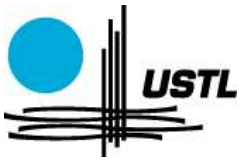
# Model-checking pour les ambients : des algèbres de processus aux données semi-structurées

Jean-Marc Talbot






# Model-checking pour les ambients : des algèbres de processus aux données semi-structurées

mémoire présenté le 9 décembre 2005

pour l'obtention de

**l'Habilitation à diriger des recherches**

**en Sciences mathématiques (spécialité informatique)**

par

Jean-Marc Talbot

**Composition du jury**

| | |
|---|---|
| *Président :* | Laurence Duchien (Professeur - Université de Lille 1) |
| *Rapporteurs :* | Roberto Amadio (Professeur - Université Paris 7)<br>Hubert Comon-Lundh (Professeur - ENS Cachan)<br>Helmut Seidl (Professeur - Technische Universität München) |
| *Examinateurs :* | Giuseppe Castagna (CR CNRS - ENS)<br>Michaël Rusinowitch (DR INRIA - LORIA) |
| *Directeur :* | Sophie Tison (Professeur - Université de Lille 1) |

**UNIVERSITÉ DES SCIENCES ET TECHNOLOGIES DE LILLE**
Laboratoire d'Informatique Fondamentale de Lille — UMR 8022
UFR d'IEEA – Bât. M3 – 59655 VILLENEUVE D'ASCQ CEDEX
Tél. : 33 (0)3 28 77 85 41 – Télécopie : 33 (0)3 28 77 85 37 – email : direction@lifl.fr

Mis en page et composé avec LaTeX et la classe thlifl.

# Note de lecture

Ce document se compose de deux parties. Ces deux parties ont été écrites de telle façon que la lecture de l'une puisse se faire indépendemment de l'autre. Seul le premier chapitre de la seconde partie sert de lien aux deux parties de ce document. Les références bibliographiques numériques (*e.g.* [19]) font référence à ma bibliographie personnelle (voir Annexe A). Les autres références sont de type alphanumérique (*e.g.* [CG98]) et sont référencées à partir de la page 87.

Certaines des publications mentionnées dans ce mémoire seront en annexe de ce document.

**Annexe C**

> Witold Charatonik, Silvano Dal Zilio, Andrew D. Gordon, Supratik Mukhopadhyay, and Jean-Marc Talbot. Model Checking Mobile Ambient. *Theoretical Computer Science*, 308(1-3) :277–331, 2003.

**Annexe D**

> Witold Charatonik, Andrew D. Gordon, and Jean-Marc Talbot. Finite-control Mobile Ambients. In *European Symposium on Programming - ESOP 2002*, volume 2305 of *Lecture Notes in Computer Science*, pages 295–313. Springer, 2002.

**Annexe E**

> Iovka Boneva and Jean-Marc Talbot. When Ambients Cannot be Opened. *Theoretical Computer Science*, 333(1-2) :127–169, 2005.

**Annexe F**

> Iovka Boneva and Jean-Marc Talbot. On Complexity of Model-Checking for the TQL Logic. In *3rd IFIP International Conference on Theoretical Computer Science (IFIP-TCS'04)*, pages 381–394. Kluwer, 2004.

**Annexe G**

> Iovka Boneva, Jean-Marc Talbot, and Sophie Tison. Expressiveness of Spatial Logic for Trees. In *Twentieth Annual IEEE Symposium on Logic in Computer Sience (LICS 2005)*, pages 280–289. IEEE Press, 2005.

# Table des matières

**Note de lecture** **i**

**Introduction** **1**

**I Vérification par model-checking du calcul des ambients** **5**

**1 Une introduction au calcul des ambients** **7**

1.1 Un bref historique . . . . . 7

1.1.1 CCS : A Calculus of Communicating Systems . . . . . 7

1.1.2 Le $\pi$-calcul . . . . . 12

1.2 Le calcul des ambients . . . . . 15

1.2.1 Une présentation graphique . . . . . 16

1.2.2 Syntaxe et sémantique . . . . . 17

1.3 Quelques variantes du calcul des ambients . . . . . 23

**2 Vérification pour le calcul des ambients sans récursion** **27**

2.1 Vérification dans les algèbres de processus . . . . . 27

2.1.1 Quelques généralités sur la vérification de programmes . . . . . 27

2.1.2 Vérification par model-checking . . . . . 28

2.2 Vérification par model-checking du calcul des ambients . . . . . 29

2.2.1 La logique des ambients . . . . . 29

2.2.2 Le problème de model-checking . . . . . 32

**3 Vérification pour le calcul des ambients à contrôle fini** **37**

3.1 Le calcul des ambients à contrôle fini . . . . . 37

3.1.1 Le système de type FC . . . . . 41

3.1.2 Typage et contrôle fini . . . . . 41

3.1.3 Communication de noms vs communication de capacités . . . . . 42

3.2 Model-checking pour les ambients à contrôle fini . . . . . 42

**4 Vérification en présence de réplication** **43**
4.1 Limitation de l'approche par model-checking en présence de réplication . . . . . 43
4.2 Le calcul pur et public sans la capacité `open` . . . . . 44
4.2.1 Motivations . . . . . 44
4.2.2 Résultats . . . . . 45
4.3 Expressivité de calculs minimalistes . . . . . 46
4.4 Le calcul pur et public sans les capacités `in` et `out` . . . . . 47
4.4.1 Model-checking de $pMA^{-\nu,mvt}$ face à la logique $\mathcal{L}^{-\triangleright}$ . . . . . 47
4.4.2 Convergence de nom pour le fragment $pMA^{-\nu,mvt}$ . . . . . 50
4.5 Un schéma général . . . . . 53

**5 Conclusion et perspectives** **55**

**II Une étude formelle du langage de requêtes TQL** **57**

**6 Des algèbres de processus aux données semi-structurées** **59**
6.1 Ambients et données semi-structurées . . . . . 59
6.2 Logique des ambients et la logique d'arbres de TQL . . . . . 60
6.3 Le langage de transformation de TQL . . . . . 61

**7 Arbres, automates, logiques et XML** **63**
7.1 Modèles formels pour la validation et l'interrogation de documents XML . . . . . 63
7.2 Arbres . . . . . 64
7.3 Logiques et arbres . . . . . 65
7.4 Logiques et automates . . . . . 68

**8 Expressivité et satisfiabilité pour la logique TQL** **71**
8.1 La logique TQL : syntaxe et sémantique . . . . . 71
8.2 Expressivité de la logique TQL . . . . . 74
8.2.1 Expressivité pour les ensembles d'arbres . . . . . 74
8.2.2 Expressivité pour les requêtes monadiques . . . . . 76
8.3 Satisfiabilité pour la logique TQL . . . . . 77

**9 Model-checking et requêtes pour la logique TQL** **79**
9.1 Le model-checking de TQL . . . . . 79
9.1.1 Un algorithme pour le problème de model-checking de TQL . . . . . 79
9.1.2 Complexité combinée pour le problème de model-checking de TQL . . . . 79
9.1.3 Complexité de données pour le problème de model-checking de TQL . . . . 80

9.2 Réponses aux requêtes . . . . . 82

**10 Conclusion et perspectives** **83**

**Bibliographie** **87**

**A Publications** **97**

A.1 Revues internationales . . . . . 97

A.2 Conférences internationales avec actes et comité de lecture . . . . . 97

**B Curriculum vitæ** **99**

**C Model Checking Mobile Ambient** **101**

**D Finite-control Mobile Ambients** **155**

**E When Ambients Cannot be Opened** **177**

**F On Complexity of Model-Checking for the TQL Logic** **223**

**G Expressiveness of Spatial Logic for Trees** **237**

# Introduction

Ce mémoire présente une partie des travaux de recherche que j'ai effectués depuis ma nomination en février 2001 comme maître de conférences à l'université des Sciences et Technologies de Lille.

Dans un soucis de cohérence, je ne présente pas dans ce document l'intégralité des travaux de recherches que j'ai pu mener depuis l'obtention de ma thèse de doctorat. Ainsi, les travaux réalisés au cours de mon séjour post-doctoral au Max-Plack-Institut für Informatik dans l'équipe "Programming Logic" (dirigée par le professeur Harald Ganzinger) ne figurent pas dans ce mémoire. Ces travaux s'inscrivaient dans la suite de ma thèse de doctorat et portaient sur les contraintes ensemblistes [1, 9, 11, 15] et l'analyse ensembliste de programmes [10]. Ils ont été réalisés en collaboration avec notamment Andreas Podelski, Witold Charatonik (MPI für Informatik), Joachim Niehren et Martin Müller (PS Lab - Université de la Sarre).

De même que ce document ne mentionne pas mes travaux les plus anciens, il ne présente pas non plus des travaux plus récents et toujours en cours. Ces travaux étudient notamment l'utilisation des calculs des automates d'arbres comme définition de requêtes n-aires [20] et sont réalisés dans le cadre de la thèse de Laurent Planque, encadrée par Sophie Tison, Joachim Niehren et moi-même. Nous n'abordons pas non plus des travaux sur les automates équationnels [21], fruits d'une collaboration avec Hitoshi Ohsaki (National Institute of Advanced Industrial Science et Technology - AIST).

Les travaux présentés dans ce mémoire s'articulent autour des ambients mobiles et, pour être plus précis, autour du problème de model-checking pour les ambients. Cependant, ces travaux présentent deux volets différents qui formeront les deux parties de ce document. Notre fil conducteur pour la présentation de nos travaux sera

$$\boxed{\mathcal{A} \models \phi}$$

$\phi$ sera une formule d'une certaine logique permettant d'exprimer des propriétés pour une classe de structures logiques dont $\mathcal{A}$ sera un des éléments. Le problème de model-checking est de déterminer si $\mathcal{A}$ satisfait la propriété $\phi$, *ie* si $\mathcal{A}$ est un *modèle* de la formule $\phi$.

Alors que le $\mathcal{A}$ de notre fil conducteur se veut être mis pour *Ambient*, nous allons maintenant instancier cette lettre de manière plus précise et de deux façons différentes ; ces deux instanciations vont décliner deux aspects des ambients et chacun de ces aspects va constituer une partie de ce mémoire.

**$\mathcal{A}$ comme agent ..** ou processus La première partie de ce document aura pour objet la vérification par model-checking du calcul des ambients, une algèbre de processus, introduite par Cardelli et Gordon, ayant pour but de modéliser l'informatique mobile. Cette première partie décrit des recherches menées dans un premier temps en compagnie de Witold Charatonik, Supratik Mukhopadhyay (MPI für Informatik), Sivano Dal Zilio et Andrew D. Gordon (Microsoft Research). Cette partie se terminera par une présentation des travaux menés dans le cadre du stage de recherche de DEA de Iovka Boneva, stage que j'ai co-encadré avec Sophie Tison.

Le model-checking est une des techniques les plus largement utilisées pour la vérification de programmes. $\mathcal{A}$ désigne un programme pour lequel on souhaite savoir s'il vérifie la propriété $\phi$. Les logiques utilisées pour décrire ces propriétés sont, par exemple, les logiques temporelles (CTL, LTL, ..) ou les logiques dynamiques (PDL, ..). Par opposition à la démonstration automatique, le terme de model-checking est, en général, réservé à une méthode purement automatique de la preuve de $\mathcal{A} \models \phi$.

Initialement dédiées aux systèmes d'états finis, les techniques de model-checking ont été, depuis quelque temps, étendues, tout au moins pour certains types de propriétés, à des systèmes comportant un nombre infini d'états comme les algèbres de processus (BPP, systèmes à pile, PA, réseaux de Petri, ..), les automates à compteurs, les automates temporisés ou les systèmes hydrides.

Notre étude a porté sur le problème de model-checking pour le calcul des ambients et une logique proposée par Cardelli et Gordon, appelée *logique des ambients*. Cette étude s'est voulue la plus exhaustive et complète possible. Nous avons en particulier essayé de définir au mieux la frontière de la décidabilité du problème de model-checking en étudiant divers fragments du calcul et de la logique.

**Chapitre 1** Le premier chapitre présente le calcul des ambients et le resitue dans la lignée de ses ancètres que sont CCS et le $\pi$-calcul. Ce chapitre est purement introductif. Il se veut didactique mais absolument pas exhautif : de nombreux travaux en relation avec le calcul des ambients, le $\pi$-calcul ou CCS ne seront pas évoqués. Sa rédaction a été un petit travail qui m'a été enrichissant.

**Chapitre 2** Ce chapitre présente tout d'abord la logique des ambients et le problème de model-checking associé, comme introduits par Cardelli et Gordon. Puis, nous présentons les résultats de complexité obtenus pour ce problème en considérant les fragments du calcul et de la logique étudiés précédemment par Cardelli et Gordon. Nous tentons ensuite de lever une à une les restrictions et montrons que seule celle concernant la gestion des noms privés permet de préserver la décidabilité du model-checking.

**Chapitre 3** La présence de la réplication (un des moyens de définir l'itération dans les algèbres de processus) dans le calcul implique l'indécidabilité du model-checking mais son absence implique la terminaison pour tout processus. Ce chapitre fait une proposition de compromis sous la forme du calcul des ambients à contrôle fini. Ce calcul fait que le graphe des processus atteignables par réduction depuis un processus donné est fini. Ceci nous permet de montrer que le model-checking demeure décidable.

**Chapitre 4** Nous considérons finalement le model-checking et des problèmes liés comme l'accessibilité, pour un calcul avec réplication mais dont certains des mécanismes opérationnels (ou capacités) ont été supprimés. Nous montrons que, même pour des fragments minimalistes de calcul des ambients et de la logique, le model-checking est indécidable.

**Chapitre 5** Le dernier chapitre de cette partie du mémoire est l'occasion de porter un regard critique sur nos travaux ; nous tentons de mesurer la portée de nos résultats tout en dégageant clairement ce qui a été fait de ce qu'il reste à accomplir.

**$\mathcal{A}$ comme arbre** La seconde partie du document a pour objet une étude formelle de la logique présente dans le langage de requêtes TQL. Le langage TQL (Tree Query Language) est un formalisme d'interrogation pour des documents semi-structurés, proposé par Cardelli et Ghelli et basé sur les ambients et leur logique.

Nos travaux ont été réalisés dans le cadre de la thèse de doctorat de Iovka Boneva, co-encadrée par Sophie Tison et moi-même. Cette thèse a débutée en même temps que la création du projet INRIA MOSTRARE, qui réunit des chercheurs de l'équipe GRAPPA (Groupe de Recherche sur l'Apprentissage Automatique) de l'université de Lille 3 et de l'équipe STC (Spécification, Test et Contraintes) de l'université

de Lille 1. Le but du projet MOSTRARE est l'utilisation des techniques d'apprentissage pour l'inférence de programmes réalisant l'extraction automatique d'informations dans des documents semi-structurés arborescents comme les documents HTML ou XML. L'objectif de MOSTRARE est notamment de répondre dans ce cadre aux questions suivantes : que doit-on apprendre ? Comment doit-on l'apprendre ?

Ainsi, l'étude des objets dont on souhaite réaliser l'apprentissage est une des activités du projet MOSTRARE : elle regroupe des travaux autour de formalismes de description de structures arborescentes, qu'ils soient opérationnels comme les automates d'arbres ou plus déclaratifs comme des logiques.

Notre activité autour de TQL s'est ainsi pleinement intégrée dans le projet MOSTRARE.

Pour reprendre notre fil conducteur, dans le cadre des bases de données, le problème de model-checking, tel que nous l'avons décrit, s'appelle le problème de *requête booléenne*. De plus, dans le cas où la formule $\phi$ possède des variables libres alors rechercher les valuations la satisfaisant est le calcul des réponses de la requête. Nous nous sommes également intéressés à l'expressivité de la logique de TQL ainsi qu'à la décision de la satisfiabilité de cette logique.

Le chapitre 6 fait le lien entre la première et la seconde partie du mémoire.

**Chapitre 7** Ce chapitre présente tout d'abord brièvement les notions formelles d'arbres, d'automates arbres. Dans un second temps, nous présentons la logique monadique du second ordre ainsi que certaines de ses extensions. Pour terminer, nous explicitons le lien entre logique et automate. Ce chapitre a pour but d'introduire les outils mathématiques utiles à nos travaux.

**Chapitre 8** Après avoir défini la logique TQL, nous évoquons l'étude que nous avons menée sur son expressivité et l'expressivité de certains de ses fragments. La majeure partie de notre étude concerne les formules closes et nous mesurons donc cette expressivité en termes d'ensembles d'arbres décrits. Nous relions d'ailleurs certains fragments à la logique monadique du second ordre. Outre l'expressivité, nous nous intérressons également au problème de satisfiabilité.

**Chapitre 9** Ce chapitre présente les résultats de complexité obtenus pour le problème de model-checking de TQL dans sa globalité mais également d'un certain nombre de fragments dont ceux évoqués au chapitre précédent. Les résultats concernent la complexité combinée ainsi que la complexité de données.

**Chapitre 10** Ce dernier chapitre expose quelques pistes de recherche à court et moyen termes sur la poursuite des travaux que nous avons présentés dans cette seconde partie.

# Première partie

# Vérification par model-checking du calcul des ambients

# Chapitre 1

# Une introduction au calcul des ambients

Dans ce chapitre, nous présentons le calcul des ambients et certains de ses ancêtres, comme CCS et le $\pi$-calcul. Cette présentation ne se veut nullement exhaustive et ne servira essentiellement qu'à définir les bases de ce calcul en rapport avec notre travail. Ainsi, de nombreux travaux sur le calcul des ambients ainsi que sur CCS, le $\pi$-calcul et bien d'autres formalismes qui leur sont liés ne seront pas mentionnés ici.

## 1.1 Un bref historique

Le calcul des ambients s'inscrit dans une longue lignée de formalismes ayant pour but de modéliser de manière formelle les systèmes ou programmes informatiques ; cette modélisation passe par la définition claire et rigoureuse d'un point de vue mathématique d'une syntaxe et d'une sémantique. L'inspiration de ces travaux est bien sûr le $\lambda$-calcul qui, basé sur une syntaxe et une sémantique opérationnelle minimaliste (la $\beta$-réduction), capture la notion de programmation fonctionnelle.

Le calcul des ambients a pour objet la modélisation de l'informatique concurrente, distribuée et mobile, tant sur le plan logiciel que matériel. Les mots "concurrente" et "mobile" sont déjà apparus comme motivation à l'introduction de deux formalismes, plus précisément deux algèbres de processus, appelés respectivement CCS et $\pi$-calcul. Nous allons tout d'abord présenter ces deux calculs et mentionner leurs limitations qui ont conduit à l'introduction du calcul des ambients.

### 1.1.1 CCS : A Calculus of Communicating Systems

CCS est une algèbre de processus proposée par Milner [Mil80, Mil89] visant à formaliser la notion de programmation concurrente.

Un système (ou programme) est décrit comme un ensemble d'agents concurrents (c'est-à-dire s'exécutant indépendamment des autres) et interagissant via un mécanisme de communications synchrones (les émissions et réceptions de messages sont bloquantes). Les agents communiquent les uns avec les autres en utilisant des *ports de communication*, qui sont identifiés par un *nom*. Dans la syntaxe d'un processus, le nom $a$ désignera l'action de recevoir un message sur le port $a$ tandis que que le *co-nom* $\overline{a}$ désignera l'action d'émettre un message sur le port $a$. Nous allons supposer dans un premier temps que ces communications ne sont en fait que des points de synchronisations par rendez-vous ; ceci correspond à des processus qui émettraient et recevraient uniquement des messages dont le contenu est vide.

**syntaxe** La construction élémentaire d'un agent est la mise en séquence d'actions : pour une action $\alpha$ ($\alpha$ étant une émission $\overline{a}$ ou une réception $a$), $\alpha.P$ désigne un agent qui effectuera l'action $\alpha$ et pour-

suivra son exécution comme $P$. Ainsi, si un agent $\overline{a}.P$ propose une émission via le port $a$ et l'agent $a.Q$ une réception pour ce même port, alors une interaction peut se produire entre ces deux agents : la synchronisation a lieu et les 2 agents poursuivent leur exécution comme $P$ et $Q$ respectivement.

Hormis la composition séquentielle d'actions, CCS dispose de bien d'autres caractéristiques que nous allons présenter au travers d'un exemple simple, celui d'un distributeur de boissons. Pour simplifier, notre distributeur n'accepte que les pièces de 50 centimes et propose comme choix (restreint) de boissons (thé ou café). Thé et café coûtent 50 centimes. Notre distributeur sera idéal puisque sa caisse ne sera jamais pleine et il ne sera jamais en rupture de boissons.

Notre modélisation sera réalisée au travers d'un ensemble d'équations (récursives) définissant les différents agents et/ou états du système.

$$\mathsf{Distributeur} \stackrel{def}{=} \mathsf{Machine} \mid \mathsf{Caisse} \tag{1.1}$$

$$\mathsf{Machine} \stackrel{def}{=} 50.\mathsf{MachineP} \tag{1.2}$$

$$\mathsf{MachineP} \stackrel{def}{=} comm\_cafe.\mathsf{SertCafe} + comm\_the.\mathsf{SertThe} + annule.(\overline{50} \mid \mathsf{Machine}) \tag{1.3}$$

$$\mathsf{SertCafe} \stackrel{def}{=} \overline{credit}.\overline{cafe}.\mathsf{Machine} \tag{1.4}$$

$$\mathsf{SertThe} \stackrel{def}{=} \overline{credit}.\overline{the}.\mathsf{Machine} \tag{1.5}$$

$$\mathsf{Caisse} \stackrel{def}{=} credit.(\overline{pc} \mid \mathsf{Caisse}) \tag{1.6}$$

Notre agent Distributeur est en fait composite et constitué de deux sous-agents concurrents, la Machine et la Caisse. L'opérateur $\mid$ désigne la composition d'agents, composition permettant la formation d'agents plus complexes (équation 1.1)

La Machine est un agent qui attend (une réception de) 50 centimes. Une fois la somme reçue, elle changera d'état pour devenir MachineP (équation 1.2).

Le comportement de MachineP est plus complexe et comporte plusieurs alternatives selon l'interaction que celle-ci va avoir avec un utilisateur potentiel (équation 1.3). La MachineP peut

- recevoir une commande de café et passer dans l'état SertCafe,
- recevoir une commande de thé et passer dans l'état SertThe ou
- recevoir un ordre d'annulation, rendre (c'est-à-dire réémettre) la pièce et retourner dans son état initial Machine.

On parle ici de choix externe (puisque le comportement de la machine est conditionné par l'interaction avec un agent externe) ; l'opérateur $+$ désigne ce choix. Il convient de remarquer que, par exemple, dès la commande de café reçue, les parties de l'agent concernant la commande de thé ou la demande d'annulation, constituant les autres alternatives, disparaîtront.

Notez aussi que la réémission de la monnaie et le retour dans l'état initial se font de manière concurrente (utilisation de l'opérateur $\mid$). La machine n'attend donc pas que la monnaie soit reprise pour retourner dans l'état initial Machine.

SertCafe et SertThe sont les deux états de la machine dans lesquels elle interagit avec la seconde composante du Distributeur à savoir, la Caisse. SertCafe (respectivement SertThe) émet un crédit en direction de la Caisse, puis un café (respectivement un thé), et enfin retourne vers l'état initial Machine (équations 1.4 et 1.5).

On notera que contrairement à la monnaie en cas d'annulation de la commande, la machine ne retourne dans son état initial qu'après que la boisson commandée soit prise (reçue).

La Caisse après réception d'un crédit, stocke simplement celui-ci sous la forme (d'une émission de) $\overline{pc}$, avant de revenir dans son état Caisse (équation 1.6).

**sémantique opérationnelle** Formellement, la sémantique opérationnelle de CCS peut être définie comme un système de transitions étiquetées (*LTS - Labelled Transition System*) qui décrit les interactions d'un agent avec son environnement (*ie* d'autres agents). Ce système de transition est construit à partir d'un ensemble de règles d'inférences définissant une sémantique opérationnelle structurelle (*SOS - Structural Operational Semantics*) [Plo81]. Nous ne définirons pas formellement cette sémantique *SOS*, nous contentant d'une illustration à partir de notre exemple.

Nous considérons un utilisateur du distributeur ayant le comportement suivant : il met 50 centimes dans le distributeur, commande un café et prend ce dernier. Nous modélisons ceci par l'agent suivant dans lequel **0** désigne le processus terminé, *ie* qui n'interagit plus [1] :

$$\mathsf{Utilisateur} \stackrel{def}{=} \overline{50}.\overline{comm\_cafe}.cafe.\mathbf{0}$$

On écrira comme une transition labellée

$$\mathsf{Utilisateur} \xrightarrow{\overline{50}} \overline{comm\_cafe}.cafe.\mathbf{0}$$

le fait que l'utilisateur peut "émettre" 50 centimes, puis passer dans l'état où il s'apprête à commander un café. Symétriquement, on aura

$$\mathsf{Distributeur} \xrightarrow{50} \mathsf{MachineP} \mid \mathsf{Caisse}$$

pour exprimer que le distributeur peut "recevoir" 50 centimes pour passer dans l'état MachineP | Caisse. Ce nouvel état peut être déduit structurellement de la définition de Distributeur comme Machine | Caisse (équation 1.1) et du fait qu'on peut déduire de l'équation 1.2 la transition $\mathsf{Machine} \xrightarrow{50} \mathsf{MachineP}$. Cette déduction est structurelle vis-à-vis de la composition puisque que si un agent $P$ peut exécuter une action $\alpha$ (pour devenir $P'$), alors le résultat de la composition de cet agent $P$ avec un agent $Q$ pourra également exécuter cette action $\alpha$ (pour devenir $P' \mid Q$).

On aura remarqué que les transitions peuvent être labellées par des noms (comme $50$) ou des co-noms (comme $\overline{50}$).

Utilisateur et Distributeur pouvant exécuter une action duale l'une de l'autre (respectivement $50$ et $\overline{50}$), on en déduit en ce qui concerne le système composé de l'utilisateur et du distributeur

$$\mathsf{Utilisateur} \mid \mathsf{Distributeur} \xrightarrow{\tau} \overline{comm\_cafe}.cafe.\mathbf{0} \mid \mathsf{MachineP} \mid \mathsf{Caisse}$$

La transition est ici labellée par l'action $\tau$. On parle alors de $\tau$-transition par opposition aux labels correspondants aux émissions ou aux réceptions. Les $\tau$-transitions ou transitions silencieuses correspondent à une évolution d'un système obtenu par interaction (interne) de certains de ses composants.

Poursuivons notre description de l'exécution de notre système. Similairement à ce que nous avons vu précédemment, nous avons

$$comm\_cafe.\mathsf{SertCafe} \xrightarrow{comm\_cafe} \mathsf{SertCafe}$$

Comme pour la composition, cette transition s'étend structurellement à l'opérateur de choix $+$ ; si une des alternatives $P_i$ d'un processus $P$ peut exécuter une action $\alpha$ et devenir $P_i'$ alors $P$ globalement peut exécuter $\alpha$ pour devenir $P_i'$. Ainsi, on aura

$$comm\_cafe.\mathsf{SertCafe} + comm\_the.\mathsf{SertThe} + annule.(\overline{50} \mid \mathsf{Machine}) \xrightarrow{comm\_cafe} \mathsf{SertCafe}$$

[1] Souvent, lorsque ce processus inactif **0** apparaît en fin d'une séquence d'actions comme dans $\overline{o}.p.\mathbf{0}$, il sera simplement omis et on écrira plus simplement $\overline{o}.p$.

Cette dernière transition peut se combiner avec l'utilisateur commandant un café, c'est-à-dire

$$\overline{comm\_cafe}.cafe.\mathbf{0} \xrightarrow{\overline{comm\_cafe}} cafe.\mathbf{0}$$

L'interaction entre l'utilisateur et le distributeur produira

$$\overline{comm\_cafe}.cafe.\mathbf{0} \mid \mathsf{MachineP} \mid \mathsf{Caisse} \xrightarrow{\tau} cafe.\mathbf{0} \mid \mathsf{SertCafe} \mid \mathsf{Caisse}$$

Comme nous avons vu précédemment, nous avons

$$\mathsf{SertCafe} \xrightarrow{\overline{credit}} \overline{cafe}.\mathsf{Machine} \qquad \text{et} \qquad \mathsf{Caisse} \xrightarrow{credit} \overline{pc} \mid \mathsf{Caisse}$$

et donc, en étendant structurellement par composition avec $cafe.\mathbf{0}$, l'interaction entre la caisse et la machine donnera

$$cafe.\mathbf{0} \mid \mathsf{SertCafe} \mid \mathsf{Caisse} \xrightarrow{\tau} cafe.\mathbf{0} \mid \overline{cafe}.\mathsf{Machine} \mid \overline{pc} \mid \mathsf{Caisse}$$

Finalement, en étendant structurellement par composition de l'interaction entre l'utilisateur et la machine $cafe.\mathbf{0} \mid \overline{cafe}.\mathsf{Machine} \mid \overline{pc} \mid \mathsf{Caisse} \xrightarrow{\tau} \mathbf{0} \mid \mathsf{Machine} \mid \overline{pc} \mid \mathsf{Caisse}$.

L'utilisateur $\mathbf{0}$ a terminé son interaction, la machine est revenue dans son état initial et le contenu de la caisse a augmenté d'une pièce.

Dans le cas classique de la programmation séquentielle, l'équivalence de deux programmes est simplement définie comme le fait de calculer la même chose : ainsi, deux fonctions sont équivalentes si pour toute entrée, elles produisent le même résultat. L'équivalence des programmes CCS, ou plus généralement concurrents, est plus complexe, puisque la notion de résultat n'y est pas évidente.

La notion d'équivalence entre agents est en fait purement comportementale. Deux agents sont équivalents s'ils permettent les mêmes interactions avec leur environnement. Le système de transitions étiquetées décrivant le comportement d'un agent dans ses interactions potentiels, il peut donc servir de mesure d'équivalence : deux agents sont équivalents s'ils possèdent des systèmes de transitions étiquetées isomorphes. Des formes similaires ou plus faibles (notamment vis-à-vis des $\tau$-transitions) d'équivalences entre agents sont données par les relations de bisimulation.

Il est à noter que des propriétés algébriques des opérateurs qui paraissent naturelles comme la commutativité des opérateurs de choix et de composition ou l'idempotence de l'opérateur de choix ne sont en réalité déductibles que vis-à-vis d'une équivalence comportementale ; ainsi, comme pour tout agent $P$ et $Q$, si $P \mid Q$ se comporte identiquement à $Q \mid P$, alors on en déduira que la composition est commutative.

**renommage et restriction** Nous terminons la présentation de CCS par deux aspects de la syntaxe que nous n'avons pas encore abordés ; le premier permet de définir de manière paramétrique un agent. Le second est plus important puisqu'il permet de restreindre la portée des interactions.

On peut remarquer que le mécanisme commande/service de la boisson est identique pour le thé et le café, seul varie l'objet de la commande et évidemment, l'objet du service. On pourrait définir de manière générique un agent SetBoisson défini comme suit :

$$\mathsf{SertBoisson} \stackrel{def}{=} commande.\overline{credit}.\overline{boisson}.\mathsf{Machine} \tag{1.7}$$

Il suffirait alors de pouvoir renommer $commande$ en $comm\_cafe$ et $boisson$ en $cafe$ dans la définition SertBoisson pour obtenir $comm\_cafe$.SetCafe. La syntaxe de CCS permet ce renommage de la manière suivante :

$$\mathsf{SertBoisson}[comm\_cafe/commande, cafe/boisson]$$

Ainsi, on obtient un système équivalent en remplaçant dans l'équation 1.3, $comm\_cafe$.SertCafe par SertBoisson$[comm_cafe/commande, cafe/boisson]$ ainsi que l'expression $comm\_the$.SertThe par SertBoisson$[comm_the/commande, the/boisson]$.

Nous avons conservé le contenu de la Caisse en utilisant une émission $\overline{pc}$ [2]. Nous associons conceptuellement $\overline{pc}$ à la caisse alors que cela n'est pas tout à fait exact. Ainsi, considérons l'agent Voleur défini comme $pc$.Voleur dont le but est de vider la caisse.

Le système Utilisateur | Distributeur | Voleur va évoluer vers $\mathbf{0}$ | Distributeur | Voleur. Le Distributeur a donc finalement une caisse vide. Pour parer ce problème, CCS dispose d'un opérateur $\backslash L$ qui pour un ensemble de noms $L$, restreint les ports du processus $P\backslash L$ à ceux ne figurant pas dans $L$. Pour tout port présent dans $L$, celui-ci sera rendu local à $P$ par la construction $P\backslash L$. Ainsi, notre distributeur sera sécurisé en étant défini comme

$$\text{Distributeur} \stackrel{def}{=} \text{Machine} \mid \text{Caisse}\backslash\{pc\}$$

Le nom $pc$ voit sa portée restreinte à la Caisse.

**CCS et communications** Bien qu'ayant distingué émission et réception, nous nous sommes restreints au cas où les interactions entre agents n'étaient que synchronisations. Il convient de noter que d'un point de vue purement calculatoire ceci est suffisant puisque le formalisme que nous avons présenté est Turing-complet. Cependant, afin de modéliser plus simplement de nombreux systèmes, un mécanisme de communications est souhaitable.

Les communications impliquent évidemment l'existence de valeurs, d'objets à communiquer comme par exemple des chaînes, des entiers. Ainsi, les communications entraînent la nécessité d'ajout de données mais également de variables, d'opérations de traitements pour ces données dans le langage CCS. Toutes ces extensions sont hors de notre propos et nous allons simplement nous contenter de formuler la syntaxe des communications :

– émission d'une valeur $v$ sur le port $a$, la continuation étant $P$ : $\overline{a}\langle v\rangle.P$
– réception d'une valeur dans la variable $x$ sur le port $a$, la continuation étant $Q$ : $a(x).Q$

L'interaction entre ces deux agents produira une communication de la valeur $v$ à $Q$ ; $Q$ verra donc sa variable $x$ remplacée par la valeur $v$. Ainsi,

$$\overline{a}\langle v\rangle.P \mid a(x).Q \xrightarrow{\tau} P \mid Q\{x \leftarrow v\}$$

**limitations de CCS** Comme conclusion à cette section, essayons d'enrichir notre système par un employé de la société d'exploitation de notre distributeur ; celui-ci a pour rôle de périodiquement récupérer l'argent se trouvant dans la caisse. Puisque nous avons sécurisé le caisse, la seule solution pour modéliser notre système est d'écrire

$$\text{Machine} \mid (\text{Caisse} \mid \text{Employe})\backslash\{pc\}$$

qui ferait de l'employé une "partie" du système Distributeur, immobilisé près de la caisse. Cette situation est due au fait que le réseaux d'interconnexion, c'est-à-dire la façon dont les agents sont connectés entre eux (via les ports), est fixée une fois pour toute dans un système CCS.

Une solution simple serait la possibilité pour la Caisse de transmettre à Employe le nom local $pc$ pour qu'il puisse œuvrer. Pour cela, les ports doivent eux-même être des valeurs qui peuvent être émises et reçues. C'est cette caractéristique qui distingue CCS du $\pi$-calcul.

[2] Nous aurions également pu parfaitement utiliser une réception.

### 1.1.2 Le $\pi$-calcul

Comme nous l'avons déjà mentionné, dans CCS, la structure d'interconnexion des agents (reflétée grossièrement par les noms de ports qu'ils possèdent) est fixe. Pour pallier ce problème, Milner, Parrow et Walker ont proposé une nouveau formalisme appelé $\pi$-calcul [MPW92].

L'idée de base du $\pi$-calcul pour permettre la dynamicité du réseau d'interconnexion des agents est d'étendre CCS en faisant des noms de ports (appelés désormais *canaux* dans le $\pi$-calcul) des valeurs possibles de la communication [3]. Les communications permettent à un agent d'acquérir de nouveaux (noms de) canaux, ce qui se traduit par l'acquisition de nouveaux interlocuteurs potentiels.

Cette idée est poussée à l'extrême puisque les canaux du $\pi$-calcul sont des noms qui ne transportent que d'autres noms (de canaux). Le $\pi$-calcul est en ce sens un *calcul nominal* qui ne considère outre les processus qu'un seul type d'objet syntaxique, les noms.

Alors que la plupart des constructions de CCS restent insensibles à cette extension, il en va différemment pour la restriction ; en effet, si un nom $a$ est restreint à un processus $P$ syntaxiquement écrit dans CCS comme $P\backslash\{a\}$, que se passe-t-il si $P$ communique $a$ à un autre agent $Q$ ?

Dans le $\pi$-calcul, la restriction change donc de nature et de syntaxe ; $(\nu n)P$ désigne un processus dont le nom $n$ est restreint et lié par le lieur $(\nu)$. Ceci implique que les noms restreints (tout comme les paramètres formels de la réception, *e.g.* $x$ dans le processus $c(x).P$) peuvent désormais subir une $\alpha$-conversion : ainsi, les processus $(\nu n)\overline{x}\langle n\rangle.n(y).\overline{a}\langle y\rangle\mathbf{0}$ et $(\nu m)\overline{x}\langle m\rangle.m(y).\overline{a}\langle y\rangle\mathbf{0}$ sont équivalents.

Détaillons la syntaxe du processus $(\nu n)\overline{x}\langle n\rangle.n(y).\overline{a}\langle y\rangle\mathbf{0}$ : celui-ci émet le nom (de canal) restreint $n$ sur le canal $x$, puis écoute sur ce canal $n$ en attente d'une réception (ayant pour paramètre formel le nom $y$). Un fois la réception effectuée, le nom reçu est réémis sur le canal $a$.

Cet exemple montre que la restriction ne peut plus être fixe : le nom $n$ est un secret que notre processus partage avec un tiers en le communicant. Ainsi, si $(\nu n)$ désigne l'espace où $n$ est partagé alors la communication $n$ entraînera une extension de la portée de $n$ (*scope extrusion*). Inversement, si un processus est censé partager un secret avec un tiers mais que ce dernier n'est pas (plus) au courant alors l'espace de partage décrit par $(\nu n)$ pourra être réduit (*scope intrusion*).

Initialement, la sémantique du $\pi$-calcul est définie comme celle de CCS par un système de transitions étiquetées provenant d'une sémantique de type SOS. Cependant inspiré par la machine chimique abstraite de Berry et Boudol [BB92], Milner propose dans [Mil90] une sémantique opérationnelle alternative pour le $\pi$-calcul. Cette sémantique se base sur deux relations définies sur l'ensemble des processus. La première notée $\equiv$ est une relation de congruence dite structurelle : elle associe des formes syntaxiques différentes d'un même processus et comme dans la machine chimique abstraite, permet aux "processus réactants" souhaitant interagir de venir au contact. Elle spécifie par exemple que la composition est une opération associative et commutative. C'est cette relation de congruence structurelle qui définit les opérations d'extension et de réduction de porté de la restriction : le processus $(\nu n)(P \mid Q)$ est congru à $((\nu n)P) \mid Q$ si $n$ n'apparaît pas (libre) dans $Q$, c'est-à-dire si $Q$ ne connaît pas le secret $n$. La seconde partie de la sémantique est définie par une relation de réduction notée $\rightarrow$ qui traduit notamment l'interaction par communication entre processus, correspondant ainsi aux $\tau$-transitions de la sémantique SOS. Ainsi, la règle de communication pour deux processus voulant interagir via le canal $x$ est donnée par [4]

$$\overline{x}\langle y\rangle.P \mid x(z).Q \rightarrow P \mid Q\{z \leftarrow y\}$$

Le processus $Q\{z \leftarrow y\}$ désigne le processus $Q$ dans lequel toutes les occurrences libres du nom $z$ ont été remplacées par $y$ en prenant garde que toutes les occurrences remplacées demeurent des occurrences libres de $y$ (ceci pouvant nécessiter une $\alpha$-conversion de certaines variables liées).

[3] En réalité c'est un petit peu plus compliqué et le chemin n'a pas été immédiatement franchi entre les deux calculs [EN00].

[4] Nous ne considérons ici que la règle pour un calcul sans opérateur de choix, opérateur qui peut être simulé dans le calcul présenté ici. Cependant, une règle plus complexe pour un calcul incluant cet opérateur de choix existe.

Cette relation de réduction comprend également des règles structurelles qui précisent sous quel contexte les communications peuvent avoir lieu, comme par exemple

$$\frac{P \to Q}{P \mid R \to Q \mid R}$$

La communication de canaux permet donc de modéliser des systèmes dans lequel le réseau de communications des différentes composantes évolue. Ceci offre donc une modélisation d'une certaine forme de mobilité basée sur la distribution des noms de canaux sur les agents. Le déplacement est en réalité traduit par le changement des noms que possède un agent ; intuitivement, deux agents, qui peuvent communiquer l'un avec l'autre, sont "proches".

Dans [Mil90], Milner introduit une nouvelle construction permettant de remplacer la récursion définie par un ensemble d'équations. Cette construction appelée *réplication* est notée $!P$ et peut être vue comme une nombre non borné de compositions de processus $P$ ; sa définition est donnée par la congruence structurelle comme

$$!P \equiv !P \mid P$$

Il convient de noter que la congruence structurelle comprenant à la fois les mécanismes de réplication (ou de récursion) et d'extension de portée de la restriction fait que le $\pi$-calcul offre un mécanisme de génération dynamique de noms "frais", *ie* distincts de tous les noms précédemment présents dans le processus. Ainsi, considérons un serveur d'impression

$$!print(x).(\nu jobid)\overline{x}\langle jobid\rangle.jobid(f).P(f)$$

Ce serveur attend sur le canal $print$ une requête d'impression sous la forme de l'identité du processus voulant imprimer ; il se sert ensuite de cette identité comme d'un canal pour émettre un identificateur unique pour le "job d'impression". Cet identificateur du job d'impression est alors utilisé comme canal pour recevoir le fichier à imprimer. Si ce serveur d'impression est composé avec les processus $Q_i$ (pour $i \in \{1, 2\}$) définis comme $\overline{print}\langle p_i\rangle.p_i(j).\overline{j}\langle f_i\rangle$, on obtiendra après réduction le processus

$$(\nu j)(\nu j')!print(x).(\nu jobid)\overline{x}\langle jobid\rangle.jobid(f).P(f) \mid j(f).P(f) \mid j'(f).P(f) \mid \overline{j}\langle f_1\rangle \mid \overline{j'}\langle f_2\rangle$$

dans lequel les noms frais $j$ et $j'$ ont été crées.

**Example 1** *Reprenons notre exemple de distributeur et plus précisément sur l'interaction entre la* Caisse *et l'*Employe*; informellement, la* Caisse *attend sur un canal* $verrou$ *l'identificateur de l'*Employe*. Elle utilise ensuite cet identificateur comme un canal privé entre elle et l'employé, canal utilisé pour transmettre le nom privé désignant les pièces de la caisse. L'*Employe *utilise alors ce nom pour retirer un certain montant de la* Caisse*. Ainsi, cet employé sera défini par le processus*

$$(\nu id)\overline{verrou}\langle id\rangle.id(p).p().p()$$

*Notez que nous utilisons simplement* $p()$ *pour désigner une synchronisation sur le canal* $p$*. La* Caisse *quant à elle est modélisée par le processus*

$$(\nu pc)!credit().\overline{pc}\langle\rangle \mid \overline{pc}\langle\rangle \mid \overline{pc}\langle\rangle \mid !verrou(i).\overline{i}\langle pc\rangle$$

*La première partie de la caisse* $!credit().\overline{pc}\langle\rangle$ *traduit son interaction avec la* Machine*, tandis que la seconde partie* $!verrou(i).\overline{i}\langle pc\rangle$ *définit celle avec l'*Employe*. Ainsi, si on compose la* Caisse *et l'*Employe*,*

*en utilisant la congruence structurelle définissant la réplication ($!P \equiv !P \mid P$) et en permettant l'extension de portée des restriction, on aura le processus*

$$(\nu id)\underbrace{(\nu pc)!credit().\overline{pc}\langle\rangle \mid !verrou(i).\overline{i}\langle pc\rangle \mid \overline{pc}\langle\rangle \mid \overline{pc}\langle\rangle \mid verrou(i).\overline{i}\langle pc\rangle}_{\text{Caisse}} \mid \underbrace{\overline{verrou}\langle id\rangle.id(p).p().p()}_{\text{Employe}}$$

*Après communication de l'identité via la canal $verrou$, le système devient*

$$(\nu id)\underbrace{(\nu pc)!credit().\overline{pc}\langle\rangle \mid !verrou(i).\overline{i}\langle pc\rangle \mid \overline{pc}\langle\rangle \mid \overline{pc}\langle\rangle \mid \overline{id}\langle pc\rangle}_{\text{Caisse}} \mid \underbrace{id(p).p().p()}_{\text{Employe}}$$

*De nouveau l'extension de portée de la restriction est utilisée permettant après communication (sur le canal $id$) à la* Caisse *et à l'*Employe *de partager le nom secret $pc$,*

$$(\nu id)(\nu pc)\underbrace{!credit().\overline{pc}\langle\rangle \mid !verrou(i).\overline{i}\langle pc\rangle \mid \overline{pc}\langle\rangle \mid \overline{pc}\langle\rangle}_{\text{Caisse}} \mid \underbrace{pc().pc()}_{\text{Employe}}$$

*Cette dernière communication reflète la dynamicité du réseau de connexions entre agents, puisque l'employé et la caisse partagent maintenant un canal privé $pc$.*

*Nous laissons au lecteur le soin de compléter la suite des réductions de notre système.*

*Notez que, bien que ne le réalisant pas, il aurait été simple que la caisse effectue une authentification de l'employé en utilisant le nom $id$.*

Jusqu'à présent dans CCS, comme dans le $\pi$-calcul, nous avons considéré les communications comme synchrones : émission et réception de messages sont bloquantes pour l'exécution d'un processus. Cependant, les communications peuvent également être asynchrones : dans ce cas, la réception reste bloquante puisque la valeur reçue va influencer la continuation du calcul, alors que l'émission ne le sera plus. Un processus voulant émettre un message, ne restera pas en attente d'un receveur potentiel. Il émettra le message (même si personne à cet instant précis n'est là pour le recevoir) et poursuivra son exécution. Pratiquement, on peut imaginer que le message ainsi émis sera alors stocké dans une file ou un ensemble de messages en attente d'être reçu. Dans [Bou92], Boudol propose une version asynchrone du $\pi$-calcul. Cette version est particulièrement simple puisque purement syntaxique : la continuation des processus souhaitant réaliser une émission disparaît. Ainsi, un processus émettant un nom $y$ sur le canal $x$ et continuant son exécution comme $P$ s'écrira simplement $\overline{x}\langle y\rangle.\mathbf{0} \mid P$. Il est, de plus, très simple de simuler le $\pi$-calcul synchrone dans le $\pi$-calcul asynchrone en faisant que les processus émetteurs se bloquent en attente d'une confirmation de la réception.

Bien qu'offrant de grandes possibilités pour la modélisation de systèmes concurrents, le $\pi$-calcul a néanmoins quelques défauts : la notion de concurrence est souvent associée à celle de distribution, c'est-à-dire à des agents ou processus s'exécutant sur un ensemble de sites distincts, sites pouvant être des machines ou simplement des processeurs. Le $\pi$-calcul n'offre cependant pas la possibilité de modéliser la notion de distribution. Ainsi, des extensions du $\pi$-calcul ont été proposées pour modéliser la distribution et des notions qui lui sont attachés comme les pannes (*failures*) ou la migration (d'un site à un autre).

Citons notamment le $\pi_l$-calcul d'Amadio et Prasad [AP94] qui considère un notion de localités sur lesquelles processus et canaux sont distribués. Ce calcul considère les localités comme des objets de premier ordre qui peuvent ainsi être créés dynamiquement et dont les noms peuvent être communiqués. L'objectif des auteurs est d'étudier le comportement de tels systèmes en présence d'apparition (non-déterministe) de pannes de certains sites, pannes que les autres processus peuvent détecter et ainsi adapter leur comportement. Amadio poursuit dans [Ama97, Ama00] cette étude dans le cas d'un calcul

asynchrone, le $\pi_{1\ell}$-calcul, et ajoute la possibilité de création de processus distants (*process spawning*), c'est-à-dire s'exécutant sur un autre site.

Mentionnons également les travaux de Riely et Hennessy [RH98] qui proposent une extension du $\pi$-calcul appelé $D\pi$, qui comprend comme le $\pi_{1\ell}$-calcul une notion de localités auxquelles les processus sont attachées ; on note $\ell[P]$ le fait que le processus $P$ se retrouve sur le site $\ell$. Ces localités sont également des objets de premier ordre qui peuvent être crées et dont le nom peut être communiqué. Cependant contrairement au $\pi_{1\ell}$-calcul où les communications pouvaient s'établir entre sites distincts, les communications dans $D\pi$ ne peuvent s'établir qu'entre processus d'une même localité. Pour établir des communications "longues distances", le calcul dispose également de la possibilité de faire migrer un processus d'un site à un autre. Mais la caractéristique principale de $D\pi$ est que, contrairement au $\pi_{1\ell}$-calcul, la structure des localités n'est pas plane mais arborescente : cette arborescence est produite soit au moment de créations d'une nouvelle localité qui peut être insérée à l'intérieur d'une autre localité, soit par l'exécution d'une instruction d'un processus qui déclenche la migration de sa localité d'appartenance au sein de l'arborescence. Cette structure d'arbre ne se retrouve pas directement dans la syntaxe du processus mais dans un environnement d'exécution qui lui est associé pour définir la sémantique opérationnelle via une relation de réduction. Cet aspect différencie $D\pi$ du calcul des ambients que nous allons présenter à la section suivante ; de même, contrairement au calcul des ambients, chaque (nom de) localité n'apparaît qu'une seule fois dans l'arbre. Ceci est notamment exprimé au niveau de la congruence structurelle de $D\pi$ par la règle

$$\ell[P] \mid \ell[Q] \equiv \ell[P \mid Q]$$

Notons également que $D\pi$ intègre la possibilité de définir une politique d'utilisation des noms transmis : permissions de réception/émission pour les noms de canaux, exécution/arrêt/migration pour les noms de localités. Ceci se retrouvera dans le calcul des ambients dans le mécanisme de communications des capacités qui transmettront non pas un nom, mais une action précise concernant ce nom.

Ces travaux préfigurent une évolution de la notion de mobilité : ce ne seront plus les échanges de noms permettant la transformation du réseau d'interconnexion des agents qui traduiront la mobilité mais bien les agents eux-même qui migreront de sites en sites ou pour reprendre les termes de Dal Zilio [Dal01] de "labile", les systèmes deviennent "motile". Cet aspect de mobilité "physique" des agents est le fondement même du calcul des ambients, calcul que nous allons décrire dans la section suivante.

## 1.2 Le calcul des ambients

La mobilité dans son sens le plus large dans la communauté informatique revêt différentes formes notamment lorsque est abordée la distinction matériel/logiciel. Les aspects "matériels" souvent regroupés sous le terme d'informatique mobile (*mobile computing*) traitent les problèmes de connectique et d'interopérabilité pour des appareils tels que les ordinateurs et téléphones portables ou les assistants personnels. Les aspects "logiciels" de la mobilité (*mobile computation*) regroupent la programmation d'applications physiquement distribuées impliquant notamment l'appel à des programmes distants par exemple via RPC/RMI (*Remote Procedure Call / Remote Method Invocation*), le téléchargement de code (mécanisme de chargement dynamique de classes de JAVA) ou le code mobile (applets, agents). De plus, cette mobilité peut désormais se réaliser à l'échelle planétaire, supportée par l'internet, où les temps de latence des communications peuvent être importants et où les connexions de type GPRS ou Wifi, soumises à des contraintes physiques, peuvent être parfois intermittentes.

Le $\pi$-calcul offre un mécanisme simpliste, uniforme et inaltérable de communications ou d'accès aux ressources, puisque le fait de posséder un nom permet de communiquer avec un tiers et cette capacité persiste tant que les deux partenaires le souhaitent. Le $\pi$-calcul serait un monde où la connaissance de l'adresse d'une machine me permettrait sans encombre de communiquer avec elle et cela de manière

permanente. En réalité, il n'en est rien. Pare-feu, réseaux locaux, privés et virtuels sont autant de barrières délimitant des domaines d'administration qui ne se laisseront pénétrer que par des entités (messages, agents) autorisées.

Cardelli affirme dans [Car99] que la problématique principale de la mobilité doit être de "franchir des barrières", barrières matérielles comme l'accès à un réseau sans fil ou barrières logicielles comme le franchissement d'un pare-feu. Il décrit en ces termes les caractéristiques fondamentales que doit posséder un calcul pour permettre la modélisation de systèmes mobiles à grande échelle, tant matériels que logiciels, puisque devant être traitées selon lui de manière uniforme. Alors que la notion de localité était ajoutée au dessus d'un formalisme existant comme dans $\pi_{1\ell}$ ou $D\pi$, il propose d'en faire l'objet de bases des systèmes mobiles. Ces localités ou *ambients* seront des barrières partitionnant "l'espace matériel et logiciel". Ces ambients seront l'unité de la base des différentes caractéristiques d'un système mobile. Un ambient sera une unité de :

- localité : un processus sera attaché à un ambient de même que récursivement d'autres ambients. Ceci créera une hiérarchie de domaines tels que portable/sous-réseau/LAN/internet.
- mobilité : une localité pourra s'échapper d'une localité englobante ou au contraire entrer dans une autre, sous réserve de posséder les autorisations de franchissement de ses barrières ; le portable interrompt sa connexion GPRS en entrant dans la gare pour se connecter au réseau Wifi de celle-ci.
- distribution : l'ambient décrit clairement un espace qui peut être associé à un processeur. Il délimite une sphère de communications, rendant celles-ci locales, évitant les problèmes de communications distribuées entre différents sites.
- sécurité : l'ambient délimite clairement deux espaces entre lesquels les interactions doivent être explicitées et autorisées.

Cardelli retient comme formalisme pour modéliser les systèmes mobiles le calcul des ambients qu'il a précédemment introduit avec Gordon [CG98].

### 1.2.1 Une présentation graphique

Le calcul des ambients a donc pour but la modélisation des aspects mobiles de l'informatique en intégrant notamment au plus bas niveau les notions de barrières, localités ou domaines.

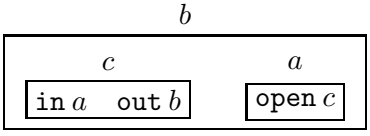
Informellement, un ambient est une boîte portant un nom, dans laquelle des processus s'exécutent ; il contient également d'autres ambients (boîtes) créant ainsi une hiérarchie d'ambients. Les actions exécutées par les processus sont des *capacités* de mobilité et des actions de communications.

Les ambients constituent ainsi des unités de distribution tant sur le plan matériel que logiciel, puisque définissant clairement un espace de calcul.

Les ambients constituent surtout une unité de déplacement, puisque lorsque ceux-ci se déplacent, ils se déplacent avec leur contenu. De plus, la mobilité dans le calcul des ambients est *subjective* dans le sens où c'est l'ambient qui bouge de son propre chef en exécutant une de ses capacités. Ceci s'oppose à une mobilité *objective* où l'ambient se trouve déplacé par un tiers.

Les capacités (de mobilité) `in`, `out`, `open` permettent à un ambient qui les exécutent de respectivement entrer dans un ambient voisin de nom $a$ (`in` $a$), de sortir de l'ambient englobant de nom $b$ (`out` $b$), d'ouvrir l'enveloppe d'un ambient contenu de nom $c$ (`open` $c$).

**Example 2** *Voici décrit de manière graphique un processus du calcul des ambients :*



*Il est composé de 3 ambients, nommés $a$, $b$ et $c$ ; les ambients $a$ et $c$ se trouvent à l'intérieur de l'ambient $b$. De plus, $c$ dispose des capacités* `in` $a$, `out` $b$ *et $a$ de la capacité* `open` $c$.

*Etudions maintenant la dynamique de ce processus selon le rôle des capacités que nous avons informellement décrit (voir figure 1.1).*

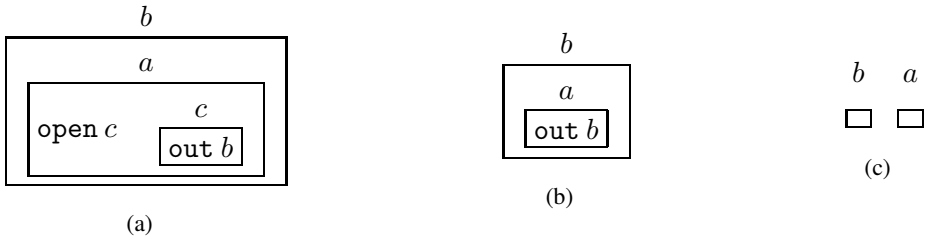



FIG. 1.1 – Dynamique des ambients

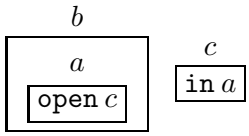
*L'ambient $c$ peut exercer sa capacité à entrer dans l'ambient $a$ qui se trouve à ses cotés : l'ambient $c$ se déplace à l'intérieur de l'ambient $a$ et perd sa capacité* `in` $a$ *(figure 1.1(a)).*

*L'ambient $a$ peut alors exercer sa capacité à ouvrir un ambient de nom $c$ ; l'enveloppe de l'ambient $c$ est alors dissoute et son contenu (*`out` $b$*) devient contenu de $a$ (figure 1.1(b)).*

*On peut remarquer que $a$ possède dorénavant la capacité à sortir de $b$ (*`out` $b$*), capacité qu'il n'avait pas initialement. Ainsi, ouvrir des ambients offre la possibilité d'acquérir de nouvelles capacités (nous verrons plus tard que les communications rendent également possible cette acquisition).*

*Finalement, l'ambient $a$ exerçant sa capacité* `out` $b$*, le processus obtenu est celui de la figure 1.1(c).*

*Il faut noter que partant du processus initial, l'ambient $c$ pouvait pleinement exercer sa capacité* `out` $b$*, donnant ainsi le processus*




*On voit ainsi apparaître le non-déterminisme de l'exécution d'un processus du calcul des ambients. Remarquons également que dans ce dernier processus, plus aucun ambient ne peut exercer une de ses capacités, l'ambient $a$ n'étant notamment plus voisin de l'ambient $c$.*

*Si on souhaite parer cette possibilité d'exécution, il convient simplement à l'intérieur de $c$ de composer les deux capacités en séquence* `in` $a$.`out` $b$*, imposant à* `in` $a$ *d'être exercer en premier.*

Décrivons maintenant de manière formelle la syntaxe des processus du calcul des ambients ainsi la notion d'exécution des ces processus.

### 1.2.2 Syntaxe et sémantique

**syntaxe du calcul des ambients** Nous supposons un ensemble dénombrable de noms $\mathcal{N}$ et on utilisera notamment $a$, $b$, $c$, $m$, $n$, ... pour désigner les éléments de $\mathcal{N}$. Les processus du calcul des ambients sont définis à la figure 1.2.

Les opérateurs de restriction (de nom) et de réception sont les seuls lieurs de la syntaxe ; le nom $n$ est lié dans les expressions $(\nu n)P$ et $(n).P$. Nous supposerons que deux processus $\alpha$-équivalents (*ie* pouvant être obtenus l'un de l'autre par renommage de leurs variables liées) sont égaux. Une occurrence d'un nom $n$ est libre si elle n'est pas dans la portée d'un lieur $(\nu n)$ ou $(n)$. On notera $fn(P)$ l'ensemble des noms ayant au moins une occurrence libre dans le processus $P$.

| $P ::=$ | processus | $M ::=$ | capacités |
|---|---|---|---|
| $\mathbf{0}$ | inactif | $n$ | nom |
| $M[P]$ | ambient | $\texttt{in}\ M$ | entrer dans $M$ |
| $P \mid Q$ | composition | $\texttt{out}\ M$ | sortir de $M$ |
| $M.P$ | préfixe de capacités | $\texttt{open}\ M$ | ouvrir $M$ |
| $(n).P$ | réception | $\epsilon$ | séquence vide |
| $\langle M \rangle$ | émission | $M.M'$ | concaténation |
| $!P$ | réplication | | |
| $(\nu n)P$ | restriction de nom | | |

FIG. 1.2 – Processus et capacités

**sémantique opérationnelle** La sémantique opérationnelle du calcul des ambients est définie à l'aide de deux relations, la relation de congruence structurelle et la relation de réduction.

La relation de *congruence structurelle*, notée $\equiv$, est définie sur l'ensemble des processus. Cette relation identifie des processus qui, bien que syntaxiquement différents, se veulent désigner le même objet et posséder ainsi le même comportement. Cette relation est définie comme étant la plus petite vérifiant les axiomes donnés à la figure 1.3.

| | |
|---|---|
| $\equiv$ est une congruence | (Struct Congr) |
| $P \mid Q \equiv Q \mid P$ | (Struct Par Comm) |
| $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$ | (Struct Par Assoc) |
| $P \mid \mathbf{0} \equiv P$ | (Struct Zero Par) |
| $(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P$ | (Struct Res Res) |
| $(\nu n)\mathbf{0} \equiv \mathbf{0}$ | (Struct Res Zero) |
| $(\nu n)(P \mid Q) \equiv P \mid (\nu n)Q$ si $n \notin fn(P)$ | (Struct Res Par) |
| $(\nu n)(m[P]) \equiv m[(\nu n)P]$ si $n \neq m$ | (Struct Res Amb) |
| $\epsilon.P \equiv P$ | (Struct $\epsilon$) |
| $(M.M').P \equiv M.M'.P$ | (Struct .) |
| $!P \equiv P \mid !P$ | (Struct Repl Copy) |
| $!\mathbf{0} \equiv \mathbf{0}$ | (Struct Repl Zero) |
| $!(P \mid Q) \equiv !P \mid !Q$ | (Struct Repl Par) |
| $!!P \equiv !P$ | (Struct Repl Repl) |

FIG. 1.3 – Congruence Structurelle $P \equiv Q$

La relation de congruence structurelle énonce notamment (outre le fait d'être une congruence) que la relation de composition est associative (Struct Par Assoc), commutative (Struct Par Comm) et que le processus inactif $\mathbf{0}$ est son élément neutre (Struct Zero Par). Les règles concernant l'opérateur de

restriction (Struct Res _) sont standarts et similaires à celles du $\pi$-calcul. La relation de congruence structurelle stipule de plus que la concaténation de capacités "coïncide" avec la construction de préfixes de capacités (Struct $\epsilon$ et Struct .).

Il convient de noter que les deux axiomes (Struct Repl Par) et (Struct Repl Repl) concernant la réplication ne figurent pas dans les articles qui ont introduit le calcul des ambients [CG98, CG00b] mais sont par exemple considérés dans [CG00a]. Ces axiomes sont souvent également considérés dans le cas du $\pi$-calcul. Ils ont l'avantage d'assurer la décidabilité de la congruence structurelle dans le cas du $\pi$-calcul [Hir99] comme dans le cas du calcul des ambients [Dal00] [5]. On peut de plus aisément prouver que, contrairement à $D\pi$, ces axiomes n'impliquent pas que $a[P] \mid a[Q] \equiv a[P \mid Q]$.

Le second élément de la sémantique opérationnelle du calcul des ambients spécifie la manière dont s'exécutent les processus et est donnée par la relation de réduction $\rightarrow$ définie comme la plus petite relation vérifiant les règles décrites à la figure 1.4.

| | |
|---|---|
| $n[\texttt{in}\, m.P \mid Q] \mid m[R] \rightarrow m[n[P \mid Q] \mid R]$ | (Red In) |
| $m[n[\texttt{out}\, m.P \mid Q] \mid R] \rightarrow n[P \mid Q] \mid m[R]$ | (Red Out) |
| $\texttt{open}\, n.P \mid n[Q] \rightarrow P \mid Q$ | (Red Open) |
| $\langle M \rangle \mid (n).P \rightarrow P\{n \leftarrow M\}$ | (Red I/O) |
| $P \rightarrow Q \Rightarrow P \mid R \rightarrow Q \mid R$ | (Red Par) |
| $P \rightarrow Q \Rightarrow n[P] \rightarrow n[Q]$ | (Red Amb) |
| $P \rightarrow Q \Rightarrow (\nu n)P \rightarrow (\nu n)Q$ | (Red Res) |
| $P' \equiv P, P \rightarrow Q, Q \equiv Q' \Rightarrow P' \rightarrow Q'$ | (Red $\equiv$) |

FIG. 1.4 – Réduction $P \rightarrow Q$

Les trois premières règles (Red In - Red Out - Red Open) définissent formellement l'exécution des capacités `in`, `out` et `open` respectivement. La règle de communication (Red I/O) est directement inspirée du mécanisme de communication du $\pi$-calcul asynchrone.

Il convient cependant de noter que les communications ne s'établissent qu'entre processus voisins, en particulier appartenant au même ambient ; cette notion de localité est suffisante et justifie l'absence de canaux de communication.

Notez aussi que l'objet des communications obéit à la syntaxe $M$, c'est-à-dire est une séquence de noms et/ou de capacités. Ainsi, comme dans le cas du $\pi$-calcul, les noms peuvent être communiqués mais également des (séquences de) capacités créant ainsi la possibilité de nouvelles interactions entre agents.

Les trois règles suivantes (Red Par - Red Amb - Red Res) définissent les contextes possibles sous lesquelles la réduction peut se produire. Finalement, la règle (Red $\equiv$) spécifie que les processus équivalents se réduisent de manière identique (ceci reviendrait à définir la relation de réduction sur les classes d'équivalence de $\equiv$).

Nous noterons $\rightarrow^*$ la clôture réflexive et transitive de la relation de réduction $\rightarrow$.

**Example 3** *Nous allons illustrer les différentes caractéristiques du calcul des ambients au travers d'un exemple ; nous allons modéliser (sommairement) l'accès d'une applet à un site sécurisé par un pare-feu.*

[5] Sans ces axiomes, le problème de décision de la congruence structurelle pour le $\pi$-calcul comme pour le calcul des ambients reste un problème ouvert.

*On retrouvera dans la modélisation des ambients* $applet$, $pare\_feu$ *et* $site\_securise$. *Nous mettrons en avant les capacités ou communications sur le point d'être exécutées.*

*Le* $site\_securise$ *est protégé car placé dans l'ambient* $pare\_feu$ *; l'*$applet$ *émet une requête (sous la forme d'un ambient* $req$*) contenant son identité vers le* $pare\_feu$ *et recevra en retour une notification (sous la forme d'un ambient* $ack$*) contenant le programme de routage vers le site sécurisé. Nous modélisons cette situation par le processus*

$$\begin{array}{c} applet \\ \boxed{\begin{array}{c} req \\ \boxed{\langle applet\rangle \mid \texttt{out}\ applet.\texttt{in}\ pare\_feu\ .\mathbf{0}} \end{array} \mid \texttt{open}\ ack.CODE\_APPLET} \mid \\ pare\_feu \\ \boxed{!(\nu garde)\texttt{open}\ req.(x).GARDE \mid \begin{array}{c} site\_securise \\ \boxed{!\texttt{open}\ navette.\mathbf{0} \mid CODE\_SECURISE} \end{array}} \end{array}$$

*dans lequel les processus* $CODE\_APPLET$ *et* $CODE\_SECURISE$ *sont quelconques et le processus* $GARDE$ *est défini comme*

$$\begin{array}{c} garde \\ \boxed{\begin{array}{c} ack[\texttt{out}\ garde.\texttt{out}\ pare\_feu.\texttt{in}\ x.\texttt{in}\ pare\_feu.\texttt{in}\ garde.\texttt{in}\ navette.\mathbf{0}] \\ \mid navette[\texttt{open}\ x.\texttt{out}\ garde.\texttt{in}\ site\_securise.\mathbf{0}] \end{array}} \end{array}$$

*L'ambient* $req$ *exerce successivement ses capacités* $\texttt{out}\ applet$ *et* $\texttt{in}\ parefeu$ *; il sort donc de* $applet$*, puis entre dans* $parefeu$*,*

$$\begin{array}{c} applet \\ \boxed{\texttt{open}\ ack.CODE\_APPLET} \mid \\ pare\_feu \\ \boxed{!(\nu garde)\texttt{open}\ req.(x).GARDE \mid \begin{array}{c} req \\ \boxed{\langle applet\rangle} \end{array} \mid \begin{array}{c} site\_securise \\ \boxed{!\texttt{open}\ navette.\mathbf{0} \mid CODE\_SECURISE} \end{array}} \end{array}$$

*La requête a ainsi été transmise au* $pare\_feu$*. Maintenant, du fait de (Struct Repl Copy),*

$$!(\nu garde)\texttt{open}\ req.(x).GARDE \equiv !(\nu garde)\texttt{open}\ req.(x).GARDE \mid (\nu garde)\ \texttt{open}\ req\ .(x).GARDE$$

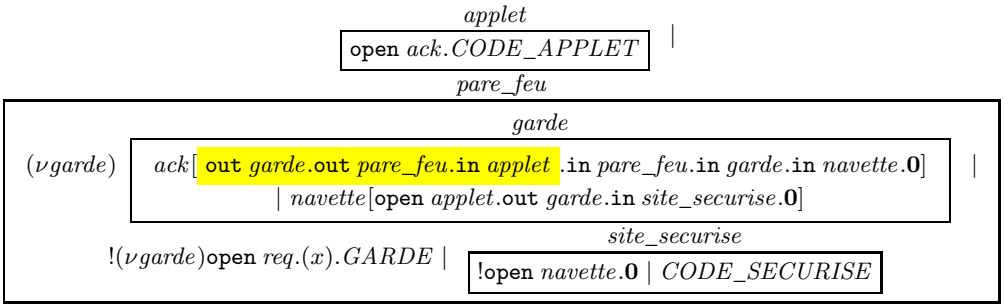
*et de (Struct Res Par),*

$$((\nu garde)\texttt{open}\ req.(x).GARDE) \mid req[\langle applet\rangle] \equiv (\nu garde)(\texttt{open}\ req.(x).GARDE \mid req[\langle applet\rangle])$$

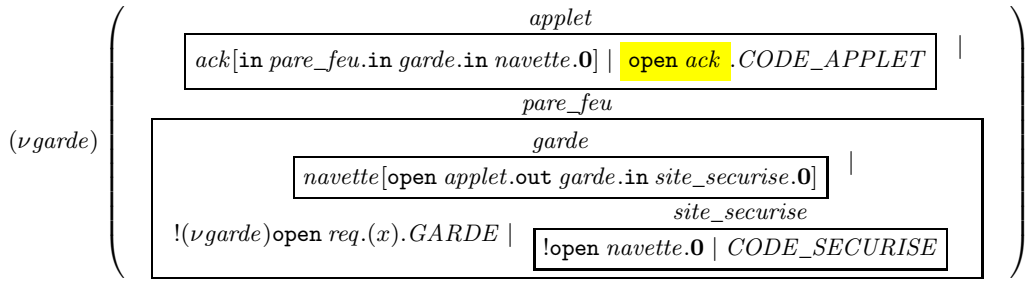
*La capacité* $\texttt{open}\,req$ *de la copie du processus, qui a été exhibée, peut être exécutée, donnant le processus*

$$\begin{array}{c} applet \\ \boxed{\texttt{open}\ ack.CODE\_APPLET} \mid \\ pare\_feu \\ \boxed{\begin{array}{c} (\nu garde)(\ (x)\ .GARDE \mid \langle applet\rangle\ ) \mid \\ !(\nu garde)\texttt{open}\ req.(x).GARDE \mid \begin{array}{c} site\_securise \\ \boxed{!\texttt{open}\ navette.\mathbf{0} \mid CODE\_SECURISE} \end{array} \end{array}} \end{array}$$

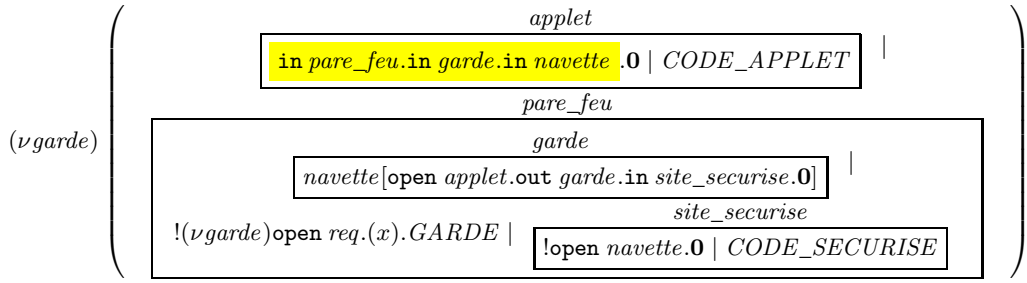
*Une communication peut maintenant avoir lieu, instanciant* $GARDE$ *avec l'identifiant de l'*$applet$



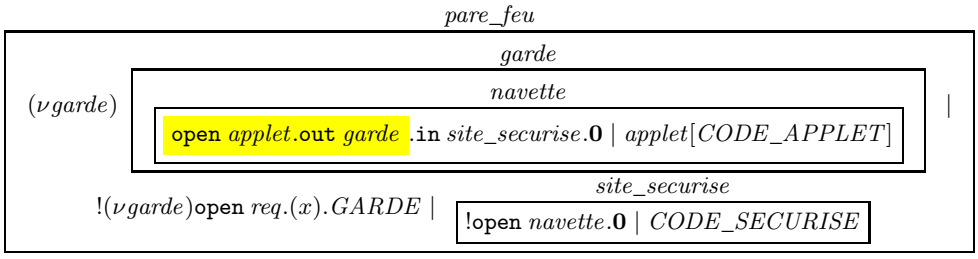

*La notification peut être transmise à l'applet, grâce à l'ambient ack qui sort de garde, puis de pare_feu et entre dans applet. On remarquera que c'est l'action de communication précédente qui a rendu possible cette dernière action,*



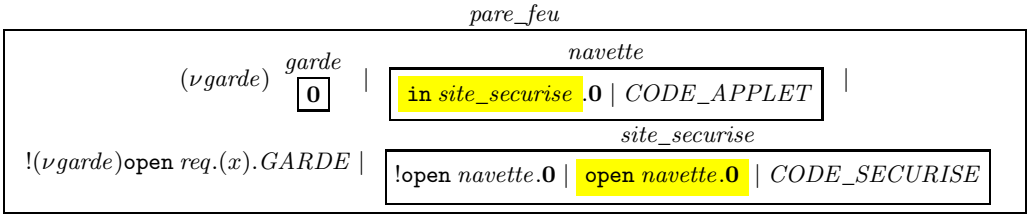

*Notez que le nom garde est maintenant partagé entre le pare_feu et l'applet ; l'ambient applet peut exercer sa capacité* open *ack, lui permettant d'acquérir les capacités pour entrer dans le pare_feu*



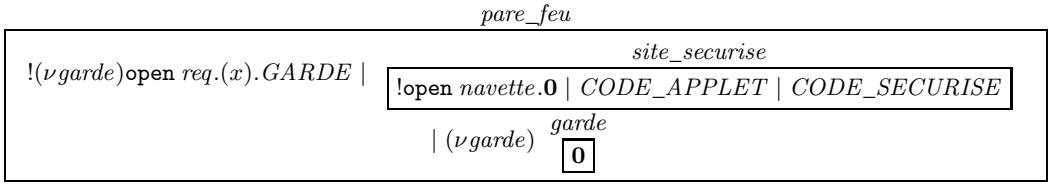

*Une fois ces capacités exécutées,*




*L'ambient navette exécute sa capacité à ouvrir applet et à sortir de garde,*



*L'ambient $navette$ peut alors entrer dans le $site\_securise$, où il sera finalement ouvert*

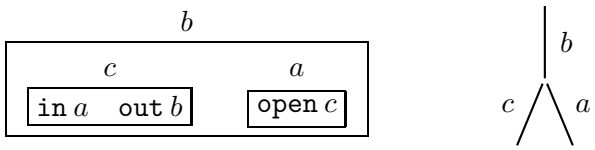



*On pourra remarquer que l'utilisation de l'opérateur de restriction pour le nom $garde$ fait que celui-ci est propre à chaque $applet$ voulant accéder au $site\_securise$. Ceci garantit d'une certaine façon un des aspects concernant la sécurité de l'accès au site. Notez qu'un reliquat de l'interaction entre $applet$ et $pare\_feu$ demeure sous la forme du processus $(\nu garde)garde[\mathbf{0}]$ ; il aurait été assez simple de mettre en œuvre un "ramasse-miette" pour se débarrasser de celui-ci. Cependant, d'un point de vue opérationnel, ce processus est inoffensif puisque ne pouvant interagir avec d'autres ambients, il se comporte comme* $\mathbf{0}$ [6].

**processus "malformés"** La syntaxe du calcul des ambients autorise des processus qu'on pourrait qualifier de malformés au regard des intuitions que nous avons données ; ainsi, le processus $\texttt{in}\, a.\texttt{out}\, b[\mathbf{0}]$ est un ambient "nommé" par une séquence de capacités. Un tel processus peut être obtenu après réduction d'une communication, par exemple de $\langle \texttt{in}\ a.\texttt{out}\ b\rangle \mid (n).n[\mathbf{0}]$. Il serait possible d'éviter que ce type de processus malformés apparaissent au cours de réductions grâce à un système de typage comme celui décrit dans [CG99]. Cependant, nous ne supposerons ici rien de tel puisque ces processus ne gêneront nullement nos développements.

**structure arborescente des processus** Dans notre exemple introductif, nous avons représenté les processus comme un ensemble de boîtes possédant des capacités (de mouvement). Une autre représentation graphique associée aux processus est possible sous la forme d'arbres. Ces arbres sont d'arité non bornée (*ie* le nombre de fils d'un nœud peut être arbitraire) et non-ordonnés (*ie* il n'existe pas d'ordre *a priori* entre les fils d'un même nœud). Ces arbres vont simplement traduire la structure hiérarchique des ambients, les arêtes de l'arbre étant étiquetées par des noms de $\mathcal{N}$.

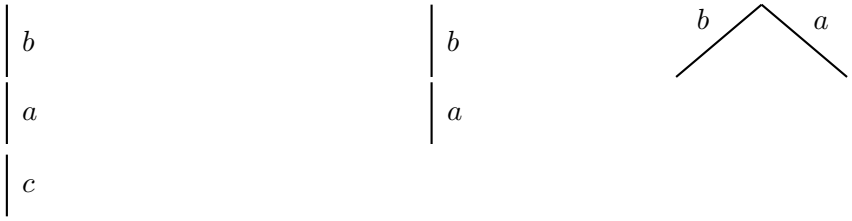
En reprenant le processus initial de l'exemple 2 (ci-dessous à gauche) qui s'écrit formellement $b[c[\texttt{in}\ a.\mathbf{0} \mid \texttt{out}\ b.\mathbf{0}] \mid a[\texttt{open}\ c.\mathbf{0}]]$, il se verra associer l'arbre suivant (ci-dessous à droite) :




L'exécution des capacités du processus modifieront la structure de l'arbre soit en réorganisant les arêtes (capacités `in` et `out`), soit en détruisant des arêtes (capacités `open`) ; en considérant l'exécution de la suite de capacités $\texttt{in}\ a$, $\texttt{open}\ c$, $\texttt{out}\ b$, on obtiendra la succession d'arbres

[6] Ce processus est un cas particulier de ce que Cardelli et Gordon appellent "un pare-feu parfait" (*perfect firewall*) [GC02], pour lequel le terme "parfait confinement" serait meilleur : un processus $(\nu n)n[P]$ tel que le nom $n$ n'est pas libre dans $P$ offre un parfait confinement dans le sens où aucun ambient ne peut interagir avec $n$ pour y entrer, en sortir ou pour l'ouvrir. De plus, dans notre cas particulier où $P$ vaut $\mathbf{0}$, le contenu de l'ambient ne peut effectuer aucune réduction.

$b[a[c[\texttt{out}\ b.\mathbf{0}] \mid \texttt{open}\ c.\mathbf{0}]]$ $b[a[\texttt{out}\ b.\mathbf{0}]]$ $b[\mathbf{0}] \mid a[\mathbf{0}]$




**réplication vs récursion** En lieu et place de l'opérateur de réplication, le calcul des ambients peut être muni d'un opérateur de récursion comme cela a été fait dans [14, San01]. On suppose un ensemble dénombrable de variables de récursion de processus, variables que nous noterons $X$, $Y$, $Z$, .... La syntaxe du calcul est alors augmentée de la manière suivante : si $P$ est un processus, alors il en va de même pour $(\mathsf{fix}\, X.P)$. L'opérateur fix est un lieur pour les variables de récursion.

La sémantique opérationnelle du calcul des ambients est adaptée simplement en ajoutant à la relation de congruence structurelle la règle

$$(\mathsf{fix}\, X.P) \equiv P\{X \leftarrow (\mathsf{fix}\, X.P)\} \qquad \text{(Struct Fix Unfold)}$$

La notation $P\{X \leftarrow (\mathsf{fix}\, X.P)\}$ désigne le processus $P$ dans lequel toutes les occurrences libres de la variable $X$ ont été remplacées syntaxiquement par $(\mathsf{fix}\, X.P)$.

Nous verrons plus tard que d'autres conditions peuvent être ajoutées à la syntaxe des processus avec récursion ainsi que d'autres axiomes à la relation de congruence structurelle concernant cette construction de récursion.

**fragments du calcul** Nous allons souvent considérer tout au long de ce mémoire divers fragments du calcul des ambients. Ces fragments sont obtenus à la fois par restriction de la syntaxe des processus et restriction en conséquence des relations de réduction et de congruence.

Lorsque l'opérateur de restriction de noms $(\nu n)$ sera omis, nous parlerons du fragment *public* du calcul des ambients et sera noté $MA^{-\nu}$. L'absence des primitives de communication ($(n)$ et $\langle M \rangle$) définira le fragment *pur* qui sera noté $pMA$. Dans ce cas, on supposera que la construction d'ambients n'utilise que des noms, *e.g.* $n[P]$. Nous considérerons fréquemment la combinaison de ces deux fragments, à savoir $pMA^{-\nu}$, le fragment pur et public.

Nous considérerons également le calcul sans réplication, noté $MA_{-!}$ et le fragment dit *statique*, noté $SA$, défini par la syntaxe suivante :

$$P ::= \mathbf{0} \mid n[P] \mid P \mid P$$

## 1.3 Quelques variantes du calcul des ambients

De nombreuses variantes du calcul des ambients ont été proposées depuis son introduction. Nous en mentionnons quelques unes ici.

**Safe Ambients** Il convient de remarquer, que dans le calcul des ambients, l'interaction des deux ambients en ce qui concerne les capacités ne se fait qu'à l'initiative uniquement de l'un des deux ; l'autre se trouve purement passif et ne peut de son côté réagir en fonction de l'interaction. Ainsi, le processus $n[\texttt{in}\ m.\texttt{out}\ m.\mathbf{0}] \mid m[P]$ peut se réduire en $n[\mathbf{0}] \mid m[P]$ sans que $m$ n'ait été "au courant" de l'entrée puis de la sortie de $n$. Levi et Sangiorgi ont également montré dans [LS00] que cette dissymétrie dans l'interaction entre deux ambients entraînait des interférences dans le calcul des ambients du fait du chevauchement de réductions potentielles de différentes natures. Afin d'éliminer ses interférences, qualifiées de "graves", et de rendre les interactions plus symétriques, Levi et et Sangiorgi ont proposé d'introduire dans le calcul des co-capabilités de la forme $\overline{\texttt{in}}\ n$, $\overline{\texttt{out}}\ n$ et $\overline{\texttt{open}}\ n$, dont l'idée est similaire aux co-actions de CCS. Les règles de réduction pour les capacités `in`, `out`, `open` deviennent

$$n[\texttt{in}\ m.P \mid Q] \mid m[\overline{\texttt{in}}\ n.R \mid S] \rightarrow m[n[P \mid Q] \mid R \mid S] \qquad \text{(Safe Red In)}$$
$$m[n[\texttt{out}\ m.P \mid Q] \mid \overline{\texttt{out}}\ m.R \mid S] \rightarrow n[P \mid Q] \mid m[R \mid S] \qquad \text{(Safe Red Out)}$$
$$\texttt{open}\ n.P \mid n[\overline{\texttt{open}}\ n.Q \mid R] \rightarrow P \mid Q \mid R \qquad \text{Safe Red Open}$$

**Boxed Ambients** Comme nous l'avons montré dans notre exemple, les communications "longues distances" comme l'applet envoyant une requête vers le pare-feu, combinent communications locales, mouvement (`in`,`out`) et dissolution (`open`).

Bugliesi, Castagna et Crafa expriment dans [BCC01] des critiques sur l'utilisation de la capacité `open`, notamment sur le fait qu'elle puisse entraîner des failles de sécurité, puisque le contenu dangereux d'un ambient ouvert peut se trouver en présence de code critique. Ils proposent un calcul alternatif appelé "Boxed Ambient" ; ce calcul élimine la capacité `open` du calcul des ambients et la remplace par un mécanisme plus sophistiqué de communications. Outre le mécanisme de communications originel, les Boxed Ambients offrent de plus la possibilité de communications père/fils. Ces communications sont modélisées dans la relation de réduction par les règles suivantes :

$$(x)^n.P \mid n[\langle M\rangle Q \mid R] \rightarrow P\{x \leftarrow M\} \mid n[Q \mid R] \qquad \text{(Red Input n)}$$
$$\langle M\rangle P \mid n[(x)^{\uparrow}.Q \mid R] \rightarrow P \mid n[Q\{x \leftarrow M\} \mid R] \qquad \text{(Red Input } \uparrow\text{)}$$
$$\langle M\rangle^n P \mid n[(x).Q \mid R] \rightarrow P \mid n[Q\{x \leftarrow M\} \mid R] \qquad \text{(Red Output n)}$$
$$(x).P \mid n[\langle M\rangle^{\uparrow} Q \mid R]P\{x \leftarrow M\} \mid n[Q \mid R] \qquad \text{(Red Output } \uparrow\text{)}$$

La première règle décrit un père qui reçoit un message émis par un fils de nom $n$, la seconde un fils recevant un message de son père. La troisième règle décrit l'émission du père pour un de ses fils de nom $n$ et la dernière l'émission d'un fils vers son père.

**Push and Pull Ambients** Alors que Cardelli et Gordon avaient privilégié dans le calcul des ambients une mobilité objective [7], Phillips et Vigliotti défendent un calcul avec une mobilité de type objective et proposent le "Push and Pull Ambient Calculus" [PV02]. Ce calcul comme son nom l'indique permet à un ambient de tirer (*pull*) en son intérieur un ambient voisin, ou au contraire de pousser (*push*) à l'extérieur un ambient qu'il contient. Ces deux actions associées à l'ouverture d'un ambient constituent les capacités du calcul et permettent les réductions suivantes :

$$m[\texttt{pull}_m\ n.P \mid Q] \mid n[R] \rightarrow m[P \mid Q \mid n[R]] \qquad \text{(Red Pull)}$$
$$m[\texttt{push}_m\ n.P \mid Q \mid n[R]] \rightarrow m[P \mid Q] \mid n[R] \qquad \text{(Red Push)}$$

[7] [CG98] contient une argumentaire en faveur de ce type de mobilité.

Pour terminer, citons également deux travaux proches des “Safe Ambients” puisque visant également à mieux contrôler les interactions entre les ambients : les ambients robustes [GYY00] utilisent un mécanisme co-actions similaires aux Safe Ambients à l’exception du fait que celui-ci renforce la notion de sécurité, puisque par exemple un ambient $n$ qui veut entrer dans un ambient $m$, doit bien sûr exercer la capacité $\mathtt{in}\, m$ mais $m$ l’autorise nominalement en exécutant la co-capacité $\overline{\mathtt{in}}\, n$. Finalement, les interactions entre ambients sont encore plus précisées dans le “Controlled Mobile Ambients” [TZH02], puisque la réduction implique une interaction tripartite entre l’ambient qui se déplace, l’ambient d’où il part et l’ambient où il arrive.

De nombreux résultats décrits dans ce mémoire restent valides pour les variantes du calcul des ambients que nous avons présentées, notamment les méthodes algorithmiques et les résultats de complexité du chapitre suivant. Cependant en ce qui concerne les résultats de décision, ceux-ci sont en général très sensibles aux calculs utilisés. Seuls les “Safe Ambients” pour lesquels il existe une transformation simple du calcul des ambients originel préservent assurément ces résultats.

# Chapitre 2

# Vérification pour le calcul des ambients sans récursion

Ce chapitre présente les travaux suivants [13, 12, 3]. Ces travaux ont été réalisés en collaboration avec W. Charatonik, S. Dal Zilio, A.D. Gordon et S. Mukhopadhyay. La publication [3] se trouve en annexe C.

## 2.1 Vérification dans les algèbres de processus

### 2.1.1 Quelques généralités sur la vérification de programmes

La vérification de programmes ou de systèmes est un domaine très vaste qui recouvre bien des aspects et vise à *vérifier* formellement la correction des programmes. Le terme de "correction" est lui-même très vague et cette notion ne saurait être absolue. On s'intéresse généralement à la correction d'un programme vis-à-vis d'une spécification : le programme satisfait-il (vérifie-t-il, réalise-t-il, implémente-t-il) la spécification ? Cependant, deux questions demeurent : qu'est-ce qu'une spécification ? quand un programme vérifie-t-il une spécification ?

De manière très abstraite, une spécification est définie par une syntaxe et une sémantique et la notion de vérification est définie comme une relation entre programmes et spécifications. On note $P \models S$ si le programme $P$ vérifie la spécification $S$. Il existe de très nombreux formalismes de spécification. Nous en donnons quelques-uns ici :

- les formalismes logiques comme les logiques temporelles LTL (*linear temporal logic*) [Pnu77] ou CTL (*computational tree logic*) [EH82] ou les logiques dynamiques comme PDL (*propositional dynamic logic*) [FL77] ou le $\mu$-calcul [Koz83]. Dans ce cas, un programme vérifie une spécification s'il est modèle de la formule logique définissant cette spécification.
- des formalismes plus opérationnels comme les automates d'entrées-sorties (*I/O automata*) ou les diagrammes d'états ou d'interaction d'UML. Vérifier la spécification peut être alors le fait d'exhiber les mêmes traces de comportements ou d'interactions, de simuler ou de bissimuler la spécification.
- la spécification peut être elle-même un programme. Le programme vérifie alors la spécification si par exemple, les deux programmes sont équivalents ou s'il y a absence de régression, c'est-à-dire si le programme à vérifier réalise au moins tout ce que peut faire le programme de spécification.

Dans le cas des algèbres de processus, les travaux sur la vérification regroupent majoritairement la preuve d'équivalence entre programmes et la vérification pour des spécifications en logique temporelle ou dynamique.

Pour CCS et le $\pi$-calcul, de nombreux travaux ont eu pour but la vérification d'équivalences comportementales de programmes, se restreignant parfois à un fragment du calcul ; citons par exemple, pour CCS [CPS93] et pour le $\pi$-calcul [Dam97]. Concernant le $\pi$-calcul, ont été également menées des recherches sur la vérification semi-automatique utilisant des assistants de preuves comme COQ [Hir99] ou Isabelle/HOL [RE99].

Nous avons déjà évoqué le fait que la sémantique d'un programme CCS pouvait être exprimée comme un système de transitions labellées, offrant ainsi un support pour l'interprétation de logiques modales. Hennessy et Milner proposent dans [HM80] une logique modale $HM$ qui étend la logique propositionnelle avec l'unique modalité $\langle\alpha\rangle\phi$ qui est vrai pour un processus $P$ si $P \xrightarrow{\alpha} P'$ et $P'$ vérifie $\phi$. La logique $HM$ bien que simpliste est puissante puisqu'elle est suffisante pour distinguer les processus qui ne possèdent pas le même comportement [8] : si deux processus $P, Q$ ne sont pas équivalents, il existe une formule de $HM$ vraie pour $P$ et fausse pour $Q$. Cette relation entre logique modale et sémantique comportementale a ensuite été étendue au $\pi$-calcul [MPW93]. Si la logique $HM$ permet de donner une caractérisation logique de l'équivalence comportementale de programmes, elle est cependant trop pauvre pour être utilisée comme langage de spécification. Des propositions ont ainsi été faites pour étendre cette logique par des opérateurs de point-fixes [Lar88, Sti89] produisant un $\mu$-calcul permettant la vérification de processus de CCS et du $\pi$-calcul. Des algorithmes et des outils ont été développés pour vérifier par model-checking des processus de CCS [SW91] et du $\pi$-calcul [Dam96], ces approches se révélant complètes pour certains fragments. Nous reviendrons sur ces travaux au chapitre 3.

Dans les systèmes d'états finis ou les algèbres de processus, le système est souvent décrit comme une composition de sous-systèmes. Alors que l'approche immédiate serait d'effectuer le produit de tous ces sous-systèmes (entraînant une explosion du nombre d'états à considérer) puis de considérer ce système au travers de logiques classiques. Une autre solution est de pouvoir raisonner directement de façon compositionnelle. On essaie par exemple de déduire une propriété du système à partir des propriétés de ses composants. Ceci permet notamment de décomposer une preuve de vérification en des sous-preuves selon la structure du système mais également d'abstraire un des sous-systèmes et de le remplacer par sa spécification, *ie* l'ensemble requis des formules qu'il doit vérifier. Concernant cette approche compositionnelle, nous pouvons citer notamment [ASW94, AD96, DG99].

### 2.1.2 Vérification par model-checking

Dans le cadre d'une spécification exprimée comme une formule logique, la relation de satisfaction liant programmes et formules est généralement donnée sous la forme d'un système formel. Prouver que le programme satisfait à la spécification revient donc à trouver une preuve correcte de ce fait au regard du système formel.

**model-checking vs démonstration (semi-)automatique** Pour montrer que le programme vérifie effectivement la spécification, deux techniques sont souvent opposées dans la littérature : le model-checking et la démonstration de propriétés (*theorem proving*).

Le model-cheching sous-entend généralement une méthode exploratoire dans l'ensemble des états d'un système et totalement automatique (méthode "presse-bouton") ; ceci implique donc l'existence d'un algorithme qui, donnés un programme et une spécification, retourne vrai si le programme est conforme à la spécification.

Les principaux succès du model-cheking sont du domaine des systèmes d'états finis. Cependant, depuis quelques années, les méthodes de model-checking ont été étendues à des systèmes dont le nombre

[8] Formellement, la logique $HM$ distingue les processus qui ne sont pas (fortement) bissimilaires.

d'états peut être infini. Ainsi, on peut prouver par model-checking des propriétés temporelles de systèmes à pile (*pushdown systems*) ou de diverses algèbres de processus comme BPP ou PA.

La démonstration de propriétés sous-entend quant à elle un mécanisme lié à la notion de preuves, notion connue pour être plus ou moins automatisable selon les types de systèmes et de propriétés auxquels on s'intéresse. Au contraire du model-checking, les systèmes infinis se prêtent bien à ce type d'approche puisque le système y est représenté de manière symbolique.

**model-checking : local vs global** On distingue deux types d'algorithmes pour résoudre le problème de model-checking :

- le model-checking global détermine la valeur de vérité de la formule considérée pour tous les états du modèle. Il se base donc sur la construction de l'ensemble des états qui vérifient la formule. L'ensemble peut par exemple être décrit de manière extensionnelle dans le cas des systèmes d'états finis ; cependant, pour les systèmes ayant un nombre d'états très grand voire infini, on construit uniquement une représentation finie de cet ensemble et on parle de *model-checking symbolique*. Pour les systèmes finis ou infinis, on utilise souvent des automates pour décrire les états dans lesquels la formule est vraie.
- le model-checking local va tenter de prouver ou de réfuter l'assertion concernant le programme et la formule donnés. La construction de la preuve amènera à considérer d'autres (sous-)problèmes de model-checking, décomposant la formule selon ses sous-formules. Cette technique permet de plus d'éviter de considérer ou de calculer explicitement l'ensemble des états du système à vérifier dans le cas où cet ensemble n'est pas donné de manière extensionnelle. On parle alors de *model-checking à la volée* (*on-the-fly model-checking*). La méthode de preuve de type tableaux est une technique fréquemment employée pour résoudre les problèmes de model-checking locaux pour les systèmes finis ou infinis.

## 2.2 Vérification par model-checking du calcul des ambients

### 2.2.1 La logique des ambients

Cardelli et Gordon introduisent dans [CG00a] une logique (dite "logique des ambients") pour exprimer des propriétés des processus du calcul des ambients. Cette logique est modale puisqu'elle dispose d'un opérateur temporel semblable à la modalité EF de la logique CTL. Cependant la partie la plus originale de la logique réside dans un ensemble d'opérateurs *spatiaux* permettant de raisonner sur la structure même des processus et offrant ainsi un mécanisme pour exprimer des propriétés logiques de manière compositionnelle. Cette logique sera ensuite enrichie dans [CG01b] par un ensemble d'opérateurs permettant d'exprimer des propriétés sur les noms restreints d'un processus.

Outre l'ensemble dénombrable des noms $\mathcal{N}$, nous considérons $\mathcal{V}$ un ensemble dénombrable de variables de noms ($\mathcal{N}, \mathcal{V}$ étant disjoints). Nous désignons ces variables par $x, y, z$ et utilisons $\eta$ pour dénoter un objet qui est soit un nom $n$, soit une variable $x$. La syntaxe de la logique des ambients $\mathcal{L}$ est présentée à la figure 2.1. Cette logique est celle de [CG00a] à l'exception des opérateurs de révélation et de masquage qui ont été ajoutés dans [CG01b].

La quantification existentielle est le seul lieur (de variables) de la syntaxe. Nous définissons de manière usuelle les notions de variables libres et liées. Notez que contrairement aux processus tous les noms apparaissant dans une formule sont libres. Une formule est close si elle ne comporte pas de variables libres.

Nous définissons la relation de *sous-localité* $\downarrow$ pour tout couple de processus $P, Q$ comme $P \downarrow Q$ ssi il existe un processus $P'$ et un nom $n$ tels que $P \equiv n[Q] \mid P'$. La relation $\downarrow_*$ désignera la clôture

| $\phi, \psi ::=$ | formule |
|---|---|
| $\mathbf{T}$ | vrai |
| $\neg\phi$ | négation |
| $\phi \vee \psi$ | disjonction |
| $\mathbf{0}$ | vide |
| $\phi \mid \psi$ | composition |
| $\phi \triangleright \psi$ | adjoint de composition |
| $\eta[\phi]$ | localité |
| $\phi @ \eta$ | placement |
| $\Diamond\phi$ | modalité “Finalement” |
| $\diamond\phi$ | modalité “Quelque part” |
| $\exists x.\phi$ | quantification existentielle |
| $\eta \circledR \phi$ | révélation |
| $\phi \oslash \eta$ | masquage |

FIG. 2.1 – La logique des ambients

réflexive et transitive de $\downarrow$. Nous notons $\phi\{x \leftarrow m\}$ la formule obtenue à partir de $\phi$ en y remplaçant toutes les occurrences libres de la variable $x$ par le nom $m$.

Nous définissons la sémantique de la logique $\mathcal{L}$ par une relation de satisfaction $\models$ entre processus et formules closes. Cette sémantique est donnée à la figure 2.2.

Décrivons informellement les caractéristiques de la logique des ambients : la logique $\mathcal{L}$ inclus les connecteurs booléens $\mathbf{T}$, $\neg$, et $\vee$ munis de leur interprétation usuelle et dispose de la quantification existentielle sur l'ensemble des noms. La logique $\mathcal{L}$ possède également une composante modale temporelle sous la forme de la modalité “finalement” $\Diamond$, similaire à la modalité EF de la logique CTL. Cette modalité permet d'exprimer des propriétés sur l'évolution (temporelle) d'un processus : un processus satisfait $\Diamond\phi$ s'il possède un “descendant temporel” (*ie* un processus dans lequel il se réduit) qui satisfait $\phi$.

L'originalité de la logique des ambients réside dans un ensemble d'opérateurs dits *spatiaux* permettant de raisonner sur la structure même des processus. Ces opérateurs sont notamment vide $\mathbf{0}$, localité $n[\,]$ et de composition $\mid$. L'opérateur de composition permet de raisonner de manière compositionnelle, puisque si $P \models \phi$ et $Q \models \psi$ alors $P \mid Q \models \phi \mid \psi$. Un tel opérateur est notamment évoqué dans [Win85, Dam90] mais y est utilisé de manière restreinte. L'opérateur de composition utilisé ici est en revanche identique à celui proposé par Caires et Monteiro dans [CM98].

Aux connecteurs spatiaux de localité et de composition sont associés des opérateurs duaux (ou adjoints) appelés respectivement placement et adjoint de composition ; cet adjoint de composition est similaire à l'implication de [Dam88] et permet le raisonnement compositionnel suivant :

$$\frac{P \models \phi \triangleright \psi \quad Q \models \phi}{P \mid Q \models \psi}$$

$P$ garantit $\psi$ sous la condition d'être placé dans un contexte vérifiant $\phi$. Cette règle montre que l'opérateur d'adjoint de composition est un opérateur d'implication. Similairement, la composition joue le rôle de conjonction ; ainsi, $P \models \phi_1 \triangleright (\phi_2 \triangleright \phi_3)$ ssi $P \models (\phi_1 \mid \phi_2) \triangleright \phi_3$.

Comme l'implication de [Dam88], l'opérateur d'adjoint de composition est voisin de l'implication linéaire [Gir87] et de l'implication de la logique de relevance [Urq72].

Il convient de remarquer que l'opérateur d'adjoint de composition permet une forme "concurrente" pour le raisonnement pré/post-condition de la logique de Hoare pour les programmes séquentiels. L'élément principal d'un système de preuves pour une logique de Hoare est un triplet $\{\phi\}P\{\psi\}$ qui énonce que l'exécution du programme $P$ dans un environnement vérifiant $\phi$ nous amène dans un environnement où $\psi$ est vrai. Autrement dit, pour tout programme $Q$ dont l'exécution, une fois terminée, mène dans un environnement où $\phi$ est vrai, composé séquentiellement avec $P$ produit un programme $Q;P$ dont l'exécution rend vraie la formule $\psi$. Toujours dans le cadre de la logique de Hoare et des programmes séquentiels, un opérateur spatial dit "de séparation" proche de l'opérateur de composition, permet le raisonnement pour des programmes manipulant des pointeurs [IO01, Rey02] ; cet opérateur de séparation permet le raisonnement sur des parties indépendantes du tas où sont allouées les variables dynamiques.

Finalement, la logique des ambients contient également une modalité spatiale "Quelque part" $\mathbin{\diamond\!\!\diamond}$ qui permet d'exprimer des propriétés sur la structure profonde d'un processus.

Absent de la logique des ambients présentée dans [CG00a] puisque celle-ci était dédiée à un calcul public, la logique que nous présentons dispose de deux opérateurs permettant de raisonner sur les noms restreints. Il convient de noter qu'un processus $P$ ne satisfait la formule de révélation $n\circledR\phi$ que si, après une éventuelle $\alpha$-conversion, la restriction $(\nu n)$ pour le nom $n$ peut être exhibée par $P$. Ainsi, $P$ ne peut satisfaire une formule $n\circledR\phi$ si le nom $n$ est libre dans $P$. L'opérateur de masquage $\phi\circledcirc n$ est dual de celui de révélation. Ces opérateurs permettant d'exprimer des propriétés sur des noms liés soumis à l'$\alpha$-conversion sont proches de l'opérateur de "fraicheur"de Gabbay et Pitts, noté $\mathcal{N}x.\phi$ [GP99]. Cet opérateur peut être ajouté à la logique avec la sémantique suivante : $P \models \mathcal{N}x.\phi$ ssi il existe un nom $n$ n'apparaissant libre ni dans $P$, ni dans $\phi$ et tel que $P \models \phi\{x \leftarrow n\}$.

| | | |
|---|---|---|
| $P \models \mathbf{T}$ | | |
| $P \models \neg\phi$ | ssi | non$(P \models \phi)$ |
| $P \models \phi \vee \psi$ | ssi | $P \models \phi$ ou $P \models \psi$ |
| $P \models \mathbf{0}$ | ssi | $P \equiv \mathbf{0}$ |
| $P \models \phi \mid \psi$ | ssi | il existe des processus $P', P''$ tels que $P \equiv P' \mid P''$, $P' \models \phi$ et $P'' \models \psi$ |
| $P \models \phi \triangleright \psi$ | ssi | pour tout processus $P'$ tel que $P' \models \phi$, $P \mid P' \models \psi$ |
| $P \models n[\phi]$ | ssi | il existe un processus $P'$ tel que $P \equiv n[P']$ et $P' \models \phi$ |
| $P \models \phi@n$ | ssi | $n[P] \models \phi$ |
| $P \models n\circledR\phi$ | ssi | il existe un processus $P'$ tel que $P \equiv (\nu n)P'$ et $P' \models \phi$ |
| $P \models \phi\circledcirc n$ | ssi | $(\nu n)P \models \phi$ |
| $P \models \Diamond\phi$ | ssi | il existe un processus $P'$ tel que $P \rightarrow^* P'$ et $P' \models \phi$ |
| $P \models \mathbin{\diamond\!\!\diamond}\phi$ | ssi | il existe un processus $P'$ tel que $P \downarrow_* P'$ et $P' \models \phi$ |
| $P \models \exists x.\phi$ | ssi | il existe un nom $m$ tel que $P \models \phi\{x \leftarrow m\}$ |

FIG. 2.2 – La relation de satisfaction $P \models \phi$ (pour un processus $P$ et une formule close $\phi$)

Nous utiliserons $\Box\mathcal{A}$ ("Toujours"), $\blacksquare\mathcal{A}$ ("Partout") comme des abbréviations pour respectivement $\neg(\Diamond\neg\mathcal{A})$ et $\neg(\mathbin{\diamond\!\!\diamond}\neg\mathcal{A})$

Nous utiliserons parfois le terme de "fragment spatial de la logique". Ce fragment est obtenu à partir de la logique des ambients en ôtant la modalité temporelle $\Diamond$, les opérateurs de révélation et de masquage et la quantification existentielle.

Notons que, dans la logique des ambients, "espace et temps" ne sont pas séparés comme dans une logique temporelle où les propositions atomiques (vraies ou fausses selon les états) seraient remplacées

par une logique exprimant des propriétés sur la structure (spatiale) des états. La logique des ambients intègre au même plan les deux types de raisonnement : il est possible de parler de l'évolution (composante temporelle) d'un fragment (composante spatiale) d'un processus obtenu après exécution d'un certain nombre de pas (composante temporelle).

Contrairement à une logique modale temporelle comme HM qui ne décrit que des propriétés extensionnelles des processus (*ie* liées à son fonctionnement), les opérateurs spatiaux de la logique des ambients permettent de considérer des propriétés intentionnnelles (*ie* liées à l'anatomie même des processus). Cependant, il est très simple de montrer que la logique ne peut distinguer deux processus qui sont structurellement congrus.

**Proposition 1 ([CGar])** *Pour tout couple de processus $P, Q$, si $P \equiv Q$ alors pour toute formule $\phi$ de la logique $\mathcal{L}$, $P \models \phi$ ssi $Q \models \phi$.*

Nous considérons les fragments suivants de la logique : $\mathcal{L}^{-\triangleright}$ est la logique privée de l'opérateur d'adjoint de composition, $\mathcal{L}^{-\exists}$ sera la logique sans quantification, et $\mathcal{L}^{-\nu}$ la logique sans les opérateurs de révélation et de masquage. Nous composerons ces restrictions ; ainsi, $\mathcal{L}^{-\triangleright,\exists,\nu}$ sera la logique sans adjoint de composition, ni quantification, ni opérateur de révélation ou de masquage.

La définition de l'équivalence entre deux processus des ambients passe parfois par la définition d'un prédicat d'observation des processus [GC02]. On dit qu'un processus $P$

- exhibe le nom $n$ s'il existe des noms $m_1, \ldots, m_k$ (avec pour tout $i$, $m_i \neq n$) et des processus $P'$ et $Q$ tel que $P \equiv (\nu m_1) \ldots (\nu m_k) n[Q] \mid P$. On dit que $n$ est une barbe (forte) de $P$.
- converge vers le nom $n$ si $P \rightarrow^* Q$ et $Q$ exhibe le nom $n$. On dit que $n$ est une barbe faible de $P$.

Il peut donc être important de décider si un processus possède une barbe (forte ou faible). Dans le cas particulier d'un calcul public des ambients (sans restriction de noms), on peut exprimer le fait que $n$ est une barbe faible d'un processus $P$ si et seulement si $P \models n[\mathbf{T}] \mid \mathbf{T}$.

Mentionnons pour terminer cette section sur la présentation de la logique des ambients qu'une logique spatiale pour le $\pi$-calcul a été proposée dans [CC02, CC03].

### 2.2.2 Le problème de model-checking

#### Un fragment décidable du problème

Cardelli et Gordon considèrent dans [CG00a] le problème de model-checking de $MA^{-\nu}$, le fragment public du calcul des ambients (sans restriction) et de $\mathcal{L}^{-\triangleright,\nu}$, la logique des ambients sans adjoint de composition, ni opérateur de révélation ou masquage. Ils montrent que

**Theorem 1 ([CG00a])** *Le problème de model-checking du fragment public du calcul des ambients sans réplication $MA^{-\nu}_{-!}$ et de la logique $\mathcal{L}^{-\triangleright,\nu}$ (sans adjoint de composition, ni opérateurs de révélation ou masquage) est décidable.*

Considérons la famille de processus $(P_i)_{0\geq i}$ définie récursivement comme $P_0 = (y).(p[y.\mathbf{0}] \mid q[\mathbf{0}])$ et pour tout $i > 0$, $P_i = (x_i).\langle x_i.x_i\rangle \mid P_{i-1}$. Il est simple de voir que $P_n \mid \langle \texttt{in}\ q.\texttt{out}\ q\rangle$ se réduit en $k+1$ étapes en $p[(\texttt{in}\ q.\texttt{out}\ q)^{2^k}.\mathbf{0}] \mid q[\mathbf{0}]$ où $(\texttt{in}\ q.\texttt{out}\ q)^{2^k}$ est une séquence constituée de $2^k$ copies des capacités $\texttt{in}\ q.\texttt{out}\ q$. Notons finalement que $p[(\texttt{in}\ q.\texttt{out}\ q)^{2^k}.\mathbf{0}] \mid q[\mathbf{0}]$ se réduit en $2^k$ étapes en $p[\mathbf{0}] \mid q[\mathbf{0}]$. Ainsi, un processus $P$ peut

1. se réduire en un processus dont la taille est exponentielle dans la taille de $P$.
2. avoir un nombre exponentiel dans sa taille de descendants via la relation de réduction.

Pour ces raisons, il est simple de voir que l'algorithme proposé dans [CG00a] répondra au model-checking en utilisant un espace exponentiel dans la taille du problème.

Cette explosion dans la taille des processus considérés, mentionnée au point 1, est bien sûr due aux substitutions liées aux communications. Nous proposons une représentation alternative des processus dans lesquels les substitutions sont gardées explicites. Nous redéfinissons sur cette représentation les notions de réduction et de sous-localités en proposant pour ces notions des algorithmes polynomiaux. Basé sur cette nouvelle représentation, pour répondre au problème du nombre de processus à considérer (point 2), nous concevons un algorithme de model-checking de type local et à la volée, qui nous permet de conclure que

**Proposition 2 ([13, 3])** *Le problème de model-checking du calcul des ambients public sans réplication et de la logique des ambients sans adjoint de composition ni opérateurs de révélation ou masquage est dans PSPACE.*

Nous montrons de plus que cette borne est optimale, puisque le problème de model-checking est PSPACE-dur pour divers fragments de la logique et du calcul des ambients.

Localité et placement permettent de coder l'égalité entre (variables de) noms ; ainsi, $\eta = \eta'$ est valide ssi $\eta[\mathbf{T}]@\eta'$ est valide. En combinant cette égalité avec les connecteurs booléens et la quantification existentielle sur les noms, il est très simple pour toute formule booléenne quantifiée $\mathcal{B}$ de construire une formule de la logique des ambients $\psi_{\mathcal{B}}$ (de taille polynomiale) telle que $\mathcal{B}$ est valide ssi $\mathbf{0} \models \psi_{\mathcal{B}}$.

Le problème $QBF$ étant un problème PSPACE-complet, on en déduit immédiatement le résultat de PSPACE-dureté pour le fragment considéré de la logique.

**Proposition 3 ([13])** *Le problème de model-checking de $MA^{-\nu}_{-!}$, le fragment public du calcul des ambients sans réplication et de la logique $\mathcal{L}^{-\triangleright,\nu}$ (sans adjoint de composition, ni opérateurs de révélation ou masquage) est PSPACE-dur.*

Ainsi en combinant les propositions 2 et 3, on en déduit que le problème de model-checking du fragment public du calcul des ambients et de la logique $\mathcal{L}^{-\triangleright,\nu}$ (sans adjoint de composition, ni opérateurs de révélation ou masquage) est PSPACE-complet. Cependant, la quantification existentielle n'est pas la seule source de PSPACE-dureté de la logique. Par réduction du problème $QBF$, nous montrons pour $\mathcal{L}^{-\triangleright,\exists,\nu}$, la logique sans quantification, ni adjoint de composition, ni opérateur de masquage ou révélation, que

**Proposition 4 ([13])** *Le problème de model-checking est PSPACE-dur pour la logique $\mathcal{L}^{-\triangleright,\exists,\nu}$ et le calcul des ambients*

- *pur et public sans replication $pMA^{-\nu}_{!}$*
- *sans réplication, public avec communications mais sans les capacités* $\{\texttt{in}, \texttt{out}, \texttt{open}\}$

De plus, dans ce dernier cas du calcul public avec communication mais sans capacité, nous raffinons notre résultat en prouvant en ce qui concerne la complexité de programme que

**Proposition 5 ([13])** *Pour chaque entier naturel $k$, il existe une formule fixée $\phi^k_\exists$ (resp. $\phi^k_\forall$) telle que le problème de model-checking de cette formule pour les processus du calcul public avec communications mais sans capacité est dur pour le $k^{ième}$ niveau existentiel (resp. universel) de la hiérarchie polynomiale.*

Nous résumons nos résultats à la figure 2.3.

| Complexité combinée | Complexité de programme | Complexité d'expression |
|---|---|---|
| PSPACE-complet | $\Sigma_k^P$-dur , $\Pi_k^P$-dur (pour tout $k \in \mathbb{N}$)<br>dur pour tous les niveaux de PH | PSPACE-dur |

FIG. 2.3 – Complexités pour le problème de model-checking du calcul des ambients public sans réplication $MA_{-!}^{-\nu}$ et de la logique des ambients sans adjoint de composition $\mathcal{L}^{-\triangleright}$.

**Au delà du fragment décidable de Cardelli et Gordon**

Nous avons considéré chacune des restrictions imposées sur le calcul et la logique dans [CG00a] et étudions la possibilité d'étendre le model-checking en levant une à une ces restrictions.

**la restriction de noms dans le calcul** L'absence de l'opérateur de restriction dans le calcul et des opérateurs correspondants dans la logique est inutile pour la décidabilité du problème de model-checking, puisque

**Proposition 6 ([12])** *Le problème de model-checking du calcul des ambients sans réplication $MA_{-!}$ et de $\mathcal{L}^{-\triangleright}$, la logique des ambients sans adjoint de composition est PSPACE-complet.*

Notons que ce résultat s'étend aisément à une logique intégrant l'opérateur de "fraîcheur" de Gabbay et Pitts $\mathsf{N}$ ; le traitement est similaire à celui de la quantification existentielle.

**l'adjoint de composition dans la logique** Il convient tout d'abord de remarquer que l'opérateur d'adjoint de composition est particulièrement puissant puisqu'il permet de réduire le problème de validité de la logique des ambients dans celui du model-checking, puisque

**Proposition 7 ([12])** $\mathbf{0} \models \mathbf{T} \triangleright \phi$ *ssi la formule $\phi$ est valide.*

Ainsi, nous sommes amenés à nous poser la question de la satisfiabilité de la logique des ambients. Nous utilisons le fait qu'une structure finie peut être décrite comme un processus statique [9] et que la logique $\mathcal{L}^{-\triangleright}$ peut encoder FO, la logique du premier ordre. En appliquant le théorème de Trakhtenbrot [Tra50], nous montrons que la satisfiabilité est indécidable pour la logique $\mathcal{L}^{-\triangleright}$ interprétée sur le calcul des ambients contenant le fragment statique. Ainsi,

**Proposition 8 ([12])** *Le problème de model-checking pour la logique $\mathcal{L}$ et tout fragment du calcul des ambients contenant le fragment statique est indécidable.*

Le résultat ci-dessus est obtenu en fait pour un fragment relativement simple de la logique contenant les opérateurs booléens et spatiaux ($\mathbf{0}$, $n[\,]$, $\mid$), l'adjoint de composition et la quantification existentielle. Cette dernière joue un rôle capital puisque, si elle devait être omise dans le fragment que nous considérons, alors le problème de model-checking et de satisfiabilité deviendrait décidable pour le fragment statique du calcul des ambients [CCG03, DZLM04]. Ces résultats ont été généralisés dans [CG04b] en étendant le calcul statique par l'opérateur de restriction et la logique par les opérateurs de révélation, masquage et de "fraîcheur". Alors que ce dernier opérateur préserve la décidabilité du problème de model-checking, il y est montré en reprenant la technique que nous avons utilisée pour montrer la proposition 8 qu'il en va différemment pour l'opérateur de révélation.

[9] On rappelle que le fragment statique contient les processus n'utilisant que les constructions suivantes : inactif $\mathbf{0}$, ambient $n[\,]$ et composition $\mid$.

**la réplication dans le calcul** Finalement, concernant cette dernière restriction, nous montrons que le problème de model-checking du calcul public et pur (*ie* sans restriction, ni communication) mais incluant la réplication est également indécidable. Nous reviendrons plus précisément sur ce résultat au chapitre 4 mais il constitue la principale motivation du calcul des ambients à contrôle fini que nous présentons dans le chapitre suivant.

# Chapitre 3

# Vérification pour le calcul des ambients à contrôle fini

Ce chapitre présente des travaux publiés dans [14] et menés en collaboration avec W. Charatonik et A.D. Gordon. L'annexe D contient cette publication.

## 3.1 Le calcul des ambients à contrôle fini

L'absence totale de récursion dans le calcul des ambients implique non seulement que l'ensemble des processus accessibles par réduction depuis un processus donné est fini mais de plus, que toute exécution de ce processus termine. Ainsi, ce fragment ne permet même pas de modéliser les systèmes d'états finis. Inversement, la présence de réplication ou de récursion dans le calcul, comme nous l'avons évoqué dans le chapitre précédent et comme nous le montrerons dans le suivant, rend impossible la mécanisation totale du model-checking.

Dans le cadre de CCS, il existe un fragment bien connu, appelé CCS à états finis (*finite-state* CCS) pour lequel comme son nom l'indique un processus décrit un système dont le nombre d'états est fini. Ce fragment est obtenu de manière purement syntaxique : les équations définissant le processus ne peuvent être récursives que si les membres droits ne contiennent pas l'opérateur de composition. Informellement, un processus à états finis possède un certain nombre de composantes s'exécutant de manière concurrente, mais ces composantes n'ont pas la possibilité de créer dynamiquement d'autres sous-processus (ou *threads*). Pour ce fragment, le problème de model-checking pour le $\mu$-calcul étendant la logique $HM$ par un opérateur de point-fixe [Lar88] est décidable [SW91]. Cette même restriction syntaxique appliquée au $\pi$-calcul produit le $\pi$-calcul à contrôle fini [10]. Comme pour CCS, le problème de model-checking du $\mu$-calcul est décidable pour le $\pi$-calcul à contrôle fini [Dam96].

Amadio et Meyssonnier proposent dans [AM02] un $\pi$-calcul asynchrone à contrôle fini. Contrairement au cas synchrone où le nombre de composantes d'un système est globalement borné, ici seuls les processus actifs, *ie* en attente de réception voient leur nombre borné. Ainsi, le nombre de messages en attente de réception, lui, peut être arbitrairement grand. Amadio et Meyssonnier montrent que dans ce cas le problème d'accessibilité de contrôle (c'est-à-dire la possibilité d'atteindre un processus contenant un certain identificateur en position non-gardée) est indécidable. Ce problème est cependant montré décidable lorsque les hypothèses d'unicité du receveur et d'entrées bornées sont ajoutées.

Nous nous donnons comme objectif de définir un fragment du calcul des ambients qui permettrait

[10] L'opérateur de restriction permettant de générer des noms "frais" différents de tous les autres fait qu'on peut considérer le nombre d'états comme infini.

d'exprimer n'importe quel système d'états finis et pour lequel le model-checking pour $\mathcal{L}^{-\triangleright}$, la logique des ambients sans adjoint de composition serait décidable.

La réplication n'est pas la forme de récursion la plus pratique lorsqu'il s'agit d'en décrire des formes limitées. Nous considérons donc un calcul muni d'un opérateur de récursion explicite noté fix (utilisant des identificateurs $A$, $B$, $C$) et défini au travers de la congruence structurelle par les deux axiomes suivants :

$$(\mathsf{fix}\, A.P) \equiv P\{X \leftarrow (\mathsf{fix}\, A.P)\} \quad \text{(Struct Fix Unfold)}$$
$$(\mathsf{fix}\, A.\mathbf{0}) \equiv \mathbf{0} \quad \text{(Struct Fix Id)}$$

De plus, nous nous restreignons à un calcul où seuls des noms peuvent être communiqués. Nous discuterons à la section 3.1.3 de cette restriction.

Il convient maintenant d'essayer de définir ce que pourrait être le calcul des ambients à contrôle fini.

Dans le cas du $\pi$-calcul synchrone à contrôle fini [Dam96], il est bien évident que la restriction syntaxique interdisant la récursion au travers de la composition fait que tout processus atteint par réduction possède un nombre de composantes (ou plus simplement, un nombre de compositions) bornées. Il en va différemment dans le calcul des ambients du fait des capacités de mobilité. Prenons, par exemple, le processus $(\mathsf{fix}\, A.m[n[\mathtt{out}\ m.A]])$. Il est évident que la réduction de ce processus crée de plus en plus de composantes susceptibles d'interagir avec d'autres composantes. Notez d'un autre côté qu'interdire la récursion au travers de la composition implique, du fait de l'asynchronisme des communications du calcul des ambients que, dans tout processus, un nombre borné de communications pourrait intervenir lors de sa réduction, les rendant de fait inutiles.

Indépendamment de la restriction syntaxique adéquate, on pourrait exiger que, de plus, le nombre de composantes du système soit globalement borné. Malheureusement, dans le cadre du calcul des ambients, cette restriction est toujours insuffisante pour obtenir un système avec un nombre fini d'états ; nous montrons à l'exemple 4 que pour cette restriction du calcul le problème de convergence de nom (et donc, le problème de model-checking de la logique $\mathcal{L}^{-\triangleright}$) est indécidable.

**Example 4** *Nous montrons ici une réduction du problème de correspondance de Post (PCP) dans celui de la convergence de nom pour un fragment du calcul des ambients pur, public avec récursion et tel que qu'il n'y a pas de récursion au travers de la composition et tel que les processus considérés possèdent un nombre borné de composition de processus (non réduits à* $\mathbf{0}$*).*

*On rappelle que le problème de correspondance de Post est donné par une famille indexée sur un ensemble fini $I$ de couples de mots sur l'alphabet $\{a, b\}$, ie $((u_i, v_i))_{i\in I}$ avec pour tout $i$, $u_i, v_i \in (a + b)^*$. Le problème est de décider s'il existe une suite finie d'indices $i_1, i_2, \ldots, i_n$ telle que :*

$$u_{i_1} u_{i_2} \ldots u_{i_n} = v_{i_1} v_{i_2} \ldots v_{i_n}$$

*Ce problème est bien connu pour être indécidable [Pos44].*

*A partir d'une instance de PCP, nous allons montrer comment construire un processus qui convergera vers le nom spécial $succes$* [11].

*Le processus possède deux modes de fonctionnement : le premier consiste à construire deux mots, l'un par concaténation des $u_i$ et l'autre des $v_i$, le choix à chaque étape de la paire $i$ utilisée étant non-déterministe. Le processus entre également de manière non-déterministe dans le second mode ; celui-ci consiste à vérifier si les deux mots construits à la première étape sont égaux et si cela est le cas alors l'ambient $succes$ est exhibé.*

*Le fonctionnement de notre processus sera orchestré par le processus* Lock *défini par*

$$\mathsf{Lock} \stackrel{def}{=} (\mathsf{fix}\, L.lock[\mathtt{open}\ complet.L])$$

[11] L'encodage présenté ici est dans l'esprit de celui de [12], mais il diffère du fait de la nature différente des deux calculs.

*La concaténation d'une nouvelle paire ouvrira le verrou $lock$, procédera, puis libérera un ambient $complet$ marquant la fin de cette itération. La phase de vérification, quant à elle débutera par l'ouverture de $lock$, stoppant ainsi le mécanisme de la première phase.*

*Nous supposons deux noms d'ambients $m_1, m_2$ et, pour les lettres $a, b$, les noms $a_1, a_2, b_1, b_2$. Intuitivement, les noms indicés par 1 (resp. par 2) sont associés aux mots $u_i$ (resp. $v_i$) et à leur concaténation. Pour simplifier l'écriture, pour un mot $w = \ell_1 \dots \ell_k$, nous utiliserons dans les processus $w[P]$ pour $\ell_1[\dots[\ell_k[P]]..]$ et* $\mathtt{cap}\, w$ *pour* $\mathtt{cap}\, \ell_1.\dots.\mathtt{cap}\, \ell_k$ *(avec* $\mathtt{cap} \in \{\mathtt{in}, \mathtt{out}, \mathtt{open}\}$*).*

*Nous définissons une famille de processus $\mathsf{N}_j^w$ pour $j \in \{1,2\}$ et $w \in \{u_i \mid 1 \le i \le n\}$ si $j = 1$ et $w \in \{v_i \mid 1 \le i \le n\}$ sinon par :*

$$\mathsf{N}_j^w \stackrel{def}{=} (\mathsf{fix}\, N.\mathtt{open}\ lconcat_j.m_j[\mathtt{open}\ cn.w[x[\mathtt{out}\ w.\mathtt{in}\ m_j.\mathtt{in}\ w.fin_j[\mathtt{out}\ m_j.\mathtt{out}\ w.\mathtt{out}\ m_j.N]]]])$$

*Ce processus a pour but de concaténer le mot $w$ à la séquence qui a déjà été construite ; il est aidé en cela par le processus $\mathsf{Aux}_j$ défini (pour $1 \le j \le 2$) par*

$$\mathsf{Aux}_j \stackrel{def}{=} (\mathsf{fix}\, A.\mathtt{open}\ laux_j.cn[\mathtt{in}\ m_j.\mathtt{open}\ n.fin'_j[\mathtt{out}\ m_j.A]])$$

*Précisons maintenant comment les concaténations des mots sont représentées : pour les $u_i$ et donc pour $j = 1$, au mot $u_3u_1u_5$ sera associé le processus $m_1[\mathsf{Guide}_1 \mid m_1[u_3[m_1[u_1[m_1[u_5[s_1[\mathbf{0}]]]]]]]]$ où le processus $\mathsf{Guide}_j$ est défini par $(\mathsf{fix}\, G.\mathtt{in}\ m_j.\mathtt{open}\ x.n[\mathtt{out}\ m_j.G])$ (pour $1 \le j \le 2$}).*

*Finalement, le contrôle sera donné aux différents processus de concaténation par le processus $\mathsf{CC}$ défini par*

$$\mathsf{CC} \stackrel{def}{=} (\mathsf{fix}\, C.\mathtt{open}lock.lconcat_1[lconcat_2[laux_1[laux_2[\mathtt{open}fin_1.\mathtt{open}fin_2.\mathtt{open}fin'_1.\mathtt{open}fin'_2.complet[C]]]]])$$

*On définit le processus $\mathsf{CONCAT}$ comme $\mathsf{Aux}_1 \mid \mathsf{Aux}_2 \mid \mathsf{N}_1^{u_1} \mid \dots \mid \mathsf{N}_1^{u_n} \mid \mathsf{N}_2^{v_1} \mid \dots \mid \mathsf{N}_2^{v_n} \mid \mathsf{CC}$*

*Détaillons ce mécanisme de concaténation en considérant les processus $\mathsf{N}_1^w$ et $\mathsf{Aux}_1$ , une fois que ceux-ci ont ouvert respectivement $lconcat_1$ et $laux_1$ ; nous étudions leur comportement par composition avec le processus $m_1[\mathsf{Guide}_1 \mid m[\mathbf{0}]]$.*

$$\begin{array}{c} m_1[\mathtt{open}\ cn.w[x[\mathtt{out}\ w.\mathtt{in}\ m_1.\mathtt{in}\ w.fin_1[\mathtt{out}\ m_1.\mathtt{out}\ w.\mathtt{out}\ m_1.\mathsf{N}_1^w]]]] \mid \\ cn[\ \mathtt{in}\ m_1\ .\mathtt{open}\ n.fin'_1[\mathtt{out}\ m_1.\mathsf{Aux}_1]] \quad \mid \quad m_1[\ \mathtt{in}\ m_1\ .\mathtt{open}\ x.n[\mathtt{out}\ m_1.\mathsf{Guide}_1] \mid m[\mathbf{0}]] \end{array}$$

*se réduit en*

$$\xrightarrow{m_1} \boxed{\begin{array}{c} \mathtt{open}\ cn\ .w[x[\mathtt{out}\ w.\mathtt{in}\ m_1.\mathtt{in}\ w.fin_1[\mathtt{out}\ m_1.\mathtt{out}\ w.\mathtt{out}\ m_1.\mathsf{N}_1^w]]] \mid \\ cn[\mathtt{open}\ n.fin'_1[\mathtt{out}\ m_1.\mathsf{Aux}_1]] \quad \mid \quad m_1[\mathtt{open}\ x.n[\mathtt{out}\ m_1.\mathsf{Guide}_1] \mid m[\mathbf{0}]] \end{array}}$$

*Notez qu'une autre exécution de ces deux capacités $\mathtt{in}\ m_1$ existe mais elle mène vers une situation de blocage. Puis*

$$\xrightarrow{m_1} \boxed{\begin{array}{c} w[x[\ \mathtt{out}\ w.\mathtt{in}\ m_1\ .\mathtt{in}\ w.fin_1[\mathtt{out}\ m_1.\mathtt{out}\ w.\mathtt{out}\ m_1.\mathsf{N}_1^w]]] \mid \\ \mathtt{open}\ n.fin'_1[\mathtt{out}\ m_1.\mathsf{Aux}_1] \quad \mid \quad m_1[\mathtt{open}\ x.n[\mathtt{out}\ m_1.\mathsf{Guide}_1] \mid m[\mathbf{0}]] \end{array}}$$

*On a alors*

$$\xrightarrow{m_1} \boxed{\begin{array}{c} w[] \quad \mid \quad \mathtt{open}\ n.fin'_1[\mathtt{out}\ m_1.\mathsf{Aux}_1] \mid \\ m_1[x[\mathtt{in}\ w.fin_1[\mathtt{out}\ m_1.\mathtt{out}\ w.\mathtt{out}\ m_1.\mathsf{N}_1^w]] \mid \ \mathtt{open}\ x\ .n[\mathtt{out}\ m_1.\mathsf{Guide}_1] \mid m[\mathbf{0}]] \end{array}}$$

*qui se réduit en*

$$\overset{m_1}{\boxed{\begin{array}{c} w[] \quad | \quad \texttt{open}\, n.fin'_1[\texttt{out}\, m_1.\mathsf{Aux}_1] \mid \\ m_1[\texttt{in}\, w.fin_1[\texttt{out}\, m_1.\texttt{out}\, w.\texttt{out}\, m_1.\mathsf{N}_1^w] \mid n[\texttt{out}\, m_1.\mathsf{Guide}_1] \mid m[\mathbf{0}]] \end{array}}}$$

*Notez qu'à cet instant, la capacité* $\texttt{in}\, w$ *de* $m_1$ *pourrait être exécutée ; cependant, ceci entraînerait un blocage. On réduit donc d'abord la capacité* $\texttt{out}\, m_1$ *de l'ambient* $n$*,*

$$\overset{m_1}{\boxed{w[] \mid \texttt{open}\, n.fin'_1[\texttt{out}\, m_1.\mathsf{Aux}_1] \mid n[\mathsf{Guide}_1] \mid m_1[\texttt{in}\, w.fin_1[\texttt{out}\, m_1.\texttt{out}\, w.\texttt{out}\, m_1.\mathsf{N}_1^w] \mid m[\mathbf{0}]]}}$$

*On a alors*

$$\overset{m_1}{\boxed{w[m_1[fin_1[\texttt{out}\, m_1.\texttt{out}\, w.\texttt{out}\, m_1.\mathsf{N}_1^w] \mid m[\mathbf{0}]]] \mid fin'_1[\texttt{out}\, m_1.\mathsf{Aux}_1] \mid \mathsf{Guide}_1}}$$

*qui se réduit finalement en*

$$m_1[\mathsf{Guide}_1 \mid w[m_1[m[\mathbf{0}]]]] \mid fin_1[\mathsf{N}_1^w] \mid fin'_1[\mathsf{Aux}_1]$$

*Considérons maintenant la seconde étape, à savoir la vérification des mots générés. Le processus* VERIF *est défini par*

$$\texttt{open}\, lock.\texttt{open}\, m_1.\texttt{open}\, m_2.\texttt{open}\, a_1.\texttt{open}\, a_2.\mathsf{SV} \mid \texttt{open}\, lock.\texttt{open}\, m_1.\texttt{open}\, m_2.\texttt{open}\, b_1.\texttt{open}\, b_2.\mathsf{SV}$$

*Ce processus ouvre l'ambient* $lock$ *pour éviter toute concaténation ultérieure, ouvre* $m_1$*,* $m_2$ *puis soit les* $a$*, soit les* $b$*; il est évident qu'un mauvais choix peut être réalisé même si les deux mots sont égaux. Cependant, dans ce dernier cas, un bon choix (selon les premières lettres des deux mots) est toujours possible. Notez également que lorsque l'exécution de* SV *débute, on est certain que les mots issus de la première étape sont non vides. Ce processus* SV *est défini par*

$$\mathsf{SV} \overset{def}{=} (\mathsf{fix}\, U.\texttt{open}\, m_1.U) \mid (\mathsf{fix}\, V.\texttt{open}\, m_2.V) \mid (\mathsf{fix}\, C.lverif\,[\texttt{open}\, term.C]) \mid \mathsf{OA} \mid \mathsf{OB} \mid \mathsf{YES}$$

*Les deux premières composantes de* SV *visent à éliminer les ambients* $m_1$ *et* $m_2$*. La composante d'identificateur* $C$ *assure le contrôle de l'effacement des lettres et le bon enchaînement de chaque itération.*

*Les processus* OA *et* OB *ouvrent chacun la même lettre dans les deux mots (là encore, un mauvais choix peut se produire, bloquant la réduction mais si les deux mots sont égaux un bon choix est toujours possible). Ces deux processus sont définis par :*

$$\mathsf{OA} \overset{def}{=} (\mathsf{fix}\, A.\texttt{open}\, lverif\,[\texttt{open}\, a_1.\texttt{open}\, a_2.term[A]])$$
$$\mathsf{OB} \overset{def}{=} (\mathsf{fix}\, B.\texttt{open}\, lverif\,[\texttt{open}\, b_1.\texttt{open}\, b_2.term[B]])$$

*Finalement, le processus* YES *exhibe l'ambient* $succes$ *si les deux mots sont égaux et est défini comme* $\texttt{open}\, s_1.\texttt{open}\, s_2.succes[\mathbf{0}]$.

*Pour terminer, nous définissons le processus* PCP *comme*

$$\mathsf{PCP} \overset{def}{=} \mathsf{Lock} \mid \mathsf{CONCAT} \mid \mathsf{VERIF} \mid m_1[\mathsf{Guide}_1 \mid s_1[\mathbf{0}]] \mid m_2[\mathsf{Guide}_2 \mid s_2[\mathbf{0}]]$$

*Il est alors très simple de montrer que le processus* PCP *converge vers le nom* $succes$ *si et seulement si l'instance associée du problème de correspondance de Post a une solution.*

A l'extrême, on pourrait interdire la récursion à la fois au travers de la composition et de la construction d'ambients ; cependant, cette restriction serait beaucoup trop drastique, puisque la structure spatiale des processus ne pourrait être que de taille bornée. Nous verrons cependant que ces processus seront à contrôle fini dans le sens où nous le définissons.

$$\frac{A \text{ est un identificateur, } (A,\tau) \in \Gamma}{\Gamma \vdash A \,:\, \tau}\ \text{(Identifier)} \qquad \frac{}{\Gamma \vdash \mathbf{0} \,:\, 0}\ \text{(Zero)} \qquad \frac{\Gamma \vdash P \,:\, \tau \,,\ \Gamma \vdash Q \,:\, \theta}{\Gamma \vdash P \mid Q \,:\, \tau + \theta}\ \text{(Par)} \qquad \frac{\Gamma \vdash P \,:\, \tau}{\Gamma \vdash (\nu n)P \,:\, \tau}\ \text{(Res)}$$

$$\frac{}{\Gamma \vdash \langle n \rangle \,:\, 1}\ \text{(Output)} \qquad \frac{\Gamma \vdash P \,:\, \tau}{\Gamma \vdash (n).P \,:\, \max(\tau - 1, 1)}\ \text{(Input)} \qquad \frac{\Gamma \vdash P \,:\, \tau \,,\ \mathtt{cap} \in \{\mathtt{in}, \mathtt{out}\}}{\Gamma \vdash \mathtt{cap}\, n.P \,:\, \max(\tau, 1)}\ \text{(In/Out)}$$

$$\frac{\Gamma \vdash P \,:\, \tau}{\Gamma \vdash n[P] \,:\, \tau + 1}\ \text{(Amb)} \qquad \frac{\Gamma \vdash P \,:\, \tau}{\Gamma \vdash \mathtt{open}\, n.P \,:\, \max(\tau - 1, 1)}\ \text{(Open)} \qquad \frac{\Gamma \cup \{(A,\tau)\} \vdash P \,:\, \theta \,,\quad \theta \leq \tau}{\Gamma \vdash (\mathsf{fix}\, A.P) \,:\, \tau}\ \text{(Fix)}$$

FIG. 3.1 – Typage des processus $\Gamma \vdash P \,:\, \tau$

### 3.1.1 Le système de type FC

Pour définir le calcul des ambients à contrôle fini, nous définissons un système de typage FC présenté à la figure 3.1. $\Gamma$ désigne un environnement de typage défini comme un ensemble de paires $(A, \tau)$ où $A$ est un identificateur et $\tau$ un entier naturel.

Le type d'un processus sera un entier naturel qui bornera supérieurement le nombre d'ambients et d'émissions actifs dans le processus [12]. Un *jugement de type* $\Gamma \vdash P \,:\, \tau$ est valide s'il existe une preuve du jugement utilisant les règles d'inférences du système FC. Un processus $P$ est *bien typé* par un environnement $\Gamma$ s'il existe un entier naturel $\tau$ tel que le jugement $\Gamma \vdash P \,:\, \tau$ est valide. Un processus $P$ est dit *à contrôle fini* s'il existe un environnement $\Gamma$ qui permet de bien typer $P$.

Il existe une notion de type principal pour le système FC.

**Proposition 9 ([14])** *Pour tout processus $P$ et tout environnement $\Gamma$ tel que $P$ est bien typé par $\Gamma$, alors il existe un plus petit $\tau$ tel que $\Gamma \vdash P \,:\, \tau$ est valide.*

Notons $\mathcal{T}^{\mathsf{FC}}(P, \Gamma)$ ce type principal. Nous avons alors la propriété de réduction du sujet suivante, qui stipule qu'au cours de l'exécution le type d'un processus ne peut pas croître :

**Proposition 10 ([14])** *Pour tout processus $P$ et tout environnement $\Gamma$ tel que $P$ est bien typé par $\Gamma$, si $P \to Q$ alors $Q$ est bien typé par $\Gamma$, et $\mathcal{T}^{\mathsf{FC}}(Q, \Gamma) \leq \mathcal{T}^{\mathsf{FC}}(P, \Gamma)$.*

Finalement, la vérification de typage et l'inférence de type sont toutes les deux décidables,

**Proposition 11 ([14])** *Pour tout processus $P$ et tout environnement $\Gamma$,*
- *pour tout entier naturel $\tau$, on peut décider si $\Gamma \vdash P \,:\, \tau$ est valide.*
- *on peut décider si $P$ peut être bien typé par $\Gamma$ et calculer $\mathcal{T}^{\mathsf{FC}}(P, \Gamma)$.*

### 3.1.2 Typage et contrôle fini

Le système de typage FC garantit bien la notion de "contrôle fini" puisque

**Proposition 12 ([14])** *Pour tout processus $P$ typable par FC, il existe un nombre fini de processus deux à deux non congrus accessibles par réduction depuis $P$.*

[12] Pour des raisons techniques, seuls les processus congrus à $\mathbf{0}$ sont de type nul.

Afin de mesurer les possibilités du calcul, nous donnons dans [14] divers exemples de processus à contrôle fini et montrons qu'il est possible de coder dans notre calcul une version sans choix externe du $\pi$-calcul à contrôle fini.

### 3.1.3 Communication de noms vs communication de capacités

Comme nous avons vu, nous nous restreignons aux communications de nom. En effet, la simple possibilité de communiquer une capacité peut entraîner un comportement infini pour un processus. Ainsi, le processus $(\mathsf{fix}\, A.(x).(\langle \mathtt{in}\ x\rangle \mid A)) \mid \langle n\rangle$ serait typable avec notre système (en supposant que la règle pour l'émission de messages reste identique) alors que ce processus admet un nombre infini de descendants, puisque

$$(\mathsf{fix}\, A.(x).(\langle \mathtt{in}\ x\rangle \mid A)) \mid \langle n\rangle \rightarrow (\mathsf{fix}\, A.(x).(\langle \mathtt{in}\ x\rangle \mid A)) \mid \langle \mathtt{in}\ n\rangle \rightarrow (\mathsf{fix}\, A.(x).(\langle \mathtt{in}\ x\rangle \mid A)) \mid \langle \mathtt{in}\ (\mathtt{in}\ n)\rangle \rightarrow \ldots$$

Ce problème peut être facilement résolu en considérant une capacité spéciale `dead`, ne permettant pas de réduction, et en remplaçant les capacités malformées comme $\mathtt{in}\ (\mathtt{in}\ n)$ par cette capacité.

La problématique reste cependant intacte pour la communication de séquences de messages. Ainsi, le processus $(\mathsf{fix}\, A.(x).(\langle x.x\rangle \mid A)) \mid \langle n\rangle$ peut se réduire en un nombre non borné de processus tous bien formés. Une réponse à ce problème pourrait être l'utilisation du système de typage avec capacités affines de [CG99] pour lequel ce dernier processus n'est pas typable.

Un calcul des ambients à contrôle fini intégrant réellement un mécanisme de communications de séquences reste à définir.

## 3.2 Model-checking pour les ambients à contrôle fini

Puisque le nombre de processus accessible par la relation de réduction est fini, on peut aisément adapter l'algorithme de model-checking de [12] pour ce calcul des ambients à contrôle fini. De plus, ceci se fait sans majoration de la complexité.

**Proposition 13 ([14])** *Le problème de model-checking du calcul des ambients à contrôle fini et de $\mathcal{L}^{-\triangleright}$, la logique des ambients sans adjoint de composition est PSPACE-complet.*

Pour terminer ce chapitre, mentionnons que nos travaux ont été étendus par Lin [Lin04]. Basé sur le calcul des ambients à contrôle fini que nous venons de présenter, Lin considère une logique des ambients plus riche que la nôtre puisque intégrant un opérateur de point-fixe. Il démontre pour cette logique que le problème de model-checking reste décidable, sans toutefois préciser la complexité.

# Chapitre 4

# Vérification en présence de réplication

Ce chapitre présente des travaux publiés dans [16, 4]. Ils étaient l'objet du stage de DEA de Iovka Boneva encadré par S. Tison et moi-même.

## 4.1 Limitation de l'approche par model-checking en présence de réplication

L'opérateur de réplication est un mécanisme simple et élégant pour exprimer l'itération et ajoute indéniablement de la puissance au calcul. Comme nous l'avons mentionné, Cardelli et Gordon ont montré que le calcul pur (*ie* sans communication) était Turing-complet [CG00b] ; la preuve de ce résultat passe par la simulation du fonctionnement d'une machine de Turing dans le calcul pur. Il est très simple de voir que ce codage d'une machine de Turing implique le fait que le problème de model-checking de la logique des ambients sans adjoint de composition est indécidable pour ce calcul.

D'autres travaux ont également eu pour but d'étudier l'expressivité du calcul des ambients : Zimmer a proposé dans [Zim03] un encodage du $\pi$-calcul synchrone dans le calcul pur des "Safe Ambients", montrant ainsi que la mobilité du calcul des ambients pouvait simuler un mécanisme de communications.

Concernant le problème de vérification par model-checking, nous avons très vite prouvé par réduction du problème de correspondance de Post (PCP) [Pos44] que

**Theorem 2** *[12] Le problème de model-checking du calcul pur et public $pMA^{-\nu}$ avec la logique des ambients (sans opérateur d'adjoint de composition) est indécidable.*

Ceci montrait que la restriction de la vérification au calcul sans réplication de [CG00a] était nécessaire pour la décidabilité du problème de model-checking de la logique $\mathcal{L}^{-\triangleright}$. On peut d'ailleurs en utilisant la même réduction montrer que le problème d'accessibilité (étant donnés deux processus, est-ce que le premier peut se réduire dans le second en un certain nombre d'étapes ?) était indécidable pour le fragment $pMA^{-\nu}$.

Finalement, il a été prouvé que contrairement à CCS et au $\pi$-calcul [13], l'opérateur de restriction était superflu pour obtenir la Turing-complétude du calcul des ambients ; dans [HLS02], Hirschkoff, Lozes et Sangiorgi proposent un encodage d'une machine de Turing dans le calcul pur et public. Le même résultat de Turing-complétude a été obtenu indépendamment par Busi et Zavattaro dans [BZ02] par encodage d'une machine à registres (*RAM - Random Access Machine*). De par ce dernier encodage, il est immédiat de voir que la convergence de nom (et donc, le problème de model-checking de $\mathcal{L}^{-\triangleright}$) est indécidable pour le calcul pur et public.

[13] CCS et le $\pi$-calcul sont bien sûr Turing-complet ; cependant sans l'opérateur de restriction, ces deux formalismes auraient l'expressivité des réseaux de Petri [Gol88, AM02].

Ainsi, la présence de l'opérateur de réplication dans un calcul semble impliquer inévitablement l'indécidabilité du problème de model-checking pour ce calcul et la logique des ambients (même sans l'opérateur d'adjoint de composition).

## 4.2 Le calcul pur et public sans la capacité `open`

### 4.2.1 Motivations

Pour surmonter les problèmes de l'indécision de la vérification par model-checking, il existe deux possibilités : restreindre la classe des systèmes étudiés et/ou restreindre la classe des propriétés à vérifier.

C'est dans cette optique qu'a débutée l'étude que nous avons menée sur le calcul des ambients pur sans la capacité `open`. Par prudence (ou pessimisme), nous avons décidé de restreindre à la fois le calcul (*ie* les systèmes) et les propriétés.

**restriction du calcul** Nous avons décidé dans un premier temps de restreindre le calcul pur et public, calcul connu pour être Turing-complet. L'affaiblissement peut passer notamment par la restriction du jeu de capacités. Notre choix s'est posé sur l'élimination de la capacité `open`, motivé par les raisons suivantes :

- la capacité `open` est pour le moins la plus controversée du calcul. S'il peut avoir un sens dans le cas de modélisation d'aspect logiciel, il en va différemment pour ce qui est du matériel. Que peut bien signifier l'"'ouverture" d'un ordinateur portable ?
- le Boxed Ambient [BCC01] est une proposition d'un calcul qui ne comporte pas la capacité `open`, remplacée par un mécanisme plus sophistiqué de communication père-fils. Le fragment pur sans `open` du calcul des ambients coïncide donc avec le fragment sans communication du Boxed Ambients ; cette restriction semble donc être un point de départ avant de s'intéresser plus avant au Boxed Ambients.
- une autre motivation est purement intuitive quant au calcul ne comportant que les capacités `in` et `out` : alors que la capacité `open` modifie profondément l'arbre décrivant le processus en détruisant certains de ses nœuds, les capacités (symétriques) `in` et `out` ne font que réorganiser cet arbre, entretenant un espoir quant aux bonnes propriétés de ce fragment pour une vérification par model-checking.
- finalement, toujours de manière informelle, la capacité `open` permet à un ambient d'acquérir de nouvelles capacités par dissolution des frontières d'autres ambients tandis que, dans le cas de la restriction aux capacités `in` et `out`, les ambients possèdent dès le départ, les capacités permettant leur déplacement.

Il s'est, de plus, trouvé que Busi et Zavattaro ont mené une étude complémentaire en considérant un calcul privé des capacités de mouvement `in` et `out` [BZ02]. Cependant, ceci n'est qu'une justification *a posteriori* de notre étude, puisque nous ignorions ces travaux quand nous avons commencé cette dernière. Nous reviendrons sur les travaux de Busi et Zavattaro dans la section 4.3.

**restriction des propriétés à vérifier** Une logique complète comme la logique des ambients, même restreinte à un fragment décidable pour un calcul sans réplication, nous paraissait dans un premier temps un peu ambitieux. Notre objectif a été de ne considérer qu'une classe bien connue de propriétés, appelée *propriété de sûreté* (*safety property*).

Une propriété de sûreté marque l'absence d'un mauvais comportement d'un système lors de son exécution ; elle est vérifiée par un processus si quelle que soit son exécution, celle-ci ne passe pas par une mauvaise configuration. On se donne donc un ensemble de "mauvaises configurations", ensemble

soit fini, soit finiment décrit ; le problème se résume alors à décider s'il existe un processus qui est une mauvaise configuration et est accessible (par réduction/exécution) depuis un processus initial donné.

Modestement, nous avons commencé par nous intéresser au problème d'accessibilité, qui correspond au cas où l'ensemble des "mauvaises configurations" est réduit à un unique processus.

**ambitions** Notre objectif était de prouver que ce problème d'accessibilité entre deux processus était décidable, puis d'étudier la possibilité d'étendre ce résultat vers la vérification de propriétés de sûreté où les "mauvais" processus seraient décrits de manière particulièrement simple : ensembles réguliers de processus décrits par une grammaire, formules du fragment spatial de la logique (sans adjoint de composition, ni placement).

Cette démarche est à rapprocher de la vérification de propriétés de sûreté pour des systèmes infinis tels que l'algèbre de processus PA ou les réseaux de Petri ; ainsi, pour l'algèbre PA, alors que la logique temporelle CTL est indécidable [EK95], le fragment EF de cette logique permettant d'exprimer des propriétés de sûreté est, lui, décidable [May97, LS02]. En ce qui concerne les réseaux de Petri, il est connu que l'accessibilité y est décidable [May84] mais que le problème de model-checking, même pour la logique EF, y est lui indécidable [Esp97]. Il est néanmoins possible de vérifier (partiellement) des propriétés de sûreté pour les réseaux de Petri avec des techniques d'abstraction qui permettent de calculer une représentation symbolique d'un sur-ensemble des états accessibles depuis une configuration ou un ensemble de configurations initiales.

Idéalement, un résultat positif pour le problème d'accessibilité aurait sans doute permis d'aborder la vérification de propriétés de sûreté pour le calcul des ambients pur et/ou les Boxed Ambients même de manière partielle. Malheureusement, ce ne fut pas le cas.

### 4.2.2 Résultats

Les résultats énoncés dans cette section ont été publiés dans [16, 4]. La publication [4] se trouve en annexe E.

Nous avons donc considéré $pMA^{-\nu,\texttt{open}}$, le calcul des ambients pur et public sans la capacité `open` et montré que :

**Theorem 3 ([16, 4])** *L'accessibilité de la relation de réduction est indécidable dans $pMA^{-\nu,\texttt{open}}$, ie donnés $P, Q$ deux processus de $pMA^{-\nu,\texttt{open}}$, le problème "est-ce que $P \rightarrow^* Q$ ?" est indécidable.*

La preuve passe par l'encodage des exécutions d'une machine à deux compteurs et de l'indécision du problème d'atteindre une configuration précise d'acceptance (état final et compteurs à zéro) pour de telles machines.

La technique employée utilise le fait qu'une "partie" d'un processus, utilisé dans le calcul (par exemple, pour décrire un incrément), peut "disparaître" grâce à la règle (Struct Repl Copy) $!P \equiv !P \mid P$ de la congruence structurelle. En effet, cette règle est généralement associée à la génération de copies du processus $P$ dans une vision orientée de la gauche vers la droite de $!P \equiv !P \mid P$ ; cependant, nous utilisons ici également le fait que (Struct Repl Copy) peut servir à éliminer des copies de $P$.

Nous avons émis l'hypothèse que la source de l'indécidabilité de la relation de réduction venait de cette possibilité d'utiliser (Struct Repl Copy) dans les deux "directions" ; pour vérifier ceci, nous avons introduit une version faible du calcul $pMA^{-\nu,\texttt{open}}$ en considérant [14]

- une relation de congruence stucturelle dans laquelle (Struct Repl Copy) $!P \equiv !P \mid P$ est omise ;
- une relation de réduction enrichie par (Red Repl) $!P \rightarrow !P \mid P$.

[14] Une telle définition apparaît notamment dans [AKPG01] et également dans [LS00] mais pour l'opérateur de récursion et non celui de réplication.

Nous validons notre hypothèse en montrant que :

**Theorem 4 ([16, 4])** *L'accessibilité de la relation de réduction est décidable dans le calcul $pMA^{-\nu,\mathtt{open}}$ faible.*

La preuve se fait par réduction vers le problème d'accessibilité dans les réseaux de Petri [May84]. Il est à noter que cette propriété de décision dépend également de manière cruciale du jeu de capacités restreint de $pMA^{-\nu,\mathtt{open}}$ ; en effet, si on définit similairement un fragment faible pour $pMA^{-\nu}$, le calcul pur et public, alors le problème d'accessibilité de la réduction pour ce calcul est indécidable. On peut facilement obtenir ce résultat en utilisant la réduction du problème de correspondance de Post (*PCP*) de [12].

Cependant, même le résultat positif du théorème 4 ne peut malheureusement pas être très utile en ce qui concerne la vérification par model-checking, car nous avons montré que :

**Theorem 5** *Le problème de convergence de nom (et donc, de model-checking pour $\mathcal{L}^{-\triangleright}$, la logique des ambients sans adjoint de composition) est indécidable pour le calcul $pMA^{-\nu,\mathtt{open}}$ et le calcul $pMA^{-\nu,\mathtt{open}}$ faible.*

Ainsi, même pour un fragment simple du calcul pour lequel l'accessibilité de la relation de réduction est décidable, le problème de convergence de nom est indécidable alors qu'il correspond, à la fois, à une formule très simple de la logique et à un ensemble facilement descriptible de processus.

Récemment, notre travail a été complété par Busi et Zavattaro [BZ05] ; alors que nous avons obtenu un résultat de décidabilité pour l'accessibilité de la relation de réduction en modifiant la sémantique du calcul, ils ont quant à eux préservé la sémantique originelle mais restreint un peu plus la syntaxe : ils limitent l'usage de la réplication aux processus qu'ils nomment "gardés", c'est-à-dire débutant par une capacité. Ainsi, la syntaxe de la réplication devient $!\mathtt{in}\ n.P$ et $!\mathtt{out}\ n.P$. S'inspirant des techniques que nous avons développées, Busi et Zavattaro montrent que dans ce cas l'accessibilité de la relation de réduction est décidable. Ils étendent également ce résultat à une forme plus faible d'accessibilité qu'ils nomment "accessibilité spatiale", où le processus qu'on souhaite atteindre n'est que partiellement décrit utilisant pour cela la décidabilité du problème de couverture dans les réseaux de Petri. Finalement, Busi et Zavattaro montrent que le jeu de capacités considéré est crucial puisque l'ajout de la capacité `open` à ce calcul avec réplication gardée ne permet pas de préserver la décidabilité de l'accessibilité de la relation de réduction.

## 4.3 Expressivité de calculs minimalistes

Comme nous l'avons déjà mentionné, notre travail a été, avant tout, motivé par des problématiques de vérification. Cependant, parallèlement à celui-ci, diverses études sur de petits fragments de calcul des ambients ont été menées. Bien que visant uniquement à mesurer l'expressivité de tels fragments, ces travaux sont néanmoins proche des nôtres.

Parallélement à notre travail, Maffeis et Phillips ont également considéré $pMA^{-\nu,\mathtt{open}}$, le fragment du calcul des ambients pur et public sans la capacité `open` et étudié son expressivité [MP05] ; ils ont démontré la Turing-complétude de ce fragment y compris pour une restriction de la réplication aux processus débutant par une capacité.

Il est à noter que

– l'encodage des machines à deux compteurs proposé dans [16] ne permettait pas de conclure à la Turing-complétude de $pMA^{-\nu,\mathtt{open}}$, puisque celui-ci construisait un processus systématiquement divergent, dont aucune des suites de réductions possibles n'était terminante. Il en va différemment dans [4] et le résultat de Turing-complétude peut être déduit de ce nouvel encodage.

– la simulation des machines à registres de [MP05] ne permet pas d'obtenir le résultat d'indécidabilité de l'accessiblité de la relation de réduction de $pMA^{-\nu,\texttt{open}}$. Bien au contraire, puisque selon [BZ05], l'accessiblité de la relation de réduction pour le fragment considéré de $pMA^{-\nu,\texttt{open}}$ (réplication restreinte aux processus débutants par une capacité) est décidable.

Nous avons évoqué le fait que dans le cas du fragment $pMA^{-\nu,\texttt{open}}$ où la réplication restreinte aux processus débutants par une capacité, non seulement l'accessibilité de la relation de réduction était décidable, mais également une propriété plus forte dite "accessibilité spatiale" [BZ05]. Ceci permet d'exprimer des propriétés sur la structure spatiale d'un processus atteignable depuis un processus initial. Cependant, cette description de la structure spatiale à atteindre ne peut pas être quelconque puisque comme mentionné explicitement dans [MP05], le problème de convergence de nom (et donc, de model-cheching de la logique $\mathcal{L}^{-\triangleright}$) demeure indécidable pour ce fragment du calcul.

Nous avons déja mentionné les travaux de Busi et Zavattaro concernant le calcul des ambients pur sans les capacités de mouvement `in` et `out` [BZ02, BZ04] ; les auteurs montrent qu'en présence de réplication [15] , un calcul pur sans capacités de mouvement `in` et `out` (intégrant ou non la restriction de nom) n'est pas Turing-complet [16]. Nous allons voir dans la section 4.4 que bien que n'étant pas Turing-complet, le problème de model-checking de ces calculs face à la logique $\mathcal{L}^{-\triangleright}$ n'est pas décidable.

## 4.4 Le calcul pur et public sans les capacités `in` et `out`

Nous nous proposons dans cette section de compléter les résultats concernant la vérification par model-checking de fragments du calcul des ambients avec réplication.

Comme nous avons évoqué précédemment, le calcul $pMA^{-\nu,mvt}$, c'est-à-dire le fragment pur, public et sans mouvement du calcul des ambients, n'est pas Turing-complet [BZ04]. Cependant nous allons montrer que le problème de model-checking de la logique $\mathcal{L}^{-\triangleright}$ est malgré tout indécidable pour ce fragment. Dans un second temps, nous montrons que le problème de convergence de nom est lui décidable.

Les réseaux de Petri vont constituer l'outil principal de la preuve de ces résultats.

**réseau de Petri** Pour une ensemble fini $E$, un multiensemble fini $m$ sur $E$ sera vu comme une application de $E$ dans $\mathbb{N}$, l'ensemble des entiers naturels, tel que $m(e) \neq 0$ pour un nombre fini d'éléments $e$ de $E$. Nous notons $\mathcal{M}(E)$ l'ensemble des multiensembles finis dont les élements sont dans $E$.

Un réseau de Petri est donné par un triplet $(\mathsf{P}, \mathsf{T}, W)$ tel que $\mathsf{P}$ est un ensemble fini de *places*, $\mathsf{T}$ est un ensemble fini de *transitions* et $W$ est une application de $\mathsf{T}$ dans $\mathcal{M}(\mathsf{P}) \times \mathcal{M}(\mathsf{P})$.

Pour une transition $t$ telle que $W(t) = (s_1, s_2)$, on note ${}^\bullet t$ le multiensemble $s_1$ et $t^\bullet$ le multiensemble $s_2$. Un *marquage* pour un réseau de Petri $PN = (\mathsf{P}, \mathsf{T}, W)$ est un multiensemble de places, *ie* un élément de $\mathcal{M}(\mathsf{P})$. Une transition $t$ est *déclenchable* pour un marquage $m$ si ${}^\bullet t \subseteq m$.

Pour un marquage $m$, si $t$ est déclenchable alors le déclenchement de $t$ produit le marquage $m'$ égal à $(m \setminus {}^\bullet t) \cup t^\bullet$. On notera dans ce cas, $m \overset{t}{\Rightarrow} m'$. On écrira $m \Rightarrow m'$ s'il existe une transition $t$ telle que $m \overset{t}{\Rightarrow} m'$. Comme d'habitude, $\Rightarrow^*$ désignera la clôture réflexive-transitive de $\Rightarrow$.

### 4.4.1 Model-checking de $pMA^{-\nu,mvt}$ face à la logique $\mathcal{L}^{-\triangleright}$

Nous montrons que le problème de model-checking du fragment pur, public et sans les capacités de mouvement du calcul des ambients est indécidable face à la logique des ambients sans adjoint de

[15] Si l'opérateur de récursion (au lieu de la réplication) est considéré, alors la restriction de nom joue un rôle très important, puisque délimitant la frontière de l'expressivité : le calcul pur sans capacité de mouvement est Turing-complet contrairement à sa version publique.

[16] La non-terminaison de la réduction des processus y est décidable.

composition.

La preuve est extrèmement simple et se réalise en deux étapes :

1. nous montrons que le fragment $pMA^{-\nu,mvt}$ permet de simuler le fonctionnement des réseaux de Petri
2. nous adaptons la preuve de l'indécidabilité du model-checking de la logique temporelle EF pour les réseaux de Petri [Esp97] à la logique des ambients $\mathcal{L}^{-\triangleright}$.

Pour être complètement précis, nous allons nous restreindre aux processus de $pMA^{-\nu,mvt}$ où l'opérateur de réplication ne s'applique qu'aux processus de la forme $\texttt{open}\, n.P$ et où les ambients sont vides, c'est-à-dire de la forme $m[\mathbf{0}]$.

Pour un processus $P$ et un entier naturel $k$, on note $P^k$ la composition de $k$ processus $P$, c'est-à-dire $\overbrace{P \mid \ldots \mid P}^{k}$ si $k$ est strictement positif et $\mathbf{0}$ si $k = 0$. Pour un (multi)ensemble fini de processus $S = \{P_1, \ldots, P_n\}$, on note $\Pi S$ le processus $P_1 \mid \ldots \mid P_n$.

**encodage des réseaux de Petri dans** $pMA^{-\nu,mvt}$ Soit $PN = (\mathsf{P}, \mathsf{T}, W)$, un réseau de Petri ; on considère pour chaque place de $\mathsf{P}$ un nom $p$ et pour chaque transition de $\mathsf{T}$ un nom $t$. Nous supposerons que l'ensemble des places (et des noms de places) est $\{p_1, \ldots, p_n\}$ et que l'ensemble des transitions (et des noms de transitions) est $\{t_1, \ldots, t_m\}$. Nous supposerons de plus un nom $lock$.

L'idée intuitive est de représenter comme un processus le couple réseau de Petri-marquage de façon à ce que le déclenchement des transitions dans le réseau (et donc l'évolution du marquage) soit simulé par des réductions du processus.

Pour tout multiensemble de places $m$, on définit $[\![m]\!]_{post}$ comme le processus $\Pi\{p_i[\mathbf{0}]^{m(p_i)} \mid 1 \leq i \leq n\}$. Nous utiliserons notamment $[\![m]\!]_{post}$ pour représenter un marquage sous la forme d'un processus.

Pour une transition $t_i$, nous définissons le processus $[\![t_i]\!]$ comme

$$!\texttt{open}\ lock.[\![{}^\bullet t_i]\!]_{pre}.(t_i[\mathbf{0}] \mid [\![t_i^\bullet]\!]_{post} \mid \texttt{open}\ t_i.lock[\mathbf{0}])$$

où pour un multiensemble de places $m$, $[\![m]\!]_{pre}$ définit récursivement une séquence de capacités comme $[\![\varnothing]\!]_{pre} = \epsilon$ et $[\![m \cup \{p\}]\!]_{pre} = \texttt{open}\, p.[\![m]\!]_{pre}$.

Finalement, pour le réseau de Petri $PN$, on définit le processus $[\![PN]\!]$ comme $lock[\mathbf{0}] \mid [\![\mathsf{T}]\!]$. L'ensemble $\mathsf{T}$ étant égal à $\{t_1, \ldots, t_m\}$, nous définissons $[\![\mathsf{T}]\!]$ comme le processus $\Pi\{[\![t_1]\!], \ldots, [\![t_n]\!]\}$.

Le fonctionnement de réseau $PN$ pour un marquage $m$ est simulé par un processus $[\![(PN, m)]\!]$ défini comme $[\![PN]\!] \mid [\![m]\!]_{post}$. L'ambient $lock$ est utilisé pour gérer l'exclusion mutuelle entre les différentes transitions. Intuitivement, la simulation du déclenchement d'une transition $t_i$ va

1. ouvrir l'ambient $lock$,
2. ouvrir les ambients correspondant à des éléments (jetons) dans les places nécessaires au déclenchement de la transition, *ie* correspondant à ${}^\bullet t_i$,
3. produire un ambient $t_i$ marquant le fait que cette transition est déclenchée et des ambients correspondant aux éléments dans les places modifiées par le déclenchement de la transition, *ie* $t_i^\bullet$,
4. finalement, ouvrir l'ambient $t_i$ et produire un nouvel ambient $lock$ marquant la complétion de la transition.

Les processus exhibant l'ambient $lock$ correspondent aux encodages des configurations du réseau. Le choix de la transition à déclencher est purement non-déterministe et peut se faire de manière erronée : une suite d'ouverture d'ambients se retrouve alors bloquée (le nombre de jetons dans une certaine place se trouve être en nombre insuffisant pour rendre la transition réellement déclenchable), auquel cas le

processus est effectivement bloqué et ne peut plus se réduire. Notre encodage est correct dans le sens suivant :

**Proposition 14** *Pour tout réseau de Petri $PN$ et tout marquage $m$ de ce réseau,*

1. *il existe un processus $P$ tel que $[\![(PN,m)]\!] \equiv lock[\mathbf{0}] \mid P$,*
2. *si $m \Rightarrow^* m'$ alors $[\![(PN,m)]\!] \rightarrow^* [\![(PN,m')]\!]$*
3. *pour tout processus $Q'$ tel que $[\![(PN,m)]\!] \rightarrow^* lock[\mathbf{0}] \mid Q'$, il existe un marquage $m'$ tel que $lock[\mathbf{0}] \mid Q' \equiv [\![(PN,m')]\!]$ et $m \Rightarrow^* m'$,*
4. *pour toute transition $t_i$ et tout processus $R'$, si $[\![(PN,m)]\!] \rightarrow^* t_i[\mathbf{0}] \mid R'$ alors il existe $m', m''$ tels que $m \Rightarrow^* m' \stackrel{t_i}{\Rightarrow} m''$ et*
   - *il existe un processus $P$ tel que $t_i[\mathbf{0}] \mid R' \rightarrow P$, et*
   - *pour tout processus $P$ tel que $t_i[\mathbf{0}] \mid R' \rightarrow P$, $P \equiv [\![(PN,m'')]\!]$*

**model-checking de la logique $\mathcal{L}^{-\triangleright}$** Comme nous l'avons déja évoqué, nous allons adapter la preuve de [Esp97] montrant l'indécidabilité du model-checking des réseaux de Petri face à la logique temporelle EF. Une des différences entre les logiques EF et $\mathcal{L}^{-\triangleright}$ est l'existence dans EF de la modalité EX (*Exists neXt*) ou pour être plus précis, dans notre cas, de la famille de modalités relativisées aux transitions $\mathrm{EX}(t_i)$ : une configuration (un marquage) vérifie une formule $\mathrm{EX}(t_i)\psi$ si la transition $t_i$ est déclenchable et après son déclenchement, la configuration obtenue vérifie la formule $\psi$. Ainsi, un marquage vérifie $\mathrm{EX}(t_i)true$ si et seulement si la transition $t_i$ y est déclenchable. Nous notons $decl(t_i)$ la formule qui affirme que $t_i$ est déclenchable.

La preuve de [Esp97] est une réduction du problème d'inclusion des marquages accessibles des réseaux de Petri, problème connu pour être indécidable. La réduction utilise une famille $\mathcal{P}$ de réseaux de Petri ayant chacun une transition distinguée $t_{AB}$ et un marquage initial. Il convient de remarquer que le réseau et son marquage initial vérifient que la transition $t_{AB}$ est initialement déclenchable et le reste jusqu'à son déclenchement, puis demeure à jamais non déclenchable quel que soit le marquage atteint. Le résultat d'indécidabilité est obtenu pour la formule suivante :

$$\Psi_u = \mathrm{AG}(decl(t_{AB}) \longrightarrow \mathrm{EX}(t_{AB})\,\mathrm{EF}\,(\bigwedge_{t\in\mathsf{T}} \neg decl(t)))$$

où $\mathrm{AG}\psi$ (*Always Globally*) est équivalent à $\neg(\mathrm{EF}\neg\psi)$. Informellement, cette formule signifie que dans quelque configuration où la transition $t_{AB}$ est déclenchable, son déclenchement nous amène dans une configuration $m$ pour laquelle il existera une configuration $m'$ accessible après un certain nombre de déclenchements de transitions (*ie* $m \Rightarrow^* m'$) telle que $m'$ ne possède plus de transitions déclenchables.

Considérons maintenant la logique des ambients $\mathcal{L}^{-\triangleright}$ et l'encodage des réseaux de Petri avec marquage que nous avons proposé.

Notons tout d'abord que la logique nous permet d'identifier les processus obtenus par réduction de $[\![(PN,m_0)]\!]$ qui correspondent effectivement à un encodage du marquage d'un réseau de Petri : de par la proposition 14, tout processus $P$ obtenu par réduction depuis $[\![(PN,m_0)]\!]$ qui vérifie la formule $lock[\mathbf{0}] \mid \mathbf{T}$ est structurellement congru à $[\![(PN,m)]\!]$ pour un certain $m$ tel que $m_0 \Rightarrow^* m$.

Essayons maintenant de définir une formule de la logique des ambients pour $decl(t_i)$ qui permettra de tester si une certaine transition est déclenchable ; il est très simple de voir de par la définition de $[\![t_i]\!]$ (et particulièrement de la séquence de capacités $[\![^\bullet t_i]\!]_{pre}$) que $decl(t_i)$ correspond précisément à la formule[17] $([\![^\bullet t_i]\!]_{post} \mid \mathbf{T})$

[17] Nous faisons ici un abus de notation utilisant le fait qu'un processus statique possède la même syntaxe qu'une formule de la logique des ambients.

Nous notons également le fait qu'une formule de la logique EF de la forme EF$\psi$ se traduira naturellement en une formule de la logique des ambients de la forme $\Diamond((lock[\mathbf{0}] \mid \mathbf{T}) \wedge \phi)$. Dualement, la formule AG$\psi$ se traduira dans une formule de la logique des ambients de la forme $\Box((lock[\mathbf{0}] \mid \mathbf{T}) \Longrightarrow \phi)$. Finalement, considérons la sous-formule de $\Psi_u$ de la forme EX$(t_{AB})$EF$\psi$ : comme nous l'avons déja mentionné la logique des ambients ne dispose pas de modalité de type EX qui permet d'exprimer notamment qu'une propriété sera vraie après un certain nombre fixé de transitions (*ie* de réduction). Cependant, nous allons pouvoir utiliser la particularité de la formule $\Psi_u$ et de notre encodage. En effet, il suffit de traduire le fait que si $t_{AB}$, est déclenchée, on finit par atteindre une configuration ayant une certaine propriété. De plus, nous sommes en mesure d'observer le déclenchement de la transition $t_{AB}$, puisque juste avant la complétion de cette transition, le processus exhibera le nom $t_{AB}$.

Utilisant le fait, pour les réseaux de la famille $\mathcal{P}$ et leur marquage initial, que $t_{AB}$ est toujours déclenchable si et seulement si elle n'a jamais été déclenchée, nous définissons donc la formule $\Phi_u$ de la logique des ambients $\mathcal{L}^{-\triangleright}$ comme

$$\Box((t_{AB}[\mathbf{0}] \mid \mathbf{T}) \Longrightarrow \Diamond \bigwedge_{t \in \mathsf{T}} \neg([\![^{\bullet} t]\!]_{post} \mid lock[\mathbf{0}] \mid \mathbf{T}))$$

**Proposition 15** *Pour tout réseau de Petri $PN$ de la famille $\mathcal{P}$ équipé d'un marquage initial $m_0$, on a $PN, m_0 \models \Psi_u$ ssi $[\![(PN, m_0)]\!] \models \Phi_u$.*

On en déduit

**Theorem 6** *le problème de model-checking du fragment $pMA^{-\nu,mvt}$ du calcul des ambients face à la logique $\mathcal{L}^{-\triangleright}$ est indécidable.*

### 4.4.2 Convergence de nom pour le fragment $pMA^{-\nu,mvt}$

Nous allons montrer dans cette section que le problème de convergence est décidable pour le fragment du calcul des ambients pur, public et sans les capacités `in`, `out`.

La preuve se fera en deux temps : tout d'abord, nous allons restreindre la relation de réduction à une version superficielle. Cette réduction restreinte fera que les ambients ne seront plus un contexte admissible pour la réduction. Nous montrons alors que si la convergence de nom a lieu avec la relation de réduction alors elle se produit également avec la relation superficielle.

Dans un second temps, nous réduisons le problème de convergence de nom dans le problème de couverture des réseaux de Petri ; ceci est réalisé en utilisant le fait que les réseaux de Petri peuvent simuler la relation de réduction superficielle.

**Remark 1** *Il convient de noter que si un processus exhibe le nom $n$ alors on peut en déduire une propriété purement syntaxique indépendante de la congruence structurelle ; si $P$ exhibe le nom $n$, alors tout processus $Q$ congru à $P$ sera de l'une des quatre formes suivantes $R_1 \mid n[R] \mid R_2$, $R_1 \mid n[R]$, $n[R] \mid R_2$ ou $n[R]$.*

**de la réduction vers la réduction superficielle** Pour définir la réduction superficielle, nous allons tout d'abord définir inductivement une famille de relations $(\rightarrow_i)_{i \in \mathbb{N}}$. Ces relations $\rightarrow_i$ sont les plus petites vérifiant les règles suivantes :

$$\begin{array}{ll}
\texttt{open}\, n.P \mid n[Q] \rightarrow_i P \mid Q & \text{(Red Open } i\text{)} \\
P \rightarrow_i Q \Rightarrow P \mid R \rightarrow_i Q \mid R & \text{(Red Par)} \\
P \equiv P', P' \rightarrow_i Q', Q' \equiv Q \Rightarrow P \rightarrow_i Q & \text{(Red } \equiv\text{)} \\
 & \\
P \rightarrow_i Q \Rightarrow n[P] \rightarrow_{i+1} n[Q] & \text{(Red Amb)}
\end{array}$$

Il est évident que $P \to Q$ si et seulement s'il existe un entier naturel $i$ tel que $P \to_i Q$.

Nous nommerons $\to_0$ la relation de réduction superficielle. Nous montrons maintenant que la convergence de nom est préservée lorsqu'on considère la réduction superficielle.

**Proposition 16** *Pour tout processus $P$, si $P$ converge vers le nom $n$ alors il existe un processus $Q'$ exhibant le nom $n$ tel que $P \to_0^* Q'$.*

**Preuve :** Si $P$ converge vers $n$ alors il existe des processus $Q_1, \ldots, Q_k$ tel que $P \to Q_1 \to Q_2 \to \ldots \to Q_k$ et $Q_k$ exhibe le nom $n$. Cette suite de réductions peut se réécrire à l'aide des relations $\to_i$ comme $P \to_{i_1} Q_1 \to_{i_2} Q_2 \to_{i_3} \ldots \to_{i_k} Q_k$. Nous définissons le poids d'une telle suite de réductions comme $\sum_{j=1}^{k} i_j$. Si le poids de la suite de réductions $P \to_{i_1} Q_1 \to_{i_2} Q_2 \to_{i_3} \ldots \to_{i_k} Q_k$ est nul, alors la preuve est terminée sinon on va transformer cette suite en une suite de poids strictement plus petit et telle que son extrémité terminale exhibera elle aussi le nom $n$.

L'intuition derrière cette transformation est la suivante : si une réduction profonde (*ie* non superficielle) a lieu et que l'ambient qui la contient est persistant (*ie* reste présent dans toute la suite de réductions) alors cette réduction n'interfère pas avec l'exhibition finale et peut donc être éliminée. Au contraire, si cet ambient est ouvert alors les réductions qui se sont produites en son sein peuvent être retardées pour se produire au plus haut niveau.

Considérons donc la suite de réductions $P \to_{i_1} Q_1 \to_{i_2} Q_2 \to_{i_3} \ldots \to_{i_k} Q_k$ et $i_l$ le plus grand indice non nul. La réduction $Q_{i_{l-1}} \to_{i_l} Q_{i_l}$ fait nécessairement intervenir deux processus $R, R'$ tels que $m[R] \to_{i_l} m[R']$. Maintenant, nous raisonnons sur cet ambient :

- cet ambient n'est pas ouvert dans la suite de la réduction. Puisque toutes les réductions restantes sont superficielles, aucune réduction ne se produit au niveau de $R'$ ; ainsi, $m[R']$ reste un sous-processus de $Q_{i_{l+1}}, \ldots, Q_{i_k}$ modulo la congruence structurelle. Pour tout $j$ tel que $0 \leq j \leq k-l$, on obtient le processus $Q'_{i_{l+j}}$ à partir de $Q_{i_{l+j}}$ en remplaçant le sous-processus $m[R']$ par $m[R]$. Il est alors simple de vérifier que

  $$P \to_{i_1} Q_1 \to_{i_2} \ldots \to_{i_{l-2}} Q_{i_{l-2}} \to_{i_{l-1}} Q_{i_{l-1}} \to_0 Q'_{i_{l+1}} \to_0 \ldots \to_0 Q'_k$$

  est bien une suite de réductions.
- cet ambient est ouvert dans la suite de la réduction. Nous supposons que cette ouverture a lieu au $h+1^{\text{ième}}$ élément de la suite. Puisque toutes les réductions restantes sont superficielles, aucune réduction ne se produit pour le sous-processus $m[R']$ ; ainsi, $m[R']$ reste un sous-processus de la suite tant que celui-ci n'est pas ouvert, *ie* de $Q_{i_{l+1}}, \ldots, Q_{i_{h-1}}$, modulo la congruence structurelle. On définit la suite de processus $Q'_{i_{l+1}}, \ldots, Q'_{i_{h-1}}$ en remplaçant dans chacun des $Q_{i_{l+1}}, \ldots, Q_{i_{h-1}}$, le sous-processus $m[R']$ par $m[R]$. Il est alors simple de montrer que

  $$P \to_{i_1} Q_1 \to_{i_2} \ldots \to_{i_{l-2}} Q_{i_{l-1}} \to_0 Q'_{i_{l+1}} \to_0 \ldots \to_0 Q'_{i_{h-1}} \to_0 S \to_{i_l - 1} Q_{i_h} \to_0 \ldots \to_0 Q'_k$$

  où $S$ est identique à $Q_{i_h}$ mais possède un sous-processus $R$ au lieu de $R'$, est bien une suite de réductions.

Dans les deux cas, la suite de réductions obtenue est de poids strictement plus petit que celle d'origine. De plus, le dernier processus de cette suite exhibe lui aussi le nom $n$. En réitérant cette transformation, on obtient une suite de réductions superficielles dont le dernier processus exhibe le nom $n$. ■

Nous n'allons plus considérer que la relation de réduction superficielle.

**de la réduction superficielle vers les réseaux de Petri** Le point le plus important ici est de remarquer que lors de la réduction de la capacité open, le processus résultant est la composition de deux sous-processus du processus initial. Ainsi, ce sont les sous-processus du processus initial qui définiront les différentes composantes des processus obtenus après réduction. Ceci permet de ne considérer qu'un nombre fini de processus ; ceux-ci vont représenter les places d'un réseau de Petri et les transitions de ce réseaux modéliseront leur interaction (superficielle).

Nous définissons $\mathsf{SubProc}(P)$ comme l'ensemble des sous-processus de $P$. Ainsi, pour le processus $P$ valant $\mathtt{open}\, M.\mathbf{0} \mid m[!\mathtt{open}\, n.\mathbf{0}] \mid n[n[\mathtt{open}\, l.\mathbf{0}]] \mid l[a[\mathbf{0}]]$, l'ensemble $\mathsf{SubProc}(P)$ est égal [18] à

$$\left\{\begin{array}{l} \mathbf{0},\ \mathtt{open}\, m.\mathbf{0},\ \mathtt{open}\, n.\mathbf{0},\ !\mathtt{open}\, n.\mathbf{0},\ m[!\mathtt{open}\, n.\mathbf{0}], \\ \ \mathtt{open}\, l.\mathbf{0},\ n[\mathtt{open}\, l.\mathbf{0}],\ n[n[\mathtt{open}\, l.\mathbf{0}]],\ a[\mathbf{0}],\ l[a[\mathbf{0}]], \\ \mathtt{open}\, m.\mathbf{0} \mid m[!\mathtt{open}\, n.\mathbf{0}], \\ \mathtt{open}\, m.\mathbf{0} \mid m[!\mathtt{open}\, n.\mathbf{0}] \mid n[n[\mathtt{open}\, l.\mathbf{0}]], \\ \mathtt{open}\, m.\mathbf{0} \mid m[!\mathtt{open}\, n.\mathbf{0}] \mid n[n[\mathtt{open}\, l.\mathbf{0}]] \mid l[a[\mathbf{0}]] \end{array}\right\}$$

Pour un processus $P$, nous définissons le réseau de Petri $PN_P$ comme $(\mathsf{P}, \mathsf{T}, W)$ où :
- l'ensemble des places $\mathsf{P}$ est $\mathsf{SubProc}(P)$ ;
- il existe dans $\mathsf{T}$ une transition pour chaque processus de la forme $!Q$ (type 1) ou $Q \mid R$ (type 2) et une transition pour chaque couple de processus de la forme $(\mathtt{open}\, n.Q, n[R])$ (type 3) ;
- l'application $W$ est définie comme :
  (1) si $t$ est type 1 et mise pour $!Q$ alors $W(t) = (\{!Q\}, \{!Q, Q\})$ ;
  (2) si $t$ est type 2 et mise pour $Q \mid R$ alors $W(t) = (\{Q \mid R\}, \{Q, R\})$ ;
  (3) si $t$ est type 3 et mise pour $(\mathtt{open}\, n.Q, n[R])$ alors $W(t) = (\{\mathtt{open}\, n.Q, n[R]\}, \{Q, R\})$.

Nous donnons à la figure 4.1 le réseau $PN_P$ pour $P$ valant $\mathtt{open}m.\mathbf{0} \mid m[!\mathtt{open}\, n.\mathbf{0}] \mid n[n[\mathtt{open}l.\mathbf{0}]] \mid l[a[\mathbf{0}]]$.

Soit $P$ un processus ; considérons son réseau de Petri $PN_P$ et $m$ un marquage de ce réseau. Nous définissons $[\![PN_P, m]\!]_{proc}$ comme le processus $\Pi\{p^{m(p)} \mid p \in \mathsf{P}\}$.

Nous montrons la correction de notre encodage en prouvant que :

**Proposition 17** *Pour tout processus $P$,*
- *soient $m, m'$ des marquages de $PN_P$. Si $m \Rightarrow^* m'$ alors $[\![PN_P, m]\!]_{proc} \equiv [\![PN_P, m']\!]_{proc}$ ou $[\![PN_P, m]\!]_{proc} \rightarrow_0^* [\![PN_P, m']\!]_{proc}$ ;*
- *pour tout processus $Q$ tel que $P \rightarrow_0^* Q$, il existe un marquage $m$ tel que $m$ est accessible depuis le marquage $\{P\}$ (ie $\{P\} \Rightarrow^* m$) et $[\![PN_P, m]\!]_{proc} \equiv Q$.*

De par la remarque 1 et la proposition 17,

**Proposition 18** *Le processus $P$ converge vers le nom $n$ si et seulement si il existe pour le réseau de Petri $PN_P$, une place de la forme $n[Q]$ et un marquage $m$ tel que $\{P\} \Rightarrow^* m$ et $1 \leq m(n[Q])$.*

Cette dernière condition est décidable puisque le réseau ne possède qu'un nombre fini de places (de la forme $n[Q]$) et que le problème de couverture dans les réseaux de Petri est décidable,

**Theorem 7 ([Rac78])** *Soit $PN$ un réseau de Petri et $m_0$, un marquage initial de ce réseau. Pour tout marquage $m$ de $PN$, on peut décider s'il existe un marquage $m'$ de $PN$ tel que $m_0 \Rightarrow^* m'$ et $m \subseteq m'$.*

Ainsi,

**Theorem 8** *Le problème de convergence de nom est décidable pour le fragment $pMA^{-\nu,mvt}$ du calcul des ambients.*

[18] Sans plus de précision, nous supposons que | induit un parenthèsage de la gauche vers la droite.

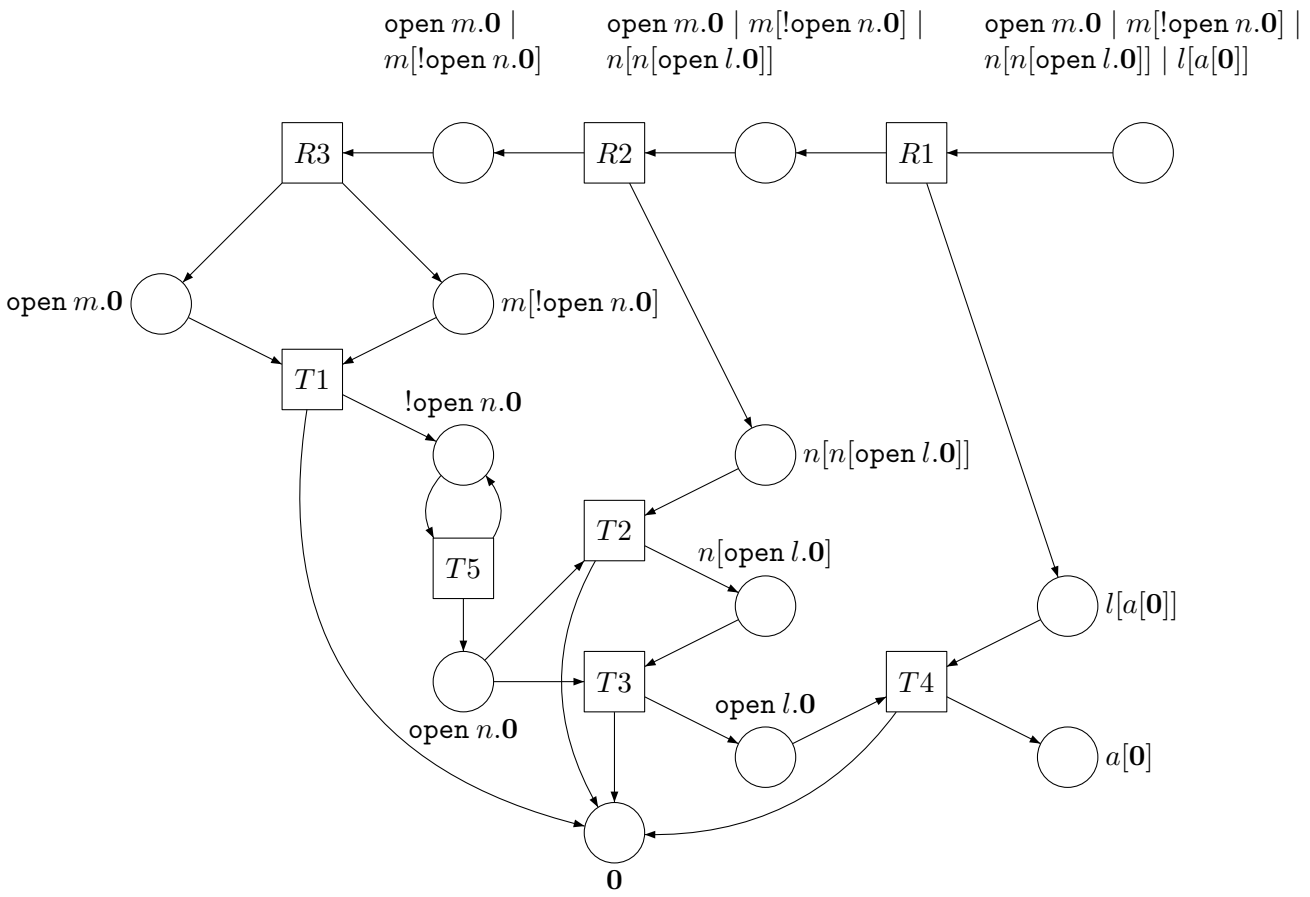



FIG. 4.1 – Le réseau de Petri pour le processus open $m.\mathbf{0}$ | $m$[!open $n.\mathbf{0}$] | $n[n[$open $l.\mathbf{0}]] | $l[a[\mathbf{0}]]$. Les transitions T1-4 sont de type 2, la transtion T5 de type 1 et les transitions R1-3 sont de type 3.

## 4.5 Un schéma général

Nous résumons à la figure 4.2 des résultats de décidabilité concernant des fragments du calcul des ambients pur, public et avec réplication. Ces résultats concernent la décidabilité du problème de model-checking face à la logique sans adjoint de composition, de la convergence de nom et de l'accessibilité. A chaque fragment est associé un tableau de trois cases, dont la première concerne le problème de model-checking, la seconde celui de la convergence de nom et la dernière celui de l'accessibilité. La lettre $D$ est mise pour décidable, $I$ pour indécidable et ? pour un problème encore ouvert. Les références $[*]$ indiquent des résultats nouveaux présentés dans ce mémoire.

L'indice $!g$ sur le nom des fragments indique des fragments où la réplication ne se trouve que devant une capacité.

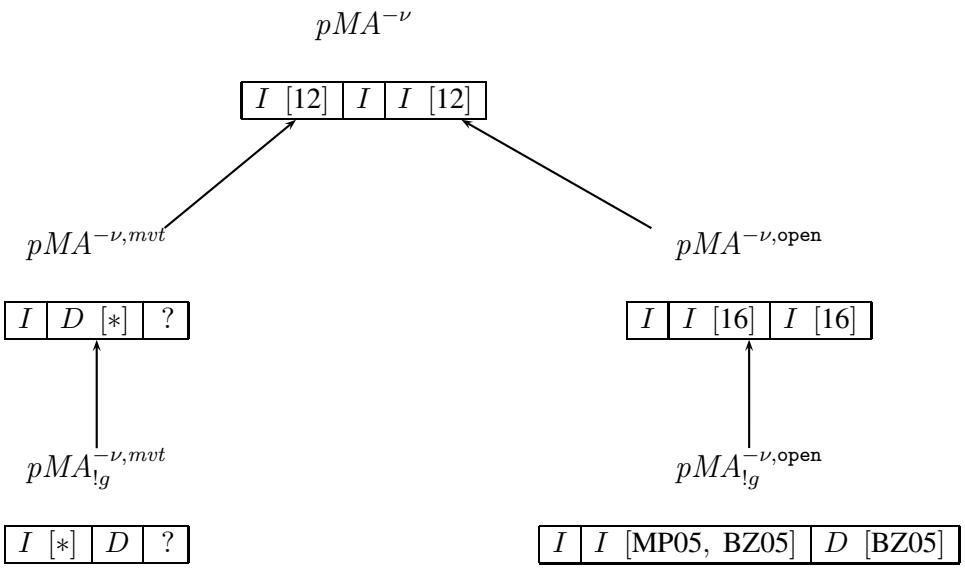



FIG. 4.2 – De la décidabilité du model-checking pour la logique $\mathcal{L}^{-\triangleright}$, de la converge de nom et de l'accessibilité pour divers fragments du calcul des ambients pur et public.

# Chapitre 5

# Conclusion et perspectives

Tentons dans cette conclusion d'avoir un regard critique sur le travail effectué. On peut dire que d'un premier abord, cette série de travaux forme un YAP [19]. En quoi ce qui a été réalisé n'est pas seulement la simple reformulation de questions classiques pour un nouveau calcul et, peut-être plus important, que peut-on en dégager ? Nous essayons de répondre à ces questions sur divers aspects de notre travail et de formuler ce qui pourrait être des pistes pour améliorer ou poursuivre ces travaux.

**bornes de complexité du model-checking** Les bornes inférieures de complexité que nous avons montrées pour le problème de model-checking sont obtenues pour des fragments du calcul et une logique très simples. Ces bornes pourront sans aucun doute être réutilisées pour d'autres calculs, en particulier pour le $\pi$-calcul. Elles tendent à montrer que la complexité combinée d'une logique spatiale équipée d'un opérateur de composition/séparation associatif et commutatif sera irrémédiablement PSPACE-dure. Les bornes obtenues pour la complexité de programmes relèvent plus de l'exercice ; bien qu'étant en pratique les plus importantes, elles sont en revanche souvent très dépendantes du formalisme utilisé, donc leur report à d'autres calculs semble difficile, voire exclu.

Nous avons montré que si la logique combine adjoint de composition et quantification existentielle (ou comme montré dans [CG04b], adjoint de composition et opérateur de révélation), le problème de model-checking devenait indécidable. Pourtant, l'adjoint de composition décrit exactement le type de problématique qu'on voudrait résoudre dans le cadre de la vérification d'agents concurrents : par exemple, "si l'applet reçue ne crée qu'un nombre borné de "threads" (vérifie une propriété $\phi$), alors le système ne produira pas d'erreur de dépassement de capacités (garantira une propriété $\psi$)". On peut se poser la question du problème de model-cheching en présence d'adjoint de composition mais sans quantification existentielle, ni révélation. Des résultats partiels concernant l'expressivité ont été obtenus dans le cas d'une logique n'ayant pas de composante temporelle [HLS02, Loz04, DGG04a]. Cependant les seuls travaux concernant le model-checking proprement dit opèrent sur un fragment du calcul purement statique [CCG03, DZLM04]. On peut dire que, dans cette direction, tout reste à faire.

**fragments minimalistes et réplication** Ce travail mentionne de nombreux encodages dans divers fragments du calcul des ambients ; ces encodages tiennent souvent plus de l'Oulipo que de la programmation, puisqu'ils obligent parfois à utiliser diverses contorsions pour obtenir le résultat désiré.

On peut toujours argumenter que de telles choses n'arrivent pas en pratique. Cependant ces travaux marquent une frontière sur les possibilités d'une vérification totalement automatique pour les aspects purement mobiles des formalismes de type ambient, comme par exemple, le Kell-calcul [BSS05].

[19] Yet Another Paper.

Notons également que ces travaux contredisent la présentation souvent faite du calcul des ambients comme une extension du $\pi$-calcul au même titre que le $\pi$-calcul avec localités ou $D\pi$. Les capacités de mobilité offrent à elles seules un calcul très puissant.

Le titre du chapitre 4 “Vérification en présence de réplication” est en fait peu révélateur des résultats obtenus. “L’impossibilité de la vérification en présence de réplication” aurait été plus exacte, puisque ce chapitre ne comporte pratiquement que des résultats négatifs.

Ces résultats amènent à la conclusion que la vérification pour le calcul des ambients avec réplication ne peut se faire que de manière approchée. Afin de vérifier le calculs des ambients, il convient de noter que celui-ci possède trois “dimensions” : la première est la hiérarchie des ambients et sa réorganisation, dimension illustrée dans $pMA^{-\nu,\texttt{open}}$, le calcul pur sans les capacités `open` . La seconde dimension est liée aux aspects de “multiplicité”, illustrée par l’encodage des réseaux de Petri dans le fragment sans mouvement $pMA^{-\nu,mvt}$. La dernière dimension est bien sûr liée aux mécanismes de communications.

Des systèmes de typage ou d’analyse offrent une première approche pour la vérification du calcul des ambients, du moins pour des propriétés simples comme les propriétés de sûreté. Cependant, aucun de ces travaux ne prend en compte l’intégralité des dimensions du calcul des ambients. Ainsi, l’analyse de forme (*shape analysis*) proposée dans [NN00] décrit la structure spatiale des ambients, mais en l’absence de communications et en approximant les aspects numériques sous une forme “un - plusieurs - une infinité”. Les communications (de noms ou de capacités) sont prises en compte dans le système de typage PolyA [AMW04] mais la forme des processus n’y est décrite que de façon très sommaire et les aspects de multiplicité sont absents.

Une vérification approchée mais de bonne qualité des propriétés de sûreté du calcul des ambients reste l’objet de travaux futurs.

**calcul à contrôle fini** Nos travaux sur le calcul des ambients à contrôle fini se trouvent cités à de nombreuses reprises dans des articles ayant pour objectif l’estimation ou le contrôle des ressources d’espace dans le cadre du calcul des ambients [TZH02, BBDCS03]. En effet, ces travaux estiment ou fixent des mesures quantitatives sur les ressources utilisables ou utilisées par les processus lors de leur exécution. Cette notion de “mesure quantitative” se retrouve dans notre système de typage définissant le calcul des ambients à contrôle fini.

# Deuxième partie

# Une étude formelle du langage de requêtes TQL

# Chapitre 6

# Des algèbres de processus aux données semi-structurées

Ce chapitre fait la transition entre la présente partie du mémoire et la précédente. Il est donc volontairement informel, les définitions des notions abordées se trouvant dans les chapitres suivants. Sa lecture n'est pas indispensable pour la compréhension de la seconde partie de ce mémoire.

Cardelli et Ghelli ont proposé dans [CG01a, CG04a] d'utiliser les processus statiques du calcul des ambients, qui ne sont finalement qu'une représentation d'arbres, comme formalisme pour modéliser des données semi-structurées. Ils proposent dualement d'utiliser une version enrichie du fragment spatial de la logique des ambients comme langage de requêtes pour ces données.

## 6.1 Ambients et données semi-structurées

**données semi-structurées** Par opposition aux données stockées dans les tables d'une base de données relationnelle dont la structure est connue, régulière et fixée, les données semi-structurées permettent d'exprimer des données irrégulières qui possèdent néanmoins un schéma sous-jacent plus ou moins strict. La structuration des données peut même être partiellement décrite dans les données elles-mêmes, comme cela est par exemple le cas dans XML (*eXtensible Markup Language*) [W3C00].

Les modèles des données semi-structurées proposés dans la littérature se basent soit sur des graphes, soit sur des arbres [ABS00]. Le modèle le plus populaire, XML, est un modèle arborescent. Grossièrement, un document XML est un mot d'un "langage de Dyck" dont l'alphabet est donné par un ensemble de balises ouvrantes et fermantes associées à des *éléments*. Du fait de cette caractéristique, le document XML induit donc un arbre ordonné d'arité non bornée dont les nœuds internes sont étiquetés par l'alphabet des éléments. Mais, dans toute sa généralité, la structure globale d'un document XML est un graphe, puisque l'existence de liens sous la forme d'*idref* de nœuds vers d'autres nœuds peut créer des cycles dans le modèle d'un document.

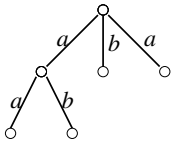
Malgré tout, dans le cadre de XML, la structure d'arbre reste le plus important et est déjà source de nombreuses problématiques. Ainsi, la majorité des travaux "formels" autour de XML ignorent ces liens *idref* et se focalisent sur la structure arborescente du document. Cardelli et Ghelli suivent cette approche et considèrent les données semi-structurées comme répondant à un modèle d'arbres.

**ambients et arbres** Les arbres sont représentés de manière indirecte par la notion d'*info-terme* ; ces info-termes sont en fait des processus purement statiques du calcul des ambients. Ainsi, $a[a[\mathbf{0}] \mid b[\mathbf{0}]] \mid a[\mathbf{0}] \mid b[\mathbf{0}]$ sera un info-terme.

Une relation de congruence est définie sur l'ensemble des info-termes, stipulant que $|$ est associatif-commutatif et que $\mathbf{0}$ est son élément neutre.

Une application associe alors les info-termes aux arbres qu'ils représentent effectivement. Ces arbres sont définis comme une imbrication de multiensembles et correspondent à des arbres non-ordonnés d'arité non bornée et dont les arêtes sont étiquetées ; cette définition en terme de multiensembles est fréquente pour les arbres non-ordonnés et se retrouve par exemple dans [SSM03]. Deux info-termes structurellement congrus représentent le même arbre. Ainsi, on se contente de raisonner sur les info-termes modulo la congruence en lieu et place des arbres.

Ainsi, on associera la représentation multiensembliste d'arbres $\{a\{b\{\varnothing\}, a\{\varnothing\}\}, a\{\varnothing\}, b\{\varnothing\}\}$ à l'info-terme précédemment donné. Cet arbre correspond au graphe suivant :




Notre présentation au chapitre 7 sera quelque peu différente ; nous définirons tout d'abord les arbres comme des graphes particuliers, puis un ensemble d'opérations permettant d'engendrer ces "graphes". Cette approche sera donc purement algébrique. Les propriétés des opérations, comme l'associativité-commutativité de $|$, seront simplement liées à l'interprétation de ce symbole sur les arbres.

Cette présentation sera très proche de celle des arbres de traits multiples (*multi-feature trees*) de [NP93].

Il convient de remarquer que les arbres considérés ici sont d'arité non bornée (un nœud peut avoir un nombre arbitraire de fils) mais contrairement aux arbres XML, les fils d'un même nœud ne sont pas ordonnés.

## 6.2 Logique des ambients et la logique d'arbres de TQL

Nous avons vu que la logique des ambients possède une composante temporelle (parlant de l'évolution d'un processus au cours du temps) et une composante spatiale (décrivant la structure du processus). C'est bien sûr la composante spatiale qui sera importante pour spécifier des requêtes sur des données semi-structurées.

Cardelli et Ghelli proposent dans [CG01a] une logique spatiale permettant d'effectuer des requêtes sur les info-termes. Cette logique correspond à la logique des ambients débarrassée des modalités temporelles, adjoint de composition et placement, opérateur de révélation et de masquage. Concernant les opérateurs spatiaux, outre les opérations de composition et de localité, l'opérateur "quelque part" $\diamond$ se voit remplacé par un opérateur de plus petit/grand point-fixe.

Une formule peut être utilisée pour tester si un arbre (un document) possède une certaine forme ou satisfait un certain nombre de contraintes. On retrouve ici la problématique pour laquelle la "communauté XML" a développé les notions de DTD (*Document Type Definition*) [W3C00] et de schémas XML [W3C04].

Cependant, les formules peuvent avoir également des variables libres : aux variables correspondant aux labels déjà présentes dans la logique des ambients, des variables correspondant à des arbres sont ajoutées à la logique, accompagnées de leur quantification existentielle. Ainsi, une formule avec des variables libres devient une requête pour un document : la valuation des variables de labels et de noms, faisant de cet arbre un modèle de la formule, est une solution de cette requête.

Le langage de la communauté XML pour définir des requêtes permettant d'identifier des nœuds dans un document est XPATH [W3C99a]. Le langage XPATH est un langage "navigationnel" qui permet de sélectionner des nœuds en fonction des chemins qui y conduisent.

## 6.3 Le langage de transformation de TQL

La logique que nous avons mentionnée à la section précédente est le cœur du système de requêtes TQL *Tree Query Language* [CGA+02]. Tout comme XQuery [W3C05], ce système est un langage de requêtes pour les documents semi-structurés.

Alors que le langage XQuery, au travers des expressions FLWOR, est plutôt opérationnel dans la description du résultat d'une requête, le langage de transformation de TQL est lui plus déclaratif, comme montré dans [CGA+02].

TQL se base sur un schéma " filtrage - capture - construction" : les formules logiques servent à filtrer le document, leurs variables libres capturant labels et sous-arbres. Ces opérations de filtrage et de capture sont utilisées dans un langage de (re)construction de documents semi-structurés, dont certaines parties se voient instanciées par les valeurs capturées. Ainsi, le résultat d'une requête TQL est lui-même un document semi-structuré. Le système TQL peut donc être utilisé comme un outil de transformation de documents semi-structurés produisant, à partir d'un document d'entrée, un document de sortie.

Ce principe de filtrage et capture dans un document XML est également le principe de base des langages XDuce [HP03] et CDuce [BCF03].

La logique sur laquelle se base TQL étant complexe (négation, récursion), l'implantation du système TQL a notamment nécessité une gestion astucieuse du traitement des captures de variables [CFG02]. Cependant, dans nos travaux, nous avons pour l'instant considéré uniquement la logique sous-jacente au système ; nous allons étudier son expressivité et les complexités pour les problèmes de satisfiabilité et model-checking (et/ou de réponses aux requêtes).

# Chapitre 7

# Arbres, automates, logiques et XML

Nous donnons dans ce chapitre les définitions nécessaires à la présentation de nos travaux. Certains des résultats mentionnés dans ce chapitre ont été publiés dans [18].

## 7.1 Modèles formels pour la validation et l'interrogation de documents XML

Comme nous l'avons déjà évoqué, la majeure partie des travaux sur une approche formelle de XML considère les documents comme des arbres ordonnés d'arité non bornée. Cet ordre provient de la représentation textuelle ou sérialisée d'un document XML qui induit naturellement un ordre entre les fils d'un même nœud.

On peut cependant abstraire ou ignorer cette relation d'ordre obtenant ainsi une approximation correcte du document d'origine. Pour reprendre une terminologie XPATH [W3C99a], on dira qu'on omettra les axes `following` − `sibling` et `preceding` − `sibling`. De très nombreux travaux ne considèrent pas ces axes induisant donc un modèle d'arbres non-ordonnés.

Les aspects formels des traitements des documents XML se divisent en trois groupes, à savoir la validation, l'interrogation et la transformation. Dans le premier cas, on s'intéresse à savoir si le document est conforme à une spécification. Cette spécification peut être donnée sous la forme d'une DTD ou d'un schéma [W3C00, W3C04]. Dans le cadre des bases de données, on parle également de requêtes booléennes puisque la réponse au problème est soit oui soit non.

Dans le second cas, on cherche toutes les parties d'un document vérifiant une certaine propriété. Cette propriété est fréquemment exprimée dans le langage (orienté "chemin") XPATH [W3C99a]. XPATH sélectionne ainsi dans le document un ensemble de nœuds, nœuds où vont par exemple s'appliquer une transformation décrite en XSLT [W3C99b] ou encore, nœuds qui vont être utilisés dans une reconstruction décrite, par exemple, en XQuery [W3C05].

Finalement, dans le troisième cas, on s'intéresse plus à des propriétés de typage du résultat d'une transformation [MPBS05] ; c'est dans ce domaine, particulièrement, que les aspects formels sont encore assez loin des aspects pratiques, comme le langage XQuery [W3C05].

La logique monadique du second ordre est une sorte d'étalon pour les logiques sur les structures induites par les mots ou les arbres d'arité bornée. Cette logique est en effet la logique des "réguliers" : sur les mots comme sur les arbres d'arité bornée [TW68], les ensembles de structures définissables par les formules closes sont précisément les langages reconnaissables respectivement de mots et d'arbres.

Dans le cadre des arbres ordonnés d'arité non bornée, la logique MSO tend également à devenir la logique de référence [NS01]. Son pouvoir d'expression pour la description d'ensembles d'arbres coïncide

avec la notion de reconnaissable. Les ensembles d'arbres reconnaissables sont, dans ce cas, définis au moyen d'automates de haies (*hedge automata*) [BKMW01, Mur99].

Pour la validation de documents, on retrouve la logique MSO au travers des automates de haies [BKMW01, Chi00] qui servent de référence pour cette tâche. Le formalisme de DTD (Document Type Definition) proposé par la communauté XML a d'ailleurs été étendu dans les *specialized DTD* [PV00] pour avoir la puissance de MSO.

Comme nous l'avons évoqué, le langage XPATH sélectionne dans un arbre un ensemble de nœuds satisfaisant une certain propriété. On parle alors de requêtes monadiques. Comme pour la validation de documents, la logique MSO sert de référence en ce qui concerne l'expressivité de formalismes exprimant des requêtes monadiques. La mauvaise complexité combinée inhérente à la logique MSO [FG02] a généré toute une série de travaux qui ont proposé des formalismes MSO-complets mais avec une complexité combinée meilleure que MSO. De manière non exhaustive, citons, par exemple, pour des formalismes basés sur la logique, DATALOG monadique [GK02], la logique "efficace" de [NS00] ou le $\mu$-calcul [BL05] et pour des formalismes plus opérationnels, les grammaires attribuées booléennes (*BAG*) [NV02], les automates de requêtes (*query automata*) de [NS02] ou les automates de forêts (*forest pushdown automata*) de [NS98].

Pour terminer, mentionnons que des travaux récents visent à étudier des formalismes permettant de définir des requêtes binaires [BE00, BS02] ou même $n$-aires [20] sélectionnant respectivement des couples et des $n$-uplets de nœuds.

## 7.2 Arbres

En informatique, le mot "arbre" est souvent utilisé pour désigner de manière génériques divers objets qui, bien qu'étant globalement semblables, divergent sur certains points.

**arbres, arbres et arbres** Un arbre est un graphe fini, non vide, orienté tel que le graphe non-orienté sous-jacent est acyclique. De plus, il existe un nœud particulier appelé *racine* qui ne possède pas d'arête entrante et, pour tout nœud, un unique chemin de la racine jusqu'à ce nœud. Ce type d'arbre est généralement appelé *arbre non-ordonné d'arité non bornée*.

On considère parfois que l'ensemble des fils de chacun des nœuds de l'arbre est équipé d'une relation d'ordre total. On parle alors d'*arbre ordonné d'arité non bornée*.

Un arbre est dit d'arité bornée s'il existe une borne fixée *a priori* sur le degré sortant des nœuds de l'arbre.

Les arbres sont souvent étiquetés, mais là aussi des variantes existent : les labels peuvent être soit sur les nœuds, soit sur les arêtes et parfois même sur les deux. L'étiquetage d'un arbre est défini comme une application des nœuds (et/ou des arêtes) vers un ensemble fini ou dénombrable de labels $\mathcal{N}$.

Lorsque l'arité d'un nœud dépend du symbole qui l'étiquette, on parle alors d'*arbre gradué* (*ranked tree*). Ce type d'arbres est souvent confondu avec la notion de termes sur une signature graduée.

Dans ce mémoire et sauf mention contraire, le mot "arbre" désignera un arbre non-ordonné d'arité non bornée dont les arêtes sont étiquetées par les éléments d'un ensemble dénombrable $\mathcal{N}$.

**une algèbre d'arbres** Nous supposons les arbres donnés sous la forme d'un graphe $(V, E, \lambda)$ tel que $V$ est un ensemble fini de nœuds, $E \subseteq V \times V$ est un ensemble fini d'arêtes et $\lambda$ est une application de $E$ dans $\mathcal{N}$.

Nous supposerons comme toujours que deux arbres (graphes) isomorphes sont égaux. Nous notons $\mathsf{Tree}$ l'ensemble de tous les arbres. Pour un arbre $\tau$, $\mathsf{root}(\tau)$ désignera la racine de $\tau$ ; pour tout nœud $v$, $\mathsf{children}(v)$ est l'ensemble des nœuds $\{v' \mid (v, v') \in E\}$.

Nous adoptons la présentation algébrique des arbres proposée dans [Cou90]. Nous supposons une signature $\Sigma$ donnée par la constante $\mathbf{0}$, les symboles unaires $a$ présents pour chaque $a$ de $\mathcal{N}$ et le symbole binaire $|$.

Soit $\mathcal{T}$ la $\Sigma$-algèbre ayant pour domaine l'ensemble de tous les arbres finis dont les arêtes sont étiquetées par des symboles de $\mathcal{N}$. La constante $\mathbf{0}$ est interprétée dans $\mathcal{T}$ comme $\mathbf{0}^{\mathcal{T}}$ l'arbre ayant un seul nœud et pas d'arête (nous ne considérons que des graphes non vides). Pour tout arbre $\tau = (V, E, \lambda)$ et tout $a$ de $\Sigma$, l'arbre $a^{\mathcal{T}}(\tau)$ est donné par $(V \cup \{r\}, E \cup \{(r, \mathsf{root}(\tau))\}, \lambda')$ où $r$ est un nouveau nœud n'appartenant pas à $V$ et $\lambda'$ étend $\lambda$ en posant que $\lambda'((r, \mathsf{root}(\tau))) = a$.

Pour tout couple d'arbres $\tau, \tau'$ définis par respectivement $(V, E, \lambda)$ et $(V', E', \lambda')$, $\tau \mid^{\mathcal{T}} \tau'$ est l'arbre défini par $(V'', E'', \lambda'')$ où (en supposant $V \cap V' = \varnothing$) :

- $V'' = (V \cup V' \cup \{r\}) \setminus \{\mathsf{root}(\tau), \mathsf{root}(\tau')\}$ (où $r \notin V \cup V'$)
- $E'' = \{(r, v) \mid v \in \mathsf{children}(\mathsf{root}(\tau)) \cup \mathsf{children}(\mathsf{root}(\tau'))\} \cup (E \setminus \{(\mathsf{root}(\tau), v) \mid v \in V\}) \cup (E' \setminus \{(\mathsf{root}(\tau'), v') \mid v' \in V'\})$.
- $\lambda''$ est défini comme $\lambda$ et $\lambda'$ pour les arêtes de $E''$ venant de $E$ et de $E'$ respectivement. De plus, $\lambda''$ satisfait que $\lambda''((r, v)) = \lambda((\mathsf{root}(\tau), v))$ si $v \in V$ et $\lambda''((r, v)) = \lambda((\mathsf{root}(\tau'), v))$ si $v \in V'$.

Informellement, $a^{\mathcal{T}}(\tau)$ ajoute une nouvelle arête étiquetée par $a$ d'une nouvelle racine vers l'ancienne racine de $\tau$. $\tau \mid^{\mathcal{T}} \tau'$ est obtenu à partir de $\tau$ and $\tau'$ en fusionnant leur racine. Ces opérations algébriques sont illustrées à la figure 7.1.

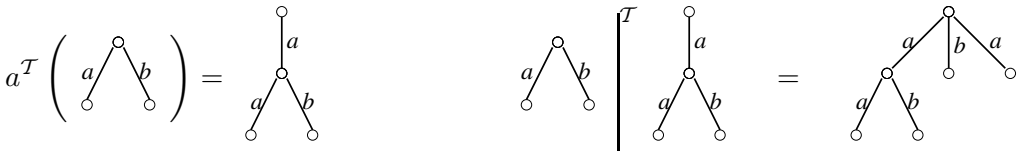



FIG. 7.1 – Opérations algébriques sur les arbres

On peut remarquer que l'ensemble des arbres Tree est finiment généré par $\Sigma$, c'est-à-dire que tout arbre de Tree peut être défini à l'aide d'un nombre fini d'opérations de la $\Sigma$-algèbre $\mathcal{T}$.

Notons également que l'opération $\mid^{\mathcal{T}}$ est associative et commutative sur les arbres et que $\mathbf{0}^{\mathcal{T}}$ est son élément neutre.

**arbres vs forêts** Nous considérons donc ici des arbres dont les arêtes portent les labels. Une présentation équivalente, et tout aussi adaptée, de nos travaux peut se faire comme suggéré dans [CGA+02] sur la base de forêts dont les nœuds portent les labels.

Ainsi, on pourrait considérer des forêts, *ie* des multiensembles d'arbres portant des labels sur leurs nœuds. Comme nous avons fait pour les arbres, on peut alors adopter un point de vue algébrique de construction de ces forêts au travers d'une algèbre $\mathcal{F}$ : $\mathbf{0}^{\mathcal{F}}$ sera la forêt (graphe) vide, $a^{\mathcal{F}}(f)$ ajoutera à la forêt $f$ un nouveau nœud étiqueté par $a$ et créera une arête entre ce nœud et les autres nœuds n'ayant pas d'arêtes entrantes dans $f$. Finalement, $f_1 \mid^{\mathcal{F}} f_2$ sera simplement l'union disjointe des deux forêts $f_1$ et $f_2$.

Les résultats que nous présenterons dans les chapitres suivants seront valables aussi bien pour une vision d'arbres que pour une vision de forêts.

## 7.3 Logiques et arbres

Nous présentons brièvement dans cette section la logique monadique du second-ordre MSO. Cette logique est l'extension de la logique du premier-ordre FO avec la possibilité de quantifier sur l'ensemble

des relations unaires (ou monadiques), *ie* sur les ensembles.

Soit $\sigma$ la signature $\{\mathsf{label}_a \mid a \in \mathcal{N}\}$ où les $\mathsf{label}_a$ sont des symboles de prédicat binaires. A un arbre $\tau = (V, E, \lambda)$, on associe une $\sigma$-structure finie $\mathcal{S}^\tau = \langle V, \{\mathsf{label}_a^\tau \mid a \in \mathcal{N}\}\rangle$, telle que $\mathsf{label}_a^\tau(v, v')$ est vrai dans $\mathcal{S}^\tau$ si $(v, v') \in E$ et $\lambda((v, v')) = a$

Nous supposons un ensemble dénombrable de variables du premier ordre notées $x, y, z, \ldots$ et un ensemble dénombrable de variables du second ordre notées $X, Y, Z, \ldots$.

**la logique MSO** Les formules de la logique monadique du second ordre sont définies par :

$$\psi ::= \mathsf{label}_a(x, y) \mid \overline{\mathsf{label}}_a(x, y) \mid x \in X \mid \psi \vee \psi \mid \neg\psi \mid \exists x.\psi \mid \exists X.\psi$$

Les formules de la logique du premier ordre FO correspondent aux formules de MSO sans quantification du second ordre $\exists X$.

Soit $\mathcal{S}$ une $\sigma$-structure de domaine $V$. Soit $\rho$ une valuation associant aux variables du premier ordre des éléments de $V$ et aux variables du second ordre des sous-ensembles de $V$. La structure $\mathcal{S}$ est modèle d'une formule MSO $\psi$ sous la valuation $\rho$ (définie notamment pour les variables libres de $\psi$), noté $\mathcal{S} \models_\rho \psi$, si :

- $\psi$ est $\mathsf{label}_a(x, y)$ et $\mathsf{label}_a(\rho(x), \rho(y))$ est vrai dans $\mathcal{S}$ ;
- $\psi$ est $\overline{\mathsf{label}}_a(x, y)$ et $\mathsf{label}_b(\rho(x), \rho(y)))$ est vrai dans $\mathcal{S}$ pour un label $b$ différent de $a$ ;
- $\psi$ est $x \in X$ et $\rho(x)$ appartient à $\rho(X)$ ;
- $\psi$ est $\psi_1 \vee \psi_2$ (resp. $\neg\psi'$) et $\mathcal{S} \models_\rho \psi_1$ ou $\mathcal{S} \models_\rho \psi_2$ (resp. $\mathcal{S} \not\models_\rho \psi'$) est vrai ;
- $\psi$ est $\exists x.\psi'$ et il existe un élément $v$ de $V$ tel que $\mathcal{S} \models_{\rho[x\to v]} \psi'$ est vrai.
- $\psi$ est $\exists X.\psi'$ et il existe un sous-ensemble $V'$ de $V$ tel que $\mathcal{S} \models_{\rho[X\to V']} \psi'$ est vrai.

Par abus de notation, nous écrirons $\tau \models \psi$ pour un arbre $\tau$ et une formule de MSO close si $\mathcal{S}^\tau \models \psi$ pour la $\sigma$-structure $\mathcal{S}^\tau$ induite par $\tau$ ; de plus, nous écrirons $[\![\psi]\!]$ pour représenter l'ensemble de tous les arbres $\tau$ tel que $\tau \models \psi$. Un ensemble d'arbres $T$ est *MSO-définissable* s'il existe une formule close de MSO $\psi$ telle que $[\![\psi]\!] = T$.

Notez que dans le cas d'un ensemble de labels $\mathcal{N}$ fini, le prédicat $\overline{\mathsf{label}}_a$ est définissable par une formule du premier ordre utilisant les prédicats $\mathsf{label}_b$.

Dans le cas des arbres ordonnés d'arité non bornée, la $\sigma$-structure de l'arbre se voit enrichie par l'interprétation d'un symbole représentant l'ordre total existant entre les fils d'un même nœud ; ceci peut se traduire par un ordre $\preceq$ tel que

- si $v \preceq v'$ est vrai dans $\mathcal{S}^\tau$ alors $v, v'$ sont fils d'un même nœud, *ie* il existe un nœud $u$ tel que $(u, v), (u, v') \in E$ ;
- $\preceq$ est un ordre total sur $\mathsf{children}(u)$ pour tout $u$ de $V$.

Une autre possibilité consiste à définir une relation next_sibling, telle que $\mathsf{next\_sibling}(u, v)$ est vrai si $v$ est le plus petit nœud strictement plus grand que $u$. La logique MSO pour les arbres ordonnés d'arité non bornée est alors augmentée d'une formule atomique permettant d'exprimer l'ordre entre les fils d'un nœud, comme $x \preceq y$ ou $\mathsf{next\_sibling}(x, y)$.

La présentation que nous avons faite des structures logiques associées aux arbres et de la logique MSO est "orientée nœud", puisque le domaine de la structure est l'ensemble des nœuds. Il est cependant possible de considérer une présentation où les objets considérés sont les arêtes. Dans ce cas, le domaine de la structure est $E$ et on considère une relation binaire succ telle que $\mathsf{succ}(e, e')$ est vrai si la destination de l'arête $e$ et la source de $e'$ coïncident. En ce qui concerne les labels des arêtes, ils sont définis par une relation monadique $\mathsf{label}_a(e)$ qui spécifie que l'arête $e$ est étiquetée par $a$. La logique MSO correspondante sera définie par les formules atomiques $\mathsf{label}_a(x)$, $\overline{\mathsf{label}}_a(x)$, et $\mathsf{succ}(x, y)$.

Le choix du type de structure considéré ("orientée nœud" ou "orientée arêtes") peut avoir une importance, comme par exemple dans le cas des graphes [20]. Dans le cas des arbres, Courcelle montre dans [Cou94] que l'expressivité pour la logique MSO est identique pour les deux types de structures.

**la logique CMSO** Courcelle propose dans [Cou90], une extension de la logique MSO, appelée "counting MSO" (CMSO). Cette extension consiste en un symbole de prédicat $Mod^i_j(X)$, où $X$ est une variable du second-ordre et $i, j$ sont deux entiers naturels tels que $0 \leq j < i$. La formule $Mod^i_j(X)$ est vraie dans une $\sigma$-structure $\mathcal{S}$ et une valuation $\rho$ associant à $X$ un sous-ensemble du domaine de $\mathcal{S}$ si la cardinalité de $\rho(X)$ modulo $i$ est égale à $j$.

Il convient de remarquer que dans le cas des arbres ordonnés d'arité non bornée, le prédicat $Mod^i_j(X)$ est définissable dans la logique MSO en utilisant l'ordre existant entre les frères. Ainsi, les expressivités de MSO et de CMSO coïncident pour les arbres ordonnés ce qui n'est pas le cas pour les arbres non-ordonnés.

**Theorem 9 ([Cou90])** *Pour les arbres non-ordonnés d'arité non bornée, la logique CMSO est strictement plus expressive que la logique MSO.*

**la logique PMSO** Seidl, Schwentick et Muscholl proposent dans [SSM03] une extension de MSO appelée "Presburger MSO" (PMSO). Cette extension est définie à l'aide d'un nouveau type de formules atomiques de la forme $x/\phi$, où $\phi$ est une formule de Presburger [21] définie sur un ensemble de variables contenant, en particulier, l'ensemble $\{\#X \mid X \text{ est une variable du second ordre}\}$.

La formule $x/\phi$ est vraie dans une $\sigma$-structure $\mathcal{S}$ sous une valuation $\rho$ si la valuation $\mu$ associant à chaque variable $\#X$ de $\phi$ le cardinal de l'ensemble $\rho(X) \cap \mathsf{children}(\rho(x))$ vérifie $(\mathbb{N}, +, =), \mu \models \phi$.

**Example 5** *La formule suivante de PMSO exprime que tous les nœuds de l'arbre ont autant d'arêtes sortantes étiquétées par un $a$ que par un $b$*

$$\forall x\ (\exists y\ \mathsf{label}_a(y,x) \iff x \in X_a) \wedge \forall x\ (\exists y\ \mathsf{label}_b(y,x) \iff x \in X_b)\ \wedge\ \forall z\ z/(\#X_a = \#X_b)$$

Notez que la logique PMSO permet d'exprimer des contraintes très riches sur les cardinalités des ensembles. Cependant, ces ensembles sont toujours "relativisés" à un nœud précis. Pour des ensembles plus généraux, la théorie monadique du second ordre associée serait indécidable [KR03].

La notion d'ensembles d'arbres *CMSO-définissables* et *PMSO-définissables* est définie comme pour la logique MSO.

**Proposition 19 ([18])** *Pour les arbres non-ordonnés d'arité non bornée, la logique PMSO est strictement plus expressive que la logique CMSO.*

Ainsi, MSO, CMSO et PMSO forment une hiérarchie stricte de logiques en terme d'expressivité pour les arbres non-ordonnés d'arité non bornée.

Mentionnons pour terminer cette section que les logiques modales peuvent être utilisées pour décrire des ensembles d'arbres ou servir de langages de requêtes. L'expressivité de PDL dans les arbres finis est étudiée dans [ABD+05], celle du $\mu$-calcul et de CTL$^*$ dans [BL05].

Une récente compilation sur l'expressivité et la complexité de différentes logiques pour les arbres d'arité non bornée a été écrite par Libkin [Lib05].

[20] Une logique MSO basée sur les arêtes permet d'exprimer l'existence d'un cycle hamiltonien dans un graphe, ce que ne permet pas de faire une logique orientée nœud.

[21] On rappelle que la théorie de Presburger est la théorie du premier ordre sur l'ensemble des entiers naturels équipé de l'addition et du prédicat d'égalité, *ie* $(\mathbb{N}, +, =)$.

## 7.4 Logiques et automates

Nous présentons ici les automates d'arbres qui ont été utiles à la preuve de certains de nos résultats, présentés dans les chapitres suivants. Ces automates étendent les automates d'arbres classiques [CDG+97] puisqu'adaptés aux arbres non-ordonnés d'arité non bornée. De plus, ils permettent de considérer un ensemble infini de symboles.

Les automates que nous considérons ici seront proches des automates de Presburger de [SSM03], des automates à faisceaux (*sheaves automata*) de [DZLM04] et de ceux pour multi-arbres de [Lug05].

Soit $\mathbb{N}$, l'ensemble des entiers naturels. On notera $\mathbb{N}^k$, l'ensemble des vecteurs d'entiers naturels de taille $k$ . Pour un ensemble $S$, nous notons $\wp^{c/f}(S)$ l'ensemble des parties finies ou co-finies de $S$.

Un *automate d'arbres à contraintes arithmétiques* sera donné par un quadruplet $(\mathcal{N}, Q, F, \Delta)$ où $\mathcal{N}$ est un ensemble dénombrable de labels (d'arêtes), $Q$ est un ensemble fini d'états qu'on considérera implicitement ordonné $(q_1, \ldots, q_n)$, $F \subseteq \mathbb{N}^n$ est la condition d'acceptance et $\Delta$ est un ensemble fini de transitions de la forme $(q, \alpha, N)$ où $q \in Q$, $\alpha \in \wp^{c/f}(\mathcal{N})$ et $N \subseteq \mathbb{N}^n$.

Soit $s$ une séquence d'états de $Q$ ; on note $\pi(s)$, la *projection de Parikh* de la séquence $s$ [Par66], c'est-à-dire l'élément de $\mathbb{N}^n$, $(\#_{q_1}, \ldots, \#_{q_n})$ où $\#_{q_i}$ est le nombre de $q_i$ dans $s$.

Pour une arête $e$ d'un arbre $\tau$, on notera $\mathsf{Succ}(e)$ l'ensemble des arêtes $e'$ ayant pour origine l'extrémité de $e$.

Un calcul $r$ d'un automate à contraintes arithmétiques sur un arbre $\tau = (V, E, \lambda)$ est une application de $E$ dans $Q$ satisfaisant la propriété suivante : pour toute arête $e$, on considère le mot $s_e$ obtenu en ordonnant de manière arbitraire les éléments de $\{r(e') \mid e' \in \mathsf{Succ}(e)\}$. On a $r(e) = q$ si et seulement s'il existe une règle de transition $(q, \alpha, N)$ telle que $\lambda(e) \in \alpha$ et $\pi(s_e) \in N$.

Un calcul est *acceptant* si le mot d'états $s$ étiquetant les arêtes ayant pour origine la racine vérifie $\pi(s) \in F$. Le langage d'un automate $A$, noté $L(A)$, est l'ensemble des arbres admettant un calcul acceptant pour l'automate $A$.

La définition des automates que nous venons de donner est totalement générique puisque les transitions y sont définies de manière abstraite. Nous allons instancier cette définition par trois descriptions possibles des ensembles de vecteurs d'entiers se trouvant dans les transitions.

On rappelle qu'un ensemble $N$ de vecteurs sur $\mathbb{N}^n$ est un ensemble linéaire s'il existe $b$ un vecteur de $\mathbb{N}^n$, appelé *base* et $p_1, \ldots, p_k$ un nombre fini $k$ de vecteurs de $\mathbb{N}^n$, appelés *périodes* tels que

$$S = \{\, b + \sum_{i=1}^{k} (\lambda_i \times p_i) \mid \lambda_1, \ldots, \lambda_k \in \mathbb{N} \,\}.$$

On écrira dans ce cas $S = \mathsf{Lin}(b, p_1, \ldots, p_k)$.

**Definition 1** *Un ensemble $S$ de vecteurs de $\mathbb{N}^n$ est dit :*
- semi-linéaire *s'il est l'union finie d'ensembles linéaires.*
- reconnaissable *s'il est l'union finie d'ensembles linéaires dont les périodes sont le produit par un scalaire de vecteurs unitaires (selon [Cou96]).*
- sans étoile *s'il est l'union finie d'ensembles linéaires dont les périodes sont des vecteurs unitaires (selon [GG99])*

Un automate à contraintes arithmétiques sera dit *semi-linéaire* (resp. *reconnaissable*, *sans étoile*) si la condition d'acceptance $F$ et, pour toute transition $(q, \alpha, N)$ de l'automate, l'ensemble $N$ est un ensemble semi-linéaire (resp. reconnaissable, sans étoile).

Nous dirons qu'un automate à contraintes semi-linéaires (reconnaissables ou sans étoile) est effectif si pour toute transition $(q, \alpha, N)$ de l'automate, les ensembles $N$ et la condition d'acceptance $F$ sont

donnés sous une forme explicite comme des ensembles finis de base et périodes ou comme une formule de Presburger.

On sait depuis les travaux de Büchi [Büc60] pour les structures linéaires (*ie* les mots) et ceux de Thacher et Wright [TW68] et de Rabin [Rab69] pour les structures arborescentes que la logique monadique du second ordre est intimement liée aux automates ; ceci est toujours le cas pour les logiques et les automates que nous avons considérés.

**Proposition 20 ([SSM03])** *Un ensemble d'arbres $L$ est PMSO-définissable ssi il existe un automate à contraintes semi-linéaires $A$ tel que $L = L(A)$.*

**Proposition 21 ([Cou96, 18])** *Un ensemble d'arbres $L$ est CMSO-définissable ssi il existe un automate à contraintes reconnaissables $A$ tel que $L = L(A)$.*

**Proposition 22 ([18])** *Un ensemble d'arbres $L$ est MSO-définissable ssi il existe un automate à contraintes sans étoile $A$ tel que $L = L(A)$.*

La preuve de ces trois résultats (propositions 20, 21 et 22) est constructive dans le sens où, donnée une formule close, il existe un algorithme pour construire un automate effectif reconnaissant les modèles de la formule.

Pour terminer, mentionnons que le problème du vide des automates à contraintes semi-linéaires (et donc, à contraintes reconnaissables ou sans étoile) est décidable. Ainsi, du fait de l'aspect constructif de l'approche, on a le théorème suivant :

**Theorem 10 ([SSM03])** *Le problème de satisfiabilité des formules closes de PMSO est décidable.*

Ceci est bien sûr également vrai pour les logiques MSO et CMSO.

# Chapitre 8

# Expressivité et satisfiabilité pour la logique TQL

Ce chapitre présente des travaux réalisés dans le cadre de la thèse de Iovka Boneva co-encadrée par Sophie Tison et moi-même. Ils ont été publiés dans [19], publication qui se trouve en annexe G.

## 8.1 La logique TQL : syntaxe et sémantique

Nous donnons ici la syntaxe et la sémantique de la logique d'arbres proposée dans [CG01a]. Comme nous l'avons évoqué, la présentation de la sémantique que nous faisons ici est légèrement différente de celle de [CG01a].

**syntaxe** Nous supposons un ensemble dénombrable de labels $\mathcal{N}$ et considérons Tree l'ensemble des arbres non-ordonnés d'arité non bornée dont les arêtes sont étiquetées par des éléments de $\mathcal{N}$.

Nous considérons un ensemble dénombrable $\mathcal{L}$ de variables de labels notées $x, y, z$, un ensemble dénombrable $\mathcal{X}$ de variables d'arbres notées $X, Y, Z$, de même qu'un ensemble dénombrable $\mathcal{R}$ de variables de récursion notées $\xi$. La syntaxe de la logique TQL est donnée à la figure 8.1, où $\eta$ désigne soit un label $a$ de $\mathcal{N}$ ou une variable de label $x$ de $\mathcal{L}$.

Pour garantir l'existence d'un plus grand point fixe $\nu\xi.\varphi$, nous demandons que chaque occurrence de la variable de récursion $\xi$ apparaisse sous un nombre pair de négations dans la formule $\varphi$.

**sémantique** Nous considérons les valuations $\rho$ et $\delta$ définies sur un sous-ensemble fini de $\mathcal{L} \cup \mathcal{X}$ et de $\mathcal{R}$ respectivement. Les valuations $\rho$ associent des labels aux variables de labels (*ie* de $\mathcal{L}$) et des arbres aux variables d'arbres (*ie* de $\mathcal{X}$). Les valuations $\delta$ associent aux variables de récursion des sous-ensembles de Tree.

Nous dirons qu'une formule $\varphi$ est $\rho$-close (resp. $\delta$-close) si les variables de labels et d'arbres (resp. de récursion) libres dans $\varphi$ sont incluses dans le domaine de $\rho$ (resp. de $\delta$).

L'interprétation d'une formule $\varphi$ est donnée par une application $[\![\varphi]\!]_{\rho,\delta}$ qui associe à $\varphi$, un sous-ensemble de Tree. Cette application est paramétrée par deux valuations $\rho$ et $\delta$ telles que $\varphi$ est $\rho$-close et $\delta$-close.

Nous supposons que la valuation $\rho$ est étendue aux labels en posant $\rho(a) = a$ pour tout $a$ de $\mathcal{N}$. La valuation $\rho[x \mapsto n]$ est identique à $\rho$ à l'exception de la variable $x$ à laquelle on associe $n$ ; les valuations $\rho[X \mapsto A]$ et $\delta[\xi \mapsto S]$ sont définies de manière identique. Pour un ensemble d'arbres $S$, on définit récursivement $S^i$ par $\{\mathbf{0}^{\mathcal{T}}\}$ pour $i = 0$ et $S^i = \{\tau \mid^{\mathcal{T}} \tau' \mid \tau \in S, \tau' \in S^{i-1}\}$, pour $1 \leq i$.

| $\varphi, \psi ::=$ | |
|---|---|
| $\mathbf{0}$ | arbre vide |
| $\eta[\varphi]$ | extension |
| $\varphi \mid \psi$ | composition |
| $\top$ | vrai |
| $\neg\varphi$ | négation |
| $\varphi \vee \psi$ | disjonction |
| $\exists x.\varphi$ | quantification sur les labels |
| $X$ | variable d'arbre |
| $\exists X.\varphi$ | quantification sur les arbres |
| $\xi$ | variable de récursion |
| $\nu\xi.\varphi$ | plus grand point fixe |
| $\eta = \eta'$ | égalité de labels |
| $\varphi^*$ | itération |

FIG. 8.1 – Syntaxe de la logique TQL

La figure 8.2 décrit l'interprétation d'une formule $\varphi$ sous les valuations $\rho$ et $\delta$.

Nous introduirons une notation de plus petit point fixe $\mu\xi.\varphi$ définie comme $\neg(\nu\xi.\neg\varphi\langle\xi \leftarrow \neg\xi\rangle)$.

Notons que l'itération $\varphi^*$ peut paraître redondante, puisque équivalente à $\mu\xi.(\varphi \mid \xi) \vee \mathbf{0}$, où $\xi$ est une variable de récursion n'apparaissant pas (libre) dans $\varphi$. Cependant, nous définirons des fragments de la logique où la construction de point-fixe sera restreinte et ne permettra pas de redéfinir l'itération.

**Definition 2 (Modèle)** *Soient $\rho$, $\delta$ deux valuations et $\varphi$ une formule de* TQL *$\rho$-close et $\delta$-close. Un arbre $\tau$ satisfait (est modèle de) la formule $\varphi$ sous les valuations $\rho$ et $\delta$ (noté $\tau \models_{\rho,\delta} \varphi$) si $\tau$ appartient à $[\![\varphi]\!]_{\rho,\delta}$.*

Nous donnons maintenant quelques exemples de propriétés exprimables dans la logique TQL.

**Example 6** *La première formule exprime que seules deux arêtes partent de la racine, une étiquetée par $a$ et l'autre par $b$*

$$a[\top] \mid b[\top]$$

*La seconde exprime qu'au moins deux arêtes, l'une étiquetée par $a$ et l'autre par $b$, partent de la racine,*

$$a[\top] \mid b[\top] \mid \top$$

*La formule suivante exprime le fait que les arêtes partant de la racine de l'arbre sont étiquetées sur l'alphabet $\{a, b\}$*

$$\mu\xi.((a[\top] \vee b[\top]) \mid \xi) \vee \mathbf{0}$$

*Ce type de propriétés est souvent appelée "horizontale". Cependant, la logique peut également exprimer des propriétés de chemin dites "verticales" : la formule suivante exprime le fait que l'arbre possède un chemin dont les étiquettes ne sont que des labels $a$*

$$\mu\xi.\mathbf{0} \vee (a[\xi] \mid \top)$$

$$
\begin{array}{lcl}
[\![\mathbf{0}]\!]_{\rho,\delta} & = & \{\mathbf{0}^{\mathcal{T}}\} \\
[\![\eta[\varphi]]\!]_{\rho,\delta} & = & \{\rho(\eta)^{\mathcal{T}}(\tau) \mid \tau \in [\![\varphi]\!]_{\rho,\delta}\} \\
[\![\top]\!]_{\rho,\delta} & = & \mathsf{Tree} \\
[\![\varphi \mid \psi]\!]_{\rho,\delta} & = & \{\tau \mid^{\mathcal{T}} \tau' \mid \tau \in [\![\varphi]\!]_{\rho,\delta}, \tau' \in [\![\psi]\!]_{\rho,\delta}\} \\
[\![\neg\varphi]\!]_{\rho,\delta} & = & \mathsf{Tree} \setminus [\![\varphi]\!]_{\rho,\delta} \\
[\![\varphi \vee \psi]\!]_{\rho,\delta} & = & [\![\varphi]\!]_{\rho,\delta} \cup [\![\psi]\!]_{\rho,\delta} \\
[\![\exists x.\varphi]\!]_{\rho,\delta} & = & \bigcup_{n\in\Lambda} [\![\varphi]\!]_{\rho[x\mapsto n],\delta} \\
[\![\eta = \eta']\!]_{\rho,\delta} & = & \mathsf{Tree} \text{ si } \rho(\eta) = \rho(\eta'), \varnothing \text{ sinon} \\
[\![X]\!]_{\rho,\delta} & = & \{\rho(X)\} \\
[\![\exists X.\varphi]\!]_{\rho,\delta} & = & \bigcup_{A\in\mathsf{Tree}} [\![\varphi]\!]_{\rho[X\mapsto A],\delta} \\
[\![\xi]\!]_{\rho,\delta} & = & \delta(\xi) \\
[\![\nu\xi.\varphi]\!]_{\rho,\delta} & = & \bigcup\{S \subseteq \mathsf{Tree} \mid S \subseteq [\![\varphi]\!]_{\rho,\delta[\xi\mapsto S]}\} \\
[\![\varphi^*]\!]_{\rho,\delta} & = & \bigcup_{i\in\mathbb{N}} ([\![\varphi]\!]_{\rho,\delta})^i
\end{array}
$$

FIG. 8.2 – Sémantique de la logique TQL

*Propriétés horizontale et verticale peuvent se combiner : la formule suivante exprime le fait que les arêtes de l'arbre sont étiquetées sur l'alphabet* $\{a, b\}$

$$\mu\xi.((a[\xi] \vee b[\xi]) \mid \xi \ \vee \ \mathbf{0})$$

*La logique* TQL *peut aussi exprimer des propriétés numériques ; la formule suivante stipule que tous les nœuds ont autant d'arêtes sortantes étiquetées par* $a$ *que par* $b$*.*

$$\mu\xi.((a[\xi] \mid b[\xi]) \mid \xi \ \vee \ (\neg(a[\top] \mid \top) \wedge \neg(b[\top] \mid \top) \wedge \neg(\exists x\ x[\neg\xi] \mid \top)) \vee \mathbf{0})$$

*On peut également exprimer des propriétés numériques plus globales ; la formule suivante est vraie pour les arbres dont le nombre d'arêtes est pair*

$$\mu\xi_p.((\exists x\ x[\xi_p] \mid impair) \vee (\exists x\ x[impair] \mid \xi_p) \vee \mathbf{0})$$

*où la sous-formule* $impair$ *est définie comme*

$$impair \stackrel{def}{=} \mu\xi_i.(\exists x\ x[\xi_p] \mid \xi_p) \vee (\exists x\ x[\xi_i] \mid \xi_i))$$

*La formule* $suivant(a, b)$ *spécifie que le document contient une arête* $b$ *se trouvant à l'extrémité d'une arête étiquetée par* $a$

$$suivant(a, b) \stackrel{def}{=} \mu\xi.((a[b[\top] \mid \top] \mid \top) \vee (\exists x\ x[\xi] \mid \top))$$

*La formule* $present(n)$ *est vraie si le label* $n$ *apparaît dans l'arbre,*

$$present(n) \stackrel{def}{=} \mu\xi((n[\top] \mid \top) \vee \exists x(x[\xi] \mid \top))$$

*Nous pouvons également spécifier qu'un nom* $n$ *est unique dans l'arbre (pouvant ainsi servir de clé dans le document) ; la formule* $unique(n)$ *est donnée par*

$$unique(n) \stackrel{def}{=} \mu\xi(n[\neg present(n)] \mid \neg present(n)) \vee (\exists x\ \neg(x = n) \wedge (x[\xi] \mid \neg present(n)))$$

*Enfin, nous spécifions que les labels étiquetant les arêtes à l'extrémité des arêtes $id$ sont uniques dans le document,*

$$\forall x.suivant(id, x) \Longrightarrow unique(x)$$

*Pour terminer, nous exprimons qu'un arbre possède deux arêtes partant de la racine étiquetée par $a$ et $b$ respectivement et que les deux sous-arbres atteints par ces deux arêtes ne sont pas isomorphes,*

$$\exists X\ a[X] \mid b[\neg X]$$

**fragments de la logique** TQL Nous définissons ici quelques fragments de la logique TQL en restreignant de diverses façons la syntaxe donnée à la figure 8.1. Ces fragments seront utilisés dans ce chapitre pour étudier finement les problèmes de satisfiabilité et d'expressivité de la logique. Ils permettront également dans le chapitre suivant de préciser les complexités pour le problème de model-checking.

Nous noterons :

- $\mathrm{TQL}^{-\exists}$, le fragment de TQL sans quantification sur les labels ou sur les arbres [22] ($\exists x, \exists X$).
- $\mathrm{TQL}^{-\nu,*}$, le fragment de la logique ne comportant pas d'opérateur de point-fixe ($\nu, \mu$), ni variable de récursion, ni itération ($*$).
- $\mathrm{TQL}^{-\exists,\nu,*}$ sera obtenu comme l'intersection de $\mathrm{TQL}^{-\exists}$ et de $\mathrm{TQL}^{-\nu,*}$.

Nous dirons qu'une formule $\varphi$ est *à récursion gardée* si pour toute sous-formule de $\varphi$ de la forme $\nu\xi.\psi$, toutes les occurrences de la variable $\xi$ apparaissent dans $\psi$ dans la portée d'un opérateur d'extension $\eta[\,]$.

Nous définissons $\mathrm{TQL}^{-\exists}_{g\nu}$ comme le fragment de TQL sans quantification et à récursion gardée et $\mathrm{TQL}^{-\exists,*}_{g\nu}$ comme la restriction de $\mathrm{TQL}^{-\exists}_{g\nu}$ aux formules sans itération[23].

## 8.2 Expressivité de la logique TQL

Nous rapportons dans cette section les résultats que nous avons obtenus sur l'expressivité de la logique TQL et divers fragments. Dans un premier temps, nous évoquerons l'expressivité en termes d'ensembles d'arbres descriptibles par des formules closes. Dans un second temps, nous discuterons de l'expressivité en ce qui concerne les requêtes monadiques.

### 8.2.1 Expressivité pour les ensembles d'arbres

Nous avons donné à la section précédente quelques exemples de propriétés exprimables avec la logique TQL. Nous allons, ici, tenter de comparer cette logique avec la logique monadique du second ordre ou son extension PMSO. Nous allons donc considérer les fragments $\mathrm{TQL}^{-\nu,*}$, $\mathrm{TQL}^{-\exists}$, $\mathrm{TQL}^{-\exists}_{g\nu}$ et $\mathrm{TQL}^{-\exists,*}_{g\nu}$.

**le fragment** $\mathrm{TQL}^{-\nu,*}$ Pour comparer l'expressivité de $\mathrm{TQL}^{-\nu,*}$ et de PMSO, nous utilisons le fait que les modèles d'une formules close de PMSO peuvent se représenter comme le langage reconnu par un automate à contraintes semi-linéaires.

Considérons la formule $distinct$ qui spécifie le fait que les labels étiquetant les arêtes partant de la racine sont tous deux à deux distincts

$$distinct \stackrel{def}{=} \neg(\exists x\ x[\top] \mid x[\top] \mid \top)$$

[22] Notez que pour les formules closes de $\mathrm{TQL}^{-\exists}$ (ou sans variables de labels), le test d'égalité entre labels est soit trivialement vrai, soit trivialement faux ; c'est pourquoi nous le supposerons absent de $\mathrm{TQL}^{-\exists}$.

[23] Notez que la récursion gardée ne permet pas de définir l'itération comme nous l'avons définie précédemment.

On peut alors montrer que :

**Proposition 23** *Il n'existe pas de formule de PMSO dont les modèles sont précisément les arbres satisfaisant* $distinct$.

Il est bien évident que dans le cas d'un ensemble fini de labels, la propriété $distinct$ est exprimable dans la logique PMSO. Cependant, même dans ce cas, les variables d'arbres de $\mathrm{TQL}^{-\nu,*}$ permettent d'exprimer des propriétés qui ne sont pas PMSO-définissables. Ainsi,

**Proposition 24** *Il n'existe pas de formule close de PMSO équivalente à la formule* $\exists X\ a[X] \mid b[X]$ *de* $\mathrm{TQL}^{-\nu,*}$.

Ceci traduit le fait que, tout comme pour la logique MSO, l'isomorphisme d'arbres ne peut être exprimé dans la logique PMSO [24].

Nous conjecturons qu'il existe au moins une formule de PMSO pour laquelle il n'existe pas de formule de $\mathrm{TQL}^{-\nu,*}$ équivalente, faisant des logiques PMSO et $\mathrm{TQL}^{-\nu,*}$ des logiques incomparables sur le plan de leur expressivité pour décrire des ensembles d'arbres.

Nous pensons qu'une formule énonçant que le nombre d'arêtes de l'arbre est pair n'est pas exprimable dans $\mathrm{TQL}^{-\nu,*}$.

**le fragment** $\mathrm{TQL}^{-\exists}$ Une nouvelle fois, nous utilisons le lien entre PMSO et les automates à contraintes semi-linéaires pour comparer l'expressivité de $\mathrm{TQL}^{-\exists}$ et de PMSO.

Pour un automate à contraintes semi-linéaires donné, il est très simple d'écrire une formule de $\mathrm{TQL}^{-\exists}$ dont l'ensemble des modèles sera précisément le langage de l'automate ; ceci montre que $\mathrm{TQL}^{-\exists}$ est au moins aussi expressive que PMSO.

Finalement, il est possible d'exhiber une formule close de $\mathrm{TQL}^{-\exists}$ dont l'ensemble des modèles ne peut être représenté comme le langage d'un automate à contraintes semi-linéaires, et donc, comme l'ensemble des modèles d'une formule close de PMSO. Cette formule est celle permettant la preuve du théorème 13 de la section suivante ; elle est donnée dans [19], qui se trouve en annexe G.

**les fragments** $\mathrm{TQL}^{-\exists}_{g\nu}$ **et** $\mathrm{TQL}^{-\exists,*}_{g\nu}$ Nous pouvons montrer tout d'abord qu'on peut construire pour tout automate à contraintes semi-linéaires (resp. à contraintes sans étoile), une formule close de $\mathrm{TQL}^{-\exists}_{g\nu}$ (resp. de $\mathrm{TQL}^{-\exists,*}_{g\nu}$) dont les modèles sont exactement le langage reconnu par l'automate.

Dans un second temps, suivant la démarche de Thatcher et Wright, nous avons proposé un algorithme qui, donnée une formule de $\mathrm{TQL}^{-\exists}$, produit un automate à contraintes arithmétiques. Les ensembles de vecteurs d'entiers des transitions sont donnés sous la forme de systèmes d'équations de point-fixes [AN01]. Dans le cas de la logique $\mathrm{TQL}^{-\exists}_{g\nu}$ (resp. de $\mathrm{TQL}^{-\exists,*}_{g\nu}$), ces systèmes d'équations sont tels que leur solution est un ensemble semi-linéaires (resp. un ensemble sans étoile), ce qui nous permet de conclure que :

**Theorem 11 ([19])** *Concernant les formules closes, les logiques* $\mathrm{TQL}^{-\exists}_{g\nu}$ *et PMSO (resp.* $\mathrm{TQL}^{-\exists,*}_{g\nu}$ *et MSO) ont la même expressivité.*

Le résultat est en réalité un peu plus fort puisque, non seulement les solutions des systèmes d'équations sont des ensembles semi-linéaires dans le cas de $\mathrm{TQL}^{-\exists}_{g\nu}$ et des ensembles sans étoile dans le cas de $\mathrm{TQL}^{-\exists,*}_{g\nu}$, mais de plus, ces systèmes peuvent être résolus produisant des automates effectifs.

[24] La preuve de ce fait peut être réalisée, comme pour MSO, à l'aide de la correspondance entre logique et automate d'arbres.

### 8.2.2 Expressivité pour les requêtes monadiques

Dans la section précédente, nous avons comparé différentes logiques d'un point de vue de leurs formules closes ; nous avons donc comparé des logiques sur leur capacité à exprimer des propriétés ou des contraintes sur des arbres (des documents).

Nous allons évoquer ici la possibilité d'utiliser les formules pour définir des requêtes et utiliser pour cela l'existence de possibles variables libres dans les formules. Cependant, nous allons devoir faire attention car les domaines d'interprétation des logiques sont différents. De plus, nous restreignons notre propos aux requêtes monadiques, c'est-à-dire ne possédant qu'une variable libre.

Considérons les formules de PMSO possédant une variable libre du premier ordre. Une telle formule définit une requête monadique dans le sens suivant : pour un arbre fixé, ce sont les nœuds de cet arbre qui serviront à instancier la variables libre et certains d'entre eux seront donc réponse à la requête.

Pour la logique TQL, il y a deux types de variables ; si la variable libre est une variable de labels, alors les valuations de labels servant à l'instanciation de cette variable seront les réponses de la requête. Dans le cas d'une variable d'arbres, la réponse à une requête sera un ensemble d'arbres, puisque l'interprétation des variables d'arbres a pour domaine l'ensemble des arbres dans son intégralité.

Nous allons évoquer ici une comparaison entre des formules de TQL et de PMSO non closes en comparant ce qui est comparable. Signalons tout d'abord qu'il ne peut y avoir dans PMSO de correspondance aux "requêtes" monadiques de TQL ayant une variable libre de labels ; considérons donc des formules de TQL avec une variable d'arbres libre. Nous supposerons également implicitement que ces formules sont en fait des formules de $\mathrm{TQL}^{-\exists}$ augmentées d'une variable d'arbres libre.

Il existe une correspondance évidente entre nœud et arbre puisque qu'un nœud détermine de manière unique un sous-arbre de l'arbre considéré, l'inverse n'étant pas vrai puisque un arbre peut être sous-arbre d'un autre en plusieurs positions, représentant ainsi plusieurs nœuds. De plus, pour la logique TQL, l'arbre instanciant la variable libre n'est pas nécessairement un sous-arbre de l'arbre considéré et cela pour deux raisons :

- du fait de la négation ; tout arbre $\tau'$, différent de l'arbre $\tau$ sur lequel on pose la requête, sera solution de $\neg X$
- du fait de l'opérateur de composition ; si on considère la formule $\varphi = a[\top] \mid X$ comme une requête appliquée à un terme $\tau$, alors une instanciation de $X$ faisant de $\tau$ un modèle de $\varphi$ ne pourra se faire avec un sous-arbre de $\tau$.

Dans les deux cas, les réponses à la requête ne pourront être identifiées par un nœud de l'arbre et donc, correspondre à une formule de PMSO monadique. Notez par ailleurs que l'ensemble des réponses à la requête $\neg X$ est infini pour tout arbre (fini) $\tau$.

Même si la variable d'arbres se trouve instanciée par un sous-arbre, la requête TQL peut malgré tout être non représentable en PMSO. Ainsi, la formule $a[X] \mid b[X]$ correspond à une requête où $X$ sera bien instancié par un sous-arbre mais avec une condition d'isomorphisme qui ne peut être définie dans PMSO. Cette condition vient ici du fait que la variable $X$ se trouve à gauche et à droite d'un opérateur de composition, mais ce n'est pas le cas dans la formule

$$(a[X] \mid \top) \wedge (b[X] \mid \top) \wedge \neg(\neg\mathbf{0} \mid \neg\mathbf{0} \mid \neg\mathbf{0})$$

qui lui est équivalente.

Dans ce dernier exemple, c'est le fait d'avoir une variable identique à droite et à gauche d'une conjonction, créant ainsi une "jointure", qui est problématique ; on pourrait alors adopter la restriction de CDuce [BCF03] qui stipule que les variables à gauche et à droite d'une conjonction doivent être différentes.

Cependant, même si cette condition est vérifiée, l'utilisation de point-fixes reste, quant à elle, problématique. Ainsi, il ne peut y avoir de formule de PMSO correspondant à la formule

$$\mu\xi.(a[X] \mid b[\xi] \vee \mathbf{0})$$

Les arbres décrits par cette formule sont des peignes dont les corps sont constitués d'une suite d'arêtes $b$ et les dents d'arêtes $a$. De plus, les arbres présents au bout des dents sont tous identiques.

Remarquons néanmoins que le "dépliage" du point-fixe laisse apparaître la présence de deux occurrences de $X$ séparées par une composition

$$\mathbf{0} \vee a[X] \mid b[\mu\xi.(a[X] \mid b[\xi] \vee \mathbf{0})]$$

Nous laissons comme travail futur la description d'un fragment syntaxique de TQL qui correspondrait aux requêtes monadiques de PMSO. De même, on pourrait étudier la décidabilité du problème de savoir si une formule de TQL (ou plus simplement, de $\mathrm{TQL}^{-\exists}_{g\nu}$ augmentée par des variables d'arbres libres) est "équivalente" à une formule (monadique) de PMSO.

## 8.3 Satisfiabilité pour la logique TQL

Dans cette section, nous allons nous intéresser au problème de satisfiabilité, c'est-à-dire, donnée une formule close $\varphi$, décider s'il existe un arbre $\tau$ tel que $\tau \models \varphi$.

En utilisant le résultat de [12] concernant le fragment statique du calcul des ambients et son lien avec les arbres non-ordonnés d'arité non bornée, on a :

**Theorem 12 ([12])** *Le problème de satisfiabilité est indécidable pour le fragment* $\mathrm{TQL}^{-\nu,*}$.

La preuve de ce résultat est établie dans [12] pour une logique contenant les opérations booléennes, les connecteurs spatiaux d'extension et de composition et la quantification sur les labels. Le résultat est néanmoins préservé si cette dernière est remplacée par une quantification sur les arbres.

On pourrait penser que la présence de quantifications est la source d'indécidabilité de la logique TQL. Ainsi, on pourrait espérer la décidabilité en l'absence de quantifications ou avec une quantification sur les labels mais pour un ensemble fini de labels. Malheureusement, ce n'est pas le cas : en adaptant la preuve de Verma sur l'indécidabilité des automates alternants modulo $AC$ [GLV02], on prouve que :

**Theorem 13 ([19])** *Le problème de satisfiabilité est indécidable pour le fragment* $\mathrm{TQL}^{-\exists}$.

Nous rappelons que nous pouvons construire algorithmiquement à partir d'une formule close de $\mathrm{TQL}^{-\exists}$ un automate à contraintes arithmétiques dont le langage accepté est précisément l'ensemble des modèles de la formule. Nous avons évoqué le fait que les contraintes arithmétiques des transitions sont données sous la forme de systèmes d'équations et que dans le cas de $\mathrm{TQL}^{-\exists}_{g\nu}$ (resp. de $\mathrm{TQL}^{-\exists,*}_{g\nu}$), ces systèmes peuvent être résolus, donnant ainsi des automates à contraintes semi-linéaires (resp. sans étoile) effectifs. Le problème du vide étant décidable pour ces classes d'automates effectifs, on a alors

**Theorem 14 ([19])** *Le problème de satisfiabilité est décidable pour les formules closes de* $\mathrm{TQL}^{-\exists}_{g\nu}$ *(et donc, de* $\mathrm{TQL}^{-\exists,*}_{g\nu}$*).*

Il est bien connu que le problème de satisfiabilité de MSO (et donc, de PMSO) est non-élémentaire. Contrairement à MSO, la satisfiabilité de $\mathrm{TQL}^{-\exists}_{g\nu}$ est, elle, élémentaire puisqu'elle peut être obtenue comme une "combinaison d'algorithmes" qui sont tous élémentaires : l'automate à contraintes semi-linéaires construit à partir de la formule possède un nombre exponentiel d'états (dans la taille de la

formule) et le système d'équations est, lui, linéaire dans la taille de la formule. Résoudre ce système revient dans un premier temps à extraire une représentation base-périodes de la clôture commutative du langage d'une grammaire algébrique. Dans un second temps, il est nécessaire de calculer des opérations booléennes sur les semi-linéaires obtenus et de tester la vacuité du résultat. Toutes ces opérations peuvent se réaliser de manière élémentaire. Une borne plus précise de cette complexité reste à calculer.

Le problème de satisfiabilité est important dans le cadre des bases de données puisqu'il permet de tester la subsomption (ou implication) de requêtes ; il conviendra d'étudier la décidabilité de ce problème pour la logique $\mathrm{TQL}^{-\exists}$ augmentée de variables libres de labels et/ou d'arbres. Ceci n'est pas immédiat puisqu'en présence d'un ensemble infini de valeurs (comme notre ensemble de noms $\mathcal{N}$), les problèmes liés à la satisfiabilité deviennent vite indécidables même dans le cas où seuls des tests d'égalité sont effectués entre ces valeurs [NSV01, BDM$^+$05].

# Chapitre 9

# Model-checking et requêtes pour la logique TQL

Nous décrivons ici des travaux réalisés dans le cadre de la thèse de Iovka Boneva co-encadrée par Sophie Tison et moi-même. Ils ont été publiés dans [17]. Cette publication se trouve en annexe F.

## 9.1 Le model-checking de TQL

### 9.1.1 Un algorithme pour le problème de model-checking de TQL

Nous nous intéressons ici au problème de model-checking pour la logique TQL. Ce problème se formule de la manière suivante : étant donnés un arbre $\tau$, une formule $\varphi$ ne comportant pas de variables de récursion libres et une valuation $\rho$ pour les variables libres de labels et d'arbres, est-ce que $\tau \models_\rho \varphi$ ?

En adaptant l'algorithme de model-checking local proposé dans [SW91] et en adoptant le contrôle d'arrêt de [Win91], nous montrons que

**Theorem 15 ([17])** *Le problème de model-checking de la logique* TQL *est dans PSPACE.*

### 9.1.2 Complexité combinée pour le problème de model-checking de TQL

La borne supérieure de complexité obtenue à la section précédente est en fait optimale. Le résultat de [13], mentionné à la section 2.2.2, nous permet d'affirmer que

**Theorem 16** *Le problème de model-checking est PSPACE-dur pour le fragment de* TQL *ne contenant que les opérations booléennes, la quantification sur les labels et le test d'égalité.*

Quantifications et égalités ne sont pas la seule source de difficulté dans le problème de model-checking, puisque par réduction du problème de validité des formules booléennes quantifiées,

**Theorem 17 ([17])** *Le problème de model-checking de la logique* $\mathrm{TQL}^{-\exists,\nu,*}$ *est PSPACE-dur.*

Nous utilisons ici le fait qu'intuitivement un arbre $\tau$ est modèle d'une formule de la forme $\varphi \mid \psi$ s'il existe une décomposition de $\tau$ en $\tau_1, \tau_2$ (*ie* $\tau_1 \mid^{\mathcal{T}} \tau_2 = \tau$) telle que $\tau_1 \models \varphi$ et $\tau_2 \models \psi$, le choix dans la décomposition étant non-déterministe et éventuellement exponentiel dans la taille de l'arbre.

### 9.1.3 Complexité de données pour le problème de model-checking de TQL

Nous rapportons ici des résultats sur la complexité de données pour le problème de model-checking de TQL ; nous rappelons qu'il s'agit de la complexité à formule fixée.

La complexité de données peut en général être bien inférieure à la complexité combinée ; ainsi, le problème de model-checking de la logique MSO pour les arbres, dont la complexité combinée est PSPACE-complète, a une complexité de données linéaire. Il en est d'ailleurs de même pour la logique PMSO comme montré dans [SSM03].

Malheureusement, il n'en est rien pour TQL. Par réduction du problème de validité des formules booléennes quantifiées, nous montrons que :

**Theorem 18 ([17])** *La complexité de données du problème de model-checking de la logique* TQL *est PSPACE-dure.*

Nous avons étudié le problème en l'absence d'opérateurs de point-fixe et d'itération et montré que

**Theorem 19 ([17])** *La complexité de données du problème de model-checking de la logique* $\mathrm{TQL}^{-\nu,*}$ *est dure pour tous les niveaux de la hiérarchie polynomiale.*

Autrement dit, pour tout entier naturel $k$, il existe une formule fixée $\varphi_k$ telle que le problème de model-checking $\tau \models \varphi_k$ est $\Sigma_k^P$-dur (resp. $\Pi_k^P$-dur).

Nous avons déjà évoqué l'équivalence d'un point de vue de l'expressivité entre $\mathrm{TQL}_{g\nu}^{-\exists}$, le fragment à récursion gardée de TQL et PMSO. Cette preuve est réalisée de manière constructive via la construction d'un automate d'arbres à contraintes semi-linéaires effectif ; ainsi, tout comme pour la logique PMSO, nous pouvons à partir d'une formule de la logique $\mathrm{TQL}_{g\nu}^{-\exists}$ construire un automate d'arbres à contraintes semi-linéaires effectif, pour lequel le test d'appartenance pour un arbre est linéaire (dans la taille de l'arbre) [SSM03] ; ainsi,

**Theorem 20 ([17])** *La complexité de données du problème de model-checking de la logique* $\mathrm{TQL}_{g\nu}^{-\exists}$ *est linéaire.*

Nous considérons également $\mathrm{TQL}^{-\exists}$ le fragment de TQL sans quantification. Nous pouvons comme pour $\mathrm{TQL}_{g\nu}^{-\exists,*}$ construire un automate à contraintes arithmétiques qui acceptera précisément les modèles de la formule. Cependant, les ensembles de vecteurs d'entiers associés aux transitions se trouvent présentés sous la forme de systèmes d'équations qu'on ne peut "résoudre" : il n'existe pas d'algorithme permettant de vérifier qu'une variable possède une dénotation vide dans la solution du système, rendant impossible le test du vide de l'automate construit. Ceci justifie l'indécidabilité de la satisfiabilité pour les formules de ce fragment [19]. Néanmoins, nous disposons d'une construction produisant un automate dont le langage est précisément l'ensemble des modèles de la formule et qui peut être utilisé pour répondre "efficacement" au problème d'appartenance ; il est possible pour un vecteur donné de tester son appartenance à la solution d'un système d'équations tel que ceux produits par la construction et cela en temps polynomial. Ceci implique que

**Theorem 21** *La complexité de données du problème de model-checking de la logique* $\mathrm{TQL}^{-\exists}$ *est dans PTIME.*

Cette bonne nouvelle doit cependant être relativisée ; en effet, l'algorithme que nous proposons est exponentiel dans la taille de la formule (qui est supposée constante dans le cadre de la complexité de données) et de la forme $O(|t|^{|\varphi|})$. Ainsi, notre méthode ne permet pas d'affirmer que la complexité de

données du model-checking de $\mathrm{TQL}^{-\exists}$ est uniformément bornée sur l'ensemble des formules par un (unique) polynôme dans la taille de l'arbre. On peut d'ailleurs se poser la question de la complexité paramétrique du problème et, en particulier, si le problème est praticable à paramètre fixé (*fixed-parameter tractable*) : on peut se demander s'il existe une fonction calculable $f$ et un polynôme $P$ tels que la complexité de données du problème de model-checking pour $\mathrm{TQL}^{-\exists}$ soit en $O(f(|\varphi|) * P(|t|))$.

Nous synthétisons nos résultats à la figure 9.1 où $\mathrm{TQL}_{\mathbb{B}ool}$ désigne le fragment de TQL comprenant les opérations booléennes, la quantification sur les labels et le test d'égalité. Tous les fragments sur cette figure sont complets pour PSPACE en ce qui concerne leur complexité combinée.

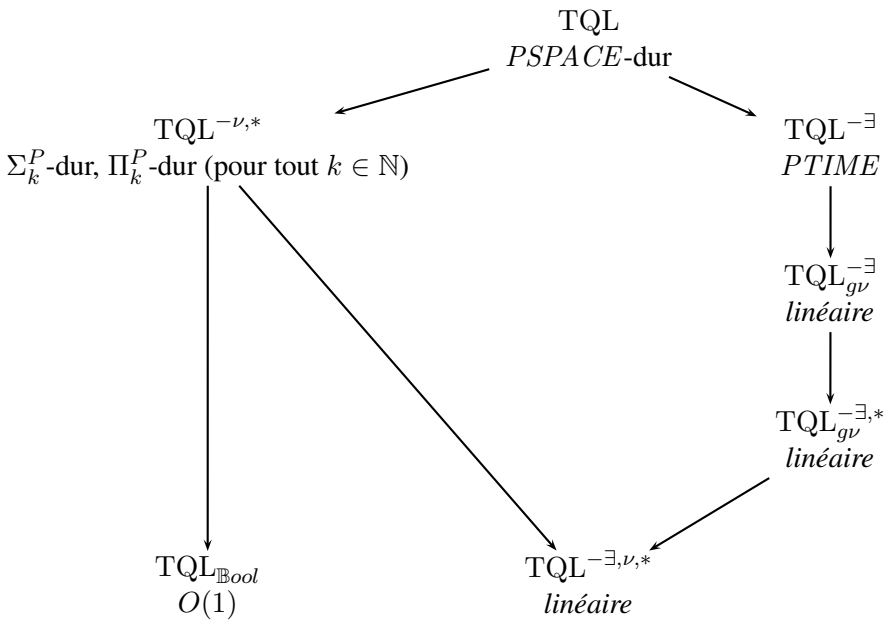



FIG. 9.1 – Complexité de données pour le problème de model-checking de formules closes de divers fragments de TQL.

Cardelli, Gardner et Ghelli ont introduit dans [CGG02] une logique spatiale pour les graphes GL. Elle est constituée d'une logique du premier ordre à deux sortes (labels d'arêtes - nœuds) augmentée par un opérateur de composition et une construction de point-fixe. Donné un ensemble de noms désignant des sommets, les (hyper-)graphes sont représentés comme un multiensemble d'arêtes entre ces sommets. La composition de deux graphes est alors simplement l'union des multiensembles d'arêtes de ces graphes. L'opérateur de composition de la logique traduit simplement cette composition sur les graphes.

Indépendamment de notre travail, une étude sur la complexité du problème de model-checking pour les graphes et cette logique a été menée [DGG04b]. Les résultats obtenus sont comparables aux nôtres : les complexités combinée et de données de la logique sont PSPACE-complètes (tout comme TQL). De plus, le fragment sans récursion a une complexité combinée qui est PSPACE-complète et une complexité de données qui peut être dure pour tous les niveaux de la hiérarchie polynomiale (comme $\mathrm{TQL}^{-\nu,*}$).

Il convient de noter que nos résultats impliquent les résultats de [DGG04b] ; il est en effet très simple de construire, à partir d'une formule de TQL sans quantification sur les arbres, une formule de GL ayant les mêmes modèles d'arbres. Le seul point délicat est la transformation de la composition puisqu'elle diffère selon les deux logiques : il faut vérifier dans le cas de GL, que les deux composantes sont toujours des arbres (chacune des composantes doit rester connexe, ce qui est définissable par une formule de GL sans récursion) et que la décomposition a eu lieu à la racine (il suffit de paramétrer la transformation

par le nœud racine de l'arbre pour en garder trace). De plus, si la formule de TQL ne comporte pas de récursion, alors il en va de même pour la formule de GL.

## 9.2 Réponses aux requêtes

Dans cette section, nous évoquons brièvement le problème de répondre aux requêtes (monadiques).

Il est très simple de transformer l'algorithme de model-checking que nous avons présenté en un algorithme pour répondre à des requêtes ; cependant, contrairement à une logique comme MSO, interprétée sur la structure finie de l'arbre, notre domaine d'interprétation est infini. Ainsi, la requête $\neg X$ aura un nombre infini de réponses quel que soit l'arbre (fini) auquel elle s'applique. Cependant, par la proposition 9 de [17], si un arbre $\tau'$ qui n'est pas sous-composante [25] de $\tau$ et que $\tau \models_{[X\mapsto\tau']} \varphi(X)$ alors tout arbre $\tau''$ également non sous-composante de $\tau$ vérifiera $\tau \models_{[X\mapsto\tau'']} \varphi(X)$. On peut donc finiment représenter l'ensemble des solutions puisque pour un arbre $\tau$, le nombre de ses sous-composantes est, lui, fini. Un résultat similaire existe pour les labels (proposition 10 de [17]).

Un algorithme (naïf) pour répondre aux requêtes est donc de tester une à une toutes les valuations possibles par l'algorithme de model-checking que nous avons proposé. Notez que ce nombre de valuations est très grand puisqu'il faut considérer toutes les sous-composantes d'un arbre qui sont en nombre exponentiel dans la taille de l'arbre.

[25] Un arbre $\tau'$ est une sous-composante de $\tau$ s'il existe $\tau''$ tel que $\tau' \mid \tau''$ est sous-arbre de $\tau$.

# Chapitre 10

# Conclusion et perspectives

Dans ce dernier chapitre, nous allons tenter de dégager des perspectives pour notre travail. Nous avons étudié TQL telle qu'elle était sans chercher à l'étendre ou à remettre en cause sa pertinence. Ceci va être fait en partie dans ce chapitre.

Nous pensons que TQL est un bon système basé sur une logique intéressante. Malheureusement, nous sommes persuadés que le modèle d'arbres, lui, n'est pas le bon.

**arbres non-ordonnés vs arbres ordonnés** On peut argumenter que l'ordre présent entre les fils d'un même nœud dans un arbre XML est quelque peu arbitraire et qu'il n'est pas souhaitable qu'un système d'interrogation s'appuie sur cet ordre, comme un système de requêtes SQL ne s'appuie pas sur l'ordre de stockage des enregistrements d'une table ; il faut avouer que la présence de cet ordre existe en pratique, représenté par exemple dans différents axes XPATH, et facilite grandement la théorie en la rendant beaucoup plus uniforme.

Que deviendrait TQL pour un modèle d'arbres ordonnés ? Notre description basée sur une algèbre d'arbres rend quasiment automatique la réponse à cette question. En effet, de par la présentation que nous avons faite de TQL, il est immédiat de voir que si on dispose d'une algèbre dont le support sera le domaine d'interprétation des formules, il suffit de faire des opérations de l'algèbre des connecteurs de la logique pour obtenir une logique spatiale.

Il convient donc juste de choisir l'algèbre permettant d'engendrer des arbres ordonnés. La solution immédiate à laquelle on peut penser et qui permettra de conserver la syntaxe de TQL est la suivante : les opérations unaires d'extension restent identiques ainsi que la constante $\mathbf{0}$. La composition $\tau_1 \mid \tau_2$ devient une composition d'arbres ordonnés fusionnant les racines de $\tau_1$ et de $\tau_2$ tout en considérant un ordre qui préserve l'ordre des fils de chacune des deux racines et fait que tous les fils de la racine de $\tau_2$ sont plus grands que les fils de la racine de $\tau_1$. Cette solution a donc l'avantage de préserver la syntaxe de TQL. Notons que, par opposition au cas non-ordonné, l'interprétation de $\mid$ n'est plus qu'associative.

Une seconde solution est de considérer une algèbre qui aurait pour signature la constante $\mathbf{0}$ interprétée comme précédemment et un ensemble de symboles $+_a$ introduit pour chacun des labels $a$ de l'ensemble $\mathcal{N}$. On interprétera [26] $\tau_1 +_a \tau_2$ comme étant l'arbre obtenu à partir de $\tau_1$ auquel on ajoute une arête étiquetée par $a$ en dernière position à la racine, l'extrémité de cette arête étant le sous-arbre $\tau_2$.

Une troisième solution est l'algèbre d'extension d'arbres de [BL02, LN03] qui consiste à étendre un arbre par ses extrémités terminales.

Les deux premières solutions sont à rapprocher des deux façons de voir l'algèbre des mots : dans le premier cas, on dispose des lettres comme constantes et d'une opération de concaténation ; dans le

[26] D'autres interprétations voisines sont possibles.

second cas, on dispose d'une opération par lettre de l'alphabet, opération qui ajoute cette lettre en fin de mot.

Il convient de remarquer que cette approche purement algébrique est à rapprocher de celle de Courcelle, présentée notamment dans [Cou90]. Alors que la reconnaissabilité algébrique (au sens de [MW67]) pour les arbres non-ordonnés d'arité non bornée correspond à la logique CMSO [Cou90], l'équationnalité (au sens de Courcelle) correspond pour ces mêmes arbres à la logique PMSO [18].

On peut ainsi voir $\mathrm{TQL}^{-\exists}$ comme un système équationnel [Cou90] où les membres gauches d'équations, outre l'union, peuvent contenir des intersections (comme les grammaires conjonctives de [Okh01]) et des négations (comme dans les équations de [Lei94]).

Pour chacune des algèbres d'arbres que nous avons proposées (en particulier la première et la seconde), il conviendrait de rééditer l'étude que nous avons menée sur l'expressivité, la satisfiabilité et la complexité du model-checking et du problème de réponses aux requêtes. Nous nous attendons dans le cas des complexités à des résultats bien meilleurs que dans le cas non-ordonné.

Concernant le problème de requêtes, notre objectif est de développer une logique possédant des variables d'arbres et de labels mais, contrairement à [BL02, LN03], sans la possibilité de quantifier sur ces variables, nous rapprochant plus d'un langage de filtrage comme celui de CDuce [BCF03].

**arbres vs haies** Nous avons dans le paragraphe précédent fait une présentation algébrique d'arbres plutôt que de haies. Ceci peut avoir une incidence en cas de présence de variables de capture dans la logique : que doivent capturer des variables ? Traditionnellement dans une approche "modèle fini", la structure étant l'arbre, il apparaît comme évident que les variables capturent des nœuds. Dans le cas d'une approche "langages de programmation", XDuce et CDuce voient leurs variables de captures instanciées par des séquences d'arbres, donc des haies. L'interprétation de TQL étant plus proche des langages comme XDuce et CDuce, il paraît donc évident que les valeurs des variables (de capture) doivent correspondre à des haies. Ceci transparaît dans la première des algèbres d'arbres ordonnés que nous avons présentée. La capture de haies dans un langage de programmation ou un langage de transformation facilite la recopie de morceaux de type "liste" du document d'origine vers le document produit.

**TQL, $\mu$-calcul, FO et LFP** Une des particularités de TQL comparée aux logiques monadiques du second ordre est de considérer comme domaine d'interprétation l'ensemble de tous les arbres et non pas l'ensemble des nœuds d'un unique arbre.

Une telle approche a également été suivie dans les travaux de Libkin et Benedikt pour les arbres d'arité bornée [BL02], étendue par Libkin et Neven pour les arbres ordonnés d'arité non bornée [LN03]. La logique décrite dans ces travaux est une logique du premier ordre FO équipée d'opérations algébriques permettant la construction des arbres et de relations définies sur les arbres. Les opérations algébriques considérées dans ces travaux sont cependant très différentes de celles que nous avons définies ici, puisque les arbres y sont notamment construits par extension au niveau des extrémités (feuilles). Il se trouve que la logique FO définie dans [LN03] est décidable (pour le problème de satisfiabilité) et que les relations définies par une formule avec $n$ variables libres sont précisément les relations reconnaissables d'arbres $n$-aires [CDG$^+$97].

On peut donc se poser la question de l'existence d'un lien entre TQL (ou plutôt TQL sans quantification sur les labels) et une logique proche de la logique du premier ordre sur une certaine signature. Nous considérons une signature purement relationnelle avec $|$, un opérateur ternaire et pour tout $a$ de $\mathcal{N}$, un symbole binaire $a$. Ces symboles sont interprétés de la manière suivante : $|\ (\tau, \tau_1, \tau_2)$ est vrai si et seulement si $\tau = \tau_1\ |^{\mathcal{T}}\ \tau_2$ et $a(\tau, \tau')$ si $\tau = a^{\mathcal{T}}(\tau')$. On note $\mathfrak{T}$ cette structure. Il est alors possible de construire à partir d'une formule $\phi$ de TQL sans quantification sur les labels avec pour variables libres $X_1, \ldots, X_n$ une formule $\tilde{\phi}$ de la logique M-LFP (*monadic least fixed-point logic* - la logique du premier

ordre étendue par un opérateur de plus petit point fixe monadique [EF95]) telle que $X_1, \ldots, X_n, Y$ sont les variables libres de $\tilde{\phi}$ et que

$$\tau \models_\rho \varphi \text{ ssi } \mathfrak{T}, \rho[Y \mapsto \tau] \models \tilde{\phi}$$

Une alternative comportant l'opérateur de point-fixe mais se basant sur la vision "un arbre - une structure" est le $\mu$-calcul. Il a été étudié dans [BL05] pour définir un langage de requêtes monadiques : un nœud est sélectionné s'il vérifie une certaine formule du $\mu$-calcul (incluant des modalités inverses). Ce formalisme est complet pour les requêtes monadiques MSO-définissables. L'extension vers un formalisme $n$-aire peut passer par l'utilisation d'un $\mu$-calcul enrichi par la présence de nominaux [SV01].

**TQL pour les arbres ordonnés** Nous souhaitons donc développer le cadre de TQL pour les arbres ordonnés en ayant comme objectif les trois points de [NS00], *ie* définir un formalisme qui soit :

1. facilement utilisable
2. aussi expressif que MSO
3. efficacement évaluable

Concernant ce dernier point, il convient de noter que la complexité combinée du problème de model-checking ou de réponses aux requêtes peut difficilement être polynomiale pour une logique avec variables quantifiées puisque le problème est déjà NP-complet pour les requêtes conjonctives et les axes XPATH [GKS04].

Les travaux que nous envisageons seront proches de ceux réalisés autour des constructions de filtrage des langages XDuce et CDuce. En particulier, ce dernier est relativement proche d'une "logique" puisque le sous-typage sémantique [FCB02] permet de considérer les opérations booléennes dans les types et les motifs.

Une différence notable entre ces langages et une logique TQL est l'objectif du filtrage ; alors qu'un langage de requêtes permet de retourner toutes les solutions (*all-matches policy*), le mécanisme de XDuce, par exemple, ne calcule qu'un filtre selon la double stratégie "*first and longest match*". De plus, XDuce et CDuce imposent des restrictions sur l'utilisation des variables de captures qui ne seraient *a priori* pas présentes dans la logique, mais dont il conviendrait d'étudier l'intérêt.

Si les résultats de complexité pour le problème de réponses aux requêtes devaient s'avérer mauvais, on pourra dans un second temps s'intéresser à des restrictions de la logique, comme la logique de chemins de [CG01a]. Il sera alors intéressant de comparer son expressivité avec, par exemple, GFOREG, la logique gardée augmentée d'expressions régulières de [Sch00] ou le langage Conditional XPATH de [Mar04].

# Bibliographie


[ABD+05] L. Afanasiev, P. Blackburn, I. Dimitriou, B. Gaiffe, E. Goris, M. Marx, and M. de Rijke. PDL for Ordered Trees. *Journal of Applied Non-Classical Logics*, 15(2) :115–135, 2005.

[ABS00] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web*. Morgan-Kaufmann, 2000.

[AD96] R. Amadio and M. Dam. Toward a Modal Theory of Types for the $\pi$-Calculus. In *Formal Techniques in Real-Time and Fault-Tolerant Systems, 4th International Symposium*, volume 1135 of *Lecture Notes in Computer Science*, pages 347–365. Springer, 1996.

[AKPG01] T. Amtoft, A. J. Kfoury, and S. M. Pericas-Geertsen. What are Polymorphically-Typed Ambients ? In *10th European Symposium on Programming (ESOP 2001)*, volume 2028 of *Lecture Notes in Computer Science*, pages 206–220. Springer, 2001.

[AM02] R. Amadio and C. Meyssonnier. On Decidability of the Control Reachability Problem in the Asynchronous $\pi$-Calculus. *Nordic Journal of Computing*, 9(1) :70–101, 2002.

[Ama97] R. Amadio. An Asynchronous model of Locality, Failure, and Process Mobility. In *Second International Conference on Coordination Languages and Models (COORDINATION'97)*, volume 1282 of *Lecture Notes in Computer Science*, pages 374–391. Springer, 1997.

[Ama00] R. Amadio. On Modelling Mobility. *TCS*, 240(1) :147–176, 2000.

[AMW04] T. Amtoft, H. Makholm, and J. B. Wells. PolyA : True Type Polymorphism for Mobile Ambients. In *3rd International Conference on Theoretical Computer Science (TCS2004)*, pages 591–604. Kluwer, 2004.

[AN01] A. Arnold and D. Niwinski. *Rudiments of $\mu$-Calculus*. North-Holland, 2001.

[AP94] R. Amadio and S. Prasad. Localities and Failures. In *14th Foundations of Software Tecchnology and Theoretical Computer Science*, volume 880 of *Lecture Notes in Computer Science*. Springer, 1994.

[ASW94] H. R. Andersen, C. Stirling, and G. Winskel. A compositional proof system for the modal mu-calculus. In *9th IEEE Annual Symposium on Logic in Computer Science (LICS'94)*, pages 144–153. IEEE Press, 1994.

[BB92] G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96(1) :217–248, 1992.

[BBDCS03] F. Barbanera, M. Bugliesi, M. Dezani-Ciancaglini, and V. Sassone. A calculus of bounded capacities. In *Advances in Computing Science - ASIAN 2003 Programming Languages and Distributed Computation, 8th Asian Computing Science Conference*, volume 2896 of *Lecture Notes in Computer Science*, pages 205–223. Springer, 2003.

[BCC01] M. Bugliesi, G. Castagna, and S. Crafa. Boxed Ambients. In *Theoretical Aspects of Computer Software (TACS 2001)*, volume 2215 of *Lecture Notes in Computer Science*, pages 38–63. Springer, 2001.

[BCF03] V. Benzaken, G. Castagna, and A. Frisch. CDuce : an XML-centric general-purpose language. In *Eighth ACM SIGPLAN International Conference on Functional Programming, ICFP 2003*, pages 51–63. ACM, 2003.

[BDM+05] M. Bojanczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on words with data. Technical Report 2005-004, LIAFA - Paris 7, 2005.

[BE00] R. Bloem and J. Engelfriet. A comparison of tree transductions defined by monadic second order logic and by attribute grammars. *Journal of Computer and System Sciences*, 61(1) :1–50, 2000.

[BKMW01] A. Brüggemann-Klein, M. Murata, and D. Wood. Regular Tree and Regular Hedge Languages over Unranked Alphabets : Version 1. Technical Report, 2001.

[BL02] M. Benedikt and L. Libkin. Tree extension algebras : Logics, automata, and query languages. In *17th IEEE Symposium on Logic in Computer Science (LICS 2002)*, pages 203–212. IEEE Computer Society, 2002.

[BL05] P. Barceló and L. Libkin. Temporal logics over unranked trees. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005)*, pages 31–40. IEEE Computer Society, 2005.

[Bou92] G. Boudol. Asynchrony and the Pi-calculus. Technical Report RR-1702, INRIA, 1992.

[BS02] A. Berlea and H. Seidl. Binary queries. In *Extreme Markup Languages 2002 Conference*, 2002.

[BSS05] P. Bidinger, A. Schmitt, and J.-B. Stefani. An abstract machine for the kell calculus. In *Formal Methods for Open Object-Based Distributed Systems, 7th IFIP WG 6.1 International Conference, FMOODS 2005*, volume 3535 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2005.

[BZ02] N. Busi and G. Zavattaro. On the expressiveness of movement in pure mobile ambients. In *Foundations of Wide Area Network Computing*, ENTCS 66(3). Elsevier, 2002.

[BZ04] N. Busi and G. Zavattaro. On the expressive power of movement and restriction in pure mobile ambients. *Theoretical Computer Science*, 322(3) :477–551, 2004.

[BZ05] N. Busi and G. Zavattaro. Deciding reachability in mobile ambients. In *14th European Symposium on Programming (ESOP 2005)*, volume 3444 of *Lecture Notes in Computer Science*, pages 248–262. Springer, 2005.

[Büc60] J. R. Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift für mathematische Logik und Grundladen der Mathematik*, 6 :66–92, 1960.

[Car99] L. Cardelli. *Secure Internet Programming : Security Issues for Mobile and Distributed Objects*, volume 1603 of *Lecture Notes in Computer Science*, chapter Abstractions for Mobile Computation, pages 51–94. Springer, 1999.

[CC02] L. Caires and L. Cardelli. A spatial logic for concurrency (part ii). In *CONCUR 2002 - Concurrency Theory, 13th International Conference*, volume 2421 of *Lecture Notes in Computer Science*, pages 209–225. Springer, 2002.

[CC03] L. Caires and L. Cardelli. A spatial logic for concurrency (part i). *Information and Computation*, 186(2) :194–235, 2003.

[CCG03] C. Calcagno, L. Cardelli, and A. D. Gordon. Deciding validity in a spatial logic for trees. In *ACM SIGPLAN Workshop on Types in Language Design and Implementation (TLDI 2003)*, volume 38(3) of *SIGPLAN Notices*, pages 62–73. ACM, 2003.

[CDG+97] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on : http://www.grappa.univ-lille3.fr/tata, 1997. release October, 1rst 2002.

[CFG02] G. Conforti, O. Ferrara, and G. Ghelli. TQL algebra and its implementation. In *2nd IFIP International Conference on Theoretical Computer Science (TCS 2002)*, volume 223 of *IFIP Conference Proceedings*, pages 422–434. Kluwer, 2002.

[CG98] L. Cardelli and A. D. Gordon. Mobile ambients. In *Foundations of Software Science and Computational Structures (FOSSACS'98)*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155. Springer, 1998.

[CG99] L. Cardelli and A. D. Gordon. Types for mobile ambients. In *26th ACM Symposium on Principles of Programming Languages (POPL'99)*, pages 79–92, 1999.

[CG00a] L. Cardelli and A.D. Gordon. Anytime, anywhere : Modal logics for mobile ambients. In *27th Symp. on Principles of Programming Languages (POPL'00)*, pages 365–377, 2000.

[CG00b] L. Cardelli and A.D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1) :177–213, 2000.

[CG01a] L. Cardelli and G. Ghelli. A query language based on the ambient logic. In *European Symposium on Programming (ESOP'01)*, volume 2028 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2001.

[CG01b] L. Cardelli and A.D. Gordon. Logical Properties of Name Restriction. In *5th International Conference on Typed Lambda Calculi and Applications (TLCA 2001)*, volume 2044 of *Lecture Notes in Computer Science*, pages 46–60. Springer, 2001.

[CG04a] L. Cardelli and G. Ghelli. TQL : a query language for semistructured data based on the ambient logic. *Mathematical Structures in Computer Science*, 14(3) :285–327, 2004.

[CG04b] G. Conforti and G. Ghelli. Decidability of freshness, undecidability of revelation. In *Foundations of Software Science and Computation Structures, 7th International Conference (FOSSACS 2004)*, volume 2987 of *Lecture Notes in Computer Science*, pages 105–120. Springer, 2004.

[CGA+02] G. Conforti, G. Ghelli, A. Albano, D. Colazzo, P. Manghi, and C. Sartiani. The query language TQL. In *WebDB*, pages 13–18, 2002.

[CGG02] L. Cardelli, P. Gardner, and G. Ghelli. A spatial logic for querying graphs. In *29th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 2380 of *Lecture Notes in Computer Science*, pages 597–610. Springer, 2002.

[CGar] L. Cardelli and A.D. Gordon. Ambient logic. *Mathematical Structures in Computer Science*, to appear.

[Chi00] B. Chidlovskii. Using regular tree automata as xml schemas. In *IEEE Advances in Digital Libraries 2000 (ADL 2000)*, pages 89–104, 2000.

[CM98] L. Caires and L. Monteiro. Verifiable and executable logic specifications of concurrent objects in $l_\pi$. In *7th European Symposium on Programming Languages and Systems (ESOP'98)*, volume 1381 of *Lecture Notes in Computer Science*, pages 42–56. Springer, 1998.

[Cou90] B. Courcelle. The Monadic Second-Order Logic of Graphs. I. Recognizable Sets of Finite Graphs. *IC*, 85(1) :12–75, 1990.

[Cou94] B. Courcelle. The monadic second order logic of graphs VI : on several representations of graphs by relational structures. *Discrete Applied Mathematics*, 54(2-3) :117–149, 1994.

[Cou96] B. Courcelle. Basic notions of universal algebra for language theory and graph grammars. *Theoretical Computer Science*, 163(1&2) :1–54, 1996.

[CPS93] R. Cleaveland, J. Parrow, and B. Steffen. The concurrency workbench : A semantics-based tool for the verification of concurrent systems. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 15(1) :36–72, 1993.

[Dal00] S. Dal Zilio. Spatial congruence for ambients is decidable. In *6th Asian Computing Science Conference (ASIAN'00)*, volume 1961 of *Lecture Notes in Computer Science*, pages 88–103. Springer, 2000.

[Dal01] S. Dal Zilio. Mobile Processes : a Commented Bibliography. In *Modeling and Verification of Parallel Processes (MOVEP 2000)*, volume 2067 of *Lecture Notes in Computer Science*, pages 206–222. Springer, 2001.

[Dam88] M. Dam. Relevance logic and concurrent composition. In *3rd IEEE Annual Symposium on Logic in Computer Science (LICS'88)*, pages 178–185. IEEE Press, 1988.

[Dam90] M. Dam. *Relevance Logic and Concurrent Composition*. PhD thesis, University of Edinburgh, 1990.

[Dam96] M. Dam. Model checking mobile processes. *Information and Computation*, 129(1) :35–51, 1996.

[Dam97] M. Dam. On the decidability of process equivalences for the pi-calculus. *Theoretical Computer Science*, 183(2) :215–228, 1997.

[DG99] M. Dam and D. Gurov. Compositional verification of CCS processes. In *Perspectives of System Informatics, Third International Andrei Ershov Memorial Conference (PSI'99)*, volume 1755 of *Lecture Notes in Computer Science*, pages 247–256. Springer, 1999.

[DGG04a] A. Dawar, P. Gardner, and G. Ghelli. Adjunct elimination through games in static ambient logic. In *FSTTCS 2004 : Foundations of Software Technology and Theoretical Computer Science, 24th International Conference*, volume 3328 of *Lecture Notes in Computer Science*, pages 211–223. Springer, 2004.

[DGG04b] Anuj Dawar, Philippa Gardner, and Giorgio Ghelli. Expressiveness and complexity of graph logic. Technical report, Imperial College, 2004.

[DZLM04] S. Dal-Zilio, D. Lugiez, and C. Meyssonnier. A logic you can count on. In *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2004)*, pages 135–146. ACM, 2004.

[EF95] H.-D. Ebbinghaus and J. Flum. *Finite model theory*. Perspectives in Mathematical Logic. Springer-Verlag, Berlin, 1995.

[EH82] E.A. Emerson and J.Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. In *Proceedings of the fourteenth annual ACM Symposium on Theory of Computing (STOC'82)*, pages 169–180. ACM Press, 1982.

[EK95] J. Esparza and A. Kiehn. On the model checking problem for branching time logics and basic parallel processes. In *7th International Conference on Computer Aided Verification (CAV'95)*, volume 939 of *Lecture Notes in Computer Science*, pages 353–366. Springer, 1995.

[EN00] U. H. Engberg and M. Nielsen. A calculus of communicating systems with label passing—ten years after. In *Proof, Language, and Interaction ; Essays in Honour of Robin Milner*, chapter V, Mobility, pages 599–622. MIT Press, 2000.

[Esp97] J. Esparza. Decidability of model-checking for infinite-state concurrent systems. *Acta Informatica*, 34 :85–107, 1997.

[FCB02] A. Frisch, G. Castagna, and V. Benzaken. Semantic subtyping. In *17th IEEE Symposium on Logic in Computer Science (LICS 2002)*, pages 137–146. IEEE Computer Society, 2002.

[FG02] M. Frick and M. Grohe. The complexity of first-order and monadic second-order logic revisited. In *17th IEEE Symposium on Logic in Computer Science (LICS 2002)*, pages 215–224. IEEE Computer Society, 2002.

[FL77] M.J. Fischer and R.E. Ladner. Propositional modal logic of programs. In *9th ACM Symposium on Theory of Computing (STOC'77)*, pages 194–211. ACM Press, 1977.

[GC02] A. D. Gordon and L. Cardelli. Equational Properties of Mobile Ambients. *Mathematical Structures in Computer Science*, 12 :1–38, 2002.

[GG99] S. Gaubert and A. Giua. Petri net languages and infinite subsets of m. *Journal of Computer and System Sciences*, 59(3) :373–391, 1999.

[Gir87] J-Y Girard. Linear logic. *Theoretical Computer Science*, 50 :1–102, 1987.

[GK02] Georg Gottlob and Christoph Koch. Monadic queries over tree-structured data. In *17th IEEE Symposium on Logic in Computer Science (LICS 2002)*, pages 189–202. IEEE Computer Society, 2002.

[GKS04] G. Gottlob, C. Koch, and K. U. Schulz. Conjunctive queries over trees. In *Proceedings of the Twenty-third ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2004)*, pages 189–200. ACM, 2004.

[GLV02] J. Goubault-Larrecq and K. N. Verma. Alternating Two-way AC-Tree Automata. Technical report, LSV, Laboratoire Spécification et Vérification, 2002.

[Gol88] U. Goltz. On Representing CCS Programs by Finite Petri Nets. In *Mathematical Foundations on Computer Science*, volume 324 of *Lecture Notes in Computer Science*, pages 339–350, 1988.

[GP99] M. Gabbay and A.M. Pitts. A new approach to abstract syntax involving binders. In *Fourteenth Annual IEEE Symposium on Logic in Computer Sience (LICS'99)*, pages 214–224. IEEE Press, 1999.

[GYY00] X. Guan, Y. Yang, and J. You. Making Ambients more Robust. In *International Conference on Software : Theory and Practice*, pages 377–384, 2000.

[Hir99] D. Hirschkoff. *Mise en œuvre de preuves de bisimulation*. PhD thesis, Ecole Nationale des Ponts et Chausses, 1999.

[HLS02] D. Hirschkoff, E. Lozes, and D. Sangiorgi. Separability, expressiveness, and decidability in the ambient logic. In *Logic in Computer Science (LICS'02)*, pages 423–432. IEEE Computer Society, 2002.

[HM80] M. Hennessy and R. Milner. On observing nondeterminism and concurrency. In *7th Colloquium on Automata, Languages and Programming*, volume 85 of *Lecture Notes in Computer Science*, pages 299–309. Springer, 1980.

[HP03] H. Hosoya and B. C. Pierce. XDuce : A statically typed xml processing language. *ACM Trans. Internet Techn.*, 3(2) :117–148, 2003.

[IO01] S. Ishtiaq and P. W. O'Hearn. Bi as an assertion language for mutable data structures. In *28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 14–26. ACM, 2001.

[Koz83] D. Kozen. Results on the propositional mu -calculus. *Theoretical Computer Science*, 27(3) :333–354, December 1983.

[KR03] F. Klaedtke and H. Rueß. Monadic Second-Order Logics with Cardinalities. In *30th International Colloquium on Automata, Languages and Programming, ICALP 2003*, volume 2719 of *Lecture Notes in Computer Science*. Springer Verlag, 2003.

[Lar88] K. G. Larsen. Proof system for hennessy-milner logic with recursion. In *13th Colloquium on Trees in Algebra and Programming (CAAP '88)*, volume 299 of *Lecture Notes in Computer Science*, pages 215–230. Springer, 1988.

[Lei94] E. L. Leiss. Unrestricted complementation in language equations over a one-letter alphabet. *Theoretical Computer Science*, 132(2) :71–84, 1994.

[Lib05] L. Libkin. Logics for unranked trees : An overview. In *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005*, volume 3580 of *Lecture Notes in Computer Science*, pages 35–50. Springer, 2005.

[Lin04] H. Lin. A predicate spatial logic and model checking for mobile processes. In *First International Colloquium on Theoretical Aspects of Computing (ICTAC 2004)*, volume 3407 of *Lecture Notes in Computer Science*, pages 36–36. Springer, 2004.

[LN03] L. Libkin and F. Neven. Logical definability and query languages over unranked trees. In *18th IEEE Symposium on Logic in Computer Science (LICS 2003)*, pages 178–187. IEEE Computer Society, 2003.

[Loz04] E. Lozes. Adjuncts elimination in the static ambient logic. In *10th International Workshop on Expressiveness in Concurrency (EXPRESS'03)*, volume 96 of *Electronic Notes in Theoretical Computer Science*, pages 51–72. Elsevier Science Publishers, 2004.

[LS00] F. Levi and D. Sangiorgi. Controlling interference in ambients. In *27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 352–364, 2000.

[LS02] D. Lugiez and P. Schnoebelen. The regular viewpoint on PA-processes. *Theoretical Computer Science*, 274(1-2) :89–115, 2002.

[Lug05] Denis Lugiez. Multitree automata that count. *Theoretical Computer Science*, 333(1-2) :225–263, 2005.

[Mar04] M. Marx. Conditional xpath, the first order complete xpath dialect. In *Proceedings of the Twenty-third ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 13–22. ACM, 2004.

[May84] E.W. Mayr. An Algorithm for the General Petri Net Reachability Problem. *SIAM Journal of Computing*, 13(3) :441–460, 1984.

[May97] R. Mayr. Model checking PA-processes. In *Conference on Concurrency Theory (CONCUR'97)*, volume 1243 of *Lecture Notes in Computer Science*, pages 332–346, 1997.

[Mil80] R. Milner. *A Calculus of Communicating Systems*. Springer, 1980.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Mil90] R. Milner. Functions as processes. In *Proceedings ICALP '90*, volume 443 of *Lecture Notes in Computer Science*, pages 167–180. Springer, 1990.

[MP05] S. Maffeis and I. Phillips. On the computational strength of pure ambient calculi. *Theoretical Computer Science*, 330 :501–551, 2005.

[MPBS05] S. Maneth, T. Perst, A. Berlea, and H. Seidl. Xml type checking with macro tree transducers. In *Proceedings of the Twenty-fourth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2005)*, 2005.

[MPW92] R. Milner, J. Parrow, and J. Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1) :1–40,41–77, 1992.

[MPW93] R. Milner, J. Parrow, and D. Walker. Modal logics for mobile processes. *Theoretical Computer Science*, 114(1) :149–171, 1993.

[Mur99] M. Murata. Hedge Automata : A Formal Model for XML Schemata. Technical report, Fuji Xerox Information Systems, 1999.

[MW67] J. Mezei and J.B. Wright. Algebraic automata and context-free sets. *Information and Control*, 11(2-3) :3–29, 1967.

[NN00] H. Riis Nielson and F. Nielson. Shape analysis for mobile ambients. In *27th ACM Symposium on Principles of Programming Languages (POPL'00)*, pages 142–154, 2000.

[NP93] J. Niehren and A. Podelski. Feature automata and recognizable sets of feature trees. In *TAPSOFT'93 : Theory and Practice of Software Development, International Joint Conference CAAP/FASE*, volume 668 of *Lecture Notes in Computer Science*, pages 356–375. Springer, 1993.

[NS98] A. Neumann and H. Seidl. Locating matches of tree patterns in forests. In *Foundations of Software Technology and Theoretical Computer Science, 18th Conference*, volume 1530 of *Lecture Notes in Computer Science*, pages 134–145. Springer, 1998.

[NS00] F. Neven and T. Schwentick. Expressive and efficient pattern languages for tree-structured data. In *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2000)*, pages 145–156. ACM, 2000.

[NS01] F. Neven and T. Schwentick. Automata-and logic-based pattern languages for tree-structured data. In *Semantics in Databases*, volume 2582 of *Lecture Notes in Computer Science*, pages 160–178. Springer, 2001.

[NS02] F. Neven and T. Schwentick. Query automata over finite trees. *Theoretical Computer Science*, 275(1-2) :633–674, 2002.

[NSV01] F. Neven, T. Schwentick, and V. Vianu. Towards regular languages over infinite alphabets. In *Mathematical Foundations of Computer Science 2001, 26th International Symposium, MFCS 2001*, volume 2136 of *Lecture Notes in Computer Science*, pages 560–572. Springer, 2001.

[NV02] F. Neven and J. Van den Bussche. Expressiveness of structured document query languages based on attribute grammars. *Journal of the ACM*, 49(1) :56–100, 2002.

[Okh01] Alexander Okhotin. Conjunctive grammars. *Journal of Automata, Languages and Combinatorics*, 6(4) :519–535, 2001.

[Par66] R. J. Parikh. On Context-Free Languages. *Journal of the ACM*, 13(4) :570–581, 1966.

[Plo81] G.D. Plotkin. A Structural Approach to Operational Semantics. Technical report, Computer Science Dept, Aarhus University, Denmark, 1981.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*, pages 46–67, 1977.

[Pos44] E. L. Post. Recursively Enumerable Sets of Positive Integers and their Decision Problems. *Bulletion of the American Mathematical Society*, 50 :284–316, 1944.

[PV00] Y. Papakonstantinou and V. Vianu. Dtd inference for views of xml data. In *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 35–46. ACM, 2000.

[PV02] I. Phillips and M. G. Vigliotti :. On Reduction Semantics for the Push and Pull Ambient Calculus. In *2nd IFIP International Conference on Theoretical Computer Science (TCS 2002)*, pages 550–562. Kluwer, 2002.

[Rab69] M. O. Rabbin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141 :1–35, 1969.

[Rac78] C. Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6 :223–231, 1978.

[RE99] C. Röckl and J. Esparza. Proof-checking protocols using bisimulations. In *Conference on Concurrency Theory (CONCUR'99)*, volume 1664 of *Lecture Notes in Computer Science*, pages 525–540. Springer, 1999.

[Rey02] J. C. Reynolds. Separation logic : A logic for shared mutable data structures. In *17th IEEE Symposium on Logic in Computer Science (LICS 2002)*, pages 55–74. IEEE Computer Society, 2002.

[RH98] J. Riely and M. Hennessy. A Typed Language for Distributed Mobile Processes. In *25th ACM Symposium on Principles of Programming Languages (POPL'98)*, pages 378–390, 1998.

[San01] D. Sangiorgi. Extensionality and intensionality of the ambient logics. *ACM SIGPLAN Notices*, 36(3) :4–13, 2001.

[Sch00] T. Schwentick. On diving in trees. In *Mathematical Foundations of Computer Science 2000, 25th International Symposium (MFCS 2000)*, volume 1893 of *Lecture Notes in Computer Science*, pages 660–669. Springer, 2000.

[SSM03] H. Seidl, T. Schwentick, and A. Muscholl. Numerical Document Queries. In *Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 155–166. ACM, 2003.

[Sti89] C. Stirling. An introduction to modal and temporal logics for CCS. In *Concurrency : Theory, Language, And Architecture*, volume 491 of *Lecture Notes in Computer Science*, pages 2–20. Springer, 1989.

[SV01] U. Sattler and M. Y. Vardi. The hybrid $\mu$-calculus. In *Automated Reasoning, First International Joint Conference, IJCAR 2001*, volume 2083 of *Lecture Notes in Computer Science*, pages 76–91. Springer, 2001.

[SW91] C. Stirling and D. Walker. Local model checking in the modal mu-calculus. *Theoretical Computer Science*, 89(1) :161–177, 1991.

[Tra50] B. A. Trakhtenbrot. The impossibility of an algorithm for the decision problem for finite models. *Doklady Akademii Nauk SSR*, 70 :569–572, 1950.

[TW68] J. W. Thatcher and J. B. Wright. Generalized finite automata with an application to a decision problem of second-order logic. *Mathematical System Theory*, 2 :57–82, 1968.

[TZH02] D. Teller, P. Zimmer, and D. Hirschkoff. Using Ambients to Control Resources. In *19th International Conference on Concurrency Theory (CONCUR'02)*, volume 2421 of *Lecture Notes in Computer Science*, pages 288–303. Springer, 2002.

[Urq72] A. Urquhart. Semantics for relevant logics. *Journal of Symbolic Logic*, 37(1) :159–169, 1972.

[W3C99a] W3C. *XML Path Language (XPATH)*. World Wide Web Consortium, Nov 1999.

[W3C99b] W3C. *XSL Transformations (XSLT)*. World Wide Web Consortium, Nov 1999.

[W3C00] W3C. *Extensible Markup Language (XML) 1.0*. World Wide Web Consortium, Oct 2000.

[W3C04] W3C. *XML Schema Part 1 : Structures - Second Edition*. World Wide Web Consortium, Oct 2004.

[W3C05] W3C. *XQuery 1.0 : An XML Query Language*. World Wide Web Consortium, Sep 2005.

[Win85] G. Winskel. A complete system for SCCS with modal assertions. In *Fifth Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 206 of *Lecture Notes in Computer Science*, pages 392–410. Springer, 1985.

[Win91] G. Winskel. A note on model checking the modal nu-calculus. *Theoretical Computer Science*, 83(1) :157–167, 1991.

[Zim03] P. Zimmer. On the Expressiveness of Pure Safe Ambients. *Mathematical Structures of Computer Science*, 13(5) :721–770, 2003.

# Annexe A

# Publications

## A.1 Revues internationales

[1] Jean-Marc Talbot. The $\exists\forall^2$ Fragment of the First-Order Theory of Atomic Set Constraints is $\Pi_1^0$-hard. *Information Processing Letters*, 74(1–2) :27–33, 2000.

[2] Jean-Marc Talbot, Philippe Devienne, and Sophie Tison. Generalized Definite Set Constraints. *CONSTRAINTS - An international Journal*, 74(1-2) :161–202, 2000.

[3] Witold Charatonik, Silvano Dal Zilio, Andrew D. Gordon, Supratik Mukhopadhyay, and Jean-Marc Talbot. Model Checking Mobile Ambient. *Theoretical Computer Science*, 308(1-3) :277–331, 2003.

[4] Iovka Boneva and Jean-Marc Talbot. When Ambients Cannot be Opened. *Theoretical Computer Science*, 333(1-2) :127–169, 2005.

## A.2 Conférences internationales avec actes et comité de lecture

[5] Nirina Andrianarivelo, Wadoud Bousdira, and Jean-Marc Talbot. On Theorem-proving in Horn Theories with Built-in Algebras. In *Third International Conference on Artificial Intelligence and Symbolic Mathematical Computation - AISMC3*, volume 1138 of *Lecture Notes in Computer Science*, pages 320–338, 1996.

[6] Philippe Devienne, Jean-Marc Talbot, and Sophie Tison. Set-Based Analysis for Logic Programming and Tree Automata. In *Fourth International Static Analysis Symposium - SAS'97*, volume 1302 of *Lecture Notes in Computer Science*, pages 127–140, 1997.

[7] Philippe Devienne, Jean-Marc Talbot, and Sophie Tison. Solving Classes of Set Constraints with Tree Automata. In *Third International Conference on Constraint Programming (CP)*, volume 1330 of *Lecture Notes in Computer Science*, pages 62–76, 1997.

[8] Philippe Devienne, Jean-Marc Talbot, and Sophie Tison. Co-definite Set Constraints with Membership Expressions. In *Proceedings of the 1998 Joint International Conference and Symposium on Logic Programming (JICSLP-98)*, pages 25–39. MIT Press, 1998.

[9] Martin Müller, Joachim Niehren, and Jean-Marc Talbot. Entailment of Atomic Set Constraints is PSPACE-Complete. In *Fourteenth Annual IEEE Symposium on Logic in Computer Sience*, pages 285–294. IEEE Press, 1999.

[10] Witold Charatonik, Andreas Podelski, and Jean-Marc Talbot. Paths vs. Tress in Set-based Program Analysis. In *Principles of Programming Languages - POPL*, pages 330–338. ACM Press, 2000.

[11] Jean-Marc Talbot. On the Alternation-Free Horn $\mu$-Calculus. In *Seventh International Conference on Logic for Programming and Automated reasoning - LPAR'2000*, volume 1955 of *Lecture Notes in Artificial Intelligence*, pages 418–435, 2000.

[12] Witold Charatonik and Jean-Marc Talbot. The Decidability of Model Checking Mobile Ambients. In *Proceedings of the 15th Annual Conference of the European Association for Computer Science Logic (CSL'01)*, volume 2142 of *Lecture Notes in Computer Science*, pages 339–354. Springer, 2001.

[13] Witold Charatonik, Silvano Dal Zilio, Andrew D. Gordon, Supratik Mukhopadhyay, and Jean-Marc Talbot. The Complexity of Model Checking Mobile Ambients. In *Proceedings FoSSaCS'01*, volume 2030 of *Lecture Notes in Computer Science*, pages 152–167. Springer, 2001.

[14] Witold Charatonik, Andrew D. Gordon, and Jean-Marc Talbot. Finite-control Mobile Ambients. In *European Symposium on Programming - ESOP 2002*, volume 2305 of *Lecture Notes in Computer Science*, pages 295–313. Springer, 2002.

[15] Witold Charatonik and Jean-Marc Talbot. Atomic Set Constraints with Projection. In *International Conference on Rewriting Techniques and Applications - RTA 2002*, volume 2378 of *Lecture Notes in Computer Science*, pages 311–325. Springer, 2002.

[16] Iovka Boneva and Jean-Marc Talbot. When Ambients Cannot be Opened. In *Sixth International Conference on Foundations of Software Science and Computation Structures - FOSSACS 2003*, volume 2620 of *Lecture Notes in Computer Science*, pages 169–184. Springer, 2003.

[17] Iovka Boneva and Jean-Marc Talbot. On Complexity of Model-Checking for the TQL Logic. In *3rd IFIP International Conference on Theoretical Computer Science (IFIP-TCS'04)*, pages 381–394. Kluwer, 2004.

[18] Iovka Boneva and Jean-Marc Talbot. Automata and logics for unranked and unordered trees. In *International Conference on Rewriting Techniques and Applications (RTA 2005)*, volume 3467 of *Lecture Notes in Computer Science*, pages 500–515. Springer, 2005.

[19] Iovka Boneva, Jean-Marc Talbot, and Sophie Tison. Expressiveness of Spatial Logic for Trees. In *Twentieth Annual IEEE Symposium on Logic in Computer Sience (LICS 2005)*, pages 280–289. IEEE Press, 2005.

[20] Joachim Niehren, Laurent Planque, Jean-Marc Talbot, and Sophie Tison. N-ary queries by tree automata. In *Databases and Programming Languages (DBLP 2005)*, volume 3774 of *Lecture Notes in Computer Science*, 2005.

[21] Hitoshi Ohsaki, Jean-Marc Talbot, Sophie Tison, and Yves Roos. Monotone AC-tree automata. In *International Conference on Logic for Programming and Automated Reasoning (LPAR 2005)*, Lecture Notes in Computer Science. Springer, 2005.

# Annexe B

# Curriculum vitæ

Jean-Marc Talbot

Laboratoire d'Informatique Fondamentale de Lille UMR CNRS 8022 (LIFL)
Université des Sciences et Technologies de Lille (USTL)
Bâtiment M3, Cité scientifique
59655 Villeneuve d'Ascq Cedex
talbot@lifl.fr

**Fonctions et titres :**

**2001-.. :** Maître de conférences - UFR Informatique, Electronique, Electrotechnique et Automatisme (IEEA) - Université des Sciences et Technologies de Lille (USTL)

**1998 - 2000 :** Post-doctorant puis Wissenschaftlicher Mitarbeiter (Chercheur) au Max-Planck-Institut für Informatik (Sarrebruck - Allemagne)

**1998 :** Doctorat en informatique de l'Université des Sciences et Technologies de Lille - Mention Très honorable - Titre : "*Contraintes ensemblistes définies et co-définies : extensions et applications*"

**Encadrement :**

- Stage de DEA :
  - **Haiyan Housroum** *"Utilisation des automates d'arbres pour la vérification de protocoles cryptographiques."* (50 % - co-encadrement avec Sophie Tison - soutenance juillet 2001).
  - **Iovka Boneva** *"Le problème d'accessibilité dans le calcul des ambients."* (70 % - co-encadrement avec Sophie Tison - soutenance juillet 2002)
  - **Emmanuel Filiot** *"Composition de requêtes monadiques dans les arbres."* (30 % - co-encadrement avec Sophie Tison et Joachim Niehren - soutenance juillet 2005)
- Thèse de doctorat :
  - **Iovka Boneva** *Les langages de requêtes pour les arbres non-ordonnés d'arité non-bornée.* (70 % - co-encadrement avec Sophie Tison - début septembre 2002 - soutenance prévue janvier 2006)
  - **Laurent Planque** *Requêtes n-aires pour les documents semi-structurés.* (30 % - co-encadrement avec Sophie Tison et Joachim Niehren - début septembre 2004)
  - **Emmanuel Filiot** *Transformations de documents XML.* (50 % - co-encadrement avec Sophie Tison - début septembre 2005)

**Animation de la recherche :**

- Membre du comité de programmes de conférences internationales :
  - 10th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2003)
  - 2004 Annual Conference of the European Association for Computer Science Logic (CSL'04)
- Rapports de lecture pour divers journaux et conférences (5 dernières années) : ACM-TOCL - Theoretical Computer Science - Information & Computation - FOSSACS'05 - ESOP'05 - ICALP'04 - TACAS'04 - RTA'03 - CONCUR'03 - FOSSACS'03 - CSL'02 - RTA'02 - CSL'01 - PPDP'01.

**Participation à des actions de recherche :**

- membre de l'action incitative du CNRS ATIP jeunes Chercheurs "STAC" (Porteur : I. Ryl - LIFL)
- membre du projet INRIA MOSTRARE (Responsable : R. Gilleron - http://www.grappa.univ-lille3.fr/mostrare)
- participation à l'action concertée incitative *Masse de Données* "TRALALA : Langages pour la manipulation de documents XML : fondement et pratique (TRAnsformation LAnguages for XML : Logics and Applications)" (Porteur : G. Castagna - http://www.cduce.org/tralala.html ).

**Vie collective :**

- Membre suppléant élu de la commission de spécialistes de la section 27 de l'université des sciences et technologies de Lille (depuis février 2004).
- Membre du conseil d'administration de l'association "Société des Personnels Enseignants et Chercheurs en Informatique de France" - SPECIF (janvier 2002- décembre 2004) et organisateur du congrès SPECIF 2004 à Lille (70 participants).

# Annexe C

# Model Checking Mobile Ambient

Witold Charatonik, Silvano Dal Zilio, Andrew D. Gordon, Supratik Mukhopadhyay, and Jean-Marc Talbot. Model Checking Mobile Ambient. *Theoretical Computer Science*, 308(1-3) :277–331, 2003.

# Annexe D

# Finite-control Mobile Ambients

Witold Charatonik, Andrew D. Gordon, and Jean-Marc Talbot. Finite-control Mobile Ambients. In *European Symposium on Programming - ESOP 2002*, volume 2305 of *Lecture Notes in Computer Science*, pages 295–313. Springer, 2002.

# Annexe E

# When Ambients Cannot be Opened

Iovka Boneva and Jean-Marc Talbot. When Ambients Cannot be Opened. *Theoretical Computer Science*, 333(1-2) :127–169, 2005.

# Annexe F

# On Complexity of Model-Checking for the TQL Logic

Iovka Boneva and Jean-Marc Talbot. On Complexity of Model-Checking for the TQL Logic. In *3rd IFIP International Conference on Theoretical Computer Science (IFIP-TCS'04)*, pages 381–394. Kluwer, 2004.

# Annexe G

# Expressiveness of Spatial Logic for Trees

Iovka Boneva, Jean-Marc Talbot, and Sophie Tison. Expressiveness of Spatial Logic for Trees. In *Twentieth Annual IEEE Symposium on Logic in Computer Sience (LICS 2005)*, pages 280–289. IEEE Press, 2005.