. Morisset. Access control and rewrite systems. In *1st International Workshop on Security and Rewriting Techniques, SecRet'06 (Satellite Workshop to ICALP'2006)*, 2006.

[51] M. Harrison, W. Ruzzo, and J. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8) :461–471, 1976.

[52] J. Hein. Completions of perpetual logic programs. *Theoretical Computer Science*, 99(1) :65–78, 1992.

[53] J. Herbrand. Recherches sur la théorie de la Démonstration (thesis 1930). *Logical Writings*, 1971.

[54] G. Hiet, L. Mé, B. Morin, and V. Viet Triem Tong. Monitoring both OS and program level information flows to detect intrusions against network servers. In *IEEE Workshop on "Monitoring, Attack Detection and Mitigation"*, 2007.

[55] G. Hiet, L. Mé, J. Zimmermann, C. Bidan, B. Morin, and V. Viet Triem Tong. Détection fiable et pertinente de flux d'informations illégaux. In *Proc of the 2nd Conference on Security in Network Architectures and Information Systems (SARSSI)*, 2007.

[56] D. J. Howe. Semantics foundations for embedding HOL in Nuprl. In M. Wirsing and M. Nivat, editors, *Proceedings of the Algebraic Methodology and Software Technology Conference AMAST'96*, volume 1101 of *Lecture Notes in Computer Science*, pages 85–101. Springer-Verlag, 1996.

[57] G. Huet. *A Uniform Approach to Type Theory*, volume Logical Foundations of Functional Programming, pages 337–398. Addison-Wesley, 1990.

[58] J. Jaffar and P.J. Stuckey. Semantics of infinite tree logic programming. *Theoretical Computer Science*, 46(2-3) :141–158, 1986.

[59] M. Jaume. Formalisation de la SLD-résolution dans le Calcul des Constructions Inductives. In *Actes des 6èmes Journées Francophones de Programmation Logique et programmation par Contraintes, JFPLC'97*, pages 277–291. Hermès, 1997.

[60] M. Jaume. Unification : a Case Study in Transposition of Formal Properties. In E.L. Gunter and A. Felty, editors, *Supplementary Proceedings of the 10th International Conference on Theorem Proving in Higher Order Logics : Poster session TPHOLs'97*, pages 79–93, 1997.

[61] M. Jaume. A full formalisation of SLD-resolution in the Calculus of Inductive Constructions. *Journal of Automated Reasoning*, 23(3–4) :347–371, 1999.

[62] M. Jaume. Logic programming and co-inductive definitions. In *Fixed Points in Computer Science, FICS'00 (Workshop to LC'2000)*, 2000.

[63] M. Jaume. Logic programming and co-inductive definitions. In P. Clote and H. Schwichtenberg, editors, *14th International Workshop, CSL'2000, Annual Conference of the European Association for Computer Science Logic*, volume 1862 of *Lecture Notes in Computer Science*, pages 343–355. Springer Verlag, 2000.

[64] M. Jaume. Preuves infinies en programmation logique. In *Actes des 9èmes Journées Francophones de Programmation Logique et programmation par Contraintes, JFPLC'2000*, pages 33–47. Hermès, 2000.

[65] M. Jaume. On greatest fixpoint semantics of logic programming. *Journal of Logic and Computation*, 12(2) :321–342, 2002.

[66] M. Jaume and C. Dubois. Formalisation of general logics in the Calculus of Inductive Constructions : Towards an abstract development to assist formal specification of logics. Research Report 37-1999, LaMI, University of Evry, France, 1999.

[67] M. Jaume and C. Morisset. Formalisation and implementation of access control models. In *Information Assurance and Security (IAS'05) International Conference on Information Technology, ITCC*, pages 703–708. IEEE CS Press, 2005.

[68] M. Jaume and C. Morisset. A formal approach to implement access control. *Journal of Information Assurance and Security*, 2 :137–148, June 2006.

[69] M. Jaume and C. Morisset. Towards a formal specification of access control. In *Joint Workshop on Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis FCS-ARSPA'06 (Workshop to LICS'2006)*, 2006.

[70] M. Jaume and C. Morisset. Contrôler le contrôle d'accès : Approches formelles. In *Approches Formelles dans l'Assistance au Développement de Logiciels, AFADL'07*, 2007.

[71] M. Jaume and C. Morisset. Un cadre sémantique pour le contrôle d'accès. *Technique et Science Informatiques*, 2008.

[72] A. Abou El Kalam, R. El Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel, and G. Trouessin. Organization based access control. In *Policy'2003*, Como, Italie, June 2003.

[73] A. Abou El Kalam, S. Benferhat, A. Miège, R. El Baida, F. Cuppens, C. Saurel, P. Balbiani, Y. Deswarte, and G. Trouessin. Organization based access control. In *IEEE 4th International Workshop on policies for distributed systems and networks*, page 120. IEEE Computer Society, 2003.

[74] R.S. Kemp and G.A. Ringwood. Reynolds and Heyting models of logic programs. In *ICLP94 Workshop on Proof Theoretical extensions of logic programming*, 1994.

[75] C. Kirchner and H. Kirchner. Rewriting, solving, proving, 2006.

[76] H.P. Ko and M.E. Nadel. Substitution and refutation revisited. In Koichi Furukawa, editor, *Proceedings of the Eighth International Conference on Logic Programming*, pages 679–692. The MIT Press, 1991.

[77] T. Kolbe and C. Walther. Proof analysis, generalization, and reuse. In W. Bibel and P. H. Schmidt, editors, *Automated Deduction : A Basis for Applications. Volume II, Systems and Implementation Techniques*. Kluwer Academic Publishers, 1998.

[78] R. Lalement. *Computation as Logic*. Prentice Hall International Series in Computer Science, 1993.

[79] B.W. Lampson. Protection. *Operating Systems Review*, 8(1) :18–24, January 1974.

[80] L.J. LaPadula and D.E. Bell. Secure Computer Systems : A Mathematical Model. *Journal of Computer Security*, 4 :239–263, 1996.

[81] J.L. Lassez, M. Maher, and K. Mariott. Unification revisited. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 587–625. M. Kaufman, Los Altos, California, 1987.

[82] J.L. Lassez and M.J. Maher. Closures and fairness in the semantics of programming logic. *Theoretical Computer Science*, 29(1-2) :167–184, 1984.

[83] H.M. Levy. *Capability-Based Computer Systems*. Digital Press, Bedford, MA, 1984.

[84] J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, second, extended edition, 1987.

[85] J.W. Lloyd and J.C. Sheperdson. Partial evaluation in logic programming. *Journal of Logic Programming*, 11(3-4) :217–242, 1991.

[86] N. Magaud and Y. Bertot. Changing data structures in type theory :a study of natural numbers. In P. Callaghan, Z. Luo, J. McKinna, and R. Pollack, editors, *Types for Proofs and Programs TYPES'2000*, volume 2277 of *Lecture Notes in Computer Science*, pages 181–196. Springer-Verlag, 2000.

[87] Z. Manna and R.J. Waldinger. Deductive synthesis of the unification algorithm. *Science of Computer Programming*, 1(1-2) :5–48, 1981.

[88] Z. Manna and R.J. Waldinger. Fundamentals of deductive program synthesis. *IEEE Transactions on Software Engineering*, 18(8) :674–704, 1992.

[89] McLean. The algebra of security. In *Proc. IEEE Symposium on Security and Privacy*, pages 2–7. IEEE Computer Society Press, 1988.

[90] J. McLean. Security models and information flow. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 180–187, 1990.

[91] J Meseguer. General logics. In H.-D. Ebbinghaus et al., editors, *Logic Colloquium '87*. North-Holland Publishing Co., 1989.

[92] A. Mikhajlova and J. Von Wright. Proving isomorphism of first-order logic proof systems in HOL. In J. Grundy and M. Newey, editors, *11th International Conference on Theorem Proving in Higher Order Logics TPHOLs'98*, volume 1479 of *Lecture Notes in Computer Science*, pages 295–314. Springer Verlag, 1998.

[93] C. Morisset. *Sémantique des systèmes de contrôle d'accès*. PhD thesis, Université Pierre et Marie Curie, 2007.

[94] C. Morisset and A. Santana de Oliveira. Automated detection of information leakage in access control. In *Proceedings of the 2nd International Workshop on Security and Rewriting Techniques (SecReT'07)*, Paris, France, 2007.

[95] P. Naumov. Formalization of isabelle meta logic in nuPRL. Technical Report TR99-1769, Cornell University, Computer Science, 1999.

[96] P. Naumov. Importing isabelle formal mathematics into nuPRL. Technical Report TR99-1734, Cornell University, Computer Science, 1999.

[97] N.J. Nilsson. *Principles of Artificial Intelligence*. Symbolic Computation. Springer-Verlag, 1982.

[98] U. Nilsson and J. Maluszynski. *Logic Programming and Prolog*. Wiley, 1990.

[99] S.L. Osborn. Information flow analysis of an RBAC system. In *7th ACM Symposium on Access Control Models and Technologies SACMAT*, pages 163–168, 2002.

[100] C. Palamidessi. Algebraic properties of idempotent substitutions. In M.S. Paterson, editor, *Proceedings, 17th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 443 of *Lecture Notes in Computer Science*, pages 386–399. Springer-Verlag, 1990.

[101] C. Paulin-Mohring. Inductive definitions in the system Coq. Rules and properties. In M. Bezem and J.F. Groote, editors, *Proceedings of the* $1^{st}$ *International Conference on Typed Lambda Calculi and Applications, TCLA'93*, volume 664 of *Lecture Notes in Computer Science*, pages 328–345. Springer-Verlag, 1993.

[102] L.C. Paulson. Verifying the unification algorithm in LCF. *Science of Computer Programming*, 5 :143–169, 1985.

[103] L.C. Paulson and A.W. Smith. Logic programming, functional programming, and inductive definitions. In P. Schroeder-Heister, editor, *International Workshop on Extensions of Logic Programming*, volume 475 of *LNCS*, pages 283–310. Springer-Verlag, 1989.

[104] A. Podelski, W. Charatonik, and M. Müller. Set-based failure analysis for logic programs and concurrent constraint programs. In S. Doaitse Swierstra, editor, *8th European Symposium on Programming, ESOP'99*, LNCS, pages 177–192. Springer-Verlag, 1999.

[105] O. Pons. Ingénierie de la preuve. In INRIA, editor, *Onzièmes Journées Francophones des Langages Applicatifs, JFLA'2000*, 2000.

[106] V. Prevosto. *Conception et Implantation du langage FoC pour le développement de logiciels certifiés*. PhD thesis, Université Paris 6, sep 2003.

[107] V. Prevosto and D. Doligez. Algorithms and proof inheritance in the Foc language. *Journal of Automated Reasoning*, 29(3-4) :337–363, dec 2002.

[108] V. Prevosto, D. Doligez, and Th. Hardin. Algebraic structures and dependent records. In C. Munoz, S. Tahar, and V. Carreno, editors, *TPHOLs : 15th International Workshop on Higher Order Logic Theorem Proving and Its Applications*, volume 2410. LNCS, Springer-Verlag, aou 2002.

[109] V. Prevosto and M. Jaume. Making proofs in a hierarchy of mathematical structures. In *11th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning, Calculemus 2003*, pages 89–100. Aracne, 2003.

[110] Focal project. *Focal, version 0.2 Tutorial and reference manual.* LIP6 – INRIA – CNAM, sept 2004. Distribution available at : `http ://focal.inria.fr`.

[111] C. Renard. *Un peu d'extensionnalité en Coq - une tactique pour les sétoïdes.* Mémoire de DEA, Université Paris 7, 2001.

[112] R. Rioboo. *Programmer le Calcul Formel, des Algorithmes à la Sémantique, Mémoire d'Habilitation.* Mémoire d'habilitation, Université Pierre et Marie Curie, Paris, France, 2002.

[113] A.J. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1), 1965.

[114] A.J. Robinson. *Logic : Form and Function. The Mechanization of Deductive Reasoning.* Edinburgh University Press, 1979.

[115] J. Rouyer. Développement de l'algorithme d'unification dans le Calcul des Constructions avec Types Inductifs. Research Report 1795, INRIA-Lorraine, 1992.

[116] Sandhu and Chen. The multilevel relational (MLR) data model. *ACMTISS : ACM Transactions on Information and System Security*, 1, 1998.

[117] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2) :38–47, 1996.

[118] E.Y. Shapiro. Alternation and the computational complexity of logic programs. *Journal of Logic Programming*, 1(1) :19–33, 1984.

[119] J.C. Shepherdson. The role of standardising apart in logic programming. *Theoretical Computer Science*, 129(1) :143–166, 1994.

[120] L. Sterling and E. Shapiro. *The Art of Prolog : Advanced Programming Techniques.* MIT Press, Cambridge, Ma, 1986.

[121] R. K. Thomas. Team-based access control (tmac) : a primitive for applying role-based access controls in collaborative environments. In *RBAC '97 : Proceedings of the second ACM workshop on Role-based access control*, pages 13–19. ACM Press, 1997.

[122] M.V. Tripunitara and N. Li. Comparing the expressive power of access control models. In *11th ACM Conf. on Computer and Communications Security.* ACM SIGSAC, 2004.

[123] M.C. Tschantz and S. Krishnamurthi. Towards reasonability properties for access-control policy languages. In D.F. Ferraiolo and I. Ray, editors, *Proceedings of SACMAT 2006*, pages 160–169. ACM, 2006.

[124] M.H. van Emden and R.A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4) :733–742, 1976.

# Annexe A

# L'atelier Focal

L'atelier Focal [110, 38, 107], et la méthodologie sous-jacente, ont initialement été conçus pour répondre au besoin d'accroître la confiance dans les résultats fournis par les algorithmes de calcul formel. C'est en fait ce domaine qui a servi de modèle et donné les lignes directrices du développement de l'atelier. Grâce à l'atelier Focal, R. Rioboo a pu construire une bibliothèque de calcul formel conséquente [112] comprenant des algorithmes complexes et dont l'efficacité est comparable à celle des meilleurs systèmes de calcul formel. Focal est maintenant utilisé dans différents domaines et en particulier dans le domaine de la sécurité. Outre les développements que nous effectuons avec Focal sur les politiques de contrôle d'accès, Focal a été utilisé avec succès pour formaliser la politique de sécurité au sol d'un aéroport [29].

L'atelier Focal [106, 107, 108, 38, 110] fournit un environnement de développement intégré (IDE) logiquement fondé, avec une sémantique claire et qui permet d'obtenir des implantations efficaces. Focal offre un langage muni de traits objets (héritage multiple, liaison tardive, redéfinition, ...) permettant non seulement de structurer un développement de manière à le rendre facilement réutilisable, mais aussi d'obtenir un logiciel par raffinements successifs en passant progressivement de la spécification à l'implantation. Focal permet d'écrire au sein d'un même cadre de travail des spécifications, des programmes et des preuves. Cet atelier permet aux programmeurs d'écrire des preuves formelles de leur code. Ils sont aidés dans leur tâche par Zenon, un démonstrateur automatique. Les preuves obtenues sont vérifiées par Coq.

Un programme Focal est la donnée d'une hiérarchie d'espèces. Une espèce peut être vue comme un ensemble de méthodes, identifiées par leur nom. Chaque méthode peut être soit déclarée (constantes, opérations et propriétés) soit définie (implantations des opérations et preuves des théorèmes), et appartient à l'une des trois classes suivantes :

- Le type support est la méthode qui représente l'ensemble sous-jacent à la structure définie par l'espèce. Chaque espèce doit avoir un seul type support, et il peut être soit déclaré soit défini. Un type support seulement déclaré correspond alors à un type abstrait de données, et un type support défini correspond à un type concret.
- Les méthodes opérationnelles sont les constantes ou les fonctions de la structure. Les méthodes qui ne sont que déclarées sont appelées des signatures. Le langage utilisé pour définir ces méthodes est similaire à celui du noyau fonctionnel d'OCaml, avec une construction permettant d'appeller une méthode d'une espèce donnée.

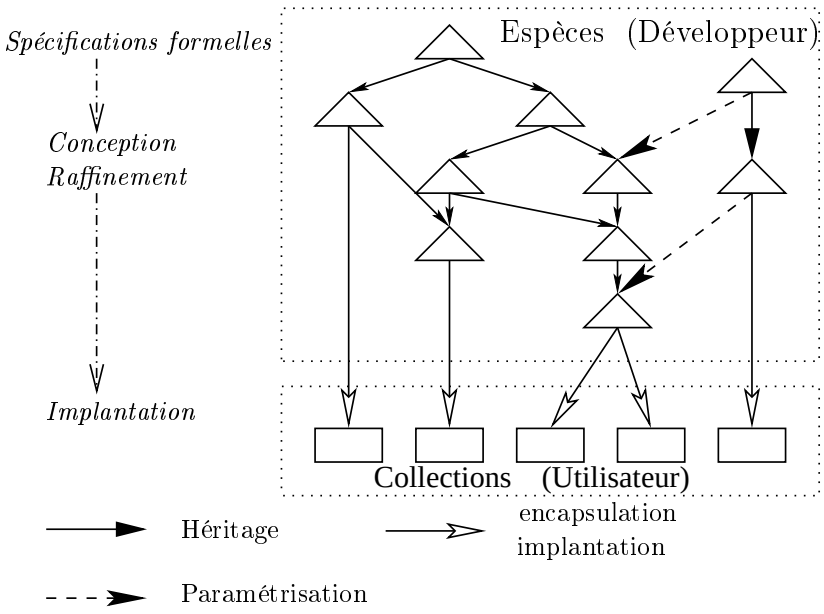


FIG. A.1 – Construction d'un programme Focal

– Les méthodes logiques décrivent les propriétés des méthodes opérationnelles. Dans ce contexte, la déclaration d'une méthode logique est simplement l'énoncé d'une propriété, tandis que la définition d'une méthode logique est la preuve que cette propriété est vraie.

Le langage utilisé pour les énoncés de propriétés est composé des connecteurs logiques de base et des quantificateurs existentiels et universels sur les types Focal. Ces propriétés sont donc des formules de logique du premier ordre, contenant les noms des méthodes liées par déclaration/définition, héritage ou paramétrisation.

Au bas de la hiérachie Focal, on trouve les collections. Une collection est construite sur une espèce complétement définie (toutes les méthodes opérationnelles sont définies et toutes les méthodes logiques sont prouvées). De plus, une collection est "figée", dans le sens où on ne peut plus en hériter, et on ne peut pas définir de nouvelles méthodes ni en redéfinir. Enfin, toutes les définitions (type support et preuves compris) sont masquées à l'utilisateur afin que ce dernier n'ait accès qu'à l'interface de la collection (noms des méthodes et propriétés). La figure A.1 illustre la construction d'une hiérarchie d'espèces Focal.

La compilation d'un programme Focal s'effectue en plusieurs étapes. Le fichier source est analysé par le compilateur, qui effectue un calcul de dépendances permettant d'éliminer des cycles dans les dépendances (condition suffisante pour éviter des incohérhences logiques [17]), de gérer l'héritage multiple et la liaison tardive (relier un nom de méthode à sa définition la plus récente), d'effacer les preuves dépendant des définitions de fonctions en cas de redéfinition de celles-ci. Puis, le fichier est compilé vers un fichier Ocaml, qui correspond, une fois compilé, à un programme exécutable. Le fichier est également compilé vers un source Coq, contenant toute la structuration du programme ainsi que toutes

les preuves. Chaque preuve écrite en Zenon est analysée par cet outil, et transformée en un terme de preuve Coq. Enfin, le fichier est compilé vers un format intermédiaire de documentation, permettant de générer du XML, du LaTeX, de l'UML, etc. Un outil permettant de démontrer la terminaison des fonctions récursives est en cours d'intégration. Focal propose également un outil de génération automatique de test et un outil graphique de représentation des dépendances entre espèces et méthodes.

# Annexe B

# Formalisations

## B.1 Unification des termes du premier ordre

### B.1.1 Formalisation de l'unification dans le système Coq

L'unification est une opération de nature syntaxique permettant de résoudre un système d'équations entre termes du premier ordre. Cette opération est la "cheville ouvrière" de nombreux algorithmes de démonstration automatique et de systèmes d'aide à la preuve. Dans le domaine de la programmation, l'unification joue un rôle central dans certains algorithmes de typage et en programmation logique. Si les algorithmes d'unification de termes du premier ordre sont aujourd'hui bien connus et bien maîtrisés, leur implantation est parfois délicate et source de confusions. En effet, l'unification repose sur la notion de substitution, notion pour laquelle, comme nous le verrons, coexistent plusieurs définitions non-équivalentes. Aussi, j'ai entrepris la mécanisation, avec le système Coq, de la preuve de la décidabilité du problème de l'unification des termes du premier ordre [60]. L'objectif de ce développement est double : tout d'abord, grâce au mécanisme d'extraction du système Coq, j'ai obtenu un programme certifié implantant l'unification ; enfin, le développement obtenu a servi de base à la formalisation de la sémantique de la programmation logique présentée dans la section B.2.

Etant donnés une signature $\Sigma$ (i.e. un ensemble dénombrable muni d'une application $ar : \Sigma \to \mathbb{N}$) et un ensemble dénombrable $X$ de symboles de variables, l'ensemble $T_\Sigma[X]$ des termes est défini de manière inductive :

- Un symbole de variable est un terme.
- Si $f \in \Sigma$ est un symbole d'arité $n > 0$, et si $t_1, \cdots, t_n$ sont des termes, alors $f(t_1, \cdots, t_n)$ est un terme.

Deux termes $t_1$ et $t_2$ sont dits *unifiables* si il existe une substitution $\theta$ telle que $\theta(t_1) = \theta(t_2)$. Décider si deux termes $t_1$ et $t_2$ sont unifiables revient à résoudre l'équation $t_1 = t_2$. Plus généralement, l'algorithme d'unification permet de résoudre un système d'équations $E$ entre termes en transformant $E$ en un système de la forme $\cup_1^n \{x_i = t_i\}$, tel que les variables $x_i$ soient deux à deux distinctes et tel que $\{x_1, \cdots, x_n\} \cap \cup_1^n var(t_i) = \emptyset$ ($var(t)$ dénote l'ensemble des variables apparaissant dans le terme $t$), si $E$ admet une solution (et dans ce cas, ce système correspond à une substitution) ou en un système noté $\{\bot\}$ si $E$ n'admet pas de solution. Un tel système est dit *résolu*. L'algorithme d'unification peut être vu comme la définition d'une relation de transition, notée $\rightsquigarrow$. Résoudre un système d'équations $E$

| | | |
|---|---|---|
| $\{f(t_1,\cdots,t_n)=f(t'_1,\cdots,t'_n)\}\cup E$ | $\overset{\text{décomposition}}{\rightsquigarrow}$ | $\{t_1=t'_1,\cdots,t_n=t'_n\}\cup E$ |
| $\{f(t_1,\cdots,t_n)=g(t'_1,\cdots,t'_n)\}\cup E$ | $\overset{\text{conflit}}{\rightsquigarrow}$ | $\{\bot\}$ si $f\neq g$ |
| $\{x=t\}\cup E$ | $\overset{\text{élimination}}{\rightsquigarrow}$ | $\{x=t\}\cup E[x\leftarrow t]$ si $x\in var(E)$ et $x\notin var(t)$ |
| $\{x=t\}\cup E$ | $\overset{\text{occurrence}}{\rightsquigarrow}$ | $\{\bot\}$ si $x\in var(t)$ |
| $\{t=t\}\cup E$ | $\overset{\text{effacement}}{\rightsquigarrow}$ | $E$ |
| $\{t=x\}\cup E$ | $\overset{\text{inversion}}{\rightsquigarrow}$ | $\{x=t\}\cup E$ si $t\notin X$ |

TAB. B.1 – Algorithme d'unification

entre termes revient alors à construire une suite de transitions $E \rightsquigarrow E_1 \rightsquigarrow \cdots \rightsquigarrow E_n$ telle que le système $E_n$ soit résolu. Cet algorithme, introduit pour la première fois par J. Herbrand [53], est classique et est rappelé dans le tableau B.1. Le développement que j'ai effectué avec le système Coq est une mécanisation des définitions classiques des termes et des substitutions, et de la preuve de la propriété suivante :

$$\forall t_1, t_2 \in T_\Sigma[X] \quad (\exists\theta\ mgu(\theta, t_1, t_2)) \vee (\forall\theta\ \theta(t_1) \neq \theta(t_2))$$

où $mgu(\theta, t_1, t_2)$ signifie que $\theta$ est une substitution minimale, idempotente, ne portant que sur les variables apparaissant dans les termes $t_1$ et $t_2$ et telle que $\theta(t_1) = \theta(t_2)$. Un programme implantant un algorithme d'unification a été extrait de la preuve obtenue. Ce travail permet donc de disposer d'une implantation certifiée de l'algorithme d'unification, qui peut être utilisée avec profit dans les nombreux domaines où l'unification constitue un mécanisme de base (programmation logique, typage, ...).

### B.1.2 Formaliser ce qui semble déjà formel ?

De nombreux ouvrages et articles décrivent les algorithmes d'unification des termes du premier ordre. Ces algorithmes portent sur des objets à l' "allure mathématique" sur lesquels il est donc possible d'établir des propriétés dans un cadre assez formel. Il peut alors paraître inutile d'utiliser un outil comme Coq pour décrire cette théorie. Toutefois, la notion de substitution n'est pas aussi simple qu'il y paraît et la littérature en fournit de nombreuses présentations différentes. Bien sûr, selon les goûts et les besoins de chacun, plusieurs définitions (et représentations) équivalentes peuvent exister. Cependant, dans le domaine de l'informatique, il est surprenant de constater que plusieurs définitions non équivalentes coexistent : en 1991, H.P. Ko et M.E. Nadel [76] recensaient les définitions suivantes.

- Une substitution est une fonction de $X$ dans $T_\Sigma[X]$.
- E. Eder [39], C.Kirchner et H.Kirchner [75] définissent une substitution comme une fonction $\theta$ de $X$ dans $T_\Sigma[X]$ pour laquelle l'ensemble $\{x \in X,\ \theta(x) \neq x\}$ est fini (i.e. $\theta$ est l'identité presque partout).

- E.Y.Shapiro [118] définit une substitution comme une fonction $\theta$ d'un sous-ensemble fini de $X$ dans $T_\Sigma[X]$ (i.e. $\theta$ est une fonction partielle).
- K.R.Apt [7], J.L.Lassez, M.Maher et K.Mariott [81] et J.W.Lloyd [84] définissent une substitution comme une fonction $\theta$ d'un sous-ensemble fini $D \subseteq X$ dans $T_\Sigma[X]$ telle que pour tout $x \in D$, $\theta(x) \neq x$.
- N.J.Nilsson [97] définit une substitution comme une fonction $\theta$ d'un sous-ensemble fini $D \subseteq X$ dans $T_\Sigma[X]$ telle que pour tout $x \in D$, $x \notin \theta(x)$.
- L.Sterling et E.Y.Shapiro [120] définissent une substitution comme une fonction $\theta$ d'un sous-ensemble fini $D \subseteq X$ dans $T_\Sigma[X]$ telle que pour tout $x \in D$ et tout $y \in D$, $x \notin \theta(y)$.

Ces définitions ne sont pas toutes équivalentes et H.P. Ko et M.E. Nadel [76] étudient l'influence du choix d'une définition en examinant les propriétés vérifiées par la relation "*est une instance de*" en fonction de la définition utilisée. Ainsi, selon la définition employée, cette relation peut ne pas être transitive ou même ne pas impliquer l'unifiabilité des deux termes qu'elle relie. Ces constatations sont surprenantes et ne correspondent pas toujours aux propriétés de l' "objet informel" que l'on croit manipuler. Les substitutions sont donc des objets qu'il faut définir formellement et manipuler avec précaution.

Evidemment ces problèmes réapparaissent lors de la définition d'un unificateur et de la caractérisation de ses propriétés lors de la conception d'algorithmes d'unification. En 1987, E. Eder [39] consacre un article entier aux propriétés algébriques des substitutions idempotentes permettant d'envisager l'unification d'ensembles de termes. En 1987, dans le même esprit, J.L. Lassez, M.J. Maher et K. Marriott [81] revisitent l'unification en considérant un treillis de contraintes d'égalité et exhibent les différences qui existent entre les définitions d'un unificateur principal que l'on peut rencontrer dans la littérature. Parmi elles, citons les quatre suivantes :

- A.J.Robinson [113] définit un unificateur principal comme étant la substitution produite par l'algorithme présenté dans [113].
- A.J.Robinson [114] définit un unificateur principal comme une substitution $\theta$ telle que pour tout autre unificateur $\sigma$, $\sigma = \sigma\theta$.
- J.W.Lloyd [84] définit un unificateur principal comme une substitution $\theta$ telle que pour tout autre unificateur $\sigma$, il existe une substitution $\mu$ telle que $\sigma = \mu\theta$.
- L.Sterling et E.Y.Shapiro [120] definissent un unificateur principal relativement à un ordre partiel sur les termes comme étant la substitution qui transforme deux termes en leur instance commune la plus générale.

Comme nous le verrons dans la section B.2, ici encore, le choix d'une définition n'est pas sans conséquence. Certaines erreurs peuvent en effet apparaître lorsque l'on "combine" des résultats sur l'unfication qui ont été établis de manière indépendante et qui reposent sur des définitions non équivalentes.

La formalisation de la preuve du théorème de décidabilité de l'unification des termes du premier ordre permet donc d'expliciter complètement les notions sur lesquelles elle repose. Enfin, j'ai obtenu cette preuve par réutilisation de la preuve de cette propriété sur un ensemble plus large : l'ensemble des quasi-termes. Ce développement a donc aussi permis de dégager une méthodologie de réutilisation de preuves formelles présentée dans la section C.1.1.

## B.2 Sémantique de la programmation logique

Définir formellement la sémantique d'un langage de programmation et établir les propriétés que garantit cette sémantique offre, d'une part à celui qui programme dans ce langage de disposer d'une sémantique claire, précise et non ambiguë, et, d'autre part, à celui qui conçoit un interpréteur ou un compilateur pour ce langage d'en garantir la correction. Cette section présente les formalisations que j'ai effectuées pour étudier la sémantique de la programmation logique.

Issue de la recherche en démonstration automatique en logique du premier ordre, et basée sur le principe de résolution de A.J. Robinson [113], la programmation logique a été introduite dans la pratique par A. Colmerauer (langage PROLOG, 1972) [25] tandis que ses fondements théoriques ont été établis par R.A. Kowalski et M.H. van Emden [124]. Ce paradigme de programmation est né de la découverte d'un sous-ensemble de la logique du premier ordre (aussi appelé le fragment Hornien de la logique des prédicats) associé à une interprétation opérationnelle correcte et complète, basée sur la SLD-résolution (*Selection Linear Definite*).

Un programme logique est un ensemble fini de clauses, notées $A \leftarrow B_1, \cdots, B_n$ où $A$ et $B_i$ $(1 \leq i \leq n)$ désignent des atomes (de la forme $p(t_1, \cdots, t_k)$ où $p$ est un symbole de prédicat et $t_1$, ..., $t_n$ sont des termes). Plusieurs lectures d'un programme logique sont possibles, donnant lieu à différentes sémantiques. Traditionnellement, on distingue la sémantique déclarative d'un programme de sa sémantique opérationnelle. D'un point de vue déclaratif, la clause $A \leftarrow B_1, \cdots, B_n$ peut s'interpréter par "*si* $B_1$ *et* $\cdots$ *et* $B_n$ *sont vrais, alors* $A$ *est vrai*". Dans ce cas, une clause spécifie une relation logique entre formules atomiques exprimée par la formule quantifiée universellement $\forall \vec{x}(A \vee \neg B_1 \vee \cdots \vee \neg B_n)$ (ou de manière logiquement équivalente par $\forall \vec{x}(B_1 \wedge \cdots \wedge B_n \Rightarrow A)$). Une clause négative (aussi appelée but) est une clause de la forme $\leftarrow B_1, \cdots, B_q$ dont la signification logique est donnée par la formule $\forall \vec{x}(\neg B_1 \vee \cdots \vee \neg B_q)$ (ou de manière logiquement équivalente par $\neg \exists \vec{x}(B_1 \wedge \cdots \wedge B_q)$). L'exécution d'un programme $P$ à partir d'un but $\leftarrow B_1, \cdots, B_q$ consiste à réfuter l'ensemble $P \cup \{\forall \vec{x}(\neg B_1 \vee \cdots \vee \neg B_q)\}$, ce qui revient, d'un point de vue déclaratif, à prouver à partir de $P$, la formule existentielle $\exists \vec{x}(B_1 \wedge \cdots \wedge B_q)$, appelée requête. Cette preuve est obtenue de manière constructive : un résultat est extrait de la preuve de $\exists \vec{x}(B_1 \wedge \cdots \wedge B_q)$ et correspond à une affectation des variables de $\vec{x}$. Il s'agit d'une substitution $\theta$, appelée solution, telle que $P \models \forall \vec{z}\theta(B_1 \wedge \cdots \wedge B_q)$ où $\models$ dénote la relation de conséquence sémantique. Cette lecture logique des programmes ne fournit aucune information quant à la manière dont le programme, en tant que processus de preuve, s'exécute.

D'un point de vue opérationnel, la clause $A \leftarrow B_1, \cdots, B_n$ peut se comprendre comme une procédure $A$ dont l'appel engendrerait l'exécution des $B_i$. Cette interprétation opérationnelle des programmes est définie par la SLD-résolution. Cette règle permet de déduire une clause à partir d'une clause et d'un but :

$$\text{(SLD-résolution)}\frac{A \leftarrow A_1, \cdots, A_n \qquad \leftarrow B_1, \cdots, B_q}{\leftarrow \theta(B_1, \cdots, B_{i-1}, A_1, \cdots, A_n, B_{i+1}, \cdots, B_q)}$$

où $\theta$ est un unificateur principal du $i$-ième atome $B_i$ $(1 \leq i \leq q)$ du but et de $A$. Cette

règle peut être vue comme une relation de transition notée :

$$\leftarrow B_1, \cdots, B_q \quad \stackrel{C,\theta}{\rightarrow} \quad \leftarrow \theta(B_1, \cdots, B_{i-1}, A_1, \cdots, A_n, B_{i+1}, \cdots, B_q)$$

où $C$ est la clause $A \leftarrow A_1, \cdots, A_n$. $C$ est donc utilisée avec le but $\leftarrow B_1, \cdots, B_q$ pour déduire $\leftarrow \theta(B_1, \cdots, B_{i-1}, A_1, \cdots, A_n, B_{i+1}, \cdots, B_q)$ à partir de la règle de SLD-résolution avec l'unificateur principal $\theta$. L'exécution d'un programme logique $P$ à partir d'un but $R_0$ est décrite par une dérivation, définie comme une succession de transitions :

$$R_0 \stackrel{C_0,\theta_0}{\rightarrow} R_1 \stackrel{C_1,\theta_1}{\rightarrow} \cdots R_{i-1} \stackrel{C_{i-1},\theta_{i-1}}{\rightarrow} R_i \stackrel{C_i,\theta_i}{\rightarrow} \cdots$$

où $C_0$, $C_1$, ... sont des variantes de clauses de $P$ (i.e. des clauses de $P$ renommées). La substitution construite par cette dérivation, appelée réponse, est la composition des substitutions calculées à chaque étape. Un aspect fondamental de la programmation logique est la correspondance complète entre ces deux lectures sémantiques. Cette correspondance se révèle aussi utile d'un point de vue plus pratique : elle permet de comprendre le résultat de l'exécution d'un programme logique à partir d'une requête sans en connaître nécessairement le mécanisme d'exécution. Signalons ici qu'il existe bien d'autres lectures de la programmation logique : réécriture, résolution d'équations, preuve de théorème, réseau de processus communicants ... D'autre part, de nombreuses extensions ont été proposées afin d'augmenter la puissance d'expression de ce modèle de programmation : négation, contraintes, concurrence, ordre supérieur ...

### B.2.1 Plongement profond : Formaliser pour corriger

#### Substitutions et unificateurs

La sémantique de la programmation logique s'exprime à l'aide d'objets "bien connus" : substitutions, unificateurs, conditions de renommage. S'agissant d'objets à l' "allure mathématique", ici encore, on peut s'interroger sur l'utilité de la formalisation de la théorie de la programmation logique. Toutefois, le rôle central que joue la notion de substitution dans cette théorie et l'existence de plusieurs définitions non équivalentes pour cette notion et celle d'unificateur principal a bien entendu des répercussions dans le domaine de la programmation logique. Par exemple, le recensement des différentes définitions des substitutions a permis à H.P. Ko et M.E. Nadel [76] de corriger certaines erreurs dans la preuve du lemme de généralisation présentée dans [84]. D'ailleurs, afin d'éviter les problèmes liés à l'utilisation des substitutions lors de la définition de la sémantique des programmes logiques, R.S. Kemp et G.A. Ringwood [74] préfèrent baser directement cette sémantique sur la relation "*est une instance de*". Dans le même esprit, C. Palamidessi [100] définit une algèbre des subtitutions idempotentes, proche de celle proposée par E. Eder dans [39], munie d'opérateurs satisfaisant de "bonnes" propriétés permettant d'envisager la définition d'une sémantique déclarative compositionnelle et la parallélisation du processus d'exécution des programmes logiques.

Le développement que j'ai effectué relève donc d'une démarche minutieuse dont l'objectif est de *détailler* tous les mécanismes des preuves de la théorie classique de la programmation logique en mettant en évidence le rôle crucial qu'occupent certains *détails* généralement ignorés. Cette formalisation correspond donc à un souci de corriger et/ou de

préciser des résultats standards incorrectement formulés et/ou prouvés. Un des exemples les plus célèbres d'énoncé inexact concerne une propriété fondamentale de la programmation logique. Il s'agit du théorème de complétude de la SLD-résolution, énoncé par J.W. Lloyd [84], dans un ouvrage qui fait référence en la matière, sous la forme suivante :

> (Completeness [84]) *Let $P$ be a definite program and $G$ a definite goal. For every correct answer $\theta$ for $P \cup \{G\}$, there exists a computed answer $\sigma$ for $P \cup \{G\}$ and a substitution $\gamma$ such that $\theta = \gamma\sigma$.*

où la notion de réponse correcte correspond à celle de solution, introduite précédemment. En 1994, J.C. Shepherdson invalide cet énoncé [119] et présente le contre-exemple suivant. Si $G$ est le but $\leftarrow p(x)$ et $P$ le programme contenant l'unique clause $p(f(y,z)) \leftarrow$, alors pour une constante $a$ du langage, la substitution[1] $\theta = \{x/f(a,a)\}$ est solution. Les variantes de la clause de $P$, de la forme $p(f(x_1,x_2)) \leftarrow$, s'unifient avec $G$ via l'unificateur le plus général $\{x/f(x_1,x_2)\}$ (lorsque $x_1 \neq x$ et $x_2 \neq x$). Aussi, la réponse calculée par une dérivation à partir de $G$ est de la forme $\sigma = \{x/f(x_1,x_2)\}$. Toutefois, il n'existe pas de substitution $\gamma$ telle que $\theta = \gamma\sigma$. En effet, les variables $x_1$ et $x_2$ devraient appartenir au domaine d'une telle substitution, alors qu'elles n'appartiennent pas au domaine de la substitution $\theta$. Pour obtenir un énoncé correct, il suffit donc de remplacer la conclusion $\theta = \gamma\sigma$ par $\theta G = \gamma\sigma G$.

**Explicitation du renommage**

Ce développement a mis en relief le rôle crucial des mécanismes de renommage, souvent implicites dans la littérature, et sources d'erreurs. Comme pour les définitions des substitutions ou des unificateurs principaux, plusieurs définitions des conditions de renommage des variables des clauses mises en jeu lors de l'exécution des programmes logiques coexistent dans la littérature. Différentes conditions de renommage des clauses lors d'une dérivation :

$$R_0 \overset{C_0,\theta_0}{\rightarrow} R_1 \overset{C_1,\theta_1}{\rightarrow} \cdots R_{i-1} \overset{C_{i-1},\theta_{i-1}}{\rightarrow} R_i \overset{C_i,\theta_i}{\rightarrow} \cdots$$

apparaissent dans la littérature :

- K.R.Apt et M.H.Van Emden [8], R.Lalement [78] et J.L.Lassez et M.J.Maher [82] imposent que chaque clause $C_i$ ne partage pas de variables avec $R_i$ : $var(C_i) \cap var(R_i) = \emptyset$.
- J.W.Lloyd [84] impose que chaque clause $C_i$ ne contienne pas de variables apparaissant déjà dans la dérivation jusqu'à $R_i$.
- U.Nilsson et J.Maluszynski [98] inposent que chaque clause $C_i$ ne partage pas de variables avec $R_i$ et $R_0$ : $var(C_i) \cap (var(R_0) \cup var(R_i)) = \emptyset$.
- K.R.Apt [7] impose que chaque clause $C_i$ ne partage aucune variable avec $R_0$, $C_0$, $C_1$ , ... , $C_{i-1}$ : $var(C_i) \cap (var(R_0) \cup var(C_0) \cup \cdots \cup var(C_{i-1})) = \emptyset$.

Considérons par exemple le programme $P = \{p(f(x)) \leftarrow p(y)\}$. Avec les conditions de renommage issues de [8, 78, 82], on peut construire la dérivation suivante :

$$p(z) \overset{C_1,\{z/f(x)\}}{\longrightarrow} p(y) \overset{C_2,\{y/f(x)\}}{\longrightarrow} p(z) \rightarrow \cdots$$

où $C_1$ est la clause $p(f(x)) \leftarrow p(y)$ et $C_2$ est la clause $p(f(x)) \leftarrow p(z)$. Si l'on considère à présent les conditions de renommage issues de [98], on ne peut plus utiliser la clause $C_2$

[1] Dans tout ce document, les substitutions sont notées $\{x_1/t_1, \cdots, x_n/t_n\}$ et dénotent la fonction qui étant donnée une variable $v$ retourne $t_i$ si $v = x_i$ et retourne $v$ sinon.

lors de la deuxième transition puisque $z \in var(C_2) \cap var(p(z))$. On peut utiliser à la place la clause $p(f(x)) \leftarrow p(w)$, notée $C_2'$, et obtenir la dérivation :

$$p(z) \overset{C_1,\{z/f(x)\}}{\longrightarrow} p(y) \overset{C_2',\{y/f(x)\}}{\longrightarrow} p(w) \rightarrow \cdots$$

Or, en utilisant les conditions de renommage issues de [7], cette clause $C_2'$ ne peut plus être utilisée puisque $x \in var(C_1) \cap var(C_2')$. On peut utiliser à la place la clause $p(f(v)) \leftarrow p(w)$, notée $C_2''$, et obtenir la dérivation :

$$p(z) \overset{C_1,\{z/f(x)\}}{\longrightarrow} p(y) \overset{C_2'',\{y/f(v)\}}{\longrightarrow} p(w) \rightarrow \cdots$$

Si le renommage des clauses permet d'établir les propriétés fondamentales de la SLD-résolution, il est aussi nécessaire pour définir la sémantique opérationnelle des programmes logiques : ce renommage a au moins deux rôles. D'une part, il permet d'éviter un échec du mécanisme d'unification. Par exemple, soit $P$ le programme défini $\{p(f(x)) \leftarrow\}$ et $R$ la requête $p(x)$. Sans un renommage de la clause de $P$ *en dehors* des variables de $R$, l'unification de $p(x)$ avec $p(f(x))$ échoue. En effet, le "test d'occurrence" est une opération fondamentale de l'algorithme d'unification qui élimine les équations de la forme $x = t[x]$ (où $t[x]$ désigne un terme fonctionnel contenant la variable $x$) car elles n'ont pas de solutions sur les termes finis. D'autre part, le mécanisme de renommage garantit l'obtention d'une substitution résultat la plus générale. Par exemple, soit $P$ le programme $\{p(y) \leftarrow\}$ et $R$ la requête $p(x), p(y)$. Supposons que la clause de $P$ ne soit pas renommée et qu'à partir de $R$, le premier atome soit sélectionné. On obtient alors la dérivation :

$$p(x), p(y) \overset{\{x/y\}}{\rightarrow_P} p(y) \rightarrow_P \square$$

dont la réponse est $\rho_1 = \{x/y\}$. Le renommage de la clause de $P$ *en dehors* des variables apparaissant dans $R$ permet d'obtenir une réponse plus générale. En effet, si l'on renomme cette clause en $p(x_1) \leftarrow$ pour la première transition puis en $p(x_2) \leftarrow$ pour la deuxième transition, on obtient la dérivation :

$$p(x), p(y) \overset{\{x/x_1\}}{\rightarrow_P} p(y) \overset{\{y/y_1\}}{\rightarrow_P} \square$$

dont la réponse est $\rho_2 = \{x/x_1, y/y_1\}$ et est plus générale que $\rho_1$.

Prendre en compte de manière explicite la "partie renommage" de la SLD-résolution complique évidemment les preuves des propriétés classiques de la programmation logique. Cependant, le processus de renommage doit être considéré comme une composante à part entière de la sémantique opérationnelle des programmes logiques. C'est lui qui permet d'établir des propriétés sur les variables apparaissant dans une dérivation. Ces propriétés sont essentielles et ne constituent pas un simple *détail* technique comme le montrent les exemples qui suivent.

**Lemme de généralisation** Le lemme de généralisation, préliminaire au théorème de complétude de la SLD-résolution, s'énonce habituellement comme suit :

> (Lifting lemma [7]) *Let $P$ be a program, $N$ a goal and $\theta$ a substitution. Suppose that there exists an SLD-refutation of $P \cup \{\theta N\}$ with the sequence of mgu's $\theta_0, \ldots, \theta_n$. Then, there exists an SLD-refutation of $P \cup \{N\}$ with the sequence of mgu's $\theta_0', \ldots, \theta_n'$ such that $\theta_n' \ldots \theta_0'$ is more general than $\theta_n \ldots \theta_0 \theta$.*

Aucune hypothèse sur la substitution $\theta$ n'est posée dans cet énoncé. Néanmoins, si l'on veut pouvoir utiliser exactement les mêmes variantes de clauses dans les deux dérivations, ce qui est toujours implicitement suggéré, certaines hypothèses sont nécessaires.

En effet, soit $C_1$ la variante de clause utilisée lors de la première étape de résolution de la dérivation en hypothèse du lemme. Tandis que les conditions de renommage de $C_1$ stipulent que $C_1$ ne partage pas de variables avec la requête $\theta N$, il faut, pour pouvoir construire la dérivation en conclusion du lemme, que $C_1$ ne partage pas non plus de variables avec $N$. Or, sans aucune hypothèse supplémentaire, il se peut très bien qu'une variable de $N$ n'apparaisse pas dans $\theta N$ et soit donc présente dans $C_1$. Par exemple, si $P$ est le programme $\{p(f(v), f(z)) \leftarrow\}$, $N$ le but $\leftarrow p(f(a), y)$ et $\theta$ est la substitution $\{y/f(x)\}$, alors à partir du but initial $\theta N = \leftarrow p(f(a), f(x))$, rien n'empêche de renommer la clause de $P$ à l'aide de la substitution de renommage $r = \{z/y\}$ et d'obtenir la dérivation :

$$\underbrace{p(f(a), f(x))}_{\theta N} \overset{\{v/a, x/y\}}{\underset{P}{\rightarrow}} \square$$

Considérons à présent la dérivation que l'on peut obtenir à partir du but $N$ : la clause de $P$ doit être renommée *en dehors* des variables de $N$. La substitution de renommage $r$, utilisée dans la dérivation précédente ne convient donc plus, puisqu'elle renomme $z$ en $y$ qui apparaît dans $N$. En effet, lors de la dérivation à partir de $\theta N$, le renommage de la clause $C$ utilisée satisfait seulement $var(\theta N) \cap var(rC) = \emptyset$ alors que la construction d'une dérivation à partir de $N$ nécessite la condition $var(N) \cap var(rC) = \emptyset$. Aussi, pour pouvoir prouver formellement le lemme de généralisation, il faut supposer que la clause $C$ utilisée est renommée *en dehors* des variables de $N$ à chaque transition.

D'autres précautions à prendre concernent le lien que doit entretenir la substitution $\theta$ avec $N$. En effet, si $\theta$ affecte des variables non présentes dans $N$, quelques problèmes peuvent apparaître. Par exemple, si $P$ est le programme $\{p(f(y), f(z)) \leftarrow\}$, $N$ le but $\leftarrow p(f(a), y)$ et $\theta$ la substitution $\{x/w\}$, alors on peut renommer la clause de $P$ à l'aide de la substitution de renommage $r = \{y/x\}$ et construire les dérivations :

$$\underbrace{p(f(a), y)}_{\theta N} \overset{\rho_1 = \{x/a\,;\, y/f(z)\}}{\underset{P}{\rightarrow}} \square \qquad \underbrace{p(f(a), y)}_{N} \overset{\rho_2 = \{x/a\,;\, y/f(z)\}}{\underset{P}{\rightarrow}} \square$$

Cependant, $\rho_2 = \{x/a\,;\, y/f(z)\} \leq \rho_1\theta = \{x/w\,;\, y/f(z)\}$ n'est pas vérifié. Pour éviter cette situation, il faut donc imposer que les clauses utilisées soient renommées *en dehors* des variables du domaine de $\theta$.

La formalisation que j'ai réalisée avec Coq a permis de mettre en évidence les hypothèses implicites sur le renommage des clauses lors de la dérivation en hypothèse du lemme de généralisation, hypothèses sans lesquelles il était bien sûr impossible de mécaniser la preuve.

**Lemme de commutation** Un problème similaire s'est posé lors de la preuve du lemme de commutation. Ce résultat classique, assurant que le choix de l'atome sélectionné lors d'une étape de résolution relève d'un non-déterminisme par indifférence (*don't care non-determinism*), s'énonce habituellement :

(Switching lemma [78]) *If during a derivation, two atoms $L_1$ and $L_2$ are successively selected, then they can also be selected in the reverse order and the derived states are the same up to renaming of variables.*

$$\begin{array}{ccc} & \underbrace{(\cdots, L_1, \cdots, L_2, \cdots)}_{R_0} & \\ & C_1 \swarrow \quad \searrow C_2 & \\ \underbrace{\theta(\cdots, C_1^-, \cdots, L_2, \cdots)}_{R_1} & & \sigma(\cdots, L_1, \cdots, C_2^-, \cdots) \\ C_2 \downarrow & & \downarrow C_1 \\ \eta\theta(\cdots, C_1^-, \cdots, C_2^-, \cdots) & \approx & \mu\sigma(\cdots, C_1^-, \cdots, C_2^-, \cdots) \end{array}$$

Ici encore, quelques précautions sont nécessaires si l'on veut pouvoir utiliser les mêmes variantes des clauses $C_1$ et $C_2$ dans les deux dérivations (ce qui est habituellement suggéré dans la preuve de ce lemme). En effet, le renommage des clauses $C_1$ et $C_2$ dans la dérivation en hypothèse du lemme garantit normalement que $var(C_1) \cap var(R_0) = \emptyset$ et $var(C_2) \cap var(R_1) = \emptyset$. De plus les variables présentes dans la requête $R_1$ proviennent soit de la clause utilisée, soit de la requête $R_0$, et on a :

$$\forall x \quad x \in var(R_1) \Rightarrow (x \in var(C_1) \vee x \in var(R_0)) \tag{B.1}$$

Cependant, si l'on veut pouvoir sélectionner l'atome $L_2$, avant $L_1$, en utilisant la même variante de la clause $C_2$, il faut que cette variante soit telle que $var(C_2) \cap var(R_0) = \emptyset$. Or rien dans l'hypothèse du lemme de commutation ne garantit une telle propriété, et selon les conditions de renommage imposées dans la définition d'une dérivation, il est tout à fait possible qu'une variable présente dans la requête $R_0$, mais n'apparaissant pas dans $R_1$, ait été utilisée dans la variante de $C_2$. Pour contourner ce problème, et d'après (B.1), il faut que le renommage des clauses soit tel que $var(C_2) \cap var(C_1) = \emptyset$. C'est heureusement le cas si l'on utilise les hypothèses de renommage formulées dans [7] (i.e. du fait du renommage de la clause $C_i$ *en dehors* des clauses $C_0, C_1, \cdots, C_{i-1}$) mais ce n'est plus le cas si l'on utilise les conditions énoncées dans [8, 78, 82, 84, 98] qui ne permettent donc pas de prouver formellement le lemme de commutation. Ici encore, c'est le travail de mécanisation des preuves qui a permis de mettre en évidence les propriétés de renommage requises pour obtenir la preuve avec Coq.

**Renommer une dérivation** Formaliser une preuve nous conduit donc à l'établir à un niveau de *détail* supérieur à celui généralement exposé. Dans le domaine de la programmation logique, un exemple typique de *détail* concerne le renommage de la clause utilisée lors d'une étape de résolution. La raison pour laquelle il est fréquent et admis de s'affranchir de ces *détails* est l'existence d'un lemme dû à J.W. Lloyd et J.C. Shepherdson [85], affirmant que si deux dérivations diffèrent seulement dans le choix des variables utilisées pour renommer les clauses (et par conséquent dans les unificateurs utilisés), alors les états dérivés sont équivalents à un renommage près des variables.

(Uniqueness) *If two SLDNF-derivations differ only in the variants of clauses and the mgu which are used, then the resultants are variants of each other.*

Autrement dit, l'existence d'une dérivation ne dépend pas du choix des variables de renommage : il suffit que de "bonnes" conditions de séparation des variables soit satisfaites.

C'est précisément ce résultat qui permet de supposer à tout moment des hypothèses sur les variables présentes dans une clause. Ainsi, la plupart du temps, ce renommage est fait implicitement et afin d'alléger les théorèmes et leurs preuves, on passe sous silence cette partie du calcul : on parle de variables "fraîches" et les théorèmes sont énoncés "à un renommage près". Toutefois, pour pouvoir supposer telle ou telle hypothèse, il est nécessaire d'appliquer ce lemme en instanciant les ensembles de variables mis en jeu. L'adaptation de ce lemme d'indépendance dans le Calcul des Constructions a donc conduit à expliciter comment, à partir d'une requête $R$ admettant une dérivation $d_1$ et d'un ensemble fini de variables $Z$, on peut construire une dérivation $d_2$, à partir de la même requête, n'utilisant pour renommer les clauses que des variables appartenant à un ensemble $Y$ tel que $Z$ et $Y$ soient disjoints. Cette construction passe bien sûr par l'explicitation des substitutions de renommage et des unificateurs utilisés et peut être vue comme une opération de renommage portant sur une dérivation.

$$\boxed{\underbrace{R \overset{*}{\to} R_1}_{d_1} \quad Z} \quad \underset{(\Rightarrow \exists)}{\to} \quad \boxed{\underbrace{R \overset{(Y)*}{\to} R_2}_{d_2} \quad Z \cap Y = \emptyset \quad R_1 \approx R_2}$$

Ce résultat est absolument indispensable si l'on veut pouvoir formaliser la preuve du théorème de complétude.

**Combiner des dérivations** En effet, la preuve du théorème de complétude s'obtient par induction sur la longueur de la requête initiale et une étape de cette preuve consiste à affirmer que si chaque atome d'un ensemble d'atomes ne contenant pas de variables admet une réfutation, alors en combinant ces réfutations, on peut obtenir une réfutation à partir de la requête constituée de tous les atomes présents dans cet ensemble :

> [84] (...) *by induction hypothesis,* $P \cup \{\theta B_i\}$ *has a refutation for* $i = 1, \ldots, k$. *Because each* $\theta B_i$ *is ground, these refutations can be combined into a refutation of* $P \cup \{\theta B_1, \ldots \theta B_k\}$ (...)

Cependant combiner des dérivations est une opération délicate, puisque, afin que la dérivation à construire satisfasse à chaque étape les hypothèses de séparation des variables, il est nécessaire de pouvoir renommer chacune des réfutations initiales. Ici encore, il s'agit d'expliciter dans Coq comment on peut construire la dérivation finale en exhibant les substitutions de renommage et les unificateurs qui vont permettre d'obtenir une telle dérivation.

$$\boxed{\begin{array}{c} \theta B_1 \overset{*}{\to} \square \\ \theta B_2 \overset{*}{\to} \square \\ \vdots \\ \theta B_k \overset{*}{\to} \square \end{array}} \quad \underset{(\Rightarrow \exists)}{\to} \quad \boxed{\theta B_1, \theta B_2, \cdots, \theta B_k \overset{*}{\to} \square}$$

Ce résultat s'obtient par induction sur le nombre de réfutations initiales et sa preuve utilise constamment le lemme de "renommage" des dérivations présenté dans le paragraphe précédent.

Aussi, si la formalisation effectuée ne conduit à aucun résultat nouveau, elle permet toutefois d'éclairer toute une partie du calcul effectué par un programme logique : le renommage des variables. La formalisation de ce processus dans le Calcul des Constructions permet de garantir la validité des résultats prouvés, en rendant explicites toutes les étapes de renommage à faire dans les preuves et dans les implantations.

### B.2.2 Plongement superficiel : Formaliser pour mieux comprendre

#### Motivations

Les résultats classiques de la programmation logique formalisés dans le système Coq qui viennent d'être présentés concernent les calculs finis (i.e., définissent une sémantique pour les SLD-réfutations). En effet, il existe une tradition en informatique qui veut que tout "bon" programme soit un programme dont l'exécution termine. La programmation logique n'échappe pas à cette tradition et les propriétés de terminaison des programmes logiques ont fait l'objet de nombreux travaux. Toutefois, certains programmes ont "naturellement" vocation à donner lieu à une exécution infinie et l'étude de la sémantique des "calculs" infinis est désormais envisagée dans divers paradigmes ($\lambda$-calcul, systèmes de réécriture, programmation logique, programmation concurrente par contraintes, ...) et permet d'aborder des notions comme la concurrence, la réactivité ou encore la programmation synchrone. En effet, certains objets sont, par nature, infinis et les programmes qui permettent de les construire, même s'ils ne terminent pas, effectuent bien un calcul "utile en un certain sens". Par exemple, dans le domaine de la programmation logique, le programme permettant de construire la liste (infinie) des entiers naturels consécutifs à partir d'un certain rang s'écrit :

$$P = \{\mathsf{LN}(x, [x|l]) \leftarrow \mathsf{LN}(S(x), l)\} \tag{B.2}$$

puisque pour tout $k \in \mathbb{N}$, une dérivation infinie à partir de la requête $\mathsf{LN}(k, l)$ fournit à chaque étape $i$ une approximation $[k, S(k), ..., S^{i-1}(k)|l_i]$ de la liste des entiers naturels consécutifs à partir de $k$ :

$$\begin{array}{ll}
\mathsf{LN}(k, l) & \\
\quad\downarrow & \theta_1 = \{x_1/k, l/[k|l_1]\} \\
\mathsf{LN}(S(k), l_1) & \\
\quad\downarrow & \theta_2 = \{x_2/S(k), l_1/[S(k)|l_2]\} \\
\mathsf{LN}(S^2(k), l_2) & \\
\quad\downarrow & \\
\quad\vdots & \\
\quad\downarrow & \\
\mathsf{LN}(S^{i-1}(k), l_{i-1}) & \\
\quad\downarrow & \theta_i = \{x_i/S^{i-1}(k), l_{i-1}/[S^{i-1}(k)|l_i]\} \\
\mathsf{LN}(S^i(k), l_i) & \\
\quad\downarrow & \\
\quad\vdots &
\end{array} \tag{B.3}$$

La limite de la suite correspondant aux approximations successives calculées à chaque étape de cette dérivation est la liste infinie des entiers consécutifs à partir de $k$. Cependant, toutes les dérivations infinies ne correspondent pas nécessairement à la construction d'un objet infini : certaines dérivations, appelées ici *dérivations infinies sur le domaine des termes finis*, sont infinies et ne "calculent à la limite" que des termes finis. De telles dérivations ne sont pas pour autant dénuées d'une sémantique et ne méritent pas, comme c'est l'usage, d'être reléguées dans une classe de dérivations qui seraient engendrées à partir de "mauvais programmes" : par exemple, il est possible d'écrire un programme logique donnant lieu à des dérivations infinies qui décrivent des séquences infinies d' "états" qui modélisent certains processus (par exemple, le célèbre problème du dîner des philosophes, ou plus généralement certains problèmes dont le graphe d'états est soit infini soit cyclique).

En programmation logique, il est courant de comprendre les dérivations comme des "preuves qui calculent" et c'est bien sûr l'aspect calculatoire de telles preuves qui permet l'utilisation de programmes logiques dans la résolution de problèmes. C'est aussi cet aspect calculatoire qui est considéré dans la littérature consacrée aux dérivations infinies [2, 3, 45, 52, 58, 84] : une dérivation infinie est alors vue comme un processus de calcul à l'infini d'un objet infini. En effet, les sémantiques proposées, qui reposent, pour la plupart, sur la notion de plus grand point fixe, cherchent toutes à capturer la notion d' "atome (infini) calculable à l'infini". Dans les deux principales approches existantes, l'univers du discours est complété de manière à disposer de termes infinis : dans [2, 3, 58, 84], une notion de distance entre termes permet de définir une complétion métrique de l'univers de Herbrand – dans [45], c'est une complétion par idéaux qui est envisagée. Hélas, ces approches sont toutes incomplètes (i.e., il existe des atomes infinis dans la dénotation de certains programmes qui ne sont pas "calculables à l'infini") : la complétude de l'approche métrique ne s'obtient qu'en rendant possible la présence de termes infinis dans les requêtes[2] puisqu'elle est alors exprimée par[3] :

$$\begin{array}{rl} & A \text{ (atome éventuellement infini)} \in \mathsf{gfp}(T_P) \\ \Leftrightarrow & \text{il existe une dérivation équitable à partir de } A \end{array}$$

tandis que l'approche par idéaux, outre le fait qu'elle restreint la classe des programmes envisagés, donne seulement une sémantique pour une classe de dérivations infinies caractérisée en termes d'objets minimaux. Il semble qu'un des obstacles à la complétude de ces approches provienne de la difficulté à prendre en compte les dérivations infinies sur le domaine des termes finis : la construction d'un plus grand point fixe ne reflète pas la manière avec laquelle les termes infinis sont construits lors d'une dérivation infinie. Par exemple, si l'on considère l'approche métrique, la dénotation du programme $P = \{p(x) \leftarrow p(x)\}$ contient $p(f^\omega)$ même si le terme $f^\omega$ ne peut être construit dans une dérivation infinie (d'où la nécessité d'autoriser la présence de termes infinis dans les requêtes pour obtenir la complétude).

[2] Cette condition, explicitement mentionnée dans [104], ne correspond pas à la sémantique opérationnelle "standard". D'autre part, la présence de termes infinis dans les requêtes nécessite d'utiliser une version modifiée de l'algorithme d'unification. Enfin, le problème de la représentation des termes infinis dans le langage (fini) des requêtes se pose, même s'il peut dans certains cas être résolu en adoptant une représentation implicite (par exemple avec des contraintes) des termes infinis.

[3] $T_P$ est l'opérateur de conséquence immédiate associé au programme $P$ et $\mathsf{gfp}(T_P)$ dénote le plus grand point fixe de cet opérateur.

La définition d'une sémantique par plus grand point fixe revient à identifier programmes logiques et définitions co-inductives et, puisque les dérivations sont avant tout des preuves, il semble pertinent d'envisager cette correspondance plus en profondeur en identifiant les dérivations (finies et infinies) et les termes du Calcul des Constructions (de type inductif et co-inductif) : il s'agit là d'un plongement superficiel des dérivations de la programmation logique dans le Calcul des Constructions. Pour ce faire, les clauses vont être considérées comme des signatures fonctionnelles de constructeurs de preuves. Cette comparaison permet de mettre en évidence les phénomènes d'incomplétude observés dans les approches existantes et montre que pour la classe des dérivations infinies sur le domaine des termes finis, il existe une correspondance directe entre dérivations (SLD) et preuves (par co-induction). Dans cette section, nous nous intéressons donc à la cohérence interne entre deux théories (programmation logique et définitions co-inductives).

### SLD-résolution et (Co-)induction

Rappelons qu'un ensemble peut être défini (co-)inductivement à l'aide d'un ensemble $\Phi$ de *constructeurs* de la forme $e \leftarrow E$ où $e$ est l'élément construit à partir d'un ensemble d'objets $E$. Dans le cas d'une *définition inductive*, l'ensemble ainsi défini, noté $\mathsf{Ind}(\Phi)$, correspond à l'intersection de tous les ensembles $\Phi$-*clos* (i.e., les ensembles $A$ vérifiant pour chaque constructeur $e \leftarrow E$ de $\Phi$, $E \subseteq A \Rightarrow e \in A$) et contient tous les éléments obtenus en appliquant un nombre fini de fois les constructeurs. Dans le cas d'une *définition co-inductive*, l'ensemble défini, noté $\mathsf{CoInd}(\Phi)$, correspond à l'union de tous les ensembles $\Phi$-*denses* (i.e., les ensembles $A$ tels que pour tout $a \in A$ il existe un ensemble $E \subseteq A$ tel que $a \leftarrow E$ est un constructeur de $\Phi$) et contient tous les éléments obtenus en appliquant un nombre de fois éventuellement infini les constructeurs. De manière équivalente, ces ensembles peuvent être définis à partir d'un opérateur monotone $T_\Phi$ défini à partir de $\Phi$ par :

$$\mathcal{B} = \bigcup_{e \leftarrow E \in \Phi} \{\{e\} \cup E\} \quad T_\Phi(A) = \{e \in \mathcal{B} \mid \exists\, e \leftarrow E \in \Phi \quad E \subseteq A\} \tag{B.4}$$

On a alors $\mathsf{Ind}(\Phi) = \cap_{T_\Phi(A) \subseteq A} A$ et $\mathsf{CoInd}(\Phi) = \cup_{A \subseteq T_\Phi(A)} A$. Ces deux ensembles correspondent respectivement au plus petit point fixe ($\mathsf{lfp}(T_\Phi)$) et au plus grand point fixe ($\mathsf{gfp}(T_\Phi)$) de $T_\Phi$.

Définitions inductives et programmes logiques présentent de nombreuses similitudes. Un programme défini $P$ peut être vu comme un ensemble de constructeurs $[P]$ correspondant aux instances fermées (i.e., sans variables) des clauses de $P$. L'opérateur $T_{[P]}$ associé à cet ensemble de constructeurs, comme indiqué en (B.4), correspond exactement à l'opérateur de "conséquence immédiate" (traditionnellement noté $T_P$). D'autre part, une interprétation de Herbrand $I$ est un modèle de $P$ ssi $T_P(I) \subseteq I$, ou en d'autres termes, ssi $I$ est $T_{[P]}$-clos. Enfin, le plus petit modèle de Herbrand $\mathcal{M}_P$ est défini comme l'intersection de tous les modèles de Herbrand ce qui, en termes de définitions inductives, revient à définir la dénotation de $P$ par l'ensemble $\mathsf{Ind}([P]) = \mathsf{lfp}(T_{[P]})$[4]. Aussi, plutôt qu'un simple outil technique, les définitions inductives constituent une alternative naturelle au paradigme "*logic programs as first-order theories*". C'est cette idée qui est défendue par G.Huet [57] et L.C.Paulson et

[4]De plus, on a $\mathsf{lfp}(T_{[P]}) = T_{[P]}^{\uparrow\omega}$ car le corps de chaque clause est fini, et donc, en termes de définitions inductives, $[P]$ est finitaire.

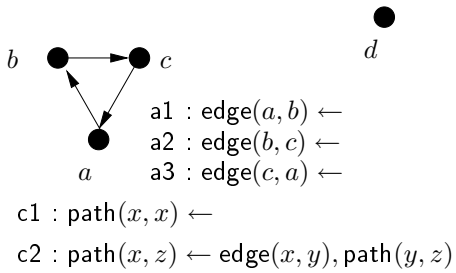


FIG. B.1 – Chemin entre deux sommets d'un graphe orienté

A.W.Smith [103, 57] et utilisée avec profit par P.Deransart et J.Maluszynski [35]. Cette lecture sémantique de la programmation logique permet de considérer un programme défini, non plus comme une théorie du premier ordre, mais comme la définition d'une "nouvelle logique". En adoptant ce point de vue, la dénotation d'un programme correspond à l'ensemble des théorèmes qu'il est possible de dériver à partir de cette logique. Cette approche conduit donc à identifier un programme à une définition inductive ou co-inductive, selon que l'on considère ou non les dérivations infinies. Malheureusement, cette identification n'est plus aussi fructueuse lors de la définition d'une sémantique pour les dérivations infinies. En effet, envisageons à présent plus en profondeur le paradigme "*logic programs as co-inductive definitions*". Pour cela, considérons les clauses $C$ de la forme $A \leftarrow A_1, \cdots, A_n$, en suivant l'isomorphisme de Curry-Howard, comme des signatures fonctionnelles de constructeurs de preuve (qui associent à partir des preuves $\pi_{A_i}$ des $A_i$ une preuve $\pi_A$ de $A$, $A$ correspond alors au type du $\lambda$-terme $\pi_A$) et comparons les preuves représentées par les dérivations SLD avec celles représentées par des $\lambda$-termes. Avant de mener cette comparaison, rappelons qu'un $\lambda$-terme de preuve établissant une proposition $\phi$ peut être défini de manière récursive, et donc utiliser récursivement la preuve de $\phi$, sous certaines conditions. Il s'agit de la notion de "preuve gardée" introduite par T. Coquand dans le contexte de la théorie des types :

> [26] *In order to establish that a proposition $\phi$ follows from other propositions $\phi_1, \cdots, \phi_q$, it is enough to build a proof term e for it, using not only natural deduction, case analysis, and already proven lemmas, but also using the proposition we want to prove recursively, provided such a recursive call is guarded by introduction rules.*

Cette condition syntaxique garantit le caractère "productif" d'une classe de $\lambda$-termes. Puisqu'à chaque transition d'une dérivation, une clause (i.e., un constructeur) est appliquée, les dérivations peuvent être vues comme des preuves "gardées". Voyons cela sur l'exemple typique du programme de la figure B.1 permettant de caractériser les chemins d'un graphe orienté. L'existence d'un cycle dans ce graphe rend possible la construction d'une dérivation infinie à partir de la requête $\mathsf{path}(s_1, s_2)$ où $s_1$ est un sommet apparaissant sur un cycle et $s_2$ est un sommet quelconque. Par exemple, on a :

$$\begin{aligned} \mathsf{path}(a,x) &\xrightarrow{\mathsf{c2}} \mathsf{edge}(a,y_1), \mathsf{path}(y_1,x) \xrightarrow{\mathsf{a1}} \mathsf{path}(b,x) \\ &\xrightarrow{\mathsf{c2}} \mathsf{edge}(b,y_2), \mathsf{path}(y_2,x) \xrightarrow{\mathsf{a2}} \mathsf{path}(c,x) \\ &\xrightarrow{\mathsf{c2}} \mathsf{edge}(c,y_3), \mathsf{path}(y_3,x) \xrightarrow{\mathsf{a3}} \mathsf{path}(a,x) \rightarrow \cdots \end{aligned} \tag{B.5}$$

Il est possible de construire le $\lambda$-terme $\pi_{ax}$ de type $\forall x\, \mathsf{path}(a,x)$ (qui vérifie bien la condition de garde puisque l'appel récursif à $\pi_{ax}$ est bien "gardé" par $\mathsf{c2}$) correspondant à la dérivation (B.5) :

$$\pi_{ax} := \lambda x.\, \mathsf{c2}(a,b,x,\mathsf{a1},\mathsf{c2}(b,c,x,\mathsf{a2},\mathsf{c2}(c,a,x,\mathsf{a3},\pi_{ax}(x))))$$

Les six premières transitions de la dérivation correspondent aux six applications des constructeurs, tandis que l'appel récursif correspond à la suite de la dérivation[5]. Un tel $\lambda$-terme peut être vu comme la définition récursive de la suite des clauses appliquées dans la dérivation et *vice versa* (néanmoins, il existe des dérivations infinies pour lesquelles cette suite de clauses n'est pas toujours calculable). Il existe donc bien une correspondance directe entre dérivations et $\lambda$-termes lorqu'aucun terme infini apparaît : les dérivations infinies qui ne construisent aucun terme infini correspondent à des preuves par co-induction. En présence de termes infinis, cette correspondance n'est plus immédiate : reprenons l'exemple du programme (B.2), vu comme une définition co-inductive. Il est alors possible de prouver, par co-induction, que la liste, notée $\mathsf{from}(k)$, des entiers consécutifs à partir de $k$ vérifie bien $\mathsf{LN}(k,\mathsf{from}(k))$. Or, pour construire le $\lambda$-terme correspondant, il est nécessaire de disposer de cette liste définie[6] par :

$$\mathsf{from} := \lambda n.\mathsf{cons}(n,\mathsf{from}(S(n)))$$

Le $\lambda$-terme s'écrit alors :

$$\begin{array}{rl} \pi_{LN} := \lambda n.\mathsf{eq\_ind}( & \mathsf{cons}(n,\mathsf{from}(S(n))),\\ & \lambda u.\mathsf{LN}(n,u),\\ & \mathsf{C}(n,\mathsf{from}(S(n)),\pi_{LN}(S(n))),\\ & \mathsf{from}(n),\\ & \mathsf{from\_unfold}(n)) \end{array}$$

où $\mathsf{from\_unfold}$ est la preuve de la proposition :

$$\forall n\ \mathsf{from}(n) = \mathsf{cons}(n,\mathsf{from}(S(n)))$$

et où $\mathsf{eq\_ind}$ correspond au schéma d'élimination de l'égalité (i.e., égalité *à la* Leibnitz) et associe une preuve de $P(y)$ à partir d'un élément $x$, d'un prédicat $P$, d'une preuve de $P(x)$, d'un élément $y$ et d'une preuve de $x = y$. La définition de ce terme est bien gardée puisque l'appel récursif à $\pi_{LN}$ est protégé par le constructeur $\mathsf{C}$. On voit donc bien sur cet exemple que le $\lambda$-terme ne reflète pas le processus de construction de la liste infinie (dérivation (B.3)) : prouver par co-induction $\mathsf{LN}(k,\mathsf{from}(k))$ revient à appliquer une infinité de fois le constructeur de preuve $\mathsf{C}$ en utilisant à chaque appel récursif la propriété $\mathsf{from\_unfold}$.

[5] Un phénomène "rassurant" peut cependant être observé si l'on rajoute un argument au prédicat $\mathsf{path}$ correspondant à la longueur du chemin qui sépare les deux sommets considérés : dans ce cas, la longueur du chemin séparant le sommet $a$ du sommet $d$ est $S^\omega$ (la longueur du chemin considéré correspond à la longueur de la preuve de l'existence d'un chemin).

[6] La définition de $\mathsf{from}$ est gardée puisque $\mathsf{from}$ est protégé par le constructeur $\mathsf{cons}$. L'utilisation de termes infinis dans l'univers du discours revient à considérer une définition co-inductive pour les termes (les symboles de fonction sont les constructeurs).

**Sémantique complète pour les dérivations infinies sur un domaine de termes finis**

La lecture sémantique de la programmation logique présentée ci-dessus et dans [62, 64, 63, 65] m'a permis de définir une sémantique valide et complète pour les dérivations infinies qui ne construisent pas de termes infinis. Plus formellement, ces dérivations sont définies comme suit.

**Définition B.1** *Une SLD-preuve sur le domaine des termes finis est soit une réfutation, soit une dérivation infinie équitable*[7] *:* $R_0 \overset{C_1,\theta_1}{\rightarrow} R_1 \rightarrow \cdots \rightarrow R_{i-1} \overset{C_i,\theta_i}{\rightarrow} R_i \rightarrow \cdots$ *telle que* $\forall k \geq 0 \quad \exists p > k \quad \forall q \geq p \quad \theta_q \cdots \theta_p \cdots \theta_{k+1} R_k \approx \theta_p \cdots \theta_{k+1} R_k$.

où $\approx$ dénote la relation d'équivalence au renommage près des variables. L'information calculée par ces dérivations doit donc l'être en un nombre fini d'étapes (i.e., pour chaque requête $R$ apparaissant dans la dérivation, il existe un certain rang à partir duquel la suite des instanciations de $R$ par les unificateurs est stationnaire).

Les propriétés de cette classe de dérivations s'établissent en identifiant un programme $P$ par l'ensemble de constructeurs défini par :

$$\lceil P \rceil = \{\theta C \mid C \in P \,,\, \theta : X \rightarrow T_\Sigma[X]\}$$

qui correspondent aux instances (pouvant contenir des variables) des clauses du programme. Les résultats de validité et de complétude prouvés s'énoncent alors comme suit.

**Proposition B.1**

- *(Validité). Si il existe une SLD-preuve sur le domaine des termes finis :*

$$A_1, \cdots, A_n \overset{C_1,\theta_1}{\rightarrow} R_1 \rightarrow \cdots \rightarrow R_{i-1} \overset{C_i,\theta_i}{\rightarrow} R_i \rightarrow \cdots$$

  *à partir du programme $P$, alors il existe $k \geq 0$ tel que pour tout $i$ $(1 \leq i \leq n)$ $\theta_k \cdots \theta_1 A_i \in \mathsf{Coln d}(\lceil P \rceil)$.*
- *(Complétude). Etant donné un programme $P$, si $A \in \mathsf{Coln d}(\lceil P \rceil)$, alors il existe une SLD-preuve sur le domaine des termes finis à partir de $A$ avec le programme $P$ :*

$$A \overset{C_1,\sigma_1}{\rightarrow} R'_1 \rightarrow \cdots \rightarrow R'_{i-1} \overset{C_i,\sigma_i}{\rightarrow} R'_i \rightarrow \cdots$$

  *telle que pour tout $i \geq 1$, $\sigma_i A = A$.*

La propriété de complétude fournit ici une contre-partie opérationnelle du plus grand point fixe de $T_{\lceil P \rceil}$.

Les approches existantes dans la définition d'une sémantique pour les programmes logiques qui ne terminent pas sont exclusivement dédiées à la caractérisation d'objets infinis et aboutissent à une sémantique valide mais non complète. L'approche que j'ai suivie prend le parti opposé puisqu'elle ne s'applique qu'aux dérivations ne construisant aucun terme infini. Il s'agit là d'une restriction sur les dérivations qui a permis d'établir une sémantique valide et complète. Quoi qu'il en soit, la définition d'une sémantique pour

[7] Rappelons qu'une dérivation est *équitable* si elle échoue ou si pour tout atome $A$ y apparaissant, $A$ ou l'un de ses résidus est sélectionné en un nombre fini d'étapes.

(toutes) les dérivations infinies ne connait pas encore de solution générale. Néanmoins, les dérivations infinies sur le domaine des termes finis semblaient constituer un obstacle à la complétude des approches existantes et les résultats présentés ici contribuent à une meilleure compréhension du problème.

# Annexe C

# Pratique des méthodes formelles

## C.1 Changement de représentation

### C.1.1 Réutilisation de preuves : définition d'isomorphismes au sein du langage de spécification

Dans la section B.1, nous avons présenté une formalisation de l'unification de termes du premier ordre. Toutefois, plutôt que de construire cette preuve formelle en suivant une approche classique (basée sur l'algorithme d'unification) et puisqu'une preuve formelle similaire avait été construite par J. Rouyer [115] pour un ensemble plus général, l'ensemble des quasi-termes, nous avons transposé la propriété d'unification, déjà établie, de l'ensemble des quasi-termes vers l'ensemble des termes. Il s'agissait donc de mettre en relation l'ensemble sur lequel porte la preuve cherchée avec le sous-ensemble correspondant de l'ensemble des quasi-termes, et de prouver la conservation du résultat. La principale difficulté provient de l'utilisation de fonctions partielles dans le formalisme du système Coq dans lequel seules les fonctions totales sont représentables.

#### Unification de quasi-termes

Afin d'illustrer leur technique de synthèse de programmes [88], Z. Manna et R.J. Waldinger ont dérivé l'algorithme d'unification à partir de la preuve de la "satisfaisabilité" des spécifications du problème de l'unification [87]. Cette preuve avait été formalisée par L.C. Paulson [102] dans le système LCF [46] mais porte sur un sur-ensemble de l'ensemble des termes. Un développement similaire a été effectué par J. Rouyer [115], avec la version 5.8 du système Coq. Cette version ne permettant pas les définitions d'ensembles mutuellement inductifs, la transcription de la définition exacte des termes, dans le calcul des constructions inductives, par la définition d'un type inductif de type ensemble (Set), ne fut pas possible. En effet, un terme de la forme $f(t_1, \cdots, t_n)$ est construit à partir d'un symbole de fonction $f$ d'arité $n$ et d'une liste de termes $\ell$ de longueur $n$. Cependant, en utilisant la définition polymorphe des listes, la position du type TERM dans la définition d'un constructeur de type $\Sigma \rightarrow \texttt{list}(\text{TERM}) \rightarrow \text{TERM}$ correspond à une occurrence non positive de TERM [101]. Or, il n'est pas permis pour un objet $A$ d'être défini inductivement à l'aide d'un constructeur dans lequel $A$ n'apparaît pas positivement. De plus, le contrôle de l'arité des symboles fonctionnels ne peut se faire sans utiliser un type dépendant, ce

qui ne permettait pas l'extraction de programmes. Une structure de données intermédiaire plus générale regroupant termes et listes de termes en un seul type, correspondant à l'ensemble $Q_\Sigma[X]$ des quasi-termes, avait alors été définie par J.Rouyer en supprimant de la définition des termes tout ce qui est "contrôle" pour ne conserver que les notions "constructives". L'ensemble $Q_\Sigma[X]$ est donc défini inductivement à l'aide des quatre constructeurs suivants :

$$\mathsf{V}\colon X \to Q_\Sigma[X] \quad \mathsf{Root}\colon \Sigma \to Q_\Sigma[X] \to Q_\Sigma[X]$$
$$\mathsf{C}\colon \Sigma \to Q_\Sigma[X] \quad \mathsf{ConsArg}\colon Q_\Sigma[X] \to Q_\Sigma[X] \to Q_\Sigma[X]$$

$\mathsf{V}$ et $\mathsf{C}$ sont utilisés pour les variables et les constantes, $\mathsf{Root}$ permet d'obtenir des quasi-termes fonctionnels et $\mathsf{ConsArg}$ est utilisé pour construire des listes (non vides) de quasi-termes, correspondant aux arguments des symboles de fonction apparaissant dans les quasi-termes fonctionnels. Cependant, certains quasi-termes ne correspondent pas à des termes : c'est le cas des quasi-termes de la forme $\mathsf{ConsArg}(q_1, q_2)$ et des quasi-termes pour lesquels le nombre des arguments associés à un symbole de fonction $f$ est différent de l'arité de $f$. De manière similaire, certains quasi-termes ne correspondent pas à des listes de termes, comme par exemple le quasi-terme $\mathsf{ConsArg}(\mathsf{ConsArg}(x, x), x)$.

### Définition de fonctions partielles

Dans les deux développements de L.C. Paulson et J. Rouyer, la propriété d'unification n'est pas obtenue directement sur l'ensemble des termes mais sur des structures de données plus générales. Toutefois, il existe une bijection entre un sous-ensemble de $Q_\Sigma[X]$ et l'ensemble des termes et il est possible de définir un prédicat caractérisant les quasi-termes "compatibles" avec la notion de termes. Afin de transposer la propriété d'unification vers l'ensemble des termes, il est nécessaire de définir une fonction ayant pour domaine un sous-ensemble de $Q_\Sigma[X]$ caractérisé par ce prédicat. Une telle restriction, sur le domaine d'une fonction, n'est pas permise dans le Calcul des Constructions. Les problèmes liés à la définition de telles fonctions dans des systèmes comme Coq, qui n'incorporent pas la notion de partialité, sont discutés par C. Dubois et V. Viguié Donzeau-Gouge dans [36].

Nous esquissons ici les deux techniques que nous avons utilisées pour représenter une fonction partielle permettant d'obtenir un terme à partir de certains quasi-termes dans le système Coq. A partir de deux ensembles, $E_1$ (quasi-termes) et $E_2$ (termes), définis inductivement, et d'un prédicat $P$ sur $E_1$, tel qu'il existe une bijection entre le sous-ensemble $E_1^P = \{e \in E_1 \mid P(e)\}$ de $E_1$ et $E_2$, on souhaite construire deux fonctions reliant $E_1^P$ et $E_2$. Les deux caractéristiques de ce problème sont :

- Le sous-ensemble $E_1^P$ est caractérisé par un prédicat, ce qui empêche toute définition "directe" de fonction ou de prédicat sur $E_1^P$ (ainsi, les formules $\forall e_1 \in E_1^P \varphi$ devront se lire $\forall e_1 \in E_1(P(e_1) \Rightarrow \varphi)$).
- On ne dispose pas de la bijection, mais seulement de la preuve de son existence, cette bijection est caractérisée via une relation $\equiv$ définie sur $E_1 \times E_2$ et vérifiant :

$$\forall e_1 \in E_1^P\,(\exists e_2 \in E_2\,(e_1 \equiv e_2 \wedge (\forall e_2' \in E_2\,(e_1 \equiv e_2' \Rightarrow e_2' = e_2))))$$
$$\forall e_2 \in E_2\ \ (\exists e_1 \in E_1\,(e_1 \equiv e_2 \wedge (\forall e_1' \in E_1\,(e_1' \equiv e_2 \Rightarrow e_1' = e_1))))$$

On cherche à construire les deux fonctions $f_{12}$ et $f_{21}$ telles que :

$$f_{12} : E_1^P \to E_2 \quad \forall e_1 \in E_1^P\ e_1 \equiv f_{12}(e_1) \qquad f_{21} : E_2 \to E_1 \quad \forall e_2 \in E_2\ f_{21}(e_2) \equiv e_2$$

La fonction $f_{21}$ se construit directement, puisqu'elle peut s'appliquer à l'ensemble $E_2$ tout entier, et vérifie $\forall e_2 \in E_2\, P(f_{21}(e_2))$. La définition de $f_{12}$ est plus délicate, puisqu'elle est définie sur un ensemble caractérisé par un prédicat. Les deux solutions considérées pour la définition de $f_{12}$ consistent à considérer simultanément un élément de $E_1$ et la preuve qu'il satisfait bien le prédicat[1] $P$.

La première solution repose sur l'utilisation de la règle d'élimination de la proposition False :

$$\mathsf{forall\ P : Set,\ False \to P}$$

qui permet d'envisager la définition de la fonction cherchée comme l'écriture d'un programme qui prend un argument $e_1$ satisfaisant une pré-condition $P$ et qui peut donc produire un résultat quelconque si son argument ne satisfait pas $P$ puisqu'un tel cas n'est pas possible (on dispose d'une preuve de $P(e_1)$) : en adoptant cette approche "langage de programmation", le résultat est construit à partir de son argument $e_1$ sachant qu'il vérifie forcément le prédicat $P$.

La deuxième solution consiste à exploiter non plus l'élément $e_1$, mais la preuve qu'il satisfait bien le prédicat $P$. Toutefois, puisque $P$ est un prédicat sur $E_1$ ($P\colon E_1 \to \mathsf{Prop}$) et puisque $E_2$ est de type Set, il n'est pas possible de définir une fonction $f_{12}$ en considérant la structure de $P(e_1)$ : cela reviendrait utiliser un objet non informatif (i.e. sans contenu calculatoire) pour construire un objet informatif. Cette approche "langage de preuves" consiste donc à construire inductivement, pour tout élément $e_1$ de $E_1^P$, l'ensemble $\mathcal{P}[e_1]$ des preuves de $P(e_1)$ ($\mathcal{P}\colon E_1 \to \mathsf{Set}$). Même si $\mathcal{P}[e_1]$ et $P(e_1)$ semblent équivalents d'un point de vue logique, ils ne sont pas équivalents d'un point de vue "calculatoire". Une fois cet ensemble construit, l'utilisation de types dépendants permet de définir la fonction :

$$f_{12} : \prod_{e_1 \in E_1} (\mathcal{P}[e_1] \to E_2)$$

Cette fonction est définie récursivement sur la structure d'une preuve et permet de prendre en compte le contenu algorithmique d'une telle preuve. La fonction ainsi construite doit vérifier :

$$\forall e_1 \in E_1 \quad \forall p \in \mathcal{P}[e_1] \quad e_1 \equiv f_{12}(e_1, p)$$

Pour simplifier, on notera $p[e]$ la preuve de $\mathcal{P}(e)$ et on écrira simplement $f_{12}(p[e_1])$ pour $f_{12}(e_1, p)$. Ces définitions permettent de considérer des preuves comme des objets d'une collection mathématique et donc de mener des raisonnements par induction sur ces preuves ou bien de construire des fonctions récursives sur ces preuves.

### Transposition de propriétés

Afin d'établir la propriété d'unification sur l'ensemble des termes à partir de la preuve de cette propriété sur l'ensemble des quasi-termes, il suffit maintenant de montrer que les

[1] Afin de considérer l'ensemble $E_1$ tout entier (plutôt que $E_1^P$) comme domaine de la fonction $f_{12}$, nous pouvons ajouter à $E_2$ un élément "artificiel" et définir $E_2' = E_2 \cup \{\mathsf{meaningless}\}$. La fonction $f_{12}$ vérifierait alors $f_{12}(e_1) = \mathsf{meaningless}$ pour tous les éléments $e_1$ de $E_1$ ne satisfaisant pas le prédicat $P$. Mais cette solution ne fait que déplacer le problème.

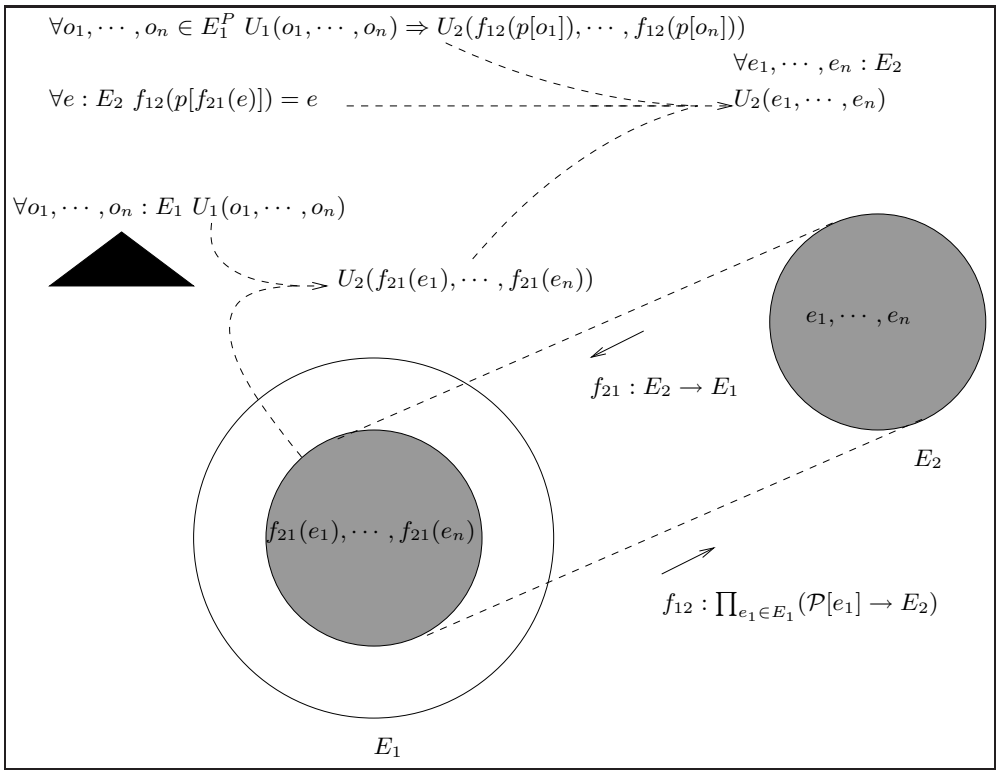


FIG. C.1 – Transposition de propriétés formelles

fonctions entre termes et quasi-termes préservent la propriété d'unification. Replaçons nous dans le cadre plus général du paragraphe précédent. Puisque $E_2$ et un sous-ensemble de $E_1$ sont isomorphes, nous souhaitons transposer une propriété exprimée sur $E_1$ vers $E_2$. Considérons deux propriétés formelles $U_1$ (quasi-unification) et $U_2$ (unification), respectivement sur $E_1$ et $E_2$, telles que :

$$\forall o_1, \cdots, o_n \in E_1^P \quad U_1(o_1, \cdots, o_n) \Rightarrow U_2(f_{12}(p[o_1]), \cdots, f_{12}(p[o_n]))$$

Supposons que $E_1$ vérifie $U_1$ et montrons la propriété $U_2$ sur $E_2$. La preuve est obtenue en conduisant le raisonnement suivant (figure C.1). Soit $e_1$, ..., $e_n$ des éléments de $E_2$, par hypothèse, on a $U_1(f_{21}(e_1), \cdots, f_{12}(e_n))$ et puisque chaque $f_{21}(e_i)$ admet une preuve $p[e_i]$ de $P(f_{21}(e_i))$ dans $\mathcal{P}[f_{21}(e_i)]$, on a alors $U_2(f_{12}(f_{21}(p[e_1])), \cdots, f_{12}(f_{21}(p[e_n])))$. Or $f_{12}(f_{21}(p[e_i])) = e_i$ ce qui prouve la propriété $U_2$ pour les éléments de $E_2$.

### C.1.2 Réutilisation de preuves : Outil externe de transformation de preuves

L'atelier Focal a été utilisé pour construire une large bibliothèque de calcul formel [112]. L'intérêt principal du calcul formel par rapport aux méthodes numériques est de donner des résultats exacts, sans erreur d'arrondi. Mais le calcul formel perd beaucoup de son intérêt si ces résultats ne sont pas fiables. En effet, comme tous les programmes, les programmes de calcul formel sont sujets à des erreurs d'implantation qui peuvent provenir soit d'une

incorrection mathématique, soit d'une mauvaise interprétation de la sémantique. Dans le domaine du calcul formel, les spécifications mathématiques sont issues de l'algèbre classique et les propriétés sont souvent des théorèmes bien connus. Certains d'entre eux ont déjà été formalisés dans un autre cadre, soit dans Coq, soit dans un autre système comme Mizar, PVS, Theorema, etc. Il est évidemment intéressant de comprendre comment réutiliser toutes ces preuves dans le cadre de Focal. Il s'agit donc de faire "communiquer" deux assistants à la preuve dans le but de faire bénéficier l'un des preuves implantées par l'autre. Pour cela, il nous faut tout d'abord mieux cerner ce que peut signifier le mot réutilisation. Une preuve s'appuie sur un énoncé, effectué dans un cadre logique précis, mais également sur certains choix de présentation des données. Par exemple, alors que les structures mathématiques implantées dans la bibliothèque de calcul formel de Focal sont paramétrées par une relation d'équivalence, la majorité des développements existants en Coq utilisent l'égalité syntaxique, qui est une forme plus contrainte d'égalité, sans toutefois utiliser dans les preuves les propriétés plus fortes que cette relation vérifie. S'agit-il alors de réutiliser uniquement la preuve d'une propriété sur une structure particulière ou bien peut-on envisager de généraliser une preuve afin qu'elle corresponde à la preuve cherchée ? Nous présentons ici, au travers d'un exemple sur les groupes, le développement d'un outil permettant d'une part de généraliser une preuve en la paramétrant par une relation d'équivalence et un ensemble de propriétés que doit vérifier cette relation (il s'agit essentiellement de propriétés de congruence) et d'autre part de réutiliser cette preuve dans l'environnement Focal.

### Implantation des groupes dans les systèmes Coq et Focal

La notion de groupe est implantée dans les systèmes Coq et Focal. Toutefois, ces deux développements utilisent une représentation différente de l'égalité.

L'implantation avec Coq, obtenue par P. Castéran [21], repose sur l'égalité de Leibniz, implantée au coeur du système Coq. L'égalité de Leibniz, aussi appelée égalité propositionnelle, est caractérisée par les deux axiomes suivants où $=_A$ dénote l'égalité propositionnelle sur le type $A$ :

$$\forall A : \texttt{Set}\ \forall x : A \quad x =_A x$$
$$\forall A : \texttt{Set}\ \forall P : (A \to \texttt{Prop})\ \forall x, y : A \quad x =_A y \Rightarrow (P\ x) \Rightarrow (P\ y)$$

En Coq, l'égalité est définie de manière inductive comme la plus petite relation binaire réflexive :

```
Inductive eq [A:Set;x:A]: A -> Prop := refl_equal : (eq A x x).
```

et le schéma d'élimination (spécialisé au cas des prédicats non dépendants) associé à cette définition correspond au deuxième axiome cité ci-dessus. On établit aisément que la relation ainsi définie est symétrique et transitive. Une des principales propriétés de cette égalité est que toute fonction $f$ de $A$ dans $B$ préserve `eq` :

$$\texttt{f\_equal} : \forall A, B : \texttt{Set}\ \forall f : A \to B\ \forall x, y : A \quad x =_A y \Rightarrow (f\ x) =_B (f\ y)$$

P. Castéran [21] fournit deux représentations différentes de la notion de groupe dans le système Coq :

- Un groupe additif est défini comme un ensemble non vide muni d'une loi de composition interne $\oplus$ associative, admettant un élément neutre 0, telle que chaque élément admette un opposé pour $\oplus$.

– Un groupe additif est défini comme un ensemble $E$ non vide muni d'une loi de composition interne $\oplus$ telle que pour tout $x$ dans $E$, les deux fonctions $f_x = (\lambda y : E.\ x\ \oplus\ y)$ et $g_x = (\lambda y : E.\ y\ \oplus\ x)$ soient surjectives.

P. Castéran montre alors l'équivalence de ces deux représentations.

L'implantation des groupes avec Focal repose exclusivement sur l'égalité des setoïdes. Un setoïde est défini comme un ensemble non vide muni d'une relation d'équivalence $\doteq$. Si $St_A$ (resp. $St_B$) est un setoïde construit sur le type $A$ (resp. $B$) et la relation d'équivalence $\doteq_A$ (resp. $\doteq_B$), un morphisme de setoïdes de $St_A$ dans $St_B$ est une fonction $f$ de $A$ dans $B$ qui préserve l'égalité, c'est-à-dire qui vérifie la propriété de compatibilité :

$$\texttt{f\_comp} : \forall x, y : A \quad x \doteq_A y \Rightarrow (f\ x) \doteq_B (f\ y)$$

Tandis qu'avec l'égalité de Leibniz, toute fonction $f : A \to B$, définissable dans le système Coq, préserve l'égalité à travers la propriété `f_equal`, un terme $f$ de type $A \to B$ est un morphisme de setoïdes si et seulement si il est accompagné d'une preuve de compatibilité `f_comp`. Dans le système Focal, un groupe additif est défini comme un ensemble non vide muni d'une relation d'équivalence $\doteq$ et d'une loi de composition interne $\oplus$ associative, admettant un élément neutre 0, telle que $\doteq$ soit une congruence pour $\oplus$ et telle que chaque élément admette un opposé pour $\oplus$.

### Schéma de réutilisation

Afin de faire bénéficier le développement Focal des résultats prouvés dans le système Coq, nous avons développé un schéma de réutilisation composé de deux parties distinctes (voir figure C.1.2).

La première partie a pour but de transformer une preuve utilisant l'égalité de Leibniz en une preuve utilisant une égalité de setoïde. Passer de l'égalité de Leibniz à celle des setoïdes revient à généraliser l'égalité de Leibniz en une relation d'équivalence particulière qui satisfait la propriété de compatibilité de toutes les opérations sur son domaine. De nombreuses propriétés de compatibilité alors engendrées ne sont pas utilisées dans les preuves à réutiliser et l'étape de généralisation portera uniquement sur les propriétés $I_1, \cdots, I_n$ dont on a réellement besoin. Alors que ces propriétés correspondent à des instances de la propriété de Leibniz dans le développement initial, elles devront être prouvées lors de la généralisation.

La transformation de la preuve que nous avons effectuée lors de cette étape repose sur les tactiques de "réécriture" associées aux relations d'égalité considérées. Deux tactiques du système Coq sont directement concernées par l'égalité. La tactique `Rewrite` du système Coq permet de mettre en œuvre la propriété de Leibniz `eq_ind`. Etant donné un but courant à prouver dans un contexte dans lequel figure en hypothèse le terme $h : (y = x)$, l'application de la tactique `Rewrite` $h$ permet de remplacer $y$ par $x$ dans le but courant. La tactique `Replace` $t_1$ `with` $t_2$ permet de remplacer les occurrences d'un terme $t_1$ par un terme $t_2$ dans un but et engendre un but supplémentaire qui consiste à prouver l'égalité $t_1 = t_2$. Lorsque la preuve en cours de construction met en jeu des setoïdes et des morphismes de setoïdes, il n'est plus possible d'utiliser ces tactiques : seules les propriétés de réflexivité, symétrie et transitivité qui définissent les relations d'équivalence ainsi que les propriétés de compatibilité des morphismes (`f_comp`) peuvent être appliquées. Afin de pouvoir réutiliser

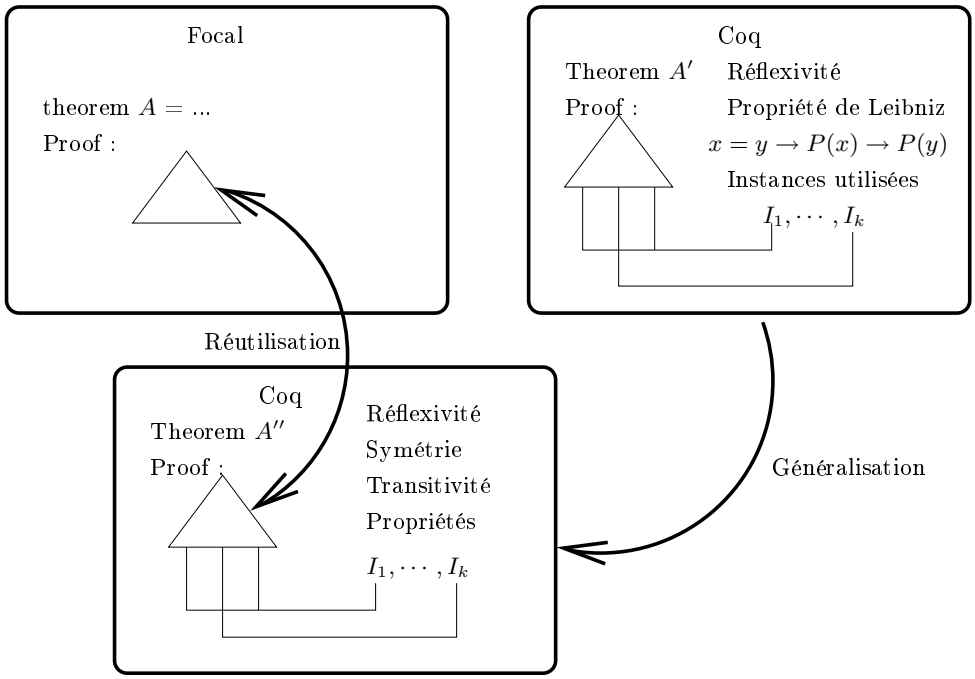


FIG. C.2 – Schéma de réutilisation

des preuves en Coq reposant sur l'égalité de Leibniz, dans le contexte des preuves en Focal reposant sur la notion de setoïde, il est nécessaire de disposer de tactiques sur les setoïdes similaires aux tactiques `Rewrite` et `Replace`. Dans [111], C. Renard propose un module d'extension Coq, qui permet de manipuler les égalités des setoïdes avec de nouveaux outils qui se comportent de façon analogue aux outils de l'égalité du système Coq. A l'aide de ce développement, il devient possible de déclarer un setoïde en fournissant le terme $A$ de type `Set` ou `Type` correspondant au type des éléments du setoïde, la relation $eq_A$ de type $A \rightarrow A \rightarrow$ `Prop` correspondant à la relation d'égalité considérée et le terme $St_A$ de type setoïde contenant une preuve que la relation $eq_A$ est bien une relation d'équivalence. Il devient également possible de définir un morphisme $f$ en fournissant la preuve de la propriété de compatibilité de $f$. L'algorithme qui engendre l'obligation de preuve relative à la propriété de compatibilité est décrit dans [111]. La seule contrainte imposée au terme $f$ est de ne pas contenir de produit dépendant. L'extension proposée par C.Renard fournit alors deux tactiques de réécriture sur les setoïdes `Setoid_rewrite` et `Setoid_replace` : si $eq$ est une égalité de setoïde, la tactique `Setoid_rewrite` appliquée à un terme de la forme $(eq\ t_1\ t_2)$ remplace toute occurrence du terme $t_1$ par le terme $t_2$ à condition que dans le but courant toutes les opérations "traversées" soient des morphismes. Munis de ces nouvelles tactiques, nous pouvons alors définir un mécanisme de généralisation de preuves. Ce mécanisme procède comme suit :

1. La définition d'une relation d'équivalence sur l'ensemble $E$ considéré est introduite.
2. Chaque occurrence de l'égalité (de Leibniz) `=` dans le développement Coq est remplacée par une occurrence de l'égalité de setoïde `=S`.
3. Les propriétés de congruence sont introduites pour les opérations que l'on souhaite

considérer comme des morphismes de setoïde.

4. Les trois tactiques `Reflexivity`, `Symmetry` et `Transitivity H` sont remplacées, dans les scripts de preuve, par les applications des propriétés de réflexivité, de symétrie et de transitivité établies pour la relation `=S`. De même, les occurrences (`Rewrite H`) et (`Replace m with n`) sont remplacées par les tactiques équivalentes (`Setoid_rewrite H`) et (`Setoid_replace m with n`).

Ainsi, le prototype réalisé automatise la généralisation de l'égalité de Leibniz au sein d'un développement Coq.

La seconde partie du schéma de réutilisation a pour but d' "encapsuler" un terme de preuve Coq de manière à l'utiliser comme preuve dans le système Focal. Ceci se traduit par la définition d'une tactique de preuve Focal qui se fonde sur la compilation en Coq d'une spécification Focal. Syntaxiquement, la tactique de réutilisation prend la forme :

`{* Reuse (`*theoreme_Coq instances*`). *}`

et se place à l'endroit où la preuve est attendue. Le terme *theoreme_Coq* est le théorème Coq à réutiliser. Les termes $t_1$, ..., $t_n$ désignés par *instances* correspondent à des termes Coq. Ce morceau de code Focal est compilé en un lemme Coq : le script généré commence par procéder à l'introduction des dépendances de déclarations et des paramètres puis applique la tactique `Exact (`*theoreme_Coq* $t_1\ t_2 \ldots t_n$`)`. Les termes $t_1\ t_2 \ldots t_n$ mentionnés font référence directement aux objets manipulés par la spécification Coq obtenue par compilation. Pour utiliser cette tactique il faut donc connaître assez précisément cette traduction ou du moins quelques correspondances simples à établir et quasi-syntaxiques. Une des perspectives de ce travail est de proposer une syntaxe qui permettrait de relier les paramètres du théorème Coq à instancier avec les objets de la spécification Focal.

Le schéma de réutilisation proposé ici a été utilisé pour prouver dans l'atelier Focal quatre des résultats prouvés sur les groupes par P. Castéran dans le système Coq. Nous avons ainsi obtenu une spécification des groupes dans le système Focal mettant en jeu les deux propriétés de surjectivité présentées plus haut. La principale difficulté de ce travail provient du fait que la preuve à réutiliser est exprimée à l'aide d'un langage de tactiques.

La réutilisation de développements formels pose des problèmes difficiles – tant du point de vue théorique que du point de vue technique – et les perspectives de ce travail sont nombreuses. Concernant l'outil présenté dans cette section, et au delà de l'aspect réutilisation, un tel outil devrait également permettre d'apporter une information sur la preuve réutilisée, en fournissant les hypothèses sur la relation d'égalité qui sont réellement utilisées. À plus long terme, il serait intéressant d'étudier la réutilisation de manière plus approfondie. Plusieurs directions sont possibles :

- Plusieurs représentations équivalentes d'un même objet peuvent coexister et les problèmes de changement de représentation méritent d'être étudiés plus en profondeur, tant d'un point de vue pratique, comme dans [86], que d'un point de vue théorique, comme dans [11].
- Abstraire un concept dans une preuve pour l'instancier de manière différente – comme nous l'avons fait pour la notion d'égalité – nécessite une étude plus poussée et une comparaison avec d'autres travaux comme par exemple le développement présenté dans [77].

– Un piste plus ambitieuse concerne la "communication" entre divers outils de développements formels. Ici encore, des problèmes théoriques se posent (il s'agit de faire "coïncider les logiques mises en jeu") [56, 95] et la mise en pratique pose également des problèmes "techniques" [27, 96, 41, 31].

### C.1.3 Preuves dans un architecture hiérarchique

Dans cette section, nous présentons une étude sur la réutilisation de preuves formelles dans le cadre de l'environnement Focal. Focal fournit un langage fonctionnel muni de traits objets avec lequel on peut écrire des spécifications, des programmes et des preuves. Les preuves portent sur des objets susceptibles d'être redéfinis lors de l'héritage. La possibilité de redéfinir une fonction pose cependant un problème puisque les preuves qui garantissent que la fonction redéfinie est conforme à sa spécification sont alors invalidées. Dans ce contexte, la question de la place des preuves dans la hiérarchie du graphe d'héritage se pose alors : faut-il prouver les spécifications au niveau le plus abstrait (au risque de devoir refaire les preuves en cas de redéfinition), ou bien faut-il prouver les spécifications au niveau le plus bas (au risque de devoir dupliquer les preuves) ? Cette question n'admet pas de réponse générale. Nous tentons ici de donner des critères sur la structure du code considéré pour répondre à cette question et nous esquissons une méthodologie de développement permettant de limiter l'impact des redéfinitions sur les preuves. Ce travail est détaillé dans [109].

#### Preuves invalidées par le mécanisme de redéfinition

Une preuve en Focal est soit un script Coq interprété dans le contexte de l'espèce où la propriété est prouvée, soit une preuve, contenant plus ou moins de détails, écrite en Cutter, le langage déclaratif sur lequel est basé le démonstrateur automatique Zenon. Il existe deux moyens d'introduire une preuve en Focal : soit en définissant un théorème, soit en démontrant une propriété déjà déclarée. L'une des formes syntaxiques possibles est la suivante :

```
theorem T : theoreme
proof : [def : def_dep_1 ... def_dep_n] ;
        [decl : decl_dep_1 ... decl_dep_n] ;
        preuve ;
```

Les termes $def\_dep_k$ désignent les noms des méthodes définies (directement dans l'espèce ou par héritage) et les termes $decl\_dep_k$ désignent les méthodes déclarées. Le mot-clé `decl` indique que la preuve du théorème $T$ dépend uniquement des types des méthodes $decl\_dep_1$, ..., $decl\_dep_n$. En revanche, elle dépend également de la définition des méthodes $def\_dep_1$, ..., $def\_dep_n$. Ces indications de dépendances explicitent donc de manière claire le contexte de la preuve[2].

Lorsqu'une redéfinition de fonction invalide les preuves qui dépendent de cette définition, le compilateur demande à l'utilisateur de les refaire. Le choix de la place des preuves

[2]Dans Focal, certaines caractéristiques de la programmation orientée-objet ont été limitées afin d'éviter des constructions non correctes, par exemple la récursion ouverte qui peut mener à des incohérences lorsqu'elle est mal utilisée.

dans la hiérarchie et la méthodologie de développement sont donc importants afin de limiter le nombre de preuves invalidées. Nous essayons ici d'éclairer ces choix et de les illustrer au travers d'un exemple. Considérons l'espèce Setoid contenant les méthodes suivantes :

Espèce Setoid

| méthode | nom | type | définition |
|---|---|---|---|
| déclarée | $=$ | self $\rightarrow$ self $\rightarrow$ bool | |
| déclarée | eq_refl | $\forall x \in$ self $x = x$ | |
| déclarée | eq_sym | $\forall x, y \in$ self $x = y \Rightarrow y = x$ | |
| déclarée | eq_trans | $\forall x, y, z \in$ self $x = y \Rightarrow y = z \Rightarrow x = z$ | |

ainsi que les deux espèces définies dans la table C.1. Dans cette présentation des espèces, self désigne le type d'un élément de l'espèce, et and et not sont les opérateurs classiques sur les booléens. L'espèce Setoid représente la notion de setoïde, elle ne contient que des

Espèce PartialOrder (hérite de l'espèce Setoid)

| méthode | nom | type | définition |
|---|---|---|---|
| déclarée | $\leq$ | self $\rightarrow$ self $\rightarrow$ bool | |
| déclarée | le_refl | $\forall x \in$ self $x \leq x$ | |
| déclarée | le_asym | $\forall x, y \in$ self<br>$x \leq y \Rightarrow y \leq x \Rightarrow x = y$ | |
| déclarée | le_trans | $\forall x, y, z \in$ self<br>$x \leq y \Rightarrow y \leq z \Rightarrow x \leq z$ | |
| définie | $<$ | self $\rightarrow$ self $\rightarrow$ bool | $x < y :=$ $x \leq y$ and $x \neq y$ |
| définie | T | $\forall x, y \in$ self<br>$x \leq y \Rightarrow (x < y \vee x = y)$ | `def` : $<$ , `decl` :$\leq$<br>...preuve... |

Espèce OrderedSet (hérite de l'espèce PartialOrder)

| méthode | nom | type | définition |
|---|---|---|---|
| déclarée | P | $\forall x, y \in$ self $x \leq y \vee y \leq x$ | |
| définie | $=$ | hérite de Setoid | $x = y :=$ $x \leq y$ and $y \leq x$ |
| définie | eq_refl<br>eq_sym<br>eq_trans | hérite de Setoid | `def` : $=$ , `decl` : ...<br>...preuve... |
| redéfinie | $<$ | hérite de PartialOrder | $x < y :=$ not $y \leq x$ |
| redéfinie | T | hérite de PartialOrder | `def` : $<$ , `decl` :$\leq$<br>...preuve... |

Tab. C.1 – Définition des espèces PartialOrder et OrderedSet

méthodes déclarées. L'espèce PartialOrder hérite de l'espèce Setoid et correspond à la notion d'ordre partiel. Elle déclare une méthode $\leq$ ainsi que les propriétés qui établissent que cette méthode correspond à un ordre partiel. Elle définit de plus une méthode $<$ correspondant à l'ordre strict associé à $\leq$ et définit également une méthode T correspondant à une preuve d'une propriété reliant $\leq$ et $<$. L'espèce OrderedSet hérite de l'espèce PartialOrder et correspond à la notion d'ordre total. Elle déclare la propriété P qui établit que $\leq$ est une relation d'ordre totale et s'appuie sur cette propriété pour redéfinir la méthode $<$

afin d'en donner une implantation plus efficace. La preuve définissant la méthode T dans l'espèce PartialOrder est donc invalidée puisqu'elle dépend de la définition de $<$. Elle est donc redéfinie. D'autre part, l'espèce OrderedSet définit les méthodes $=$, eq_refl, eq_sym et eq_trans, seulement déclarées jusqu'alors. Cet exemple illustre une situation où une preuve est invalidée suite à la redéfinition d'une méthode. Si l'on considère uniquement la hiérarchie contenant ces trois espèces, il est clair que la preuve du théorème T développée dans l'espèce PartialOrder est inutile puisqu'aucune autre espèce hérite de PartialOrder. Une première indication pour déterminer la place d'une preuve au sein de la hiérarchie est donc donnée par la réponse à la question "*existe-t-il des espèces qui héritent de la méthode dans laquelle figure la preuve et qui ne redéfinissent pas l'objet sur lequel porte la preuve ?*". Evidemment, dans le cas d'une réponse négative, cette preuve est inutile et il peut être intéressant de la différer. Toutefois, ce seul critère n'est pas satisfaisant puisqu'il préjuge des développements futurs basés sur la hiérarchie considérée.

Si le problème du choix de la place d'une preuve au sein d'une hiérarchie n'admet pas de solution générale, il est en revanche possible d'adopter une méthodologie de développement permettant de minimiser le nombre de preuves invalidées. Cette méthodologie procède en trois temps. Elle consiste à déclarer, puis spécifier et enfin implanter chaque méthode introduite[3]. Si l'on considère l'exemple précédent, la méthode $<$ est seulement implantée (elle n'est pas spécifiée) et n'a donc pas été introduite en respectant cette méthodologie. Les définitions des espèces PartialOrder et OrderedSet obtenues en adoptant la méthodologie proposée sont données dans la table C.2. La méthode $<$ est seulement déclarée et spécifiée par la méthode spec dans l'espèce PartialOrder. La méthode T ne dépend donc plus de la définition de $<$ mais seulement du type de la méthode spec. La définition de $<$ est alors introduite dans l'espèce OrderedSet et permet de définir la méthode spec. Cette preuve dépend bien sûr de la définition de $<$. Toutefois, comme on le voit sur cet exemple, la méthode T est insensible aux éventuelles redéfinitions de $<$ qui n'invalideront que la méthode spec. En suivant cette approche, les preuves de résultats "généraux" ne dépendent donc que des spécifications des fonctions de base et ne sont pas invalidées lors de la redéfinition de ces fonctions.

La méthodologie de développement suggérée ici peut être adaptée lorsque le développement à réutiliser n'a pas été obtenu en respectant les étapes successives de déclaration, de spécification et d'implantation. En effet, dans ce cas, pour chaque preuve invalidée par le mécanisme de redéfinition, il suffit d'ajouter une spécification pour chaque méthode définie dont la preuve invalidée dépend. Il faut d'autre part modifier les indications de dépendances de cette preuve en remplaçant les dépendances aux méthodes définies par des dépendances aux spécifications de ces méthodes. La démarche est donc similaire à celle de la méthodologie proposée : il s'agit de minimiser le nombre de dépendances aux définitions et pour cela de trouver un ensemble de spécifications "minimales" pour chaque méthode définie de manière à ce que les redéfinitions invalident les preuves les plus "simples" possibles.

[3]On peut noter que l'on retrouve dans ce schéma "nommer, spécifier, implanter" la démarche habituellement préconisée dans le développement logiciel sous la dénomination de "cycle en V". Cette méthode est d'ailleurs prescrite dans certaines normes comme la norme CEI61508.

Espèce PartialOrder (hérite de l'espèce Setoid)

| méthode | nom | type | définition |
|---|---|---|---|
| déclarée | $\leq$ | self $\rightarrow$ self $\rightarrow$ bool | |
| déclarée | le_refl | $\forall x \in$ self $x \leq x$ | |
| déclarée | le_asym | $\forall x, y \in$ self<br>$x \leq y \Rightarrow y \leq x \Rightarrow x = y$ | |
| déclarée | le_trans | $\forall x, y, z \in$ self<br>$x \leq y \Rightarrow y \leq z \Rightarrow x \leq z$ | |
| déclarée | $<$ | self $\rightarrow$ self $\rightarrow$ bool | |
| déclarée | spec | $\forall x, y \in$ self $x < y \Leftrightarrow (x \leq y \wedge \neg(x = y))$ | |
| définie | T | $\forall x, y \in$ self<br>$x \leq y \Rightarrow (x < y \vee x = y)$ | `decl` :$\leq$, spec<br>...preuve... |

Espèce OrderedSet (hérite de l'espèce PartialOrder)

| méthode | nom | type | définition |
|---|---|---|---|
| déclarée | P | $\forall x, y \in$ self $x \leq y \vee y \leq x$ | |
| définie | $=$ | hérite de Setoid | $x = y :=$ $x \leq y$ and $y \leq x$ |
| définie | eq_refl<br>eq_sym<br>eq_trans | hérite de Setoid | `def` : $=$ , `decl` : ...<br>...preuve... |
| définie | $<$ | hérite de PartialOrder | $x < y :=$ not $y \leq x$ |
| définie | spec | hérite de PartialOrder | `def` : $<$ , `decl` :$\leq$<br>...preuve... |

Tab. C.2 – Déclarer, Spécifier, puis Implanter

## C.2 Un cadre générique pour la spécification de formalismes logiques

Afin de faciliter la spécification formelle, il est souhaitable, lorsque c'est possible, de développer ces spécifications au sein d'un cadre générique. Lorsque ce cadre existe déjà, il permet d'assister la spécification en rendant possible cette spécification par assemblage de "briques formelles" convenablement instanciées.

Dans cette section, nous présentons une implantation dans le système Coq d'un cadre formel destiné à assister la spécification et l'implémentation de "logiques". Nous considérons l'implantation dans le système Coq d'une variante de la notion de logiques générales introduite par J. Meseguer [91, 44]. Le développement obtenu a été utilisé pour implanter la logique des propositions et la logique équationnelle. Ce travail est décrit en détail dans [66].

### C.2.1 Spécification d'un cadre générique

La notion de logique générale permet d'abstraire les notions de syntaxe, de sémantique et de preuve d'une logique particulière :

– Syntaxiquement, une logique introduit un langage, défini par une collection $\{\mathcal{S}_i\}$ de

signatures (les symboles), une collection $\{\sigma\}$ de morphismes entre certaines signatures, une collection $\{\mathcal{F}_i\}$ de formules construites sur $\{\mathcal{S}_i\}$ et une collection $\{\overline{\sigma}\}$ de morphismes étendant $\{\sigma\}$ aux ensembles de formules.
- Les aspects sémantiques, capturés par la notion d'institution introduite par J.A.Goguen et R.M. Burstall [44], sont définis par une collection $\{\mathcal{M}_i\}$ de modèles de $\{\mathcal{S}_i\}$, d'une collection d'applications $\{U_\sigma\}$ qui, pour chaque morphisme $\sigma\colon \mathcal{S}_i \to \mathcal{S}_j$, permet de transformer chaque modèle de $\mathcal{S}_j$ en un modèle de $\mathcal{S}_i$, et d'une collection $\{\models_i\}$ de relations de satisfaction entre modèles et formules sur une signature.
- Les aspects "preuves" sont définis par une collection $\{\vdash_i\}$ de relations de déduction permettant de caractériser les formules qu'il est possible de déduire à partir d'ensembles de formules.

A ces définitions s'ajoute un ensemble de propriétés qui pourront être réutilisées lors de la spécification d'une logique particulière. Tout d'abord, au niveau sémantique, la notion d'institution est accompagnée de la définition de la "condition de satisfaction" qui stipule que la notion de vérité induite par la relation de satisfaction est préservée par les changements de notations représentés par les morphismes. Plus formellement, cette propriété s'écrit :

$$\forall \sigma\colon \mathcal{S}_i \to \mathcal{S}_j\ \forall \varphi : \mathcal{F}_i\ \forall M : \mathcal{M}_j\ (U_\sigma(M) \models_i \varphi \Leftrightarrow M \models_j \overline{\sigma}(\{\varphi\}))$$

Une propriété similaire, appelée propriété de translation, est également introduite pour la relation de déduction : s'il existe un morphisme $\sigma : \mathcal{S}_i \to \mathcal{S}_j$, alors, pour tout ensemble $\Gamma$ de formules construites sur $\mathcal{S}_i$ et pour toute formule $\varphi$ aussi construite sur $\mathcal{S}_i$, si $\Gamma \vdash_i \varphi$ alors $\overline{\sigma}(\Gamma) \vdash_j \overline{\sigma}(\{\varphi\})$. A ce niveau, les propriétés de réflexivité, de transitivité et de monotonie de la relation $\vdash_i$ sont aussi formellement spécifiées. Jusqu'à présent les notions d'institutions et de systèmes de déduction, même si elles partagent une partie "syntaxe" commune, sont indépendantes. Les propriétés de validité et de complétude garantissent la cohérence entre ces deux volets d'une logique.

| | | |
|---|---|---|
| Validité | $\forall n : \mathbb{N}\ \forall \varphi : \mathcal{F}_n\ \forall \Gamma$ | $\Gamma \vdash_n \varphi \Rightarrow (\forall M : \mathcal{M}_n\ M \models_n \Gamma \Rightarrow M \models_n \varphi)$ |
| Complétude | $\forall n : \mathbb{N}\ \forall \varphi : \mathcal{F}_n\ \forall \Gamma$ | $(\forall M : \mathcal{M}_n \quad M \models_n \Gamma \Rightarrow M \models_n \varphi) \Rightarrow \Gamma \vdash_n \varphi$ |

Toutes ces définitions permettent de spécifier ce qu'est une logique générale : c'est la donnée d'une syntaxe, d'une institution et d'un système de déduction correct relativement à l'institution. On obtient ainsi une "brique formelle" paramétrée qui, une fois instanciée, permet la définition d'une logique particulière (figure C.3). Les définitions formelles que nous avons implantées dans le système Coq diffèrent un peu de celles que l'on peut trouver dans la littérature sur les logiques générales [91, 44]. En effet, notre objectif est de fournir un cadre le plus souple possible pour la spécification formelle de logiques pour l'informatique – en vue, par exemple, de la conception d'un langage de spécifications dédié à un domaine particulier. Par exemple, afin d'alléger notre formalisation, nous ne nous sommes pas placés dans le cadre de la théorie des catégories pour formaliser les notions de signatures et de morphismes entre signatures et par conséquent nous n'imposons ni l'existence d'un morphisme identité sur chaque signature, ni la propriété de composition des morphismes. D'autre part, les institutions qui pourront être définies dans notre cadre ne doivent pas nécessairemenent satisfaire la condition de satisfaction. En effet, on veut que certaines logiques qui ne satisfont pas cette condition – comme certaines logiques observables/raffinement – puissent être obtenues dans notre cadre. Par contre, nous imposons

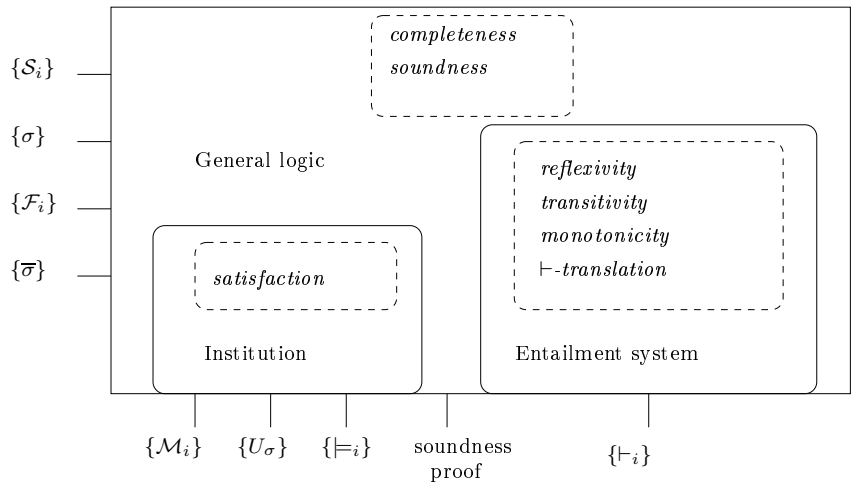


FIG. C.3 – Logiques générales

la propriété de correction pour une logique, qui n'est pas nécessairement requise dans les définitions originales.

### C.2.2 Instanciations

S'il est très général, le cadre défini ci-dessus n'est pas toujours facilement utilisable directement et il convient parfois de définir de nouvelles "briques formelles" intermédiaires entre le cadre et l'objet à formaliser. Ces nouvelles "briques formelles", également réutilisables, permettent d'automatiser certains aspects de la spécification. Par exemple, en pratique, les relations de déduction sont généralement définies à partir de systèmes formels constitués de règles de déduction. Aussi, nous avons défini une "brique formelle" réutilisable pour spécifier un système formel quelconque. Les paramètres de cette brique sont un alphabet $A$, un ensemble de formules $\mathbb{F}$ et un prédicat $\mathbb{R}$ caractérisant les instances des règles du système. Deux prédicats sont alors définis : le premier, noté $\mathcal{T}_{SF}$, correspond à la notion de formules théorèmes qu'il est possible de dériver *via* une preuve arboresente à partir d'un ensemble de formules, le second, noté $\Vdash$, correspond à la notion de déduction qui caractérise les formules qu'il est possible de dériver *via* une preuve "linéaire" (représentée par liste de formules) à partir d'un ensemble de formules. Ces deux prédicats sont prouvés équivalents ($\varphi \in \mathcal{T}_{SF}[\Gamma] \Leftrightarrow \Gamma \Vdash \varphi$) ce qui permet à l'utilisateur de choisir la forme de ses dérivations tout en pouvant passer d'une forme à l'autre. D'autre part, les propriétés de réflexivité, de transitivité et de monotonie de $\Vdash$ sont prouvées. Enfin les spécifications formelles des propriétés de cohérence, de décidabilité, de saturation et d'indépendance des axiomes ainsi que les propriétés de validité et de complétude relativement à un ensemble de formules donné sont spécifiées. A partir de la "brique formelle" ainsi obtenue, nous avons défini un "pont formel" (figure C.4) qui permet de définir la partie "preuve" d'une logique générale à partir d'un système formel. La relation $\vdash$ correspond alors à la relation de déduction $\Vdash$ du système formel. L'utilisation de ce "pont formel" permet de transposer automatiquement les propriétés de réflexivité, de transitivité et de monotonie sans avoir à les reprouver.

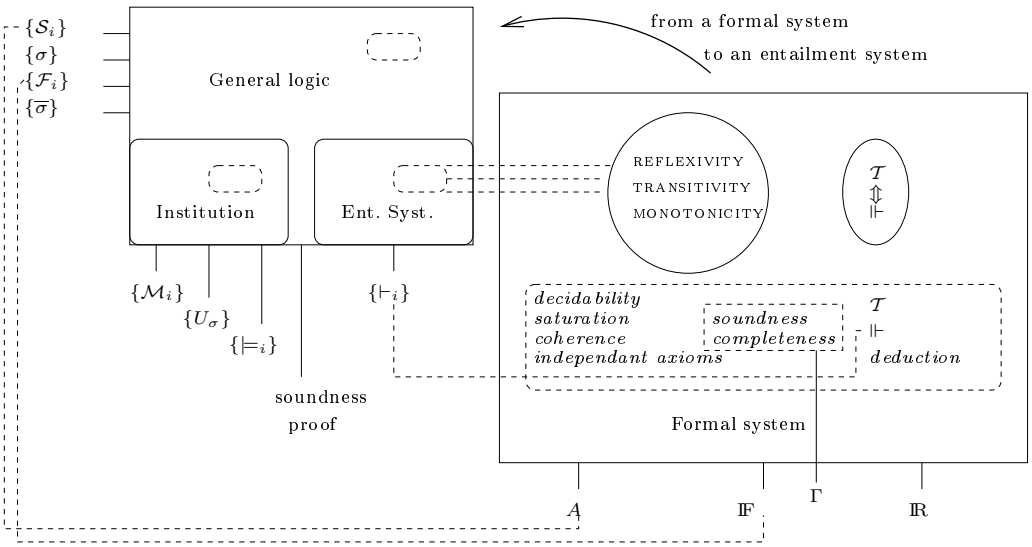


FIG. C.4 – Relations de déductions définies à partir de systèmes formels

L'utilisation des deux briques formelles présentées a permis de définir complètement deux instances de logique générale : la logique des propositions et la logique équationnelle. Chacun de ces deux développements à été obtenu en suivant le même schéma d'assemblage de "briques formelles" illustré par la figure C.5. Toutefois, ce schéma d'assemblage ne présume en rien du style à adopter pour instancier sa propre logique : l'utilisateur peut développer sa propre logique avec le style qui lui convient, il suffit ensuite qu'il enrobe ses définitions. C'est par exemple ce qui a été fait lors de la définition des systèmes formels. Une des perspectives de ce travail consiste à définir une collection d'enrobages génériques.

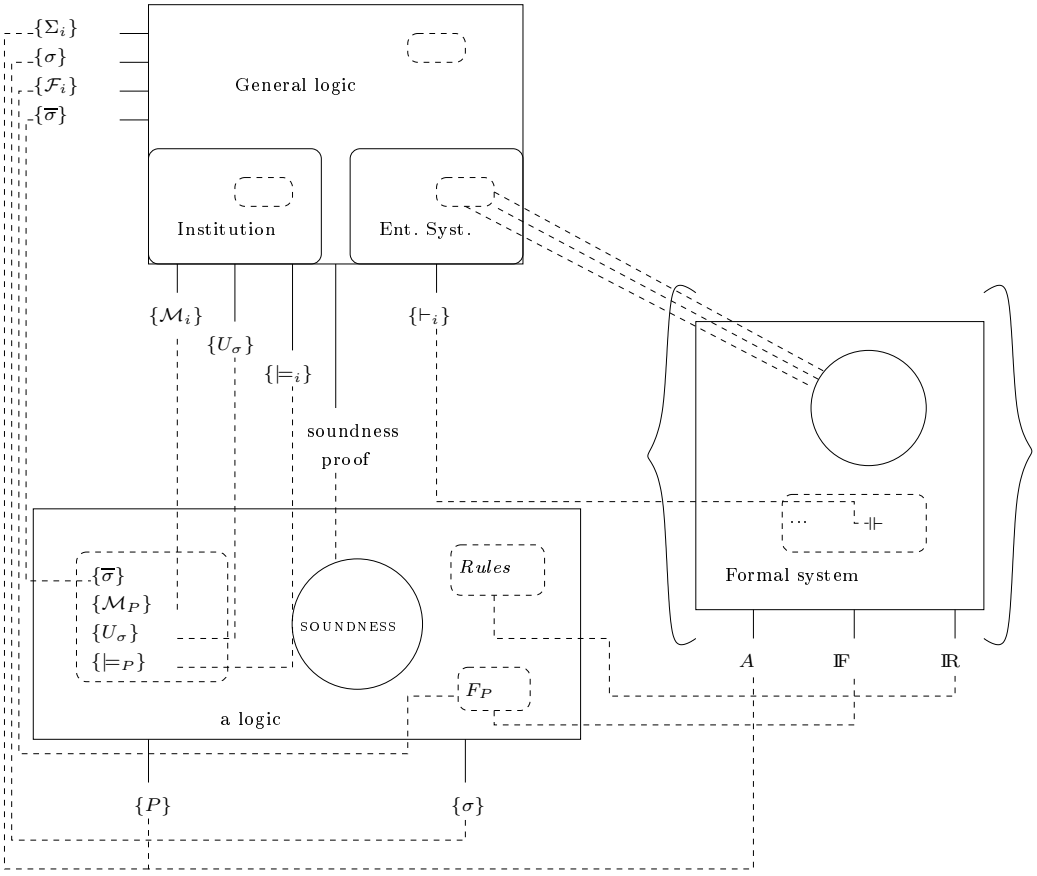


FIG. C.5 – Définition d'une logique par assemblage

# Annexe D

# Contrôle d'accès

Nous détaillons ici les résultats décrits dans le chapitre 2, et illustrons les concepts introduits à l'aide de modèles classiques de contrôle d'accès.

## D.1 Modèles de contrôle d'accès

### D.1.1 Exemples de Politiques de contrôle d'accès

Dans le chapitre 2, nous avons défini une politique de contrôle d'accès $\mathbb{P}[\rho]$, basée sur un paramètre de sécurité $\rho$, par un quintuplet $\mathbb{P}[\rho] = (\mathcal{S}, \mathcal{O}, \mathcal{A}, \Sigma, \Omega)$ où $\mathcal{S}$ est un ensemble non vide de sujets, $\mathcal{O}$ est un ensemble d'objets, $\mathcal{A}$ est un ensemble de modes d'accès, $\Sigma$ est l'ensemble des états du système sur lequel la politique est mise en œuvre et $\Omega$ est le prédicat de sécurité définissant les états sûrs. Nous présentons ici la formalisation de quatre politiques classiques de contrôle d'accès qui nous serviront d'exemples par la suite.

#### Politique HRU

La politique HRU [51] est simplement spécifiée par un ensemble $m_D$ d'accès autorisés. Selon que l'on autorise ou non cet ensemble à évoluer durant la vie du système, cet ensemble peut être vu comme un paramètre de sécurité ou comme une fonction de sécurité. Evidemment, il peut être souhaitable de permettre la modification de cet ensemble par certains utilisateurs (et dans certaines conditions). La formalisation de la politique HRU que nous donnons ici considère donc $m_D$ comme une fonction de sécurité et ne spécifie aucun paramètre de sécurité. Un état peut donc être représenté par une paire $(m, m_D)$ où $m$ désigne l'ensemble des accès courants. On note $\Sigma_{HRU}$ l'ensemble des états. Le prédicat de sécurité $\Omega_{HRU}$ définit un état sûr comme un état pour lequel chaque accès courant est autorisé :

$$\forall \sigma = (m, m_D) \in \Sigma_{HRU} \quad \Omega_{HRU}(\sigma) \Leftrightarrow m \subseteq m_D$$

On note $\mathbb{P}_{HRU}[\,] = (\mathcal{S}, \mathcal{O}, \mathcal{A}, \Sigma_{HRU}, \Omega_{HRU})$ la politique ainsi définie.

#### Politique RBAC

La politique à base de rôles que nous formalisons ici correspond à la version RBAC96 [117]. Chaque utilisateur du système est associé à un ensemble de rôles qu'il peut endosser, et

chaque rôle est associé à un ensemble de permissions. Cette politique est paramétrée par un triplet $\rho_{RBAC} = (\mathsf{U}, \mathsf{R}, \leq_{\mathsf{R}})$ où $\mathsf{U}$ est un ensemble d'utilisateurs (ici la notion de sujets est distincte de celle d'utilisateurs), et $\mathsf{R}$ est un ensemble de rôles muni d'un ordre partiel $\leq_{\mathsf{R}}$. Les fonctions de sécurité sont $user$, $\mathsf{UA}$, $\mathsf{PA}$ et $roles$, où $user : \mathcal{S} \to \mathsf{U}$ permet de connaître l'utilisateur associé à un sujet, $\mathsf{UA} \subseteq \mathsf{U} \times \mathsf{R}$ est une relation permettant de spécifier les rôles autorisés des utilisateurs, $\mathsf{PA} \subseteq (\mathcal{O} \times \mathcal{A}) \times \mathsf{R}$ est une relation permettant d'associer des permissions (c'est-à-dire des paires $(o, a) \in \mathcal{O} \times \mathcal{A}$) aux rôles et $roles : \mathcal{S} \to \wp(\mathsf{R})$ spécifie l'ensemble des rôles activés par un sujet. Un état est donc représenté par un quintuplet $(m, user, \mathsf{UA}, \mathsf{PA}, roles)$ où $m$ désigne l'ensemble des accès courants. On note $\Sigma_{RBAC}$ l'ensemble des états. Un état $\sigma = (m, user, \mathsf{UA}, \mathsf{PA}, roles)$ vérifie le prédicat $\Omega_{RBAC}$ ssi les deux conditions suivantes sont satisfaites :

$$\begin{array}{l} \forall s \in \mathcal{S} \quad roles(s) \subseteq \mathsf{ER}(s, \mathsf{UA}) \\ \forall s \in \mathcal{S} \quad \forall o \in \mathcal{O} \quad \forall a \in \mathcal{A} \quad (s, o, a) \in m \Rightarrow (o, a) \in \mathsf{EP}(s, \mathsf{PA}, roles) \end{array}$$

où, étant donné un sujet $s \in \mathcal{S}$, $\mathsf{ER}(s, \mathsf{UA})$ définit l'ensemble des rôles que $s$ peut activer en respectant $\mathsf{UA}$ et $\leq_{\mathsf{R}}$, et $\mathsf{EP}(s, \mathsf{PA}, roles)$ désigne l'ensemble des permisssions associées au rôles activés par $s$ et à leurs sous-rôles :

$$\mathsf{ER}(s, \mathsf{UA}) = \{r \in \mathsf{R} \mid \exists r' \in \mathsf{R} \quad r \leq_{\mathsf{R}} r' \wedge \ (user(s), r') \in \mathsf{UA}\}$$

$$\mathsf{EP}(s, \mathsf{PA}, roles) = \bigcup_{r \in roles(s)} \left\{ (o, a) \in (\mathcal{O} \times \mathcal{A}) \mid \begin{array}{l} \exists r'' \in \mathsf{R} \quad r'' \leq_{\mathsf{R}} r \\ \wedge \ ((o, a), r'') \in \mathsf{PA} \end{array} \right\}$$

On note $\mathbb{P}_{RBAC}[\rho_{RBAC}] = (\mathcal{S}, \mathcal{O}, \mathcal{A}, \Sigma_{RBAC}, \Omega_{RBAC})$ la politique ainsi définie.

### Politique de Bell et LaPadula

La politique de Bell et LaPadula [12] repose sur une notion de niveaux de sécurité. Elle est donc paramétrée par un treillis $\rho_{BLP} = (\mathcal{L}, \preceq, \curlyvee, \curlywedge)$. Chaque sujet et chaque objet est associé à un niveau de sécurité au moyen des fonctions de sécurité $f_s : \mathcal{S} \to \mathcal{L}$ et $f_o : \mathcal{O} \to \mathcal{L}$. Un état est donc représenté par un triplet $(m, f_s, f_o)$ où $m$ désigne l'ensemble des accès courants. On note $\Sigma_{BLP}$ l'ensemble des états. Le prédicat de sécurité $\Omega_{BLP}$ est défini par les deux propriétés spécifiées en (1.1) et (1.2). On note $\mathbb{P}_{BLP}[\rho_{BLP}] = (\mathcal{S}, \mathcal{O}, \mathcal{A}, \Sigma_{BLP}, \Omega_{BLP})$ la politique ainsi définie.

### Politique de la Muraille de Chine

Dans le contexte de la politique de la Muraille de Chine [19], chaque objet du système appartient à une compagnie, et chaque compagnie appartient à une classe de conflit d'intérêt. Il existe en outre une classe spéciale de conflit d'intérêt, avec seulement une compagnie, contenant les informations "sanitisées" (c'est-à-dire publiques). Ces informations peuvent être lues par n'importe quel sujet, quels que soient les autres accès effectués par ce dernier. L'idée intuitive de cette politique est que les sujets peuvent accéder à de l'information tant que cette dernière n'est pas en conflit avec de l'information qu'ils possèdent déjà. Cette politique est paramétrée par un quintuplet $\rho_{CW} = (\mathbb{C}, \mathbb{E}, f_{\mathbb{C}}, c_0, d_0)$ où $\mathbb{C} = \{c_1, c_2, \cdots, c_n\}$ est un ensemble de *classes de conflit d'intérêt* et $\mathbb{E}$ est un ensemble de *compagnies*. Chaque compagnie $d$ est associée à une classe de conflit d'intérêt $c_i = f_{\mathbb{C}}(d)$. Notons qu'à partir

d'une telle fonction $f_{\mathbb{C}} : \mathbb{E} \to \mathbb{C}$, il est possible de construire une fonction $f_{\mathbb{E}} : \mathbb{C} \to \wp(\mathbb{E})$ telle que $f_{\mathbb{E}}(c) = \{d \mid f_{\mathbb{C}}(c) = d\}$. Cette fonction associe à chaque classe de conflit un ensemble de compagnies. Les informations "sanitisées" peuvent être considérées comme appartenant à la compagnie $d_0$, de la classe de conflit particulière $c_0$. Un état est représenté par une paire $(m, f_o)$ où $m$ désigne l'ensemble des accès courants et $f_o : \mathcal{O} \to \mathbb{E}$ est une fonction de sécurité associant à chaque objet une compagnie. On note $\Sigma_{CW}$ l'ensemble des états. Un état $\sigma = (m, f_o)$ vérifie le prédicat de sécurité $\Omega_{CW}$ ssi les deux conditions suivantes sont vérifiées :

$$\begin{array}{l}\forall s \in \mathcal{S}\ \forall o_1, o_2 \in \mathcal{O}\ \forall a_1, a_2 \in \mathcal{A} \\ (s, o_1, a_1) \in m \wedge (s, o_2, a_2) \in m \Rightarrow f_o(o_1) = f_o(o_2) \vee f_{\mathbb{C}}(f_o(o_1)) \neq f_{\mathbb{C}}(f_o(o_2)) \\ \forall s \in \mathcal{S}\ \forall o_1, o_2 \in \mathcal{O} \\ (s, o_1, \mathsf{write}) \in m \wedge (s, o_2, \mathsf{read}) \in m \Rightarrow f_o(o_2) = d_0 \vee f_o(o_1) = f_o(o_2)\end{array}$$

La première propriété exprime que si un sujet accède à deux objets, alors soit ces objets appartiennent à la même compagnie, soit ces objets appartiennent à des objets associés à des classes de conflit différentes. La deuxième propriété exprime que dès que l'on accède en écriture à un objet, les seuls accès en lecture qui sont permis sont ceux qui portent sur des objets "sanitisés" ou appartenant à la compagnie de l'objet sur lequel un accès en écriture est effectué. Cette propriété exprime également que lorsque plusieurs objets associés à des classes de conflit différentes sont accédés en écriture, les seuls objets accessibles en lecture sont alors les objets de la compagnie $d_0$. On note $\mathbb{P}_{CW}[\rho_{CW}] = (\mathcal{S}, \mathcal{O}, \mathcal{A}, \Sigma_{CW}, \Omega_{CW})$ la politique ainsi définie.

A partir de ces définitions on montre le lemme suivant.

**Lemme D.1**

1. $\mathbb{P}_{HRU}[]$, $\mathbb{P}_{RBAC}[\rho_{RBAC}]$, $\mathbb{P}_{BLP}[\rho_{BLP}]$ *et* $\mathbb{P}_{CW}[\rho_{CW}]$ *sont des politiques compactes.*
2. $\mathbb{P}_{HRU}[]$ *et* $\mathbb{P}_{RBAC}[\rho_{RBAC}]$ *sont des politiques libres.*
3. $\mathbb{P}_{BLP}[\rho_{BLP}]$ *et* $\mathbb{P}_{CW}[\rho_{CW}]$ *ne sont pas des politiques libres.*

### D.1.2 Modèles et implantations

#### Définitions et propriétés

Dans le chapitre 2, un modèle a été défini comme la donnée d'une paire $\mathbb{M}[\rho] = (\mathbb{P}[\rho], \mathcal{R})$ et d'une sémantique pour l'ensemble $\mathcal{R}$ de requêtes. Une implantation de $\mathbb{M}[\rho] = (\mathbb{P}[\rho], \mathcal{R})$ est une paire $(\tau, \Sigma_I)$ où $\Sigma_I$ est un ensemble d'états initiaux, où $\tau : \mathcal{R} \times \Sigma \to \mathcal{D} \times \Sigma$ est une fonction de transition, et où $\mathcal{D} = \{\mathsf{yes}, \mathsf{no}\}$ est un ensemble de réponses.

On note $\Gamma_\tau(E)$ l'ensemble des états du système atteignables à partir d'un état appartenant à l'ensemble $E$ en appliquant un nombre fini de fois la fonction $\tau$. De plus, on associe à toute implantation un ensemble de séquences d'états correspondant aux traces des exécutions qu'elle engendre :

$$Exec(\tau, \Sigma_I) = \bigcup_{n \in \mathbb{N}} \left\{ (\sigma_1, \cdots, \sigma_n) \mid \begin{array}{ll} & \sigma_1 \in \Sigma_I \\ \wedge & \forall i\ (1 \leq i \leq n-1)\ \exists R \in \mathcal{R}\ \exists d \in \mathcal{D} \\ & \tau(R, \sigma_i) = (d, \sigma_{i+1}) \end{array} \right\}$$

Les propriétés de correction d'une implantation concernent essentiellement les propriétés de sécurité induites par le prédicat $\Omega$ et les propriétés relatives à la sémantique du langage de requêtes :

- $(\tau, \Sigma_I)$ est dite $\mathbb{P}[\rho]$-*correcte* (resp. $\mathbb{P}[\rho]$-*complète*) ssi tout état atteignable est sûr (resp. ssi tout état sûr est atteignable) :

$$\Gamma_\tau(\Sigma_I) \subseteq \Sigma_{|\Omega} \qquad (\text{resp. } \Sigma_{|\Omega} \subseteq \Gamma_\tau(\Sigma_I))$$

- $\tau$ est dite $[\![\mathcal{R}]\!]_\Sigma$-*correcte* (resp. $[\![\mathcal{R}]\!]^+_\Sigma$-*correcte*) ssi :

$$\forall \sigma_1, \sigma_2 \in \Sigma\ \forall R \in \mathcal{R}\ \ \tau(R, \sigma_1) = (\mathsf{yes}, \sigma_2) \Rightarrow (R, \sigma_2) \in [\![\mathcal{R}]\!]_\Sigma$$
$$(\text{resp. } \forall \sigma_1, \sigma_2 \in \Sigma\ R \in \mathcal{R}\ \tau(R, \sigma_1) = (\mathsf{yes}, \sigma_2) \Rightarrow (\sigma_1, R, \sigma_2) \in [\![\mathcal{R}]\!]^+_\Sigma)$$

- $\tau$ est dite $\mathcal{W}$-*conforme* ssi :

$$\begin{array}{l} \forall \sigma_1, \sigma_2 \in \Sigma\ \forall d \in \mathcal{D}\ \forall R \in \mathcal{R} \\ \tau(R, \sigma_1) = (d, \sigma_2) \Rightarrow \\ \left( \begin{array}{ll} & d = \mathsf{yes} \Rightarrow \left( \begin{array}{ll} & R \in \mathcal{R}^{\ominus} \Rightarrow \mathcal{W}(\sigma_1) \subseteq \mathcal{W}(\sigma_2) \\ \wedge & R \in \mathcal{R}^{\oplus} \Rightarrow \mathcal{W}(\sigma_2) \subseteq \mathcal{W}(\sigma_1) \end{array} \right) \\ \wedge & d = \mathsf{no} \Rightarrow \mathcal{W}(\sigma_2) \subseteq \mathcal{W}(\sigma_1) \end{array} \right) \end{array}$$

On écrira $\mathbb{M}[\rho] \vdash (\tau, \Sigma_I)$ lorsque l'implantation $(\tau, \Sigma_I)$ du modèle $\mathbb{M}[\rho]$ est à la fois $\mathbb{P}[\rho]$-correcte et $[\![\mathcal{R}]\!]^+_\Sigma$-correcte (resp. et $[\![\mathcal{R}]\!]_\Sigma$-correcte et $\mathcal{W}$-conforme).

Nous introduisons également les deux propriétés supplémentaires suivantes :

- Etant donnée une relation d'équivalence $\equiv$ sur les états, $\tau$ est dite $\equiv$-préservante, ce que l'on note $\equiv \vdash \tau$, ssi :

$$\begin{array}{l} \forall \sigma_1, \sigma_2, \sigma'_1, \sigma'_2 \in \Sigma\ \forall R \in \mathcal{R}\ \forall d \in \mathcal{D} \\ \left( \begin{array}{ll} & \sigma_1 \equiv \sigma_2 \\ \wedge & \tau(R, \sigma_1) = (d_1, \sigma'_1) \wedge \tau(R, \sigma_2) = (d_2, \sigma'_2) \end{array} \right) \Rightarrow \left( \begin{array}{ll} & \sigma'_1 \equiv \sigma'_2 \\ \wedge & d_1 = d_2 \end{array} \right) \end{array}$$

- $(\tau, \Sigma_I)$ est dite $\Gamma$-complète ssi :

$$\forall \sigma_1, \sigma_2 \in \Gamma_\tau(\Sigma_I) \quad \Gamma_\tau(\sigma_1) = \Gamma_\tau(\sigma_2)$$

**Exemples**

Nous reprenons ici les politiques introduites dans la section D.1.1 et notons :

$$\begin{array}{cc} \mathbb{M}^{acc}_{HRU}[\,] = (\mathbb{P}_{HRU}[\,], \mathcal{R}^{acc}) & \mathbb{M}^{acc}_{RBAC}[\rho_{RBAC}] = (\mathbb{P}_{RBAC}[\rho_{RBAC}], \mathcal{R}^{acc}) \\ \mathbb{M}^{acc}_{BLP}[\rho_{BLP}] = (\mathbb{P}_{BLP}[\rho_{BLP}], \mathcal{R}^{acc}) & \mathbb{M}^{acc}_{CW}[\rho_{CW}] = (\mathbb{P}_{CW}[\rho_{CW}], \mathcal{R}^{acc}) \end{array}$$

les modèles qui leur sont associés en considérant l'ensemble des requêtes $\mathcal{R}^{acc}$ défini en (2.1) dont la sémantique est présentée dans le chapitre 2. Les fonctions de transition permettant d'implanter ces modèles sont données dans les tables D.1, D.2, D.3 et D.4. On montre que ces fonctions de transition permettent d'obtenir des implantations correctes.

**Lemme D.2**

$$\begin{array}{rcl} \Sigma^I_{HRU} \subseteq \Sigma_{|\Omega_{HRU}} & \Rightarrow & \mathbb{M}^{acc}_{HRU}[\,] \vdash (\tau_{HRU}, \Sigma^I_{HRU}) \\ \Sigma^I_{RBAC} \subseteq \Sigma_{|\Omega_{RBAC}} & \Rightarrow & \mathbb{M}^{acc}_{RBAC}[\rho_{RBAC}] \vdash (\tau_{RBAC}, \Sigma^I_{RBAC}) \\ \Sigma^I_{BLP} \subseteq \Sigma_{|\Omega_{BLP}} & \Rightarrow & \mathbb{M}^{acc}_{BLP}[\rho_{BLP}] \vdash (\tau_{BLP}, \Sigma^I_{BLP}) \\ \Sigma^I_{CW} \subseteq \Sigma_{|\Omega_{CW}} & \Rightarrow & \mathbb{M}^{acc}_{CW}[\rho_{CW}] \vdash (\tau_{CW}, \Sigma^I_{CW}) \end{array}$$

$$\tau_{HRU}(R,\sigma) = \begin{cases} (\mathsf{yes}, \sigma \oplus (s,o,a)) & \text{if } R = \langle +,s,o,a\rangle \wedge (s,o,a) \in m_D \\ (\mathsf{yes}, \sigma \ominus (s,o,a)) & \text{if } R = \langle -,s,o,a\rangle \\ (\mathsf{no}, \sigma) & \text{otherwise} \end{cases}$$

TAB. D.1 – Implantation du modèle $\mathbb{M}^{acc}_{HRU}[]$

$$\begin{array}{l} \tau_{RBAC}(R,(m,user,\mathsf{UA},\mathsf{PA},roles)) \\ = \begin{cases} (\mathsf{yes},(m \cup \{(s,o,a)\},user,\mathsf{UA},\mathsf{PA},roles)) \\ \quad \text{if} \quad R = \langle +,s,o,a\rangle \wedge (o,a) \in \mathsf{EP}(s,\mathsf{PA},roles) \\ (\mathsf{yes},(m \setminus \{(s,o,a)\},user,\mathsf{UA},\mathsf{PA},roles)) \\ \quad \text{if} \quad R = \langle -,s,o,a\rangle \\ (\mathsf{no},(m,user,\mathsf{UA},\mathsf{PA},roles)) \quad \text{otherwise} \end{cases} \end{array}$$

TAB. D.2 – Implantation du modèle $\mathbb{M}^{acc}_{RBAC}[\rho_{RBAC}]$

$$\begin{array}{ll} \tau_{BLP}(R,\sigma) & \sigma = (m,f_s,f_o) \\ = \begin{cases} (\mathsf{yes}, \sigma \oplus (s,o,\mathsf{read})) & \text{if } R = \langle +,s,o,\mathsf{read}\rangle \\ \quad \wedge f_o(o) \preceq f_s(s) \wedge \{o' \in \mathcal{O} \mid (s,o',\mathsf{write}) \in \Lambda(\sigma) \wedge \neg(f_o(o) \preceq f_o(o'))\} = \emptyset \\ (\mathsf{yes}, \sigma \oplus (s,o,\mathsf{write})) & \text{if } R = \langle +,s,o,\mathsf{write}\rangle \\ \quad \wedge \{o' \in \mathcal{O} \mid (s,o',\mathsf{read}) \in \Lambda(\sigma) \wedge \neg(f_o(o') \preceq f_o(o))\} = \emptyset \\ (\mathsf{yes}, \sigma \ominus (s,o,x)) & \text{if } R = \langle -,s,o,x\rangle \\ (\mathsf{no}, \sigma) \text{ otherwise} \end{cases} \end{array}$$

TAB. D.3 – Implantation du modèle $\mathbb{M}^{acc}_{BLP}[\rho_{BLP}]$

$$\begin{array}{l} \tau_{CW}(R,(m,f_o)) \\ = \begin{cases} (\mathsf{yes},(m,f_o)) \text{ if } \quad R = \langle +,s,o,x\rangle \wedge (s,o,x) \in m \\ (\mathsf{yes},(m \cup \{(s,o,\mathsf{read})\},f_o)) \text{ if } \quad R = \langle +,s,o,\mathsf{read}\rangle \wedge f_o(o) = d_0 \\ (\mathsf{yes},(m \cup \{(s,o,\mathsf{read})\},f_o)) \\ \quad \text{if} \quad R = \langle +,s,o,\mathsf{read}\rangle \wedge f_o(o) \neq d_0 \\ \quad \wedge \quad \{o' \in \mathcal{O} \mid (s,o',\mathsf{write}) \in m \ \wedge f_o(o) \neq f_o(o')\} = \emptyset \\ \quad \wedge \quad \left\{ \begin{array}{ll} o' \in \mathcal{O} \mid & (s,o',\mathsf{read}) \in m \wedge f_o(o') \neq f_o(o) \\ & \wedge \quad f_{\mathbb{C}}(f_o(o')) = f_{\mathbb{C}}(f_o(o)) \end{array} \right\} = \emptyset \\ (\mathsf{yes},(m \cup \{(s,o,\mathsf{write})\},f_o)) \\ \quad \text{if} \quad R = \langle +,s,o,\mathsf{write}\rangle \\ \quad \wedge \quad \left\{ \begin{array}{ll} o' \in \mathcal{O} \mid & (s,o',\mathsf{write}) \in m \wedge f_o(o') \neq f_o(o) \\ & \wedge \quad f_{\mathbb{C}}(f_o(o')) = f_{\mathbb{C}}(f_o(o)) \end{array} \right\} = \emptyset \\ \quad \wedge \quad \{o' \in \mathcal{O} \mid (s,o',\mathsf{read}) \in m \wedge f_o(o) \neq f_o(o') \wedge f_o(o') \neq d_0\} = \emptyset \\ (\mathsf{yes},(m \setminus \{(s,o,x)\},f_o)) \text{ if } \quad R = \langle -,s,o,x\rangle \\ (\mathsf{no},(m,f_o)) \quad \text{otherwise} \end{cases} \end{array}$$

TAB. D.4 – Implantation du modèle $\mathbb{M}^{acc}_{CW}[\rho_{CW}]$

**Préordre sur les implantations**

Nous introduisons ici un préordre sur les implantations d'un même modèle lorsque ce modèle repose sur une sémantique faible de l'ensemble des requêtes. Ce préordre nous permettra de comparer deux modèles en limitant le nombre d'implantations à considérer. Nous définissons tout d'abord le préordre $\sqsubseteq_\Gamma$ exprimant qu'une implantation $(\tau_1, \Sigma_I^1)$ est plus restrictive en termes d'états atteignables qu'une implantation $(\tau_2, \Sigma_I^2)$ ssi tout état atteignable par $(\tau_1, \Sigma_I^1)$ l'est également par $(\tau_2, \Sigma_I^2)$. Nous introduisons ensuite le préordre $\sqsubseteq_\mathcal{W}$ sur les fonctions de transitions.

- $(\tau_1, \Sigma_I^1) \sqsubseteq_\Gamma (\tau_2, \Sigma_I^2) \Leftrightarrow \Gamma_{\tau_1}(\Sigma_I^1) \subseteq \Gamma_{\tau_2}(\Sigma_I^2)$
- Etant données deux fonctions de transition $\tau_1$ et $\tau_2$ de $\mathcal{R} \times \Sigma$ vers $\mathcal{D} \times \Sigma$, $\tau_1 \sqsubseteq_\mathcal{W} \tau_2$ ssi :

$$\begin{array}{l} \forall \sigma, \sigma_1 \in \Sigma \; \forall R \in \mathcal{R} \\ \quad \tau_1(R, \sigma) = (\mathsf{yes}, \sigma_1) \\ \quad \Rightarrow \left( \begin{array}{lll} \exists \sigma_2 \in \Sigma & \tau_2(R, \sigma_2) = (\mathsf{yes}, \sigma_1) \\ & \wedge \quad R \in \mathcal{R}^{\circledcirc} \Rightarrow \mathcal{W}(\sigma_2) \subseteq \mathcal{W}(\sigma) \\ & \wedge \quad R \in \mathcal{R}^{\circledcirc} \Rightarrow \mathcal{W}(\sigma) \subseteq \mathcal{W}(\sigma_2) \end{array} \right) \\ \wedge \quad \tau_1(R, \sigma) = (\mathsf{no}, \sigma_1) \Rightarrow \mathcal{W}(\sigma_1) \subseteq \mathcal{W}(\sigma) \end{array}$$

- La relation de préordre sur les implantations d'un modèle est définie par :

$$(\tau_1, \Sigma_I^1) \sqsubseteq (\tau_2, \Sigma_I^2) \Leftrightarrow ((\tau_1, \Sigma_I^1) \sqsubseteq_\Gamma (\tau_2, \Sigma_I^2) \wedge \tau_1 \sqsubseteq_\mathcal{W} \tau_2)$$

On peut alors prouver que toute implantation inférieure (selon $\sqsubseteq$) à une implantation correcte est elle-même correcte. Ce résultat sera utile lorsque nous envisagerons la comparaison de modèles de contrôle d'accès.

**Proposition D.1** *Soit $\mathbb{M}[\rho]$ un modèle de contrôle d'accès et $(\tau_1, \Sigma_I^1)$ et $(\tau_2, \Sigma_I^2)$ deux implantations de $\mathbb{M}[\rho]$. Si $(\tau_1, \Sigma_I^1) \sqsubseteq (\tau_2, \Sigma_I^2)$ et $\mathbb{M}[\rho] \vdash (\tau_2, \Sigma_I^2)$ alors $\mathbb{M}[\rho] \vdash (\tau_1, \Sigma_I^1)$.*

## D.2 Comparaison de modèles de contrôle d'accès

### D.2.1 Simulation d'implantations

**Préordre sur les modèles**

Nous détaillons ici le mécanisme de comparaison introduit dans la section 2.2.1. Rappelons que ce mécanisme est utilisable pour comparer deux modèles partageant le même ensemble de requêtes muni d'une sémantique faible. La relation de préordre partiel $\leqq$ introduite dans la définition 2.3 fait intervenir d'une part une relation d'équivalence $\equiv_\iota$ sur les états, et d'autre part une relation de simulation.

La relation d'équivalence $\equiv_\iota$ est définie comme l'intersection $\equiv_\mathcal{W} \cap \equiv_{\mathcal{W}_\emptyset} \cap \equiv_\mathcal{R}$ où :

$$\begin{array}{l} \sigma_1 \equiv_\mathcal{W} \sigma_2 \Leftrightarrow \mathcal{W}(\sigma_1) = \mathcal{W}(\sigma_2) \\ \sigma_1 \equiv_{\mathcal{W}_\emptyset} \sigma_2 \Leftrightarrow \mathcal{W}_\emptyset(\sigma_1) = \mathcal{W}_\emptyset(\sigma_2) \\ \sigma_1 \equiv_\mathcal{R} \sigma_2 \Leftrightarrow (\forall R \in \mathcal{R} \; (\sigma_1, R) \in [\![\mathcal{R}]\!]_\Sigma \Leftrightarrow (\sigma_2, R) \in [\![\mathcal{R}]\!]_\Sigma) \end{array}$$

Intuitivement, deux états sont équivalents par $\equiv_\iota$ ssi ils ont le même passé (ils auraient pu être construits à partir des mêmes requêtes et leurs fonctions de sécurité permettent les mêmes ensembles d'accès) et le même futur (ils autorisent les mêmes ensembles d'accès).

Dans la définition 2.3, nous avons imposé que la relation de simulation $\kappa_\Sigma$ utilisée vérifie certaines propriétés (ce que nous avons noté $P_s(\kappa_\Sigma)$). Ces propriétés sont les suivantes :

- $\kappa_\Sigma$ est totale à gauche[1] : intuitivement, $\mathbb{M}_1[\rho_1]$ est plus restrictif que $\mathbb{M}_2[\rho_2]$ ssi tout ce qui peut être fait avec $\mathbb{M}_1[\rho_1]$ peut également l'être avec $\mathbb{M}_2[\rho_2]$, il faut donc que tous les états de $\Sigma_1$ soient en relation avec des états de $\Sigma_2$.
- une autre contrainte consiste à imposer que deux états de $\Sigma_1$ équivalents selon $\equiv_{\iota_1}$ soient en relation avec des états de $\Sigma_2$ équivalents selon $\equiv_{\iota_2}$ et *vice-versa*. Intuitivement, deux états sont reliés s'il est possible de "faire les mêmes choses" à partir de ces deux états. Nous introduisons donc les deux propriétés suivantes requises pour $\kappa_\Sigma$ :
  - $\kappa_\Sigma \subseteq \Sigma_1 \times \Sigma_2$ est dite $\iota$-fonctionnelle ssi :

$$\begin{array}{l}\forall \sigma_1, \sigma_1' \in \Sigma_1 \ \forall \sigma_2, \sigma_2' \in \Sigma_2 \\ \sigma_1 \equiv_{\iota_1} \sigma_1' \wedge (\sigma_1, \sigma_2) \in \kappa_\Sigma \wedge (\sigma_1', \sigma_2') \in \kappa_\Sigma \Rightarrow \sigma_2 \equiv_{\iota_2} \sigma_2'\end{array}$$

  - $\kappa_\Sigma \subseteq \Sigma_1 \times \Sigma_2$ est dite $\iota$-injective ssi :

$$\begin{array}{l}\forall \sigma_1, \sigma_1' \in \Sigma_1 \ \forall \sigma_2, \sigma_2' \in \Sigma_2 \\ \sigma_2 \equiv_{\iota_2} \sigma_2' \wedge (\sigma_1, \sigma_2) \in \kappa_\Sigma \wedge (\sigma_1', \sigma_2') \in \kappa_\Sigma \Rightarrow \sigma_1 \equiv_{\iota_1} \sigma_1'\end{array}$$

  Notons que si, pour les deux modèles, la relation $\equiv_\iota$ est l'égalite, alors la $\iota$-fonctionnalité correspond intuitivement à la notion de fonctionnalité[2], et la $\iota$-injectivité à la notion d'injectivité[3].

### Modèles réduits

La relation d'équivalence $\equiv_\iota$ permet de "réduire" un modèle afin de considérer les classes d'équivalence d'états plutôt que les états. Il s'agit ici de simplifier les raisonnements et d'identifier les informations pertinentes au regard de la politique de contrôle d'accès.

Etant donné un modèle $\mathbb{M}[\rho]$ et le modèle réduit associé $\mathbb{M}^\sharp[\rho]$, les implantations du premier peuvent être reliées aux implantations du deuxième, et inversement. En effet, pour toute implantation de $\mathbb{M}[\rho]$, il est possible de construire une implantation de $\mathbb{M}^\sharp[\rho]$ à l'aide de l'opérateur :

$$\begin{array}{l}\sharp : (\mathcal{R} \times \Sigma \to \mathcal{D} \times \Sigma) \to (\mathcal{R} \times \hat{e}(\Sigma) \to \mathcal{D} \times \hat{e}(\Sigma)) \\ \forall \sigma \in \hat{e}(\Sigma) \ \forall R \in \mathcal{R} \quad \sharp(\tau)(R, \sigma) = (d, e(\sigma')) \quad \text{avec} \quad \tau(R, \sigma) = (d, \sigma')\end{array}$$

Réciproquement, pour toute implantation de $\mathbb{M}^\sharp[\rho]$, il est possible de construire une implantation de $\mathbb{M}[\rho]$ à l'aide de l'opérateur :

$$\begin{array}{l}\flat : (\mathcal{R} \times \hat{e}(\Sigma) \to \mathcal{D} \times \hat{e}(\Sigma)) \to (\mathcal{R} \times \Sigma \to \mathcal{D} \times \Sigma) \\ \forall \sigma \in \Sigma \ \forall R \in \mathcal{R} \quad \flat(\tau)(R, \sigma) = \tau(R, e(\sigma))\end{array}$$

On montre que les opérateurs $\sharp$ et $\flat$, lorsqu'ils sont appliqués à des implantations correctes, permettent d'obtenir des implantations correctes.

---

[1] Une relation $R \subseteq X \times Y$ est dite totale à gauche ssi pour tout $x$ dans $X$ il existe un $y$ dans $Y$ tel que $(x, y) \in R$.

[2] Une relation $R \subseteq X \times Y$ est dite fonctionnelle ssi pour tout $x$ dans $X$ et pour tout $y$ et $z$ dans $Y$ si $(x, y) \in R$ et $(x, z) \in R$ alors $y = z$.

[3] Une relation $R \subseteq X \times Y$ est injective ssi pour tout $x$ et $z$ dans $X$ et $y$ dans $Y$, si $(x, y) \in R$ et $(z, y) \in R$ alors $x = z$.

**Proposition D.2** *Soient* $\mathbb{M}[\rho]$ *un modèle et* $\mathbb{M}^\sharp[\rho]$ *la réduction de ce modèle.*

1. $\mathbb{M}^\sharp[\rho] \vdash (\tau, \Sigma_I) \Rightarrow (\mathbb{M}[\rho] \vdash (\flat(\tau), \Sigma_I) \wedge \equiv_\iota \vdash \flat(\tau))$
2. $(\mathbb{M}[\rho] \vdash (\tau, \Sigma_I) \wedge \equiv_\iota \vdash \tau) \Rightarrow \mathbb{M}^\sharp[\rho] \vdash (\sharp(\tau), \hat{e}(\Sigma_I))$

On montre d'autre part que l'opérateur $\sharp$ est monotone pour le préordre $\sqsubseteq$ sur les implantations.

Prouver qu'un modèle est plus restrictif qu'un autre nécessite de prendre en compte les propriétés relatives à la relation $\equiv_\iota$ ce qui peut compliquer considérablement la preuve. Cette difficulté disparaît dans le cas où cette relation correspond à l'égalité, c'est-à-dire lorsque les modèles considérés sont des modèles réduits. En effet, pour les modèles réduits, nous pouvons introduire un préordre $\leqq^\sharp$ défini comme suit : $\mathbb{M}_1^\sharp[\rho_1] \leqq^\sharp \mathbb{M}_2^\sharp[\rho_2]$, ssi il existe une relation $\kappa_\Sigma$ totale à gauche, injective et fonctionnelle telle que :

$$\begin{array}{ll} & \forall \tau_1 : \mathcal{R} \times \Sigma_1 \to \mathcal{D} \times \Sigma_1 \ \forall \Sigma_I^1 \subseteq \Sigma_1 \\ & \mathbb{M}_1^\sharp[\rho_1] \vdash (\tau_1, \Sigma_I^1) \\ \Rightarrow & \exists \tau_2 : \mathcal{R} \times \Sigma_2 \to \mathcal{D} \times \Sigma_2 \ \exists \Sigma_I^2 \subseteq \Sigma_2 \\ & \mathbb{M}_2^\sharp[\rho_2] \vdash (\tau_2, \Sigma_I^2) \wedge (\tau_1, \Sigma_I^1) \overset{\kappa_\Sigma}{\to} (\tau_2, \Sigma_I^2)) \end{array}$$

Il est alors possible de montrer qu'un modèle est plus restrictif qu'un autre (selon la définition 2.3) ssi le modèle réduit du premier est plus restrictif que le modèle réduit du deuxième (selon $\leqq^\sharp$). Pour cela, on montre tout d'abord que si une implantation $I_1$ est simulée par une implantation $I_2$, alors l'implantation "réduite" de $I_1$ est simulée par l'implantation "réduite" de $I_2$ :

$$(\tau_1, \Sigma_I^1) \overset{\kappa_\Sigma}{\to} (\tau_2, \Sigma_I^2)) \Rightarrow (\sharp(\tau_1), \hat{e}_1(\Sigma_I^1)) \overset{\kappa_\Sigma^\uparrow}{\to} (\sharp(\tau_2), \hat{e}_2(\Sigma_I^2))$$

où $\kappa_\Sigma^\uparrow = \{(e_1(\sigma_1), e_2(\sigma_2)) \mid (\sigma_1, \sigma_2) \in \kappa_\Sigma\}$.

Ce résultat nous permet d'établir que deux modèles sont ordonnés par $\leqq$ ssi leurs modèles réduits respectifs le sont par $\leqq^\sharp$.

**Proposition D.3** $\mathbb{M}_1[\rho_1] \leqq \mathbb{M}_2[\rho_2] \Leftrightarrow \mathbb{M}_1^\sharp[\rho_1] \leqq^\sharp \mathbb{M}_2^\sharp[\rho_2]$

### Propriétés sur les relations de simulation

En fonction des propriétés vérifiées par la relation de simulation $\kappa_\Sigma \subseteq \Sigma_1 \times \Sigma_2$, la proposition D.4, que nous présentons ici, permet de comparer deux modèles $\mathbb{M}_1[\rho_1]$ et $\mathbb{M}_2[\rho_2]$ sans avoir à considérer toutes les implantations de $\mathbb{M}_1[\rho_1]$. Les "bonnes propriétés" que la relation $\kappa_\Sigma$ doit vérifier sont définies comme suit :

- $\kappa_\Sigma$ est $\mathcal{W}$-monotone ssi :

$$\begin{array}{l} \forall \sigma_1, \sigma_1' \in \Sigma_1 \ \forall \sigma_2, \sigma_2' \in \Sigma_2 \\ (\mathcal{W}(\sigma_1) \subseteq \mathcal{W}(\sigma_1') \wedge (\sigma_1, \sigma_2) \in \kappa_\Sigma \wedge (\sigma_1', \sigma_2') \in \kappa_\Sigma) \Rightarrow \mathcal{W}(\sigma_2) \subseteq \mathcal{W}(\sigma_2') \end{array}$$

- $\kappa_\Sigma$ est $\Omega$-préservante ssi :

$$\forall \sigma_1 \in \Sigma_1 \ \forall \sigma_2 \in \Sigma_2 \quad ((\sigma_1, \sigma_2) \in \kappa_\Sigma \wedge \Omega_1(\sigma_1)) \Rightarrow \Omega_2(\sigma_2)$$

– $\kappa_\Sigma$ est $[\![\mathcal{R}]\!]_\Sigma$-préservante ssi :

$$\forall \sigma_1 \in \Sigma_1\ \forall \sigma_2 \in \Sigma_2\ \forall R \in \mathcal{R}$$
$$((\sigma_1, \sigma_2) \in \kappa_\Sigma \wedge (R, \sigma_1) \in [\![\mathcal{R}]\!]_{\Sigma_1}) \Rightarrow (R, \sigma_2) \in [\![\mathcal{R}]\!]_{\Sigma_2}$$

La proposition suivante montre que si une relation de simulation $\kappa_\Sigma$ entre deux modèles $\mathbb{M}_1[\rho_1]$ et $\mathbb{M}_2[\rho_2]$ est $\mathcal{W}$-monotone, il est possible de prouver que toute implantation correcte de $\mathbb{M}_1[\rho_1]$ inférieure selon $\sqsubseteq$ à une implantation simulable par une implantation correcte de $\mathbb{M}_2[\rho_2]$ est également simulable. Si la relation $\kappa_\Sigma$ est $\Omega$-préservante et $[\![R]\!]_\Sigma$-préservante, alors on peut montrer directement que toute implantation correcte de $\mathbb{M}_1[\rho_1]$ est simulable par une implantation correcte de $\mathbb{M}_2[\rho_2]$.

**Proposition D.4** *Soient $\mathbb{M}_1[\rho_1]$ et $\mathbb{M}_2[\rho_2]$ deux modèles de contrôle d'accès et $\kappa_\Sigma \subseteq \Sigma_1 \times \Sigma_2$ une relation totale à gauche, injective, fonctionnelle et $\mathcal{W}$-monotone.*

1. $$\forall (\tau_1, \Sigma_I^1), (\tau_1', \Sigma_I'^1), (\tau_2', \Sigma_I'^2)$$
$$\left( \begin{array}{ll} & \mathbb{M}_1[\rho_1] \vdash (\tau_1, \Sigma_I^1) \wedge \mathbb{M}_1[\rho_1] \vdash (\tau_1', \Sigma_I'^1) \wedge \mathbb{M}_2[\rho_2] \vdash (\tau_2', \Sigma_I'^2) \\ \wedge & (\tau_1, \Sigma_I^1) \sqsubseteq (\tau_1', \Sigma_I'^1) \wedge (\tau_1', \Sigma_I'^1) \overset{\kappa_\Sigma}{\rightarrow} (\tau_2', \Sigma_I'^2) \end{array} \right)$$
$$\Rightarrow \quad \exists (\tau_2, \Sigma_I^2) \quad \mathbb{M}_2[\rho_2] \vdash (\tau_2, \Sigma_I^2) \wedge (\tau_1, \Sigma_I^1) \overset{\kappa_\Sigma}{\rightarrow} (\tau_2, \Sigma_I^2)$$
2. *Si $\kappa_\Sigma$ est $\Omega$-préservante et $[\![\mathcal{R}]\!]_\Sigma$-préservante, alors pour toute implantation correcte $(\tau_1, \Sigma_I^1)$, il existe une implantation correcte $(\tau_2, \Sigma_I^2)$ telle que $(\tau_1, \Sigma_I^1) \overset{\kappa_\Sigma}{\rightarrow} (\tau_2, \Sigma_I^2)$.*

Ce résultat permet de montrer que si une implantation correcte d'un modèle est simulable, alors toute implantation qui lui est inférieure est aussi simulable. Il permet également de montrer que si la relation de simulation respecte des "bonnes propriétés", alors toute implantation du modèle est simulable. Nous pouvons à présent montrer que si pour toute implantation d'un modèle $\mathbb{M}_1[\rho_1]$, il existe une implantation qui lui est supérieure par $\sqsubseteq$ et qui est simulable par une implantation de $\mathbb{M}_2[\rho_2]$, alors $\mathbb{M}_1[\rho_1] \leqq \mathbb{M}_2[\rho_2]$. Nous montrons également que si la relation de simulation vérifie de "bonnes propriétés", on obtient directement $\mathbb{M}_1[\rho_1] \leqq \mathbb{M}_2[\rho_2]$.

**Proposition D.5** *Soient $\mathbb{M}_1[\rho_1]$ et $\mathbb{M}_2[\rho_2]$ deux modèles de contrôle d'accès et $\kappa_\Sigma \subseteq \Sigma_1 \times \Sigma_2$ une relation totale à gauche, $\mathcal{W}$-monotone, $\iota$-fonctionnelle et $\iota$-injective.*

1. *Si pour toute implantation $(\tau_1, \Sigma_I^1) \equiv_{\iota_1}$-préservante telle que $\mathbb{M}_1[\rho_1] \vdash (\tau_1, \Sigma_I^1)$,*
   – *il existe une implantation $(\tau_1', \Sigma_I'^1) \equiv_{\iota_1}$-préservante telle que $\mathbb{M}_1[\rho_1] \vdash (\tau_1', \Sigma_I'^1)$ et $(\tau_1, \Sigma_I^1) \sqsubseteq (\tau_1', \Sigma_I'^1)$*
   – *et s'il existe une implantation $(\tau_2', \Sigma_I'^2) \equiv_{\iota_2}$-préservante telle que $\mathbb{M}_2[\rho_2] \vdash (\tau_2', \Sigma_I'^2)$ vérifiant $(\tau_1', \Sigma_I'^1) \overset{\kappa_\Sigma}{\rightarrow} (\tau_2', \Sigma_I'^2)$,*
   *alors $\mathbb{M}_1[\rho_1] \leqq \mathbb{M}_2[\rho_2]$.*
2. *Si $\kappa_\Sigma$ est $\Omega$-préservante et $[\![\mathcal{R}]\!]_\Sigma$-préservante, alors $\mathbb{M}_1[\rho_1] \leqq \mathbb{M}_2[\rho_2]$.*

Ainsi, avec la proposition D.5, pour montrer qu'un modèle est plus restrictif qu'un autre, il suffit de montrer que toutes les implantations $\sqsubseteq$-maximales[4] sont simulables. De plus, si la relation de simulation préserve la politique de sécurité ainsi que la sémantique des requêtes, alors il est possible de montrer directement, sans considérer les implantations, qu'un modèle est plus restrictif qu'un autre.

[4]Une implantation $(\tau, \Sigma_I)$ d'un modèle $\mathbb{M}[\rho]$ est $\sqsubseteq$-maximale ssi il n'existe pas une autre implantation $(\tau', \Sigma_I')$ de $\mathbb{M}[\rho]$ telle que $(\tau, \Sigma_I) \sqsubset (\tau', \Sigma_I')$.

**Application : Méthodologie et exemples**

La démarche que nous avons adoptée pour établir qu'un modèle $\mathbb{M}_1[\rho_1]$ est plus restrictif qu'un modèle $\mathbb{M}_2[\rho_2]$ se décompose en deux étapes principales. Dans un premier temps, on construit un modèle intermédiaire $\mathbb{M}_{12}[\rho_2]$ correspondant au modèle $\mathbb{M}_1[\rho_1]$ exprimé dans le formalisme de $\mathbb{M}_2[\rho_2]$. Pour ce faire, il faut bien sûr commencer par interpréter le paramètre de sécurité $\rho_1$ par un paramètre de sécurité $\rho_2 = \kappa_\rho(\rho_1)$, puis considérer les états. Cette interprétation permet de définir un prédicat de sécurité $\Omega_{12}$ sur $\Sigma_2$ à partir duquel le modèle $\mathbb{M}_{12}[\kappa_\rho(\rho_1)]$ est défini. Ce mécanisme de traduction permet de définir une relation de simulation à partir de laquelle on établit $\mathbb{M}_1[\rho_1] \leqq \mathbb{M}_{12}[\kappa_\rho(\rho_1)]$. Il faut ensuite montrer que la politique définie par le prédicat $\Omega_{12}$ est plus restrictive que celle définie par $\Omega_2$, c'est-à-dire :

$$\forall \sigma \in \Sigma_2 \quad \Omega_{12}(\sigma) \Rightarrow \Omega_2(\sigma)$$

Dans ce cas, on montre facilement que $\mathbb{M}_{12}[\kappa_\rho(\rho_1)] \leqq \mathbb{M}_2[\kappa_\rho(\rho_1)]$, ce qui nous permet d'établir :

$$\forall \rho_1 \; \exists \rho_2 \quad \mathbb{M}_1[\rho_1] \leqq \mathbb{M}_2[\rho_2]$$

La méthodologie esquissée ci-dessus a été utilisée avec succès pour comparer entre eux les modèles de la Muraille de Chine, de Bell et LaPadula et RBAC. Nous présentons ici les grandes lignes du raisonnement permettant de montrer que le modèle $\mathbb{M}^{acc}_{CW}[\rho_{CW}]$ de la Muraille de Chine est strictement plus restrictif que le modèle $\mathbb{M}^{acc}_{BLP}[\rho_{BLP}]$ de Bell et LaPadula.

*Traduction : construction d'un modèle intermédiaire.* Pour montrer que le modèle de la Muraille de Chine est plus restrictif que le modèle de Bell et LaPadula, il faut tout d'abord donner une interprétation des concepts de $\mathbb{M}^{acc}_{CW}[\rho_{CW}]$ par des concepts de $\mathbb{M}^{acc}_{BLP}[\rho_{BLP}]$. Il s'agit d'exprimer à l'aide d'un treillis de niveaux de sécurité $\rho_{LCW}$ les notions de compagnies et de classes de conflits présentes dans le paramètre $\rho_{CW}$ de la Muraille de Chine. Ce treillis a bien sûr une forme particulière puisqu'il est issu d'une fonction de traduction $\kappa_\rho$ qui permet d'obtenir un treillis $\kappa_\rho(\rho_{CW}) = \rho_{LCW}$ à partir du paramètre $\rho_{CW}$. On cherche donc à montrer que $\mathbb{M}^{acc}_{CW}[\rho_{CW}] \leqq \mathbb{M}^{acc}_{BLP}[\rho_{LCW}]$. Il s'agit alors de spécifier comment représenter un état du système décrit dans le formalisme de la Muraille de Chine par un état "équivalent" décrit dans le formalisme de Bell et LaPadula. Concrètement, on définit donc une relation $\kappa_\Sigma \subseteq \Sigma_{CW} \times \Sigma_{BLP}$ qui permet de relier les états des deux formalismes et qui sera utilisée comme relation de simulation par la suite. Il reste alors à reformuler le prédicat $\Omega_{CW}$ qui spécifie la politique de la Muraille de Chine par un prédicat $\Omega_{LCW}$ exprimé dans le formalisme de Bell et LaPadula. On obtient finalement un nouveau modèle $\mathbb{M}^{acc}_{LCW}[\rho_{LCW}]$. C'est à partir de ce modèle intermédiaire que nous allons pouvoir montrer que $\mathbb{M}^{acc}_{CW}[\rho_{CW}] \leqq \mathbb{M}^{acc}_{BLP}[\rho_{LCW}]$. Nous procédons en deux étapes : nous montrons tout d'abord $\mathbb{M}^{acc}_{CW}[\rho_{CW}] \leqq \mathbb{M}^{acc}_{LCW}[\rho_{LCW}]$ puis nous montrons $\mathbb{M}^{acc}_{LCW}[\rho_{LCW}] \leqq \mathbb{M}^{acc}_{BLP}[\rho_{LCW}]$.

*Comparaison du modèle original avec le modèle intermédiaire.* Nous avons montré que $\mathbb{M}^{acc}_{CW}[\rho_{CW}] \leqq \mathbb{M}^{acc}_{LCW}[\rho_{LCW}]$ en utilisant la proposition D.5. Nous avons donc tout d'abord montré que la relation $\kappa_\Sigma$ définie à l'étape précédente était totale à gauche, $\mathcal{W}$-monotone, $\iota$-fonctionnelle et $\iota$-injective. Nous avons ensuite prouvé le résultat de deux manières différentes.

1. Nous avons prouvé que toute implantation correcte de la Muraille de Chine est inférieure (selon $\sqsubseteq$) à l'implantation classique $(\tau_{CW}, \Sigma^I_{CW})$ de ce modèle (i.e. $(\tau_{CW}, \Sigma^I_{CW})$ est la seule implantation maximale de la Muraille de Chine), et qu'il existe une implantation $(\tau_{LCW}, \Sigma^I_{LCW})$ correcte du modèle de de Bell et LaPadula qui simule $(\tau_{CW}, \Sigma^I_{CW})$. Le point 1) de la proposition D.5 nous permet alors de conclure que $\mathbb{M}^{acc}_{CW}[\rho_{CW}] \leqq \mathbb{M}^{acc}_{LCW}[\rho_{LCW}]$.
2. Nous avons prouvé que la relation $\kappa_\Sigma$ est $\Omega$-préservante et $[\![\mathcal{R}]\!]_\Sigma$-préservante. Le point 2) de la proposition D.5 nous permet alors de conclure que $\mathbb{M}^{acc}_{CW}[\rho_{CW}] \leqq \mathbb{M}^{acc}_{LCW}[\rho_{LCW}]$.

*Comparaison du modèle intermédiaire avec le modèle "cible".* Une fois la politique de la Muraille de Chine exprimée par le prédicat $\Omega_{LCW}$ dans le formalisme du modèle de Bell et LaPadula, il est facile de montrer que tout état vérifiant le prédicat $\Omega_{LCW}$ vérifie aussi le prédicat définissant la politique de Bell et LaPadula et on montre alors facilement que $\mathbb{M}^{acc}_{LCW}[\rho_{LCW}] \leqq \mathbb{M}^{acc}_{BLP}[\rho_{LCW}]$ ce qui, par transitivité, nous permet finalement d'établir que $\mathbb{M}^{acc}_{CW}[\rho_{CW}] \leqq \mathbb{M}^{acc}_{BLP}[\rho_{LCW}]$.

Cette méthodologie repose sur l'idée de séparer de manière claire les étapes de traduction et de comparaison. En revanche, pour montrer qu'un modèle $\mathbb{M}_1[\rho_1]$ n'est pas plus restrictif qu'un modèle $\mathbb{M}_2[\rho_2]$, on procède de manière différente. En effet, pour ce faire, on montre que :

$$\exists \rho_1 \ \forall \rho_2 \quad \mathbb{M}_1[\rho_1] \not\leqq \mathbb{M}_2[\rho_2]$$

Il suffit donc de concevoir une valeur pour $\rho_1$ qui ne puisse pas être associée à un paramètre $\rho_2$ tel que $\mathbb{M}_1[\rho_1] \leqq \mathbb{M}_2[\rho_2]$. On montre en fait par un argument de cardinalité sur les ensembles d'états qu'il n'existe pas de relation de simulation permettant d'établir $\mathbb{M}_1[\rho_1] \leqq \mathbb{M}_2[\rho_2]$. Cette étape est grandement facilitée si l'on considère les modèles réduits (la proposition D.3 garantit la validité de ce procédé). Ainsi, pour montrer que $\mathbb{M}^{acc}_{BLP}[\rho_{BLP}] \not\leqq \mathbb{M}^{acc}_{CW}[\rho_{CW}]$, nous nous sommes appuyés sur le fait que dans le modèle de la Muraille Chine, il n'était pas possible d'interdire un accès en lecture si aucun autre accès n'est présent, contrairement au modèle de Bell et La Padula. Ainsi, pour la Muraille de Chine, le cardinal de l'ensemble des classes d'équivalence selon $\equiv_{\mathcal{W}}$ est inférieur au cardinal de l'ensemble des classes d'équivalence selon $\equiv_{\mathcal{W}}$ pour le modèle de Bell et LaPadula, ce qui permet de montrer facilement qu'il n'existe pas de relation de simulation $\iota$-injective.

En suivant cette méthodologie, nous avons également montré que :

$$\mathbb{M}^{acc}_{BLP}[\rho_{BLP}] \leqq \mathbb{M}^{acc}_{RBLP}[\rho_{RBLP}] \leqq \mathbb{M}^{acc}_{RBAC}[\rho_{RBLP}]$$

$$\mathbb{M}^{acc}_{RBAC}[\rho_{RBAC}] \not\leqq \mathbb{M}^{acc}_{BLP}[\rho_{BLP}]$$

où $\mathbb{M}^{acc}_{RBLP}[\rho_{RBLP}]$ représente l'interprétation à base de rôles du modèle de Bell et LaPadula. Il est alors possible de déduire que $\mathbb{M}^{acc}_{CW}[\rho_{CW}] \leqq \mathbb{M}^{acc}_{RBAC}[\rho_{RBAC}]$.

Ainsi, ces comparaisons nous permettent d'établir formellement qu'un système régi par le modèle de la Muraille de Chine peut être obtenu à partir d'un système régi par le modèle de Bell & LaPadula qui peut lui même être obtenu à partir d'un système régi par un modèle à base de rôle. De plus, nous avons également montré qu'il n'était pas possible d'implanter un système à base de rôles dans le formalisme de Bell & LaPadula et qu'il n'était pas

non plus possible d'implanter un système régi par le modèle de Bell & LaPadula dans le formalisme de la Muraille de Chine. Ces résultats fournissent des critères de choix entre ces trois modèles en indiquant ce qu'il est possible de faire avec. Il reste alors au concepteur d'un système à choisir le formalisme le mieux adapté à ses besoins en tenant compte de ces résultats.

### D.2.2 Simulation faible d'implantations

Nous détaillons à présent le mécanisme de comparaison introduit dans la section 2.2.2 et illustrons son utilisation pour la comparaison des modèles HRU, de Bell & LaPadula et à base de rôles.

#### Propriétés sur les relations de simulation

La relation de préordre $\trianglelefteq$ introduite dans la section 2.2.2 repose sur deux relations : la première, notée $\kappa_\Sigma$, permet de relier les états des deux modèles à comparer, la deuxième, notée $\kappa_\mathcal{R}$, permet d'établir une correspondance "sémantique" entre les langages de requêtes des deux modèles à comparer. Nous introduisons ici les propriétés que doivent vérifier ces relations pour permettre la comparaison de deux modèles sans avoir à considérer toutes les implantations. D'une part, la relation $\kappa_\Sigma$ doit être $\Omega$-préservante et $\mathcal{W}$-préservante, et d'autre part la relation $\kappa_\mathcal{R}$ doit satisfaire les deux propiétés suivantes.

- La sémantique des requêtes, qui sont en relation par $\kappa_\mathcal{R}$, doit préserver le prédicat de sécurité lorsque ces requêtes sont appliquées sur des états en relation par $\kappa_\Sigma$. Dans ce cas, on dit que $\kappa_\mathcal{R}$ est $\Omega$-préservante :

$$\begin{array}{l}\forall R_1 \in \mathcal{R}_1 \quad \forall (R_2^1,\ldots,R_2^n) \in \mathcal{R}_2^\star \quad \forall \sigma_1,\sigma_1' \in \Sigma_1 \\ \big((\sigma_1,R_1,(R_2^1,\ldots,R_2^n)) \in \kappa_\mathcal{R} \wedge \Omega_1(\sigma_1) \wedge (\sigma_1,R_1,\sigma_1') \in [\![\mathcal{R}_1]\!]_{\Sigma_1} \wedge \Omega_1(\sigma_1')\big) \\ \Rightarrow \left( \begin{array}{l} \forall \sigma_2^0,\sigma_2^1,\ldots,\sigma_2^n \in \Sigma_2 \\ \left( \begin{array}{rl} & \Omega_2(\sigma_2^0) \wedge (\sigma_1,\sigma_2^0) \in \kappa_\Sigma \\ \wedge & (\sigma_2^0,R_2^1,\sigma_2^1) \in [\![\mathcal{R}_2]\!]_{\Sigma_2} \\ \wedge & \cdots \wedge (\sigma_2^{n-1},R_2^n,\sigma_2^n) \in [\![\mathcal{R}_2]\!]_{\Sigma_2} \end{array} \right) \Rightarrow \big( \quad \Omega_2(\sigma_2^1) \wedge \cdots \wedge \Omega_2(\sigma_2^n) \ \big) \end{array} \right) \end{array}$$

- La sémantique des requêtes, qui sont en relation par $\kappa_\mathcal{R}$, doit préserver la relation $\kappa_\Sigma$. Dans ce cas, on dit que $\kappa_\mathcal{R}$ est $\kappa_\Sigma$-préservante :

$$\begin{array}{l}\forall R_1 \in \mathcal{R}_1 \ \ \forall (R_2^1,\ldots,R_2^n) \in \mathcal{R}_2^\star \ \ \forall \sigma_1,\sigma_1' \in \Sigma_1 \ \ \forall \sigma_2^0,\sigma_2^1,\ldots,\sigma_2^{n-1},\sigma_2^n \in \Sigma_2 \\ \left( \begin{array}{rl} & (\sigma_1,R_1,\sigma_1') \in [\![\mathcal{R}_1]\!]_{\Sigma_1} \wedge (\sigma_2^0,R_2^1,\sigma_2^1) \in [\![\mathcal{R}_2]\!]_{\Sigma_2} \wedge \cdots \\ \wedge & (\sigma_2^{n-1},R_2^n,\sigma_2^n) \in [\![\mathcal{R}_2]\!]_{\Sigma_2} \wedge (\sigma_1,R_1,(R_2^1,\ldots,R_2^n)) \in \kappa_\mathcal{R} \wedge (\sigma_1,\sigma_2^0) \in \kappa_\Sigma \end{array} \right) \\ \Rightarrow (\sigma_1',\sigma_2^n) \in \kappa_\Sigma \end{array}$$

On montre alors la proposition suivante.

**Proposition D.6** *Soit deux modèles $\mathbb{M}_1[\rho_1] = (\mathbb{P}_1[\rho_1],\mathcal{R}_1)$ et $\mathbb{M}_2[\rho_2] = (\mathbb{P}_2[\rho_2],\mathcal{R}_2)$ tels que $[\![\mathcal{R}_1]\!]_{\Sigma_1}$ et $[\![\mathcal{R}_2]\!]_{\Sigma_2}$ soient fonctionnelles et totales à gauche*[5]*. Si il existe deux relations totales à gauche $\kappa_\Sigma \subseteq \Sigma_1 \times \Sigma_2$ et $\kappa_\mathcal{R} \subseteq \Sigma_1 \times \mathcal{R}_1 \times \mathcal{R}_2^\star$ telles que $\kappa_\Sigma$ est $\Omega$-préservante et $\mathcal{W}$-préservante et $\kappa_\mathcal{R}$ est $\Omega$-préservante et $\kappa_\Sigma$-préservante, alors $\mathbb{M}_1[\rho_1] \trianglelefteq \mathbb{M}_2[\rho_2]$.*

[5]Une relation ternaire $R \subseteq A \times B \times C$ est dite totale à gauche ssi pour tout $a \in A$ et $b \in B$, il existe $c \in C$ tel que $(a,b,c) \in R$.

La preuve s'obtient en montrant que chaque implantation $(\tau_1, \Sigma_1^I)$ du modèle $\mathbb{M}_1[\rho_1]$ peut être simulée à partir de $\kappa_\Sigma$ et de la relation $\kappa'_\mathcal{R}$ définie à partir de $\kappa_\mathcal{R}$ et de $\tau_1$ comme suit :

$$\forall \sigma_1 \in \Sigma_1 \quad \forall R_1 \in \mathcal{R}_1 \quad \forall (R_2^1, \ldots, R_2^n) \in \mathcal{R}_2^\star$$
$$(\sigma_1, R_1, (R_2^1, \ldots, R_2^n)) \in \kappa'_\mathcal{R} \Leftrightarrow \left( \begin{array}{lll} & \left( \begin{array}{ll} & \tau_1(R_1, \sigma_1) = (\mathsf{yes}, \_) \\ \wedge & (\sigma_1, R_1, (R_2^1, \ldots, R_2^n)) \in \kappa_\mathcal{R} \end{array} \right) \\ \vee & \left( \begin{array}{ll} & \tau_1(R_1, \sigma_1) = (\mathsf{no}, \sigma_1) \\ \wedge & (R_2^1, \ldots, R_2^n) = (Fail) \end{array} \right) \end{array} \right)$$

En pratique, il n'est donc pas nécessaire de définir une relation $\kappa'_\mathcal{R}$ pour chaque implantation à simuler, cette relation s'obtient directement à partir de l'implantation à simuler et de la relation $\kappa_\mathcal{R}$ définie indépendamment des implantations.

**Applications**

Nous illustrons ici l'utilisation de la proposition D.6 pour comparer les modèles HRU, de Bell & LaPadula et RBAC.

*Langages de requêtes administratives.* Les requêtes appartenant à l'ensemble $\mathcal{R}^{acc}$, défini en (2.1), permettant à un sujet de demander un accès ou le relâchement d'un accès, sont généralement accompagnées par un ensemble de requêtes, appelées requêtes administratives, permettant la modification des fonctions de sécurité. Bien sûr, l'application de ces requêtes doit être régie par une politique administrative. Nous donnons ici trois langages de requêtes administratives $\mathcal{R}^{adm}_{HRU}$, $\mathcal{R}^{adm}_{BLP}$ et $\mathcal{R}^{adm}_{RBAC}$ pour les politiques HRU, RBAC et de Bell et LaPadula ainsi que la définition d'une sémantique pour ces requêtes. La syntaxe (accompagnée d'une sémantique informelle) des requêtes considérées pour chacune de ces politiques est introduite dans la table D.5 tandis que la sémantique formelle de ces requêtes est définie dans la table D.6. Les modèles de contrôle d'accès que nous considérons ici sont obtenus à partir des ensembles de requêtes suivantes

$$\begin{array}{rcl} \mathcal{R}_{HRU} & = & \mathcal{R}^{acc} \cup \mathcal{R}^{adm}_{HRU} \cup \{Success, Fail\} \\ \mathcal{R}_{RBAC} & = & \mathcal{R}^{acc} \cup \mathcal{R}^{adm}_{RBAC} \cup \{Success, Fail\} \\ \mathcal{R}_{BLP} & = & \mathcal{R}^{acc} \cup \mathcal{R}^{adm}_{BLP} \cup \{Success, Fail\} \end{array}$$

où $Success$ et $Fail$ sont les requêtes présentées page 30.

*Paramètres de sécurité.* Prouver qu'un modèle $\mathbb{M}_1[\rho_1]$ est plus restrictif qu'un autre modèle consiste tout d'abord à interpréter le paramètre de sécurité $\rho_1$ par un paramètre de sécurité $\rho_2 = \kappa_\rho(\rho_1)$ exprimé dans le formalisme du deuxième modèle. Il s'agit alors de montrer que $\mathbb{M}_1[\rho_1] \trianglelefteq \mathbb{M}_2[\kappa_\rho(\rho_1)]$.

Lors de la comparaison des modèles HRU et RBAC, il suffit de montrer comment obtenir un paramètre de sécurité pour RBAC : les utilisateurs sont les sujets, chaque sujet est associé à un rôle (les rôles sont donc identifiés aux sujets) et la relation d'ordre sur les rôles est vide. Plus formellement, nous considérons le paramètre $\rho_1 = (\mathcal{S}, \mathcal{S}, \emptyset)$, et nous montrerons que :

$$\begin{array}{rcl} \mathbb{M}_{HRU}[\,] = (\mathbb{P}_{HRU}[\,], \mathcal{R}_{HRU}) & \trianglelefteq & \mathbb{M}_{RBAC}[\rho_1] = (\mathbb{P}_{RBAC}[\rho_1], \mathcal{R}_{RBAC}) \\ \mathbb{M}_{RBAC}[\rho_{RBAC}] = (\mathbb{P}_{RBAC}[\rho_{RBAC}], \mathcal{R}_{RBAC}) & \trianglelefteq & \mathbb{M}_{HRU}[\,] = (\mathbb{P}_{HRU}[\,], \mathcal{R}_{HRU}) \end{array}$$

| | |
|---|---|
| $\mathcal{R}^{adm}_{HRU}$ | |
| $\langle +_{m_D}, s, s', o, a\rangle$ | $s$ demande l'ajout de $(s', o, a)$ dans $m_D$ |
| $\langle -_{m_D}, s, s', o, a\rangle$ | $s$ demande le retrait de $(s', o, a)$ de $m_D$ |
| $\mathcal{R}^{adm}_{BLP}$ | |
| $\langle +_{f_s}, s, s', l\rangle$ | $s$ demande à affecter à $f_s(s')$ la valeur $l$ |
| $\langle +_{f_o}, s, o, l\rangle$ | $s$ demande à affecter à $f_o(o)$ la valeur $l$ |
| $\mathcal{R}^{adm}_{RBACB}$ | |
| $\langle +_{\mathsf{UA}}, s, u, r\rangle$ | $s$ demande l'ajout de $(u, r)$ dans UA |
| $\langle -_{\mathsf{UA}}, s, u, r\rangle$ | $s$ demande le retrait de $(u, r)$ de UA |
| $\langle +_{\mathsf{PA}}, s, o, a, r\rangle$ | $s$ demande l'ajout de $((o, a), r)$ dans PA |
| $\langle -_{\mathsf{PA}}, s, o, a, r\rangle$ | $s$ demande le retrait de $((o, a), r)$ de PA |
| $\langle +_{roles}, s, s', r\rangle$ | $s$ demande l'ajout de $r$ dans $roles(s')$ |
| $\langle -_{roles}, s, s', r\rangle$ | $s$ demande le retrait de $r$ de $roles(s')$ |

TAB. D.5 – Requêtes administratives

ce qui permet d'établir que les modèles HRU et RBAC ont le même pouvoir d'expression. Ainsi, le choix d'un de ces deux modèles n'a pas d'incidence sur ce qu'il sera possible de spécifier avec, la seule différence réside dans le formalisme fourni par ces modèles.

Nous montrerons également que le modèle de Bell et LaPadula est strictement plus restrictif que le modèle RBAC. Aussi, étant donné le paramètre de sécurité $\rho_{BLP} = (\mathcal{L}, \preceq, \curlyvee, \curlywedge)$, nous définissons le paramètre $\rho_3 = (\mathcal{S}, \mathcal{L}, \preceq)$ utile pour définir un modèle à base de rôles : les utilisateurs sont les sujets, chaque rôle correspond à un niveau de sécurité et la hiérarchie des rôles coïncide avec l'ordre $\preceq$ sur les niveaux de sécurité. Nous montrerons donc que :

$$\mathbb{M}_{BLP}[\rho_{BLP}] = (\mathbb{P}_{BLP}[\rho_{BLP}], \mathcal{R}_{BLP}) \triangleleft \mathbb{M}_{RBAC}[\rho_3] = (\mathbb{P}_{RBAC}[\rho_3], \mathcal{R}_{RBAC})$$

*Relation de simulation entre états.* Les relations $\kappa^1_\Sigma \subseteq \Sigma_{HRU} \times \Sigma_{RBAC}$, $\kappa^2_\Sigma \subseteq \Sigma_{RBAC} \times \Sigma_{HRU}$ et $\kappa^3_\Sigma \subseteq \Sigma_{BLP} \times \Sigma_{RBAC}$, permettant de relier les états "sémantiquement" équivalents, sont définies dans la table D.7. Nous montrons que ces relations satisfont les propriétés requises pour l'application de la proposition D.6.

**Lemme D.3** *$\kappa^1_\Sigma$, $\kappa^2_\Sigma$ et $\kappa^3_\Sigma$ sont $\Omega$-préservantes et $\mathcal{W}$-préservantes.*

A partir de la définition de la relation $\kappa_\Sigma \subseteq \Sigma_1 \times \Sigma_2$, il est possible d'exprimer le prédicat de sécurité $\Omega_1$ sur $\Sigma_1$ par un prédicat $\Omega_{12}$ sur $\Sigma_2$. Le prédicat $\Omega_{12}$ doit vérifier les deux propriétés suivantes :

$$\begin{array}{l} \forall \sigma_1 \in \Sigma_1\ \forall \sigma_2 \in \Sigma_2 \quad ((\sigma_1, \sigma_2) \in \kappa_\Sigma \wedge \Omega_1(\sigma_1)) \Rightarrow \Omega_{12}(\sigma_2) \\ \forall \sigma_2 \in \Sigma_2\ \Omega_{12}(\sigma_2) \Rightarrow \Omega_2(\sigma_2) \end{array} \tag{D.1}$$

La table D.8 contient les définitions des prédicats $\Omega^1$ exprimant la politique HRU dans le formalisme de la politique RBAC paramétrée par $\rho_1 = (\mathcal{S}, \mathcal{S}, \emptyset)$, $\Omega^2$ exprimant la politique RBAC dans le formalisme de la politique HRU, et $\Omega^3$ exprimant la politique de Bell et LaPadula dans le formalisme de la politique RBAC paramétrée par $\rho_3 = (\mathcal{S}, \mathcal{L}, \preceq)$. On montre que ces trois prédicats satisfont les propriétés (D.1).

$[\![\mathcal{R}^{adm}_{HRU}]\!]^{+}_{\Sigma_{HRU}}$

$((m_1, m^1_D), \langle +_{m_D}, s, s', o, a\rangle, (m_2, m^2_D)) \in [\![\mathcal{R}^{adm}_{HRU}]\!]^{+}_{\Sigma_{HRU}}$
$\Leftrightarrow (m_1 = m_2 \wedge m^2_D = m^1_D \cup \{(s', o, a)\})$

$((m_1, m^1_D), \langle -_{m_D}, s, s', o, a\rangle, (m_2, m^2_D)) \in [\![\mathcal{R}^{adm}_{HRU}]\!]^{+}_{\Sigma_{HRU}}$
$\Leftrightarrow (m_1 = m_2 \wedge m^2_D = m^1_D \backslash \{(s', o, a)\})$

$[\![\mathcal{R}^{adm}_{BLP}]\!]^{+}_{\Sigma_{BLP}}$

$((m_1, f^1_s, f^1_o), \langle +_{f_s}, s, s', l\rangle, (m_2, f^2_s, f^2_o)) \in [\![\mathcal{R}^{adm}_{BLP}]\!]^{+}_{\Sigma_{BLP}}$
$\Leftrightarrow (m_1 = m_2 \wedge f^2_s = f^1_s[s' \leftarrow l])$

$((m_1, f^1_s, f^1_o), \langle +_{f_o}, s, o, l\rangle, (m_2, f^2_s, f^2_o)) \in [\![\mathcal{R}^{adm}_{BLP}]\!]^{+}_{\Sigma_{BLP}}$
$\Leftrightarrow (m_1 = m_2 \wedge f^2_o = f^1_o[o \leftarrow l])$

$[\![\mathcal{R}^{adm}_{RBAC}]\!]^{+}_{\Sigma_{RBAC}}$

$((m_1, user^1, \mathsf{UA}^1, \mathsf{PA}^1, roles^1), \langle +_{\mathsf{UA}}, s, u, r\rangle, (m_2, user^2, \mathsf{UA}^2, \mathsf{PA}^2, roles^2)) \in [\![\mathcal{R}^{adm}_{RBAC}]\!]^{+}_{\Sigma_{RBAC}}$
$\Leftrightarrow (m_1 = m_2 \wedge user^1 = user^2 \wedge \mathsf{PA}^1 = \mathsf{PA}^2 \wedge roles^1 = roles^2 \wedge \mathsf{UA}^2 = \mathsf{UA}^1 \cup \{(u, r)\})$

$((m_1, user^1, \mathsf{UA}^1, \mathsf{PA}^1, roles^1), \langle -_{\mathsf{UA}}, s, u, r\rangle, (m_2, user^2, \mathsf{UA}^2, \mathsf{PA}^2, roles^2)) \in [\![\mathcal{R}^{adm}_{RBAC}]\!]^{+}_{\Sigma_{RBAC}}$
$\Leftrightarrow (m_1 = m_2 \wedge user^1 = user^2 \wedge \mathsf{PA}^1 = \mathsf{PA}^2 \wedge roles^1 = roles^2 \wedge \mathsf{UA}^2 = \mathsf{UA}^1 \backslash \{(u, r)\})$

$((m_1, user^1, \mathsf{UA}^1, \mathsf{PA}^1, roles^1), \langle +_{\mathsf{PA}}, s, o, a, r\rangle, (m_2, user^2, \mathsf{UA}^2, \mathsf{PA}^2, roles^2)) \in [\![\mathcal{R}^{adm}_{RBAC}]\!]^{+}_{\Sigma_{RBAC}}$
$\Leftrightarrow (m_1 = m_2 \wedge user^1 = user^2 \wedge \mathsf{UA}^1 = \mathsf{UA}^2 \wedge roles^1 = roles^2 \wedge \mathsf{PA}^2 = \mathsf{PA}^1 \cup \{((o, a), r)\})$

$((m_1, user^1, \mathsf{UA}^1, \mathsf{PA}^1, roles^1), \langle -_{\mathsf{PA}}, s, o, a, r\rangle, (m_2, user^2, \mathsf{UA}^2, \mathsf{PA}^2, roles^2)) \in [\![\mathcal{R}^{adm}_{RBAC}]\!]^{+}_{\Sigma_{RBAC}}$
$\Leftrightarrow (m_1 = m_2 \wedge user^1 = user^2 \wedge \mathsf{UA}^1 = \mathsf{UA}^2 \wedge roles^1 = roles^2 \wedge \mathsf{PA}^2 = \mathsf{PA}^1 \backslash \{((o, a), r)\})$

$((m_1, user^1, \mathsf{UA}^1, \mathsf{PA}^1, roles^1), \langle +_{roles}, s, s', r\rangle, (m_2, user^2, \mathsf{UA}^2, \mathsf{PA}^2, roles^2)) \in [\![\mathcal{R}^{adm}_{RBAC}]\!]^{+}_{\Sigma_{RBAC}}$
$\Leftrightarrow (m_1 = m_2 \wedge user^1 = user^2 \wedge \mathsf{UA}^1 = \mathsf{UA}^2 \wedge \mathsf{PA}^1 = \mathsf{PA}^2 \wedge roles^2 = roles^1[s' \leftarrow roles^1(s') \cup \{r\}])$

$((m_1, user^1, \mathsf{UA}^1, \mathsf{PA}^1, roles^1), \langle -_{roles}, s, s', r\rangle, (m_2, user^2, \mathsf{UA}^2, \mathsf{PA}^2, roles^2)) \in [\![\mathcal{R}^{adm}_{RBAC}]\!]^{+}_{\Sigma_{RBAC}}$
$\Leftrightarrow (m_1 = m_2 \wedge user^1 = user^2 \wedge \mathsf{UA}^1 = \mathsf{UA}^2 \wedge \mathsf{PA}^1 = \mathsf{PA}^2 \wedge roles^2 = roles^1[s' \leftarrow roles^1(s') \backslash \{r\}])$

TAB. D.6 – Sémantique des requêtes administratives

$$
\begin{array}{l}
\kappa^1_\Sigma \subseteq \Sigma_{HRU} \times \Sigma_{RBACB} \\
\qquad ((m_1, m_D), (m_2, user, \mathsf{UA}, \mathsf{PA}, roles)) \in \kappa^1_\Sigma \\
\quad \Leftrightarrow \quad m_1 = m_2 \wedge user = \lambda s.s \wedge \mathsf{UA} = \{(s,s) \mid s \in \mathcal{S}\} \wedge \mathsf{PA} = \{((o,a),s) \mid (s,o,a) \in m_D\} \\
\qquad \wedge\ roles = \lambda s.\{s\} \\
\hline
\kappa^2_\Sigma \subseteq \Sigma_{RBAC} \times \Sigma_{HRU} \\
\qquad ((m_1, user, \mathsf{UA}, \mathsf{PA}, roles), (m_2, m_D)) \in \kappa^2_\Sigma \\
\quad \Leftrightarrow \quad m_1 = m_2 \wedge m_D = \{(s,o,a) \mid \exists r, r' \in \mathsf{R}\ r' \leq_{\mathsf{R}} r \wedge ((o,a),r') \in \mathsf{PA} \wedge r \in roles(s)\} \\
\hline
\kappa^3_\Sigma \subseteq \Sigma_{BLP} \times \Sigma_{RBAC} \\
\qquad ((m_1, f_s, f_o), (m_2, user, \mathsf{UA}, \mathsf{PA}, roles)) \in \kappa^3_\Sigma \\
\quad \Leftrightarrow \quad m_1 = m_2 \wedge user = \lambda s.s \wedge \mathsf{UA} = \{(s, f_s(s)) \mid s \in \mathcal{S}\} \wedge roles = \lambda s.\{f_s(s)\} \\
\qquad \wedge\ \mathsf{PA} = \{((o,\mathsf{read}), f_o(o)), ((o,\mathsf{write}), f_o(o)), ((o,\mathsf{write}), \bot) \mid o \in \mathcal{O}\}
\end{array}
$$

TAB. D.7 – Etats équivalents

$$
\begin{array}{l}
\forall \sigma \in \Sigma_{RBAC}\ \Omega^1(\sigma) \Leftrightarrow \Lambda(\sigma) \subseteq \{(s,o,a) \in \mathbb{A} \mid (s,s) \in \mathsf{UA} \wedge ((o,a),s) \in \mathsf{PA} \wedge s \in roles(s)\} \\
\hline
\forall \sigma \in \Sigma_{RBAC}\ \Omega^2(\sigma) \Leftrightarrow \Omega_{HRU}(\sigma) \\
\hline
\forall \sigma \in \Sigma_{RBAC}\ \Omega^3(\sigma) \Leftrightarrow \left( \begin{array}{ll}
 & \forall s \in \mathcal{S} \quad \forall o \in \mathcal{O} \quad (s,o,\mathsf{read}) \in \Lambda(\sigma) \Rightarrow \\
 & \left( \forall r, r' \in \mathcal{L} \left( \begin{array}{ll} & r \in roles(s) \\ \wedge & ((o,\mathsf{read}), r') \in \mathsf{PA} \end{array} \right) \Rightarrow r' \preceq r \right) \\
\wedge & \forall s \in \mathcal{S}\ \ \forall o_1, o_2 \in \mathcal{O} \\
 & (s, o_1, \mathsf{read}) \in \Lambda(\sigma) \wedge (s, o_2, \mathsf{write}) \in \Lambda(\sigma) \Rightarrow \\
 & \left( \begin{array}{l} \forall r, r' \in \mathcal{L} \\ \left( \begin{array}{l} ((o_1,\mathsf{read}), r) \in \mathsf{PA} \ \wedge \\ r' = \bigvee \{r'' \in \mathcal{L} \mid ((o_2,\mathsf{write}), r'') \in \mathsf{PA}\} \end{array} \right) \Rightarrow r \preceq r' \end{array} \right)
\end{array} \right)
\end{array}
$$

TAB. D.8 – Prédicats de sécurité

*Relation de simulation sur les requêtes.* Nous introduisons à présent les relations $\kappa^1_{\mathcal{R}} \subseteq \Sigma_{HRU} \times \mathcal{R}_{HRU} \times \mathcal{R}^\star_{RBAC}$, $\kappa^2_{\mathcal{R}} \subseteq \Sigma_{RBAC} \times \mathcal{R}_{RBAC} \times \mathcal{R}^\star_{HRU}$ et $\kappa^3_{\mathcal{R}} \subseteq \Sigma_{BLP} \times \mathcal{R}_{BLP} \times \mathcal{R}^\star_{RBAC}$, permettant d'établir une correspondance entre les requêtes des modèles considérés. La définition de ces relations est donnée dans la table D.9 (si $E$ est un ensemble fini, $List(E)$ dénote une liste contenant une énumération des éléments de $E$). Pour la relation $\kappa^2_{\mathcal{R}}$, nous supposons que la relation PA vérifie la propriété suivante :

$$\forall r_1, r_2 \in \mathsf{R}\ r_1 <_{\mathsf{R}} r_2 \Rightarrow \{(o,a) \mid ((o,a), r_1) \in \mathsf{PA}\} \cap \{(o,a) \mid ((o,a), r_2) \in \mathsf{PA}\} = \emptyset$$

En effet, un rôle hérite des permissions de ses sous rôles et la propriété ci-dessus permet d'une part d'éviter une définition redondante de PA et d'autre part de simplifier la définition de $\kappa^2_{\mathcal{R}}$. Cette propriété peut être obtenue à partir d'une relation PA quelconque en supprimant les informations redondantes, sans que cela ne change le comportement du système. Nous montrons que ces relations satisfont les propriétés requises pour l'application de la proposition D.6.

**Lemme D.4** *$\kappa^i_{\mathcal{R}}$ est $\Omega$-préservante and $\kappa^i_\Sigma$-préservante ($i \in \{1,2,3\}$).*

$$\kappa^1_{\mathcal{R}} \subseteq \Sigma_{HRU} \times \mathcal{R}_{HRU} \times \mathcal{R}^{\star}_{RBAC}$$

$$\kappa^1_{\mathcal{R}} = \left\{ \begin{array}{l} ((m, m_D), \langle +, s, o, a\rangle, (\langle +, s, o, a\rangle)), \\ ((m, m_D), \langle -, s, o, a\rangle, (\langle -, s, o, a\rangle)), \\ ((m, m_D), \langle +_{m_D}, s, s', o, a\rangle, (\langle +_{\mathsf{PA}}, s, o, a, s'\rangle)), \\ ((m, m_D), \langle -_{m_D}, s, s', o, a\rangle, (\langle -_{\mathsf{PA}}, s, o, a, s'\rangle)) \end{array} \right\}$$

$$\kappa^2_{\mathcal{R}} \subseteq \Sigma_{RBAC} \times \mathcal{R}_{RBAC} \times \mathcal{R}^{\star}_{HRU}$$

$$\kappa^2_{\mathcal{R}} = \left\{ \begin{array}{l} ((m, user, \mathsf{UA}, \mathsf{PA}, roles), \langle +, s, o, a\rangle, (\langle +, s, o, a\rangle)), \\ ((m, user, \mathsf{UA}, \mathsf{PA}, roles), \langle -, s, o, a\rangle, (\langle -, s, o, a\rangle)), \\ ((m, user, \mathsf{UA}, \mathsf{PA}, roles), \langle +_{\mathsf{UA}}, s, u, r\rangle, (Success)), \\ ((m, user, \mathsf{UA}, \mathsf{PA}, roles), \langle -_{\mathsf{UA}}, s, u, r\rangle, (Success)), \\ \left((m, user, \mathsf{UA}, \mathsf{PA}, roles), \langle +_{\mathsf{PA}}, s, o, a, r\rangle, \left(List\left(\bigcup_{s' \in \mathcal{S}} \left\{ \begin{array}{l} \langle +_{m_D}, s, s', o, a\rangle \mid \exists r' \in \mathsf{R} \\ r \leq_{\mathsf{R}} r' \wedge r' \in roles(s') \end{array} \right\}\right)\right)\right), \\ \left((m, user, \mathsf{UA}, \mathsf{PA}, roles), \langle -_{\mathsf{PA}}, s, o, a, r\rangle, \left(List\left(\bigcup_{s' \in \mathcal{S}} \left\{ \begin{array}{l} \langle -_{m_D}, s, s', o, a\rangle \mid \exists r' \in \mathsf{R} \\ r \leq_{\mathsf{R}} r' \wedge r' \in roles(s') \end{array} \right\}\right)\right)\right), \\ \left((m, user, \mathsf{UA}, \mathsf{PA}, roles), \langle +_{roles}, s, s', r\rangle, \left(List\left(\bigcup_{s' \in \mathcal{S}} \left\{ \begin{array}{l} \langle +_{m_D}, s, s', o, a\rangle \mid \exists r' \in \mathsf{R} \\ r' \leq_{\mathsf{R}} r \wedge ((o, a), r') \in \mathsf{PA} \end{array} \right\}\right)\right)\right), \\ \left((m, user, \mathsf{UA}, \mathsf{PA}, roles), \langle -_{roles}, s, s', r\rangle, \left(List\left(\bigcup_{s' \in \mathcal{S}} \left\{ \begin{array}{l} \langle -_{m_D}, s, s', o, a\rangle \mid \exists r' \in \mathsf{R} \\ r' \leq_{\mathsf{R}} r \wedge ((o, a), r') \in \mathsf{PA} \end{array} \right\}\right)\right)\right) \end{array} \right\}$$

$$\kappa^3_{\mathcal{R}} \subseteq \Sigma_{BLP} \times \mathcal{R}_{BLP} \times \mathcal{R}^{\star}_{RBAC}$$

$$\kappa^2_{\mathcal{R}} = \left\{ \begin{array}{l} ((m, f_s, f_o), \langle +, s, o, a\rangle, (\langle +, s, o, a\rangle)), \\ ((m, f_s, f_o), \langle -, s, o, a\rangle, (\langle -, s, o, a\rangle)), \\ \left((m, f_s, f_o), \langle +_{f_s}, s, s', l\rangle, \left( \begin{array}{l} \langle +_{\mathsf{UA}}, s, s', l\rangle, \langle +_{roles}, s, s', l\rangle, \\ \langle -_{roles}, s, s', f_s(s')\rangle, \langle -_{\mathsf{UA}}, s, s', f_s(s')\rangle \end{array} \right)\right), \\ \left((m, f_s, f_o), \langle +_{f_o}, s, s', l\rangle, \left( \begin{array}{l} \langle +_{\mathsf{PA}}, s, o, \mathsf{read}, l\rangle, \langle +_{\mathsf{PA}}, s, o, \mathsf{write}, l\rangle, \\ \langle -_{\mathsf{PA}}, s, o, \mathsf{read}, f_o(o)\rangle, \langle -_{\mathsf{UA}}, s, o, \mathsf{write}, f_o(o)\rangle \end{array} \right)\right) \end{array} \right\}$$

TAB. D.9 – Relations entre requêtes

*Comparaison des modèles.* Les lemmes D.3 et D.4, nous permettent d'utiliser la proposition D.6, pour établir formellement le lemme suivant.

**Lemme D.5**

$$\begin{array}{rcl} \mathbb{M}_{HRU}[\,] & \trianglelefteq & \mathbb{M}_{RBAC}[(\mathcal{S}, \mathcal{S}, \emptyset)] \\ \mathbb{M}_{RBAC}[\rho_{RBAC}] & \trianglelefteq & \mathbb{M}_{HRU}[\,] \\ \mathbb{M}_{BLP}[\rho_{BLP}] & \trianglelefteq & \mathbb{M}_{RBAC}[(\mathcal{S}, \mathcal{L}, \preceq)] \end{array}$$

Nous montrons par ailleurs que le modèle RBAC n'est pas plus restrictif que le modèle de Bell et LaPadula.

**Lemme D.6** $\mathbb{M}_{RBAC}[\rho_{RBAC}] \not\trianglelefteq \mathbb{M}_{BLP}[\rho_{BLP}]$

Il existe donc des situations pour lesquelles il n'est pas possible de simuler un système à base de rôles par un système mettant en œuvre la politique de Bell et LaPadula. Considérons par exemple un système pour lequel $\mathcal{S} = \{s\}$ et $\mathcal{O} = \{o\}$. Nous pouvons construire un état du système à base de rôles où $s$ n'a aucun droit sur $o$, alors que le système reposant sur la politique de Bell et LaPadula autorisera toujours l'accès en écriture de $s$ à $o$. Il n'existe donc pas de relation $\kappa_{\Sigma} \subseteq \Sigma_{RBAC} \times \Sigma_{BLP}$ qui soit $\mathcal{W}$-préservante.

## D.3 Analyse de flots

Nous complétons ici la présentation de l'analyse des flots d'information des modèles de contrôle d'accès faite dans la section 2.3 du chapitre 2. Nous illustrons également cette analyse en considérant les modèles HRU et de Bell et LaPadula.

### D.3.1 Flots engendrés par les exécutions d'un moniteur de référence

Dans la section 2.3.1, nous avons introduit des relations permettant de caractériser les flots d'information qui ont lieu lors d'une séquence d'états. La proposition suivante permet d'exprimer des propriétés de composition de ces relations lorsque les séquences d'états sont issues des exécutions d'une implantation d'un modèle de contrôle d'accès.

**Proposition D.7** *Soit $\mathbb{P}[\rho] = (\Sigma, \Omega)$ une politique de contrôle d'accès, $\mathbb{M}[\rho] = (\mathbb{P}[\rho], \mathcal{R})$ un modèle de contrôle d'accès, $I = (\tau, \Sigma^I)$ une implantation de $\mathbb{M}[\rho]$ et $F = Exec(\tau, \Sigma^I)$.*

1. *Si $I$ est $\mathbb{P}[\rho]$-complète, alors $\hookrightarrow^{OO}_{\Sigma_{|\Omega}} \subseteq \hookrightarrow^{OO}_{F}$, $\hookrightarrow^{OS}_{\Sigma_{|\Omega}} \subseteq \hookrightarrow^{OS}_{F}$ et $\hookrightarrow^{SO}_{\Sigma_{|\Omega}} \subseteq \hookrightarrow^{SO}_{F}$.*
2. *Si $I$ est $\mathbb{P}[\rho]$-correcte, alors $\hookrightarrow^{OS}_{F} \subseteq \rightsquigarrow^{OS}_{\mathbb{P}[\rho]} \circ \hookrightarrow^{OO}_{F}$ et $\hookrightarrow^{SO}_{F} \subseteq \hookrightarrow^{OO}_{F} \circ \rightsquigarrow^{SO}_{\mathbb{P}[\rho]}$.*
3. *Si $I$ est $\mathbb{P}[\rho]$-correcte et $\hookrightarrow^{OO}_{\Sigma_{|\Omega}}$ est transitive, alors $\hookrightarrow^{OO}_{F} \subseteq \hookrightarrow^{OO}_{\Sigma_{|\Omega}}$, $\hookrightarrow^{OS}_{F} \subseteq \rightsquigarrow^{OS}_{\mathbb{P}[\rho]} \circ \hookrightarrow^{OO}_{\Sigma_{|\Omega}}$ et $\hookrightarrow^{SO}_{F} \subseteq \hookrightarrow^{OO}_{\Sigma_{|\Omega}} \circ \rightsquigarrow^{SO}_{\mathbb{P}[\rho]}$.*
4. *Si $I$ est $\mathbb{P}[\rho]$-complète est $\mathbb{P}[\rho]$ est libre, alors $\rightsquigarrow^{OS}_{\mathbb{P}[\rho]} \circ \hookrightarrow^{OO}_{\Sigma_{|\Omega}} \subseteq \hookrightarrow^{OS}_{F}$ et $\hookrightarrow^{OO}_{\Sigma_{|\Omega}} \circ \rightsquigarrow^{SO}_{\mathbb{P}[\rho]} \subseteq \hookrightarrow^{SO}_{F}$.*
5. *Si $I$ est $\mathbb{P}[\rho]$-complète et $\Gamma$-complète, alors $\rightsquigarrow^{OS}_{\mathbb{P}[\rho]} \circ \hookrightarrow^{OO}_{F} \subseteq \hookrightarrow^{OS}_{F}$, et $\hookrightarrow^{OO}_{F} \circ \rightsquigarrow^{SO}_{\mathbb{P}[\rho]} \subseteq \hookrightarrow^{SO}_{F}$.*
6. *Si $\mathbb{P}[\rho]$ est libre et $I$ est $\mathbb{P}[\rho]$-correcte, alors $\hookrightarrow^{OS}_{F} \subseteq \hookrightarrow^{OS}_{\Sigma_{|\Omega}}$ et $\hookrightarrow^{SO}_{F} \subseteq \hookrightarrow^{SO}_{\Sigma_{|\Omega}}$.*

### D.3.2 Mécanisme de détection de flots

Dans la section 2.3.2, nous avons introduit un formalisme permettant de spécifier un mécanisme de détection de flots. Nous définissons ici un tel mécanisme dans ce formalisme. Il s'agit de la formalisation d'un système de détection d'intrusion présenté dans [55, 54].

Le mécanisme de détection de flots que nous introduisons ici permet de détecter les flots engendrés par des séquences d'états produisant des flots ne respectant pas une certaine politique de flots $\rightsquigarrow^{OO}$ entre objets. Ce mécanisme peut être aussi utilisé pour des politiques de flots entre sujets et objets. Il suffit pour cela d'associer à chaque sujet $s$ un objet $o_s$. Dans ce cas, les objets du système sont les objets de l'ensemble $\mathcal{O}$ et les objets d'un ensemble $\mathcal{O}_\mathcal{S}$ contenant les objets associés aux sujets. Bien sûr, les objets considérés par le prédicat de sécurité sont les objets de $\mathcal{O}$ et pour tout état $\sigma$, nous supposons :

$$\begin{array}{rl} & \{(s, o_s, \mathsf{read}), (s, o_s, \mathsf{write})\} \subseteq \Lambda(\sigma) \\ \text{et} & ((s, o, a) \in \Lambda(\sigma) \wedge o \in \mathcal{O}_\mathcal{S}) \Rightarrow o = o_s \end{array}$$

Dans ce cadre, on montre facilement que :

$$\begin{array}{l} \hookrightarrow^{OS}_{(\sigma_1,\ldots,\sigma_n)} = \{o_1 \hookrightarrow^{OO}_{(\sigma_1,\ldots,\sigma_n)} o_2 \mid o_1 \in \mathcal{O} \wedge o_2 \in \mathcal{O}_\mathcal{S}\} \\ \hookrightarrow^{SO}_{(\sigma_1,\ldots,\sigma_n)} = \{o_1 \hookrightarrow^{OO}_{(\sigma_1,\ldots,\sigma_n)} o_2 \mid o_1 \in \mathcal{O}_\mathcal{S} \wedge o_2 \in \mathcal{O}\} \end{array}$$

La politique de flots $\rightsquigarrow^{OO}$ entre objets que nous cherchons à vérifier est représentée par une relation $\twoheadrightarrow \subseteq \mathcal{P}(\mathcal{O}) \times \mathcal{O}$. $\{o_1, \cdots, o_n\} \twoheadrightarrow o$ signifie que l'information initiallement contenue dans les objets $o_1$, ..., $o_n$ peut se propager dans l'objet $o$. La relation $\twoheadrightarrow$ est définie comme suit :

$$\twoheadrightarrow = \bigcup_{o \in \mathcal{O}} (\{o\} \cup \{o_i | o_i \rightsquigarrow^{OO} o\}, o)$$

*Séquences d'états observables.* Le mécanisme $\mathbb{F}[\rho, E, F, \rightsquigarrow_{\mathbb{F}}]$ que nous définissons est générique. Il est paramétré par un paramètre de sécurité $\rho$, par un ensemble d'états observables $E$, et par un ensemble de séquences d'états $F \subseteq E^\star$ tel que pour toute séquence $(\sigma_1, \cdots, \sigma_n) \in F$ :

- aucun accès n'est effectué dans l'état initial $\sigma_1$ : $\Lambda(\sigma_1) = \emptyset$
- l'état $\sigma_{i+1}$ est obtenu à partir de l'état $\sigma_i$ en ajoutant ou en supprimant un accès de l'ensemble des accès courants de $\sigma_i$ :

$$\forall (\sigma_1, \cdots, \sigma_n) \in F \; \forall i \in \mathbb{N} \; (1 \leq i \leq n-1) \quad \sigma_{i+1} = \sigma_i \oplus (s, o, a) \vee \sigma_{i+1} = \sigma_i \ominus (s, o, a)$$

*Définition de* $\rightsquigarrow_{\mathbb{F}}$. Puisque l'on cherche à détecter les flots ne respectant pas la politique de flots $\rightsquigarrow^{OO}$, la relation $\rightsquigarrow_{\mathbb{F}}$ spécifiant les flots recherchés est définie comme l'ensemble des flots possibles privé des flots autorisés. Plus formellement, $\rightsquigarrow_{\mathbb{F}}$ est défini comme suit :

$$\rightsquigarrow_{\mathbb{F}} = \hookrightarrow_F^{OO} \setminus (\{o \hookrightarrow_F^{OO} o\} \cup \rightsquigarrow^{OO})$$

Cette définition permet d'établir facilement la proposition suivante.

**Proposition D.8** *$\mathbb{F}[\rho, E, F, \rightsquigarrow_{\mathbb{F}}]$ est correcte et complète pour $\rightsquigarrow^{OO}$.*

*Etats d'alerte.* Le mécanisme de détection de flots que nous définissons repose sur un système de "tags". Ainsi, l'information $\Psi(\sigma)$ associée à un état $\sigma$ et permettant de définir le prédicat $\mho$ est la donnée pour chaque objet du système de deux "tags" $T_\sigma^R(o)$ et $T_\sigma^W(o)$. Ces "tags" sont définis comme suit :

- Pour toute séquence d'états $(\sigma_1, \cdots, \sigma_n) \in F$ et pour tout objet $o$, les "tags" sont initiallement définis dans $\sigma_1$ comme suit :

$$T_{\sigma_1}^R(o) = \{(\mathsf{O}, o') \in \twoheadrightarrow | \, o \in \mathsf{O}\} \quad T_{\sigma_1}^W(o) = \{(\mathsf{O}, o') \in \twoheadrightarrow | \, o = o'\}$$

- Lors de chaque transition de l'état $\sigma_i$ à l'état $\sigma_{i+1}$, les "tags W" ne changent pas et les "tags R" des objets dont le contenu a été modifié est modifié comme suit. Si $\sigma_{i+1}$ a été obtenu en ajoutant un accès aux accès courants de $\sigma_i$, alors :

$$T_{\sigma_{i+1}}^R(o') = \begin{cases} \bigcap\limits_{\{o_i \in \mathcal{O} | o_i \hookrightarrow_{\sigma_{i+1}}^{OO} o'\}} T_{\sigma_i}^R(o_i) & \text{si } \exists o'' \in \mathcal{O} \; o'' \hookrightarrow_{\sigma_{i+1}}^{OO} o' \\ T_{\sigma_i}^R(o') & \text{sinon} \end{cases}$$

  Si $\sigma_{i+1}$ a été obtenu en supprimant un accès aux accès courants de $\sigma_i$, alors les "tags" ne changent pas.

Nous pouvons prouver que pour tout état $\sigma$ et tout objet $o$, le "tag R" de $o$ peut être exprimé comme l'intersection des "tags R" d'un ensemble d'objets dans l'état initial.

**Proposition D.9** $\forall(\sigma_1, \sigma_2, \ldots, \sigma_n) \in F\ \forall o \in \mathcal{O} \quad T^R_{\sigma_n}(o) = \bigcap_{\{o_j | o_j \hookrightarrow^{OO}_{(\sigma_1,\cdots,\sigma_n)} o\}} T^R_{\sigma_1}(o_j)$

Le système de "tags" permet de définir le prédicat $\mho$ caractérisant les états d'alerte comme suit :

$$\mho(\sigma) \Leftrightarrow \exists o \in \mathcal{O}\ T^R_\sigma(o) \cap T^W_\sigma(o) = \emptyset$$

Nous montrons alors que le mécanisme de détection de flots ainsi obtenu est correct et complet.

**Proposition D.10** $\mathbb{F}[\rho, E, F, \rightsquigarrow_{\mathbb{F}}] = (\Sigma, \mho)$ *est correct et complet.*

### D.3.3 Application

Dans cette section, nous illustrons l'analyse des flots d'information d'un modèle de contrôle d'accès en considérant des variantes du modèle HRU et du modèle de Bell et LaPadula présentés précedemment. Ces variantes sont obtenues en imposant que toutes les informations de sécurité soient statiques durant la vie du système. Un état consiste donc juste en la description des accès courants effectués dans le système. On note ainsi $\Sigma^{acc} = \wp(\mathbb{A})$ l'ensemble des états de ces deux modèles (la définition des prédicats de sécurité $\Omega_{HRU}$ et $\Omega_{BLP}$ est inchangée). Il s'agit donc d'analyser les flots d'information pour les modèles définis comme suit :

| | |
|---|---|
| $\rho'_{HRU} = m_D$ | $\rho'_{BLP} = (\mathcal{L}, \preceq, \curlyvee, \curlywedge, f_s, f_o)$ |
| $\mathbb{P}'[\rho'_{HRU}] = (\mathcal{S}, \mathcal{O}, \mathcal{A}, \Sigma^{acc}, \Omega_{HRU})$ | $\mathbb{P}'[\rho'_{BLP}] = (\mathcal{S}, \mathcal{O}, \mathcal{A}, \Sigma^{acc}, \Omega_{BLP})$ |
| $\mathbb{M}'[\rho'_{HRU}] = (\mathbb{P}'[\rho'_{HRU}], \mathcal{R}^{acc})$ | $\mathbb{M}'[\rho'_{BLP}] = (\mathbb{P}'[\rho'_{BLP}], \mathcal{R}^{acc})$ |

Les instances de ces modèles que nous utiliserons dans les exemples sont définies comme suit. Pour HRU, l'ensemble des accès autorisés peut être représenté par une matrice de droits d'accès et l'ensemble $m_D$ que nous considérons est spécifié par la matrice suivante :

| | $o_1$ | $o_2$ | $o_3$ | $o_4$ |
|---|---|---|---|---|
| Alice | read, write | | read | |
| Bob | read | read, write | | |
| Charlie | | read, write | | write |

Pour le modèle de Bell et LaPadula, nous considerons le paramètre $\rho'_{BLP}$ représenté sur la figure D.1.

#### Analyse de flots

*Politiques de confidentialité et d'intégrité.* Les politiques de confidentialité et d'intégrité, exprimées en termes de flots d'information, induites pas les politiques de contrôle d'accès

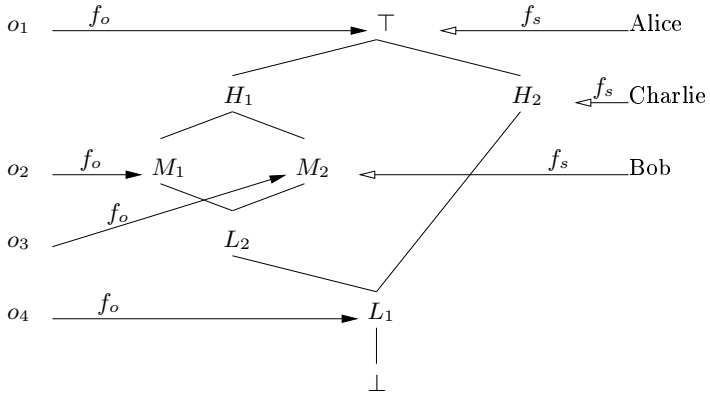


FIG. D.1 – $\rho'_{BLP}$

sont définies comme suit :

$$\rightsquigarrow^{OS}_{\mathbb{P}'_{HRU}[m_D]} = \left\{ \begin{array}{lll} o_1 \hookrightarrow^{OS} \text{Alice}, & o_3 \hookrightarrow^{OS} \text{Alice}, & o_1 \hookrightarrow^{OS} \text{Bob}, \\ o_2 \hookrightarrow^{OS} \text{Bob}, & o_2 \hookrightarrow^{OS} \text{Charlie} & \end{array} \right\}$$

$$\rightsquigarrow^{OS}_{\mathbb{P}'_{BLP}[\rho'_{BLP}]} = \left\{ \begin{array}{llll} o_1 \hookrightarrow^{OS} \text{Alice}, & o_2 \hookrightarrow^{OS} \text{Alice}, & o_3 \hookrightarrow^{OS} \text{Alice}, & o_4 \hookrightarrow^{OS} \text{Alice}, \\ o_3 \hookrightarrow^{OS} \text{Bob}, & o_4 \hookrightarrow^{OS} \text{Bob}, & o_4 \hookrightarrow^{OS} \text{Charlie} & \end{array} \right\}$$

$$\rightsquigarrow^{SO}_{\mathbb{P}'_{HRU}[m_D]} = \left\{ \text{Alice} \hookrightarrow^{SO} o_1, \quad \text{Bob} \hookrightarrow^{SO} o_2, \quad \text{Charlie} \hookrightarrow^{SO} o_2, \quad \text{Charlie} \hookrightarrow^{SO} o_4 \right\}$$

$$\rightsquigarrow^{SO}_{\mathbb{P}'_{BLP}[\rho'_{BLP}]} = \left\{ \begin{array}{llll} \text{Alice} \hookrightarrow^{SO} o_1, & \text{Alice} \hookrightarrow^{SO} o_2, & \text{Alice} \hookrightarrow^{SO} o_3, & \text{Alice} \hookrightarrow^{SO} o_4, \\ \text{Charlie} \hookrightarrow^{SO} o_1, & \text{Charlie} \hookrightarrow^{SO} o_2, & \text{Charlie} \hookrightarrow^{SO} o_3, & \text{Charlie} \hookrightarrow^{SO} o_4, \\ \text{Bob} \hookrightarrow^{SO} o_1, & \text{Bob} \hookrightarrow^{SO} o_2, & \text{Bob} \hookrightarrow^{SO} o_3, & \text{Bob} \hookrightarrow^{SO} o_4 \end{array} \right\}$$

Pour la politique de Bell et LaPadula, nous pouvons montrer que les relations $\rightsquigarrow^{OS}_{\mathbb{P}'_{BLP}[\rho'_{BLP}]}$ et $\hookrightarrow^{OO}_{Exec(\tau_{BLP},\Sigma^I_{BLP})}$ peuvent s'exprimer directement à partir des niveaux de sécurité.

**Lemme D.7**

1. $\forall o \in \mathcal{O}\ \forall s \in \mathcal{S} \quad o \rightsquigarrow^{OS}_{\mathbb{P}'_{BLP}[\rho'_{BLP}]} s \Leftrightarrow f_o(o) \preceq f_s(s)$
2. *Soit $o_1$ et $o_2$ deux objets.*
   (a) $o_1 \hookrightarrow^{OO}_{Exec(\tau_{BLP},\Sigma^I_{BLP})} o_2 \Rightarrow f_o(o_1) \preceq f_o(o_2)$
   (b) *Si il existe un sujet $s \in \mathcal{S}$ tel que $f_o(o_1) \preceq f_s(s)$, alors :*

$$f_o(o_1) \preceq f_o(o_2) \Rightarrow o_1 \hookrightarrow^{OO}_{Exec(\tau_{BLP},\Sigma^I_{BLP})} o_2$$

Nous montrons également que l'ensemble des exécutions de $(\tau_{BLP}, \Sigma^I_{BLP})$ est correct et complet pour les politiques de flots $\rightsquigarrow^{OS}_{\mathbb{P}'_{BLP}[\rho'_{BLP}]}$ and $\rightsquigarrow^{SO}_{\mathbb{P}'_{BLP}[\rho'_{BLP}]}$.

**Lemme D.8** *$Exec(\tau_{BLP}, \Sigma^I_{BLP})$ est correct et complet pour $\rightsquigarrow^{OS}_{\mathbb{P}'_{BLP}[\rho'_{BLP}]}$ et $\rightsquigarrow^{SO}_{\mathbb{P}'_{BLP}[\rho'_{BLP}]}$.*

Cette propriété n'est pas vérifiée par les exécutions de $(\tau_{HRU}, \Sigma^I_{HRU})$. En effet, si l'on considère par exemple l'ensemble $m_D$ introduit précédemment, nous avons $o_3 \hookrightarrow^{OS}_{Exec(\tau_{HRU},\Sigma^I_{HRU})}$ Bob (car Alice peut lire $o_3$ et écrire dans $o_1$ qui est accessible en lecture par Bob) mais nous n'avons pas $o_3 \rightsquigarrow^{OS}_{\mathbb{P}'_{HRU}[m_D]}$ Bob. De manière similaire, nous avons Bob $\hookrightarrow^{SO}_{Exec(\tau_{HRU},\Sigma^I_{HRU})}$ $o_4$ (car Bob peut écrire dans $o_2$ et Charlie peut lire $o_2$ et écrire dans $o_4$) mais nous n'avons pas Bob $\rightsquigarrow^{SO}_{\mathbb{P}'_{HRU}[m_D]}$ $o_4$. Il est toutefois possible de caractériser les paramètres de sécurité de la politique HRU pour lesquels la propriété de correction est vérifiée.

**Lemme D.9**

1. *$Exec(\tau_{HRU}, \Sigma^I_{HRU})$ est complet et correct pour $\rightsquigarrow^{OS}_{\mathbb{P}'_{HRU}[m_D]}$ ssi :*

$$\forall s_1, s_2 \in \mathcal{S}\ \forall o_1, o_2 \in \mathcal{O}$$
$$\{(s_1, o_1, \mathsf{read}), (s_1, o_2, \mathsf{write}), (s_2, o_2, \mathsf{read})\} \subseteq m_D \Rightarrow (s_2, o_1, \mathsf{read}) \in m_D$$

2. *$Exec(\tau_{HRU}, \Sigma^I_{HRU})$ est complet et correct pour $\rightsquigarrow^{SO}_{\mathbb{P}'_{HRU}[m_D]}$ ssi :*

$$\forall s_1, s_2 \in \mathcal{S}\ \forall o_1, o_2 \in \mathcal{O}$$
$$\{(s_1, o_1, \mathsf{write}), (s_2, o_1, \mathsf{read}), (s_2, o_2, \mathsf{write})\} \subseteq m_D \Rightarrow (s_1, o_2, \mathsf{write}) \in m_D$$

**Détection de flots "illégaux"**

Puisque les exécutions de $(\tau_{HRU}, \Sigma^I_{HRU})$ ne sont pas correctes pour les politiques de flots $\rightsquigarrow^{OS}_{\mathbb{P}'_{HRU}[\rho'_{HRU}]}$ et $\rightsquigarrow^{SO}_{\mathbb{P}'_{HRU}[\rho'_{HRU}]}$, nous utilisons maintenant le mécanisme de détection de flots défini dans la section D.3.2 afin que le système lève une alerte dès qu'il détecte un flot ne respectant pas les politiques $\rightsquigarrow^{OS}_{\mathbb{P}'_{HRU}[\rho'_{HRU}]}$ et $\rightsquigarrow^{SO}_{\mathbb{P}'_{HRU}[\rho'_{HRU}]}$. Ici ces deux politiques de flots sont exprimées en associant un objet $o_s$ à chaque sujet $s$. Plus formellement, le mécanisme de détection de flots est obtenu en considérant :

- l'ensemble $m_D$ comme paramètre de sécurité $\rho$,
- l'ensemble $\Sigma_{|\Omega_{HRU}}$ comme ensemble d'états observables $E$,
- l'ensemble $Exec(\tau_{HRU}, \Sigma^I_{HRU})$, où $\Sigma^I_{HRU} \subseteq \{\sigma \in \Sigma \mid \Lambda(\sigma) = \emptyset\}$, comme ensemble de séquences d'états observables,
- la relation :

$$\rightsquigarrow^{OO} = \left\{ o_1 \hookrightarrow^{OO} o_2 \mid \begin{array}{ll} & (o_1 \in \mathcal{O} \wedge o_2 = o_s \in \mathcal{O}_{\mathcal{S}} \wedge o_1 \rightsquigarrow^{OS}_{\mathbb{P}'_{HRU}[\rho'_{HRU}]} s) \\ \vee & (o_1 = o_s \in \mathcal{O}_{\mathcal{S}} \wedge o_2 \in \mathcal{O} \wedge s \rightsquigarrow^{SO}_{\mathbb{P}'_{HRU}[\rho'_{HRU}]} o_2) \end{array} \right\}$$

  comme politique de flots $\rightsquigarrow^{OO}$.

Les propositions D.8, D.10 et 2.1, nous garantissent alors que les états d'alerte sont exactement les états issus de séquences engendrant un flot ne respectant pas les politiques $\rightsquigarrow^{OS}_{\mathbb{P}'_{HRU}[\rho'_{HRU}]}$ et $\rightsquigarrow^{SO}_{\mathbb{P}'_{HRU}[\rho'_{HRU}]}$.

Par exemple, considérons à nouveau le modèle HRU introduit précédemment. La politique de contrôle d'accès induit la politique de confidentialité et d'intégrité suivante, exprimée par des relations de flots entre objets :

$$\twoheadrightarrow_{HRU} = \left\{ \begin{array}{lll} (\{o_1, o_3, o_A\}, o_1) & (\{o_1, o_2, o_B, \}, o_2) & (\{o_2, o_C\}, o_2) \\ (\{o_3\}, o_3) & (\{o_2, o_4, o_c\}, o_4) & (\{o_1, o_3, o_A\}, o_A) \\ (\{o_1, o_2, o_B\}, o_B) & (\{o_2, o_C\}, o_C)\} & \end{array} \right\}$$

où $o_A$, $o_B$ et $o_C$ sont les objets respectivement associés aux sujets Alice, Bob et Charlie. Considérons la séquence d'états $(\sigma_1, \sigma_2, \sigma_3, \sigma_4)$ telle que $\sigma_1$ est un état dont l'ensemble des accès courants est vide, $\sigma_2$ est obtenu en ajoutant un accès en lecture sur $o_3$ pour Alice, $\sigma_3$ en ajoutant un accès en écriture sur $o_1$ pour Alice, et $\sigma_4$ en ajoutant un accès en lecture sur $o_1$ pour Bob. Aucun des états $\sigma_1$, $\sigma_2$ et $\sigma_3$ n'est un état d'alerte, mais nous avons :

$$T^R_{\sigma_4}(o_B) \cap T^W_{\sigma_4}(o_B)$$
$$= \{(\{o_1, o_3, o_3\}, o_1), (\{o_3\}, o_3), \{o_1, o_3, o_A\}, o_A)\} \cap \{(\{o_1, o_2, o_B\}, o_B)\} = \emptyset$$

et donc $\sigma_4$ est un état d'alerte ce qui correspond bien au fait que Bob accède au contenu de l'objet $o_3$ alors que la politique de confidentialité ne l'y autorise pas.