THESE

présentée à

l'Université de la Méditerranée - Aix-Marseille II

Ecole Doctorale de Mathématiques et Informatique

par

Matthieu Martel

pour obtenir le grade de **DOCTEUR**

DE L'UNIVERSITÉ D'AIX-MARSEILLE II

spécialité : **INFORMATIQUE**

Analyse Statique et Evaluation Partielle de Systèmes de Processus Mobiles

---

Date de soutenance : 29 Novembre 2000

Composition du jury :

- Françoise BELLEGARDE, Prof., Université de Franche-Comté, Rapporteur
- Giovanni CORAY, Prof., Ecole Polytechnique Fédérale de Lausanne, Rapporteur
- Marc GENGLER, Prof., Université d'Aix-Marseille II, Directeur de thèse
- Michael GOLDSMITH, Prof., Oxford University, Examinateur
- Traian MUNTEAN, Prof., Université d'Aix-Marseille II, Président du jury

Table des matières

Chapitre 1

Introduction

Outre les performances, les garanties concernant la sûreté des logiciels, c.à.d. leur bon fonctionnement, deviennent peu à peu un critère majeur de qualité. Ces garanties sont apportées par l'utilisation de méthodes de preuve formelle permettant d'assurer que certaines propriétés sont vérifiées par un programme, indépendamment des données qui lui sont fournies, et, par conséquent, pour toutes ses exécutions. De telles vérifications sont déjà effectuées pour des systèmes critiques (c.à.d. dont une défaillance pourrait avoir des répercutions graves) comme les programmes gérant des moyens de transport (trains, avions, etc.) (BBFM99) ou pour les systèmes dont une défaillance peut avoir des conséquences économiques importantes, tels que les systèmes de cartes bancaires (CL99).

Avec l'émergence d'applications distribuées fonctionnant sur des réseaux de grande taille, tels qu'Internet, apparaissent de nouveaux problèmes concernant la vérification formelle des programmes dont les différents composants sont répartis sur plusieurs sites et échangent des informations. Aux propriétés de sûreté, pour lesquelles les techniques de preuve doivent être adaptées, il convient souvent d'ajouter des propriétés de sécurité, visant à garantir la confidentialité des données, comme c'est le cas pour les applications en rapport avec le commerce électronique (DNFP99).

1.1 Vérification formelle de programmes

Plusieurs approches sont possibles pour vérifier formellement le bon fonctionnement d'un programme, que l'on peut classer selon leur apparition chronologique dans le cycle de développement d'un logiciel (spécification, programmation, test, etc.).

Les méthodes fondées sur le raffinement, telles que FDR (Ros94) ou la méthode B (Abr96), employée par exemple pour garantir la sûreté des programmes régissant la ligne de métro Météor à Paris (BBFM99), consistent à spécifier de plus en plus précisément le

comportement d'un programme jusqu'à l'obtention du code final. En prouvant formellement que chaque raffinement est cohérent par rapport aux spécifications antérieures, la méthode garantit que le code final est conforme aux spécifications de départ.

Une autre approche, l'analyse statique, consiste à considérer un programme déjà écrit et à vérifier s'il satisfait certaines propriétés de bon fonctionnement pour toutes les exécutions possibles. Une telle vérification étant en général très complexe pour des propriétés non triviales, et par nature indécidable, plusieurs méthodes ont été proposées, dont la complexité varie en fonction de celle des propriétés à vérifier.

Les techniques de typage (MTHM97) et d'interprétation abstraite (CC77), en calculant une valeur approchée du résultat à partir de valeurs approchées pour les paramètres, permettent de vérifier la cohérence des résultats fournis par un programme. Des vérificateurs de types sont disponibles pour la plupart des langages de programmation et les techniques d'interprétation abstraite sont par exemple employées pour vérifier les programmes embarqués dans les fusées Ariane-5 (LJRDA98).

Les techniques de vérification de modèles (model checking) permettent de prouver le bon fonctionnement d'un système en calculant l'ensemble de tous ses états et en vérifiant la cohérence de chacun d'entre eux. Cela permet éventuellement d'extraire des traces d'exécution menant le système dans un état erroné. Par exemple, le vérificateur de modèle SPIN (Hol97) a été employé pour vérifier la correction de protocoles de communications (Hol99). Cependant, le problème majeur de cette approche est l'explosion combinatoire du nombre d'états à vérifier, même si celui-ci peut être limité, par exemple par l'utilisation de techniques de repliage utilisant des symétries (BDH00).

1.2 Evaluation partielle

L'évaluation partielle est une méthode consistant à optimiser un programme en fonction d'une partie de ses données pour obtenir un nouveau programme spécialisé par rapport à ces dernières (JGS93). Le fait que cette transformation repose sur des règles sémantiques formelles assure que le programme spécialisé auquel on fournit la partie des données absente lors de l'évaluation partielle produit le même résultat que le programme initial exécuté avec le même ensemble de données.

Ainsi, étant donné un programme pour lequel certaines garanties de sûreté ont été apportées, l'évaluation partielle permet d'obtenir automatiquement des versions spécialisées de celui-ci, optimisées pour certains contextes d'exécution et vérifiant les mêmes propriétés de sûreté que le programme initial auquel on fournit les mêmes données que lors de la spécialisation. Par exemple, prouver la correction d'un protocole de communication est souvent une tâche complexe et développer des versions de celui-ci optimi-

sées par rapport à certaines conditions d'utilisation demande une révision complète des preuves, ce qui est en général prohibitif. De telles versions optimisées peuvent être obtenues par évaluation partielle, comme cela a été fait pour le protocole RPC (Remote Procedure Call (MVM97)).

Une autre application de l'évaluation partielle est la possibilité de générer automatiquement, par auto-application, un générateur de compilateurs à partir d'un évaluateur partiel (Fut71). Ce générateur prend en entrée un interpréteur décrivant la sémantique d'un langage et produit un compilateur pour ce langage, vers le langage produit par l'évaluateur partiel. La simple correction de ce dernier est suffisante pour garantir celle du générateur et des compilateurs obtenus.

Assurer la correction d'un compilateur est en général une tâche difficile, rarement accomplie, et générer un code final correct à partir d'un programme écrit dans un langage de haut niveau et dont on a vérifié la sûreté est une étape importante souvent mise entre parenthèses. Des générateurs de compilateurs ont été obtenus par évaluation partielle, principalement pour des langages fonctionnels séquentiels (JGS93), et cette technique semble être une approche intéressante à ce problème.

Généralement, les programmes fournis à un évaluateur partiel sont annotés de façon à indiquer quelles parties du code peuvent être évaluées statiquement et lesquelles doivent être résidualisées, c.à.d. reconstruites dans le programme spécialisé. Ces annotations sont produites automatiquement par une analyse statique du programme source, appelée analyse des temps de liaison. D'autres analyses peuvent être employées afin de fournir des indications supplémentaires à l'évaluateur partiel, ou afin d'améliorer la précision de l'analyse des temps de liaison, telles que les analyses de flot de contrôle.

1.3 Systèmes de processus mobiles

Les systèmes de processus mobiles permettent de décrire des applications composées d'un nombre de processus variant au cours du temps, et communiquant entre eux à travers un réseau reconfigurable dynamiquement. Ces systèmes permettent de représenter la plupart des applications distribuées actuelles, telles que celles utilisant le réseau Internet, les systèmes de cartes à puces ou de téléphonie mobile.

Dans les modèles permettant de décrire de tels systèmes, la mobilité est généralement modélisée par la possibilité de créer dynamiquement et de transmettre des noms de canaux, ce qui permet de créer et supprimer des liens de communication en cours d'exécution (Mil93). De plus, ces modèles disposent de mécanismes pour implanter les créations et suppressions dynamiques de processus.

Enfin, ces modèles peuvent être classés selon qu'ils proposent des primitives de communication synchrones, ce qui veut dire que les émissions et les réceptions sont bloquantes, ou asynchrones. Dans ce dernier cas, on peut distinguer les modèles dans lesquels l'ordre des messages est préservé, c.à.d. que le premier message envoyé sur un canal est celui qui est lu par la première réception sur ce canal, de ceux ne préservant pas l'ordre des messages. Le choix d'un modèle dépend de l'application à implanter ou du niveau d'abstraction auquel on se place. Généralement, des communications asynchrones ne préservant pas l'ordre des messages sont implantées dans les protocoles des couches basses des réseaux (Tan96), tandis que des applications de plus haut niveau utilisent des communications synchrones.

Parmi les langages qui ont été proposés pour décrire des systèmes de processus mobiles, le π-calcul (Mil93) est un langage formel permettant de représenter des applications et de décrire leurs interactions de manière algébrique, à travers de nombreuses relations d'ordre et d'équivalence. Des versions synchrones et asynchrones de ce langage ont été proposées.

Concurrent ML (Rep99) est une extension du langage ML (MTHM97) offrant un modèle de parallélisme proche de celui du π-calcul. Des bibliothèques de communication telles que PVM (GBD$^+$94), utilisable à partir des langages C ou Java, et conçues initialement pour le calcul numérique, permettent de décrire des systèmes de processus mobiles.

1.4 Analyse statique et évaluation partielle de systèmes de processus mobiles

Si les techniques d'analyse statique et d'évaluation partielle de programmes séquentiels, principalement écrits dans des langages fonctionnels, ont été largement étudiées, peu de travaux ont été consacrés aux programmes décrivant des applications distribuées. Or les langages parallèles introduisent de nouvelles instructions pour la gestion des processus et de leurs interactions, pour lesquelles il est nécessaire de définir des méthodes spécifiques.

L'objet de cette thèse est l'étude des techniques d'analyse statique et d'évaluation partielle de programmes composés de processus mobiles.

Dans ce contexte, l'évaluation partielle permet de supprimer statiquement les communications d'un programme pour lesquelles on connaît le contenu du message, ainsi que l'émetteur et le récepteur, ce qui implique de connaître les lieux d'utilisation de chaque canal dans le code des processus. On obtient ainsi un programme spécialisé réalisant moins de communications que le programme initial, ce qui peut invalider la cor-

rection de la version spécialisée si des communications avec des processus extérieurs au système traité ont été supprimées. En effet, dans ce cas, le système initial et la version réduite peuvent être distinguées l'un de l'autre. Nous étudions l'impact de l'évaluation partielle des communications sur le comportement des systèmes vis-à-vis de l'extérieur et définissons les cas dans lesquels cette technique peut être appliquée, ainsi que les relations existant entre les systèmes initiaux et réduits.

Par ailleurs, comme cela a été mentionné à la section 1.2, la génération de compilateurs par auto-application d'un évaluateur partiel est un résultat important. Nous montrons qu'il est possible de définir des évaluateurs partiels auto-applicables pour des langages décrivant des systèmes de processus mobiles et nous présentons les compilateurs générés à partir de nos implantations.

Enfin, l'évaluation partielle de systèmes de processus mobiles repose sur la connaissance des lieux précis d'utilisation des canaux créés par une application ainsi que sur la connaissance des communications pouvant être réalisées par celle-ci. Aussi, nous définissons une analyse de flot de contrôle calculant une approximation de la topologie des communications réalisées par les programmes. Si cette analyse est utile pour l'évaluation partielle, elle peut par ailleurs être utilisée en tant que telle pour vérifier des propriétés de sûreté et de sécurité pour des systèmes distribués. Nous illustrons ceci sur différents exemples et montrons comment les résultats de l'analyse peuvent être utilisés pour assurer de telles propriétés.

Les caractéristiques des langages que nous considérons sont les suivantes. Premièrement, ils disposent d'instructions de communication synchrones (les émissions et les réceptions sont bloquantes) sur des canaux. Cependant, les résultats présentés dans ce document sont directement applicables à tout autre modèle assurant que les messages communiqués sont reçus dans l'ordre dans lequel ils ont été émis.

Ce modèle, qui est utilisé par la plupart des langages de haut niveau contraint le contrôle des instructions de communication et permet de construire des techniques d'analyse statique et d'évaluation partielle d'un plus grand intérêt que pour un modèle de communications ne préservant pas l'ordre des messages. En effet, dans ce dernier cas, le comportement d'une application est entièrement dépendant de son exécution et son comportement peut difficilement être prédit de manière statique.

La seconde caractéristique des langages traités dans ce document est la présence d'instructions pour la création dynamique de nouveaux processus. Cette fonctionnalité complique sensiblement l'analyse des programmes, le nombre de processus n'étant pas établi a priori, mais correspond à un mécanisme utilisé par de nombreuses applications et qu'il est important de prendre en compte.

Ensuite, ces langages permettent de créer dynamiquement des nouveaux noms de canaux ainsi que leur communication. Ces caractéristiques, qui permettent de modéliser la mobilité des processus (Mil93), induisent plusieurs difficultés tant pour l'analyse statique que pour la spécialisation. Non seulement les canaux utilisés par un programme ne sont pas connus à l'avance, mais de plus, les instructions de communication les utilisant ne sont pas données explicitement. Concernant l'analyse statique cela impose de déterminer le plus précisément possible les communications pouvant être réalisées par un programme. Par ailleurs, cela complique les preuves de correction des évaluateurs partiels qui consistent à comparer le comportement des programmes initiaux à ceux des programmes spécialisés (notions d'observabilité).

1.5 Contribution

Tout d'abord, nous nous intéressons à l'évaluation partielle auto-applicable pour le π-calcul. Dans un premier temps, nous définissons un méta-interpréteur pour ce langage, puis nous introduisons la notion de bonne annotation des programmes afin de déterminer les opérations (communications) pouvant être évaluées statiquement. Finalement, nous introduisons un évaluateur partiel auto-applicable prenant pour entrée des π-termes bien annotés et nous prouvons sa correction vis-à-vis de l'interpréteur. Les preuves de correction sont fondées sur la notion de congruence barbelée faible.

Cette étude permet de mettre en évidence qu'il est possible d'écrire des évaluateurs partiels auto-applicables pour des langages contenant des instructions de communication et de formaliser les relations existant entre les programmes initiaux et réduits. De plus, à travers la notion de bonne annotation des termes, nous spécifions les conditions nécessaires pour qu'une communication puisse être réduite au moment de l'évaluation partielle. Notamment, ces conditions imposent que le canal utilisé lors d'une communication statique soit défini localement dans le programme et jamais exporté hors de celui-ci lors de son exécution.

Ensuite, nous proposons une analyse statique permettant de calculer une approximation de la topologie des communications réalisées par des programmes écrits en Concurrent ML. Ces programmes peuvent contenir des créations dynamiques de noms de canaux ou de processus, et peuvent communiquer des fonctions. L'analyse proposée est une analyse de flot de contrôle qui construit un automate d'états finis pour chacun des processus du système afin de connaître l'ordre d'exécution des instructions de communication. Nous calculons la topologie des communications à partir du produit des automates précédents.

Nous introduisons une version réduite de l'automate produit, dont la taille est po-

lynômiale, et nous prouvons que l'on obtient des annotations correctes en substituant l'automate réduit à l'automate produit. Enfin, nous montrons comment générer automatiquement un système de contraintes en fonction du code d'un programme de telle manière qu'une solution à ce système soit une analyse correcte pour le programme. Nous présentons un algorithme permettant de trouver la plus petite solution pour ce type de systèmes.

Enfin, nous définissons un méta-interpréteur puis un évaluateur partiel auto-applicable pour un noyau non typé du langage Concurrent ML. Nous définissons aussi une analyse des temps de liaison permettant d'annoter les programmes fournis à l'évaluateur partiel et qui utilise les résultats fournis par l'analyse de flot de contrôle décrite précédemment. Ces différentes analyses ainsi que l'évaluateur partiel ont été implantés et nous obtenons les résultats suivants.

D'une part, l'analyse des temps de liaison permet d'annoter l'évaluateur partiel en vue des projections de Futamura et, d'autre part, nous obtenons par auto-application un compilateur pour notre langage ainsi qu'un générateur de compilateurs. On démontre ainsi que les résultats classiques établis pour les langages fonctionnels séquentiels restent valides pour les langages que nous considérons. Enfin, nous montrons comment notre évaluateur partiel peut être utilisé pour spécialiser un protocole de communication en fonction de certaines hypothèses sur le fonctionnement du réseau.

1.6 Plan de ce document

Le reste de ce document est organisé de la manière suivante. Le chapitre 2 introduit les formalismes et les techniques que nous utilisons par la suite, et décrit les autres travaux ayant été réalisés sur le sujet. Nous donnons les sémantiques formelles du π-calcul et de Concurrent ML, les langages que nous utilisons pour décrire des systèmes de processus mobiles. Nous définissons aussi les analyses de flot de contrôle et des temps de liaison pour les langages auxquels nous nous intéressons. Finalement nous présentons les techniques d'évaluation partielle et de génération de compilateurs.

Le chapitre 3 est consacré à l'évaluation partielle auto-applicable pour le π-calcul. Au chapitre 4, nous présentons l'analyse de flot de contrôle calculant une approximation de la topologie des communications réalisées par les programmes afin d'accroître sa précision. Le chapitre 5 est consacré à l'évaluation partielle de programmes écrits en Concurrent ML. Le chapitre 6 conclut.

Chapitre 2

Notions préliminaires

Dans ce chapitre, nous présentons les notions utilisées dans le reste de ce document. La section 2.1 décrit le π-calcul et Concurrent ML, les langages auxquels nous nous intéressons par la suite. La section 2.2 présente les techniques d'analyse statique que nous utilisons, insistant plus particulièrement sur les analyses de flot de contrôle et des temps de liaison. Enfin, la section 2.3 est consacrée à l'évaluation partielle.

2.1 Description de programmes distribués

Nous présentons ici le π-calcul et Concurrent ML, les langages utilisés dans ce document pour décrire des programmes distribués. Tous deux utilisent des primitives de communications *synchrones* (ou *par rendez-vous*), ce qui veut dire qu'un processus qui réalise une émission ou une réception reste bloqué tant que la communication n'a pas eu lieu.

Succédant à des langages tels que CCS (Mil89), le π-calcul (Mil93; Mil99) permet de tenir compte de la mobilité des composants d'un programme distribué, principalement en autorisant la communication de noms de canaux. Notre choix s'est porté sur ce langage, notamment à cause des nombreuses relations d'équivalence entre processus qui ont été étudiées dans ce contexte. Comme l'évaluation partielle d'un programme distribué supprime certaines communications, nous étudierons les conséquences de cette transformation sur le comportement du programme en utilisant ces équivalences.

Concurrent ML (Rep99) est une version du langage ML (MTHM97) étendue par des primitives permettant de gérer le parallélisme. Nous avons choisi ce langage car il est muni d'une sémantique opérationnelle simple et il offre un modèle de parallélisme proche de celui du π-calcul. Ce langage est décrit à la section 2.1.2.

2.1.1 Le pi-calcul

Nous présentons ici une version simple du π-calcul, le π-calcul monadique du premier ordre (Mil93). De nombreuses déclinaisons de ce langage ont été étudiées, en particulier le π-calcul polyadique (Mil93), le π-calcul d'ordre supérieur (San92), le π-calcul asynchrone (Bou92; HT91) ou le $b\pi$-calcul pour les communications de groupe (EM99). Les programmes sont des π-termes générés par la grammaire suivante :

$$P ::= \sum_{i=0}^{n} \pi_i.P_i \mid P\|Q \mid !P \mid (\boldsymbol{\nu} x)P \qquad \text{avec} \quad \pi_i ::= \overline{x}[y] \mid x(y) \tag{2.1}$$

Les *noms de canaux* sont les seules valeurs présentes dans le langage. Ils servent à indiquer à la fois le canal utilisé pour une communication entre deux processus et la valeur transmise lors de celle-ci. Nous utilisons indifféremment des lettres grecques ou romaines pour nommer les canaux. $\overline{x}[y]$ et $x(y)$ représentent respectivement l'émission d'un nom y sur un canal x et la réception sur le canal x d'un nom formellement nommé y. Le nom x utilisé dans les deux communications précédentes s'appelle le *sujet* de la communication. Dans $\overline{x}[y]$, y est l'*objet* de l'émission.

$P\|Q$ représente l'exécution parallèle des processus P et Q. Le terme $!P$ représente la *réplication* de P, c.à.d. $P\|P\|P\|\ldots$ avec un nombre strictement positif de copies de P. La *restriction*, notée $(\boldsymbol{\nu} x)$, permet à un processus de créer un nouveau nom de canal x, différent de tous les noms déjà définis. $\sum_{i=0}^{n} \pi_i.P_i$ représente le choix non-déterministe entre les processus. Remarquons que les sous-termes d'un choix commencent obligatoirement par une communication. $\alpha(x).P$ et $(\boldsymbol{\nu} x)P$ sont les deux manières de lier les occurrences d'une variable x dans un terme P. Sinon, x est *libre* dans P. Si $\rightarrow$ est la relation de réduction, la sémantique du choix est donnée grâce à l'équation (2.2).

$$\left(\ldots + \ldots + \overline{\alpha}[x].P\right) \parallel \left(\ldots + \ldots + \alpha(y).Q\right) \longrightarrow P \parallel (Q\{y \leftarrow x\}) \tag{2.2}$$

$Q\{y \leftarrow x\}$ représente le terme Q dans lequel les occurrences libres de la variable y ont été remplacées par x. On a aussi les autres règles de réduction suivantes :

$$\frac{P \rightarrow P'}{P\|Q \rightarrow P'\|Q} \qquad \frac{Q \rightarrow Q'}{P\|Q \rightarrow P\|Q'} \qquad \frac{P \rightarrow P'}{(\boldsymbol{\nu} x)P \rightarrow (\boldsymbol{\nu} x)P'} \tag{2.3}$$

Notons qu'aucune réduction n'a lieu derrière une communication tant que celle-ci n'a pas été réduite. La fermeture réflexive et transitive de $\rightarrow$ est notée $\rightarrow^*$. Nous utilisons les abréviations syntaxiques suivantes. $x(y_1 \ldots y_n)$ représente $x(y_1)\ldots x(y_n)$, $\overline{x}[y_1 \ldots y_n]$ représente $\overline{x}[y_1]\ldots\overline{x}[y_n]$ et $(\boldsymbol{\nu} x_1 \ldots x_n)$ représente $(\boldsymbol{\nu} x_1)\ldots(\boldsymbol{\nu} x_n)$.

0 est le processus vide, c.à.d. le choix non-déterministe entre zéro processus. x et $\overline{x}$ sont des abréviations pour $x()$ et $\overline{x}[]$, c.à.d. la réception et l'émission de messages vides,

aussi appelés signaux. Formellement, $\overline{x}.P$ est utilisé au lieu de $(\boldsymbol{\nu} z)\overline{x}[z].P$ où z n'est pas libre dans P et $x().P$ est un abréviation pour $x(z).P$ avec z non libre dans P.

Dans ce qui suit, nous définissons deux relations classiques de congruence sur l'ensemble des processus, ainsi qu'une notion de simulation (BS98; MS92). La congruence structurelle, notée $\equiv$, identifie les termes dont la syntaxe diffère mais qui ont le même sens (Mil93; MS92). Deux termes structurellement congruents agissent identiquement dans tous les contextes. Il est possible de les substituer les uns aux autres sans modifier le comportement du programme. Rappelons qu'une occurrence d'une variable x est libre si elle ne se trouve pas sous la portée d'une restriction $(\boldsymbol{\nu} x)$ ou d'un préfixe $\alpha(x)$.

Définition 1 (Congruence structurelle) *La congruence structurelle $\equiv$ est la plus petite relation de congruence sur les termes telle que*

(i) $\equiv$ *contient la relation d'alpha-équivalence,*

(ii) $\|$ *est associatif, commutatif et admet* $\mathbf{0}$ *comme élément neutre,*

(iii) $!P \equiv P\|!P$,

(iv) $(\boldsymbol{\nu} x)\mathbf{0} \equiv \mathbf{0}$ *et* $(\boldsymbol{\nu} x)(\boldsymbol{\nu} y)P \equiv (\boldsymbol{\nu} y)(\boldsymbol{\nu} x)P$,

(v) $(\boldsymbol{\nu} x)(P\|Q) \equiv P\|(\boldsymbol{\nu} x)Q$ *si* P *ne contient pas d'occurrence libre de* x. □

La relation de réduction $\rightarrow$ est naturellement étendue par la règle ci-dessous.

$$\frac{P \equiv Q \quad Q \rightarrow Q' \quad Q' \equiv P'}{P \rightarrow P'} \tag{2.4}$$

De plus, remarquons que $(\boldsymbol{\nu} x)P \equiv P$ si x n'est pas libre dans P. La seconde congruence, notée $\approx$, est construite à partir de la bisimulation barbelée faible $\approx_b$ (MS92; San92). $\approx$ identifie les termes qui apparaissent identiques d'un point de vue extérieur au programme. Si $P \approx Q$, alors aucun processus R ne peut distinguer P de Q en communicant avec l'un ou l'autre de ces termes. Nous présentons ci-dessous les notions de processus non gardé et d'observabilité qui sont nécessaires à la définition de $\approx_b$ et $\approx$.

Définition 2 (Processus non gardé et observabilité) *Soit P et Q deux processus et x un nom de canal.*

- *Q apparaît non gardé dans P si Q possède une occurrence dans P qui n'est pas sous un préfixe π.*
- *Q apparaît non restreint sur le nom de canal x dans P si Q possède une occurrence dans P qui n'est pas sous une restriction $(\boldsymbol{\nu} x)$.*
- *P est* observable *en x, noté $P{\downarrow_x}$, s'il existe des termes Q et π, où x est le sujet de π, tel que $\pi.Q$ apparaît dans P, non gardé et non restreint sur x.* □

Par exemple, pour $P \widehat{=} (x(y).Q_1 \| Q_2)$, nous avons $P\downarrow_x$, car P peut immédiatement communiquer sur x. De même, $((\boldsymbol{\nu} y)x(y).P)\downarrow_x$, si $x \neq y$. Par contre, $(x(y).u(v).P)\not\downarrow_u$ si $x \neq u$, car aucune communication ne peut survenir sur u avant que $x(y)$ ne soit réduit. Enfin, pour $P \widehat{=} ((\boldsymbol{\nu} x)x(y).P)$, $P\not\downarrow_x$, même si $x(y)$ est non gardé, car x est un canal restreint, inconnu en dehors de P. La bisimulation barbelée faible définie ci-dessous se fonde sur la notion d'observabilité.

Définition 3 (Bisimulation barbelée faible) *La bisimulation barbelée faible $\approx_b$ est la plus grande relation d'équivalence $\mathcal{R}$ sur l'ensemble des processus telle que $\mathcal{R}(P, Q)$ implique*

(i) $(P \rightarrow P') \Rightarrow (\exists Q' \,:\, (Q \rightarrow^* Q' \text{ et } \mathcal{R}(P', Q')))$,

(ii) $\forall x,\ (P \downarrow_x \Rightarrow (\exists Q' \,:\, (Q \rightarrow^* Q' \text{ et } Q' \downarrow_x)))$. □

La bisimulation barbelée faible identifie les termes qui demeurent observables sur les mêmes canaux après réduction. Cependant, cette propriété n'est pas conservée par composition de processus. Par exemple, $ab \approx_b ac$ mais $ab\|\overline{a} \not\approx_b ac\|\overline{a}$. Aussi, nous définissons la relation de congruence barbelée faible $\approx$ qui identifie les termes restant $\approx_b$-équivalents par composition.

Définition 4 (Congruence barbelée faible) *Soit $C[]$ un contexte de processus, c.à.d. un $\boldsymbol{\pi}$-terme avec un seul trou, tel que placer un processus dans le trou forme un nouveau processus bien formé. Deux processus P et Q sont congruents au sens de la congruence barbelée faible, ce qui est noté $P \approx Q$, si pour tout contexte $C[]$, $C[P] \approx_b C[Q]$.* □

Pour reprendre l'exemple précédent, $ab \approx_b ac$ mais $ab \not\approx ac$ (le contexte utilisé ici est $C[] = []\|\overline{a}$). Finalement, nous définissons la simulation barbelée simple $\prec$ qui se fonde aussi sur l'observabilité.

Définition 5 (Simulation barbelée faible) *La simulation barbelée faible $\prec$ est le plus grand pré-ordre $\mathcal{R}$ sur l'ensemble des processus tel que $\mathcal{R}(P, Q)$ implique*

(i) $(P \rightarrow P') \Rightarrow (\exists Q' \,:\, (Q \rightarrow^* Q' \text{ et } \mathcal{R}(P', Q')))$,

(ii) $\forall x,\ (P \downarrow_x \Rightarrow (\exists Q' \,:\, (Q \rightarrow^* Q' \text{ et } Q' \downarrow_x)))$. □

$P \prec Q$ exprime le fait que l'ensemble des comportements de P est un sous-ensemble de l'ensemble de ceux de Q. Par exemple, $\beta \prec \alpha\beta\|\overline{\alpha}$. Remarquons que $\prec$ est stable par composition de processus, c.à.d. que si $P \prec Q$ alors pour tout processus R, $P\|R \prec Q\|R$.

La simulation barbelée faible $\prec$ et la congruence barbelée faible $\approx$ sont les deux relations que nous utilisons pour prouver la correction des programmes évalués partiellement par rapport aux programmes originaux. Dans le reste de cette section, nous justifions rapidement ces choix. Si P est le programme initial et P_r le programme résiduel

résultant de l'évaluation partielle de P, nous disons que P_r est correct si $P_r \approx P$ ou $P_r \prec P$, selon le point de vue que l'on adopte.

Tout d'abord, les relations de simulation et de congruence barbelées fortes $\prec_s$ et $\approx_s$, (Mil93) sont obtenues en substituant $\rightarrow$ à $\rightarrow^*$ dans les définitions 3 et 5. Pour utiliser $\prec_s$ et $\approx_s$, nous devrions avoir une correspondance pas à pas entre les réductions effectuées dans P et P_r. Ceci n'est clairement pas désirable, puisque nous souhaitons supprimer les communications statiques à l'évaluation partielle. Aussi avons-nous besoin d'une notion faible d'équivalence ou de simulation.

Ensuite, la bisimulation barbelée faible $\approx_b$ identifie des termes qui se comportent différemment en fonction de leur contexte. A nouveau, ceci n'est pas le comportement que nous souhaitons voir adopter par P_r vis à vis de P, ce qui nous amène à utiliser la congruence induite par $\approx_b$.

Si $P \approx P_r$ alors les processus P et P_r ne peuvent pas être distingués par un processus externe Q qui interagit avec eux. Ceci est encore une notion forte d'équivalence. Remarquons que P_r ne réalise pas nécessairement autant de pas de réduction que P, car certaines communications internes à P peuvent avoir été réduites dans P_r.

$P_r \prec P$ offre une notion plus faible de correction, en affirmant simplement que toutes les communications observables pour P_r le sont aussi pour P. Cependant, P peut être observable sur d'autres canaux. Par exemple, considerons $P \mathrel{\widehat{=}} (\nu a)(a.c + b \| \overline{a})$ et $P_r \mathrel{\widehat{=}} c$. Le terme P_r est un terme résiduel souhaitable car P peut se réduire à P_r sans interagir avec un environnement extérieur. Mais P_r n'est pas observable en b alors que P l'est, et par conséquent $P_r \not\approx P$. Le non-determinisme est réduit pour $P_r \| Q$ par rapport à $P \| Q$ pour tout processus externe Q. En conséquence, Q peut faire la distinction entre P et P_r. Cependant, à cause du non-déterminisme, les séquences d'opérations ainsi que l'état final de $P \| Q$ ne sont pas uniques en général et il n'y a pas de raison de privilégier une séquence d'action et son état final. Si $P_r \prec P$ alors les séquences d'operations possibles pour $P_r \| Q$ le sont aussi pour $P \| Q$, le programme initial P aurait pu se comporter comme P_r et en conséquence $P \| Q$ comme $P_r \| Q$.

2.1.2 Concurrent ML

Dans cette section, nous présentons un sous-ensemble du langage Concurrent ML (BMT92; NN99; Rep91; Rep92; Rep99). Ce langage est une extension de ML (MTHM97) offrant un modèle de parallélisme proche de celui que nous avons vu précédemment pour le π-calcul. Concurrent ML permet la création dynamique de processus et de noms de canaux et est muni de primitives de communication synchrones. Concurrent ML étant un langage d'ordre supérieur, les fonctions sont des valeurs qui peuvent être communiquées. Ceci représente une forme de mobilité du code. Tout comme dans le π-calcul, les

noms de canaux sont des valeurs et peuvent être communiqués.

$$
\begin{array}{rcl}
\text{Expr} \ni e & ::= & v \mid x \mid e_0\ e_1 \mid \texttt{if}\ e_0\ e_1\ e_2 \mid \texttt{let}\ x = e_0\ \texttt{in}\ e_1 \\
& & \mid \texttt{channel()} \mid \texttt{fork}\ e_0 \mid \texttt{send}\ e_0\ e_1 \mid \texttt{receive}\ e_0 \\
\\
\text{Context} \ni E & ::= & [] \mid E\ e_1 \mid v\ E \mid \texttt{if}\ E\ e_1\ e_2 \mid \texttt{let}\ x = E\ \texttt{in}\ e_1 \\
& & \mid \texttt{send}\ E\ e_1 \mid \texttt{send}\ v\ E \mid \texttt{receive}\ E \\
\\
\text{Val} \ni v & ::= & B \mid \texttt{fun}\ x\ \texttt{=>}\ e_0 \mid \texttt{rec}\ f\ x\ \texttt{=>}\ e_0 \\
\\
\text{Base} \ni B & ::= & \texttt{()} \mid i \mid b \mid k \\
\\
\text{Unit} \ni \texttt{()} & & \\
\text{Int} \ni i & ::= & \ldots -1 \mid 0 \mid 1 \ldots \\
\text{Bool} \ni b & ::= & \texttt{true} \mid \texttt{false} \\
\text{Chan} \ni k & ::= & k_0 \mid k_1 \ldots \\
\\
\text{ProcPool} \ni P & ::= & \langle p : e \rangle \mid P :: P' \\
\text{ProcName} \ni p & ::= & p \mid q \ldots
\end{array}
$$

FIG. 2.1 – **Syntaxe de Concurrent ML.**

Le langage, dont la définition est donnée dans la figure 2.1, contient des expressions conditionnelles, l'instruction `let` et un opérateur `rec` pour la construction de fonctions récursives. `channel()` décrit un appel à une fonction qui retourne un nouveau nom de canal k, différent de tous les noms existants. `fork` e_0 crée un nouveau processus pour calculer e_0. `send` e_0 e_1 décrit l'émission de la valeur de e_1 sur le canal résultant de l'évaluation de e_0. e_0 est le *sujet* de la communication, et e_1 son *objet*. `receive` e_0 est la réception d'une valeur sur le nom de canal décrit par e_0 (le sujet de la réception). Les valeurs sont dans le domaine des types de base (comprenant `()` de type `unit`), ou des noms de canaux, ou des fonctions.

La notation $\langle p_1 : e_1 \rangle :: \langle p_2 : e_2 \rangle :: \ldots \langle p_n : e_n \rangle$ est utilisée pour décrire un groupe de n processus $p_1,\ p_2, \ldots p_n$ tel que e_i est l'expression calculée par le processus p_i, $1 \leq i \leq n$. La figure 2.2 présente un programme écrit en Concurrent ML. Il s'agit d'un système composé de deux processus p_1 et p_2 échangeant un message découpé en paquets. p_1 transmet tout d'abord à p_2 le nombre de paquets à transmettre puis construit et envoie les différents paquets. Le processus p_2 reçoit au cours d'une première communication le nombre de paquets, réalise le nombre correspondant de réceptions, et enfin reconstitue le message. Au chapitre 5, nous montrons comment spécialiser ce programme pour le cas particulier dans lequel la taille des messages, et par conséquent, le nombre de paquets sont connus statiquement, mais pas le contenu des messages.

Pour le besoin des analyses de flot de contrôle et des temps de liaison définies aux cha-

```
p1 : let foo = send γ size
     in (rec f m s =>
           if s=1 then
             send γ (head m)
           else
             let foo = send γ
                               (head m)
             in f (tail m) (s-1)
         ) msg size

p2 : let size = receive γ
     in (rec g s =>
           if s=1 then
             receive γ
           else
             let h= receive γ
             in append h (g (s-1))
         ) size
```

FIG. 2.2 – **Exemple de programme écrit en Concurrent ML : protocole de communication de messages découpés en paquets.**

pitres 4 et 5, nous annotons les différentes expressions d'un programme par des étiquettes appartenant à l'ensemble LAB. Ces analyses supposent que chaque sous-expression du programme initial possède une étiquette unique (cette propriété n'est pas maintenue par réduction). La grammaire donnée dans la figure 2.1 se réécrit alors comme décrit à la figure 2.3.

$$
\begin{array}{rcl}
\text{Expr} \ni e & ::= & v^l \mid x^l \mid (e_0^{l_0}\ e_1^{l_1})^l \mid (\texttt{if}\ e_0^{l_0}\ e_1^{l_1}\ e_2^{l_2})^l \mid (\texttt{let}\ x^{l_2} = e_0^{l_0}\ \texttt{in}\ e_1^{l_1})^l \\
& & \mid \texttt{channel()}^l \mid (\texttt{fork}\ e_0^{l_0})^l \mid (\texttt{send}\ e_0^{l_0}\ e_1^{l_1})^l \mid (\texttt{receive}\ e_0^{l_0})^l \\
\text{Context} \ni E & ::= & [] \mid (E\ e_1^{l_1})^l \mid (v^{l_0}\ E)^l \mid (\texttt{if}\ E\ e_1^{l_1}\ e_2^{l_2})^l \mid (\texttt{let}\ x^{l_2} = E\ \texttt{in}\ e_1^{l_1})^l \\
& & \mid (\texttt{send}\ E\ e_1^{l_1})^l \mid (\texttt{send}\ v^{l_0}\ E)^l \mid (\texttt{receive}\ E)^l \\
\text{Val} \ni v & ::= & B \mid \texttt{fun}\ x^{l_1} \texttt{=>}\ e_0^{l_0} \mid \texttt{rec}\ f^{l_1}\ x^{l_2} \texttt{=>}\ e_0^{l_0} \\
\text{Base} \ni B & ::= & \texttt{()} \mid i \mid b \mid k \\
\text{Unit} \ni \texttt{()} & & \\
\text{Int} \ni i & ::= & \ldots -1 \mid 0 \mid 1 \ldots \\
\text{Bool} \ni b & ::= & \texttt{true} \mid \texttt{false} \\
\text{Chan} \ni k & ::= & k_0 \mid k_1 \ldots \\
\text{ProcPool} \ni P & ::= & \langle p : e \rangle \mid P :: P' \\
\text{ProcName} \ni p & ::= & p \mid q \ldots
\end{array}
$$

FIG. 2.3 – **Syntaxe de Concurrent ML dans laquelle les expressions sont étiquetées.**

La sémantique opérationnelle, donnée dans la figure 2.4 pour les termes étiquetés, reprend celle du langage λ_{cv} defini par Reppy (Rep91). $\hookrightarrow$ définit les pas de réduction séquentiels. $e_0^{l_0}\{x \leftarrow e_1^{l_1}\}$ représente le terme obtenu en supprimant les étiquettes des occurrences de x dans $e_0^{l_0}$ puis en substituant $e_1^{l_1}$ à x.

$\xrightarrow{[\ell]}$ décrit les pas de réduction parallèle. Ces pas de réduction sont annotés par des étiquettes $\ell \in \text{LAB}^2 \cup \{\varepsilon\}$ qui seront uniquement utilisées dans les preuves de correc-

tion de l'analyse de flot de contrôle présentée au chapitre 4. Un pas étiqueté ε correspond à un pas de réduction séquentiel dans un des processus du programme. Le pas de réduction correspondant à une communication entre deux points étiquetés l_s et l_r est annoté (l_s, l_r) et celui correspondant à la création d'un nouveau processus par une instruction `fork` est annoté (l_f, l_f) où l_f est l'étiquette du `fork`. Comme Reppy (Rep91), nous utilisons des contextes d'évaluation E définis dans la figure 2.1.

$$\frac{e_0^{l_0} \hookrightarrow e_2^{l_2}}{(e_0^{l_0}\ e_1^{l_1})^l \hookrightarrow (e_2^{l_2}\ e_1^{l_1})^l} \qquad \frac{e_1^{l_1} \hookrightarrow e_2^{l_2}}{(v^{l_0}\ e_1^{l_1})^l \hookrightarrow (v^{l_0}\ e_2^{l_2})^l}$$

$$((\texttt{fun}\ x^{l_0} \texttt{=>}\ e_1^{l_1})^{l_2}\ v^{l_3})^l \hookrightarrow e_1^{l_1}\{x \leftarrow v^{l_3}\}$$

$$(\texttt{rec}\ f^{l_0}\ x^{l_1} \texttt{=>}\ e_2^{l_2})^l \hookrightarrow (\texttt{fun}\ x^{l_1} \texttt{=>}\ e_2^{l_2}\{f \leftarrow (\texttt{rec}\ f^{l_0}\ x^{l_1} \texttt{=>}\ e_2^{l_2})^l\})^l$$

$$\frac{e_0^{l_0} \hookrightarrow e_3^{l_3}}{(\texttt{if}\ e_0^{l_0}\ e_1^{l_1}\ e_2^{l_2})^l \hookrightarrow (\texttt{if}\ e_3^{l_3}\ e_1^{l_1}\ e_2^{l_2})^l}$$

$$(\texttt{if}\ \texttt{true}^{l_0}\ e_1^{l_1}\ e_2^{l_2})^l \hookrightarrow e_1^{l_1} \qquad (\texttt{if}\ \texttt{false}^{l_0}\ e_1^{l_1}\ e_2^{l_2})^l \hookrightarrow e_2^{l_2}$$

$$\frac{e_0^{l_0} \hookrightarrow e_3^{l_3}}{(\texttt{let}\ x^{l_2} = e_0^{l_0}\ \texttt{in}\ e_1^{l_1})^l \hookrightarrow (\texttt{let}\ x^{l_2} = e_3^{l_3}\ \texttt{in}\ e_1^{l_1})^l}$$

$$(\texttt{let}\ x^{l_2} = v^{l_0}\ \texttt{in}\ e_1^{l_1})^l \hookrightarrow e_1^{l_1}\{x \leftarrow v^{l_0}\}$$

$$\frac{e_0^{l_0} \hookrightarrow e_1^{l_1}}{K, P :: \langle p : E[e_0^{l_0}]\rangle \xrightarrow{[\varepsilon]} K, P :: \langle p : E[e_1{}^{l_1}]\rangle}$$

$$\frac{k \notin \mathrm{dom}(K)}{K, P :: \langle p : E[(\texttt{channel()})^l]\rangle \xrightarrow{[\varepsilon]} K[k \mapsto l], P :: \langle p : E[k^l]\rangle}$$

$$\frac{q \notin \mathrm{dom}(P)}{K, P :: \langle p : E[(\texttt{fork}\ e_0^{l_0})^l]\rangle \xrightarrow{[l,l]} K, P :: \langle p : E[()^l]\rangle :: \langle q : e_0^{l_0}\rangle}$$

$$K, P :: \langle p_s : E_s[(\texttt{send}\ k^{l_0}\ v^{l_1})^{l_s}]\rangle :: \langle p_r : E_r[(\texttt{receive}\ k^{l_2})^{l_r}]\rangle \xrightarrow{[l_s,l_r]} K, P :: \langle p_s : E_s[v^{l_1}]\rangle :: \langle p_r : E_r[v^{l_1}]\rangle$$

FIG. 2.4 – **Sémantique opérationnelle de Concurrent ML.**

De plus, nous utilisons la notion de *configuration* K, P pour représenter un programme distribué, K étant un environnement pour les canaux et P, le *groupe de processus* étant défini dans la figure 2.1.

Un processus est défini par $\langle p : e^l\rangle$, p étant le nom associé au processus qui calcule e^l. Dans la figure 2.4, $\xrightarrow{[c]}$ est défini sur l'ensemble des groupes de processus. Pour un

processus, les pas de réduction séquentiels sont décrits par $\hookrightarrow$.

Lorsqu'on exécute une instruction $(\texttt{channel()})^l$, un nouveau nom de canal k est créé et K est enrichi avec k. Lorsqu'un nouveau processus est créé, l'instruction `fork` prend la valeur `()` et le nouveau processus est nommé q, où q est un nouveau nom de processus. Finalement, une communication peut avoir lieu entre un processus p_s réalisant une émission et un processus p_r réalisant une réception sur le même nom de canal. Dans ce cas l'émission et la réception s'évaluent en la valeur de l'objet de la communication.

Notons que des notions de bisimulation faible correspondant à celles données à la section 2.1.1 ont été définies pour Concurrent ML (FHJ95).

2.2 Analyse statique

Les techniques d'analyse statique sont largement utilisées afin d'établir des propriétés satisfaites par les programmes, à partir d'un examen de leur code. Parmi les applications les plus répandues figurent celles liées aux optimisations à la compilation et la vérification de programmes (NNH99).

2.2.1 Description générale

Une analyse statique calcule un sur-ensemble des résultats pouvant être produits par un programme, sous l'hypothèse que les entrées de celui-ci appartiennent à des ensembles particuliers de valeurs. Les valeurs manipulées par les programmes (données et résultats) sont remplacées par des ensembles de valeurs appelés *valeurs abstraites*. On prouve la correction d'une analyse en montrant que le résultat de l'exécution appartient à l'ensemble des valeurs représentées par la valeur abstraite calculée par l'analyse, comme décrit par le schéma de la figure 2.5.

De nombreuses méthodes d'analyse statique ont été développées pour les langages séquentiels, dont les analyses de types (HM88), la *strictness analysis* (CC94), ainsi que les analyses de flot de contrôle et des temps de liaison détaillées dans les sections 2.2.2 et 2.2.3.

Concernant les langages parallèles, des analyses de types ont aussi été proposées. Par exemple, Reppy a étendu le système de types de ML à Concurrent ML (Rep92) et Pierce et Turner ont proposé un langage fortement typé fondé sur le π-calcul, Pict (PT96). Tous deux proposent une approche similaire qui consiste à associer à chaque canal de communication un type pour les valeurs qu'il transmet. Ce type sera par la suite associé aux réceptions sur ce canal. Une autre approche, notamment utilisée par Colaço (Col97), De Nicola et al. (DNFP99; DNFPV00) et par Nielson et Nielson (NN99) consiste à introduire de nouveaux types décrivant de manière abstraite certains comportements d'un

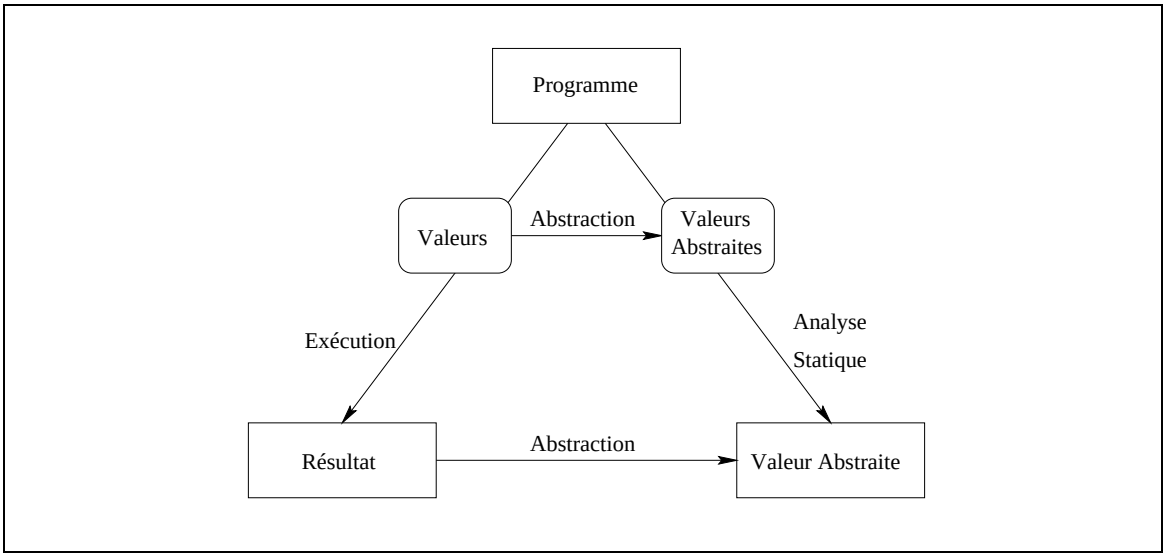


FIG. 2.5 – **Principe de fonctionnement d'une analyse statique.**

processus, comme par exemple les communications qu'il réalise ou les sites sur lesquels il migre.

De plus, outre les analyses de flot de contrôle et des temps de liaison décrites ultérieurement, d'autres analyses spécifiques ont été développées. Nielson et Nielson ont proposé une analyse permettant de détecter si un programme écrit en Concurrent ML génère un nombre fini de processus et de canaux (NN94b). Des *analyses de localité* ont été proposées pour le placement des processus en fonction de leurs interactions (Col95; Jag94; NN95).

Les *analyses de comportement* ont pour but d'assurer la sécurité des applications distribuées en définissant plusieurs niveaux de confidentialité et en vérifiant que les processus d'un certain niveau ne réalisent pas d'opération d'un niveau plus élevé (BDNN98; BDNN99; ANN98; DNFPV00). La plupart de ces analyses reposent sur une analyse de flot de contrôle.

Kobayashi et al. ont proposé une analyse des communications réalisées par des programmes distribués écrits en HACL, un langage à base de communications asynchrones utilisant des canaux (KNY95; SK94). Cette analyse permet de majorer le nombre de messages présents dans les files d'attentes des récepteurs. D'autres analyses de ce type ont été proposées pour le π-calcul (BDNN98; Ven96) et Concurrent ML (Col95; SNN97). Certaines d'entre elles sont détaillées à la section 2.2.2.

Divers outils ont été développés pour prouver la correction d'une analyse statique incluant les relations logiques (Lau91; Wan93), les relations d'équivalence partielles (PER) (HS91; Hun91) ou encore les connexions de Galois (CC77; CC92; CC94). Dans les chapitres 4 et 5 nous construisons des analyses statiques de la manière suivante.

(i) Nous donnons une spécification de l'analyse en énonçant les conditions devant être vérifiées par les valeurs abstraites. Pour l'analyse développée au Chapitre 4, ceci correspond aux Sections 4.2 et 4.3.

(ii) Prouver la correction de cette spécification. Pour cela, nous réalisons une preuve par réduction (*subject reduction*) (WF94). Cela consiste à prouver que si l'on a des valeurs abstraites correctes pour un programme p et que p devient p' après un pas de réduction, alors les valeurs abstraites sont correctes pour p'. En répétant ce raisonnement, on obtient qu'à partir d'annotations correctes pour le programme initial, on aura des annotations correctes pour tous les résultats produits en un temps fini. Pour l'analyse développée au chapitre 4, ceci correspond aux Propositions 26 et 31.

(iii) Générer un système de contraintes en fonction de la syntaxe du programme et prouver que les solutions à ce système vérifient la spécification du point (*i*), c.à.d. qu'elles sont des analyses correctes. Pour l'analyse du chapitre 4, cela correspond à la section 4.4. De plus, nous donnons à la section 4.4 un algorithme permettant de calculer la plus petite solution pour les systèmes en question.

Cette méthodologie est utilisée par Solberg et al. (SNN97) et par Nielson et al. (NNH99; NN94a). Dans notre contexte, cette approche nous permet de générer aisément les contraintes relatives à chaque processus du groupe et de calculer une analyse pour le groupe dans sa totalité à partir de la réunion des ensembles de contraintes générées.

2.2.2 Analyse de flot de contrôle

Une analyse de flot de contrôle (*control flow analysis* ou CFA) détermine l'ensemble des points d'un programme susceptibles de succéder à un point donné dans l'ordre imposé par l'exécution (NN97). Si ce problème est simple dans le cas des langages séquentiels dépourvus de procédures, il en est autrement pour des langages plus évolués.

De nombreux travaux ont été consacrés à l'analyse du flot de contrôle de langages fonctionnels séquentiels du premier ordre ou d'ordre supérieur (NN97; NNH99; Shi88; Shi91). Dans ce cadre, l'analyse consiste à déterminer l'ensemble des fonctions susceptibles d'être appelées à un point donné.

Cette tâche se trouve compliquée par l'adjonction de primitives de gestion du parallélisme (pour les communications ou la création de nouveaux processus par exemple). En effet, ces primitives définissent de nouvelles structures de contrôle que les analyses doivent traiter au mieux afin de conserver leur précision. Dans un langage fonctionnel d'ordre supérieur, il est par ailleurs possible de communiquer des noms de canaux et

des fonctions (ces dernières peuvent contenir d'autres communications). Cette forme de mobilité du code complique à son tour le calcul statique du flot de contrôle.

Considérons par exemple le programme de la figure 2.6, écrit en Concurrent ML, composé de deux processus p_1 et p_2. Chaque instruction est annotée par une étiquette unique. p_1 envoie une fonction sur le canal α et reçoit ensuite, toujours sur α, une fonction dont il calcule la valeur au point 1. p_2 reçoit une fonction sur le canal α qu'il nomme f et renvoie sur α une fonction g ou une fonction h, selon la valeur de f en 0.

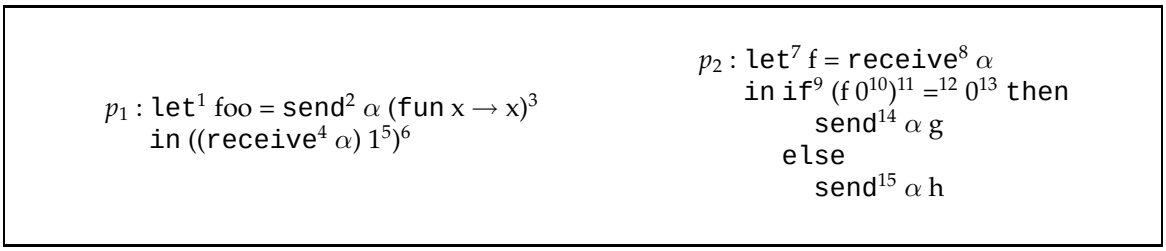


FIG. 2.6 – **Exemple de programme écrit en Concurrent ML.**

Un résultat possible pour une analyse de flot de contrôle de ce programme est présenté dans le tableau de la figure 2.7, qui associe à chaque expression e d'étiquette ℓ du programme, l'ensemble $\widehat{\mathcal{C}}(\ell)$ des étiquettes des fonctions en lesquelles e peut s'évaluer. $\widehat{\mathcal{C}}(\ell)$ est appelé *valeur abstraite* associée au point ℓ. $\widehat{\mathcal{C}}(\ell) = \varnothing$ pour les valeurs simples. Ces étiquettes n'apparaissent pas dans le tableau. $\widehat{\mathcal{E}}(id)$ représente la valeur abstraite de l'identificateur *id* dans l'environnement de l'analyse, c.à.d. l'ensemble des étiquettes des fonctions pouvant être affectées à *id*.

Etiquette	2	3	4	7	8	9	12	14	15
Valeur	$\{3\}$	$\{3\}$	$\widehat{\mathcal{E}}(g) \cup \widehat{\mathcal{E}}(h)$	$\widehat{\mathcal{E}}(g) \cup \widehat{\mathcal{E}}(h)$	$\{3\}$	$\widehat{\mathcal{C}}(14) \cup \widehat{\mathcal{C}}(15)$	$\widehat{\mathcal{E}}(=)$	$\widehat{\mathcal{E}}(g)$	$\widehat{\mathcal{E}}(h)$

FIG. 2.7 – **Valeurs abstraites résultant d'une analyse de flot de contrôle du programme de la figure 2.6.**

$\widehat{\mathcal{C}}(3) = \{3\}$ indique que seule la fonction étiquetée 3 peut apparaître au point 3. $\widehat{\mathcal{C}}(2) = \{3\}$ car une instruction `send` s'évalue en la valeur envoyée. La valeur reçue en 8 est celle qui a été envoyée en 2, on a donc $\widehat{\mathcal{C}}(8) = \widehat{\mathcal{C}}(3) = \{3\}$. Une analyse statique n'exécutant pas les calculs présents dans le programme, la condition du `if` ne peut être évaluée. La seule approximation possible consiste alors à associer au point 9 l'union des résultats possibles pour les deux alternatives, $\widehat{\mathcal{C}}(9) = \widehat{\mathcal{C}}(14) \cup \widehat{\mathcal{C}}(15)$. Pour les mêmes raisons, il n'est pas possible de déterminer laquelle des émissions présentes dans la conditionnelle aura lieu. Ainsi, la valeur reçue en 4 est parmi celles envoyées en 14 et en 15, c.à.d. $\widehat{\mathcal{C}}(4) = \widehat{\mathcal{E}}(g) \cup \widehat{\mathcal{E}}(h)$.

Pour les langages à parallélisme explicite, la précision d'une analyse de flot de contrôle dépend sensiblement des approximations réalisées aux points de réception. Une pre-

Etiquette	2	3	4	7	8	9	12	14	15
Valeur	$\{3\}$	$\{3\}$	$\mathcal{K}(\alpha)$	$\widehat{\mathcal{E}}(g) \cup \widehat{\mathcal{E}}(h)$	$\mathcal{K}(\alpha)$	$\widehat{\mathcal{E}}(g) \cup \widehat{\mathcal{E}}(h)$	$\widehat{\mathcal{E}}(=)$	$\widehat{\mathcal{E}}(g)$	$\widehat{\mathcal{E}}(h)$

FIG. 2.8 – **Résultats produits par une CFA se fondant sur les équations (2.5) et (2.6) pour le programme de la figure 2.6.**

mière méthode consiste à collecter dans un environnement $\widehat{\mathcal{K}}(\alpha)$ l'ensemble des valeurs abstraites potentiellement émises sur le canal α et à affecter le résultat à tous les points de réception susceptibles d'utiliser ce canal (BDNN98; SNN97). Ceci est résumé par les équations suivantes dans lesquelles PRG représente le programme que nous analysons.

$$(\texttt{send}^{\ell_s}\ e_0^{\ell_0}\ e_1^{\ell_1} \in \text{PRG}) \Rightarrow \forall \alpha \in \widehat{\mathcal{C}}(\ell_0),\ \widehat{\mathcal{C}}(\ell_1) \subseteq \widehat{\mathcal{K}}(\alpha) \tag{2.5}$$

$$(\texttt{receive}^{\ell_r}\ e^{\ell_2} \in \text{PRG}) \Rightarrow \widehat{\mathcal{C}}(\ell_r) = \bigcup_{\alpha \in \widehat{\mathcal{C}}(\ell_2)} \widehat{\mathcal{K}}(\alpha) \tag{2.6}$$

L'équation (2.5) indique que, pour chaque émission $\texttt{send}^{\ell_s}\ e_0^{\ell_0}\ e_1^{\ell_1}$ rencontrée dans le programme, et tout canal α tel que α est éventuellement le sujet de la communication, la valeur abstraite $\widehat{\mathcal{C}}(\ell_1)$ envoyée doit être ajoutée à l'ensemble $\widehat{\mathcal{K}}(\alpha)$ des valeurs potentiellement transmises sur le canal α. L'équation (2.6) indique que la valeur abstraite pouvant être reçue en un point de réception d'étiquette ℓ_r est l'union, pour tout les canaux α éventuellement utilisé comme sujet de la réception, de tous les ensembles $\widehat{\mathcal{K}}(\alpha)$ de valeurs pouvant transiter à un moment donné de l'exécution du programme sur le canal α.

Appliquée au programme de la figure 2.6, cette méthode produit le résultat donné dans la tableau de la figure 2.8, avec $\widehat{\mathcal{K}}(\alpha) = \{3\} \cup \widehat{\mathcal{E}}(g) \cup \widehat{\mathcal{E}}(h)$.

Suivant l'équation (2.5), on a $\widehat{\mathcal{K}}(\alpha) = \{3\} \cup \widehat{\mathcal{E}}(g) \cup \widehat{\mathcal{E}}(h)$, ce qui correspond à l'union des valeurs abstraites potentiellement émises sur le canal α par les émissions présentes aux points 2, 14 et 15. Suivant l'équation (2.6), la valeur abstraite associée aux points de réception 4 et 8 est $\widehat{\mathcal{K}}(\alpha)$, car α est le seul canal pouvant être utilisé comme sujet pour ces communications. On obtient donc que la valeur abstraite associée aux points de réception 4 et 8 est l'union des valeurs abstraites émises aux points 2, 14 et 15, ce qui est moins précis que le résultat présenté dans la figure 2.7. La raison en est que cette analyse ne tient pas compte du fait que, les communications étant synchrones, la réception étiquetée 8 (resp. 4) peut seulement communiquer avec l'émission d'étiquette 2 (resp. 14 et 15).

Ceci met en évidence que, dans le cas des langages à parallélisme explicite, une analyse de flot de contrôle doit, pour être précise, utiliser une approximation fine de la topologie des communications réalisées par le programme. Nous développons au chapitre 4 une CFA tenant compte de ces critères. Appliquée au programme de la figure 2.6, notre analyse produit les résultats de la figure 2.7.

2.2.3 Analyse des temps de liaison

Etant donnés un programme p et une partie $\mathcal{P}$ de ses entrées, le but d'une analyse des temps de liaison (*binding time analysis* ou BTA) est de déterminer les parties de p dont les données sont parfaitement déterminées grâce à $\mathcal{P}$. Les données et les expressions connues sont dites *statiques*, les autres *dynamiques*. De nombreuses BTA ont été proposées pour des langages fonctionnels séquentiels typés (BW94; HM94) ou non typés (HS91; Hun91; HS95; Lau91; Mog89; PS94), ainsi que pour des langages impératifs (HN97; HCN97).

Les résultats produits par une analyse des temps de liaison sont utilisés pour indiquer à un évaluateur partiel les parties du programme qu'il peut exécuter et celles dont le code doit être conservé pour former un programme résiduel.

Appliquée à un programme p et à un *environnement abstrait* indiquant quelles variables de p sont statiques et lesquelles sont dynamiques, une analyse des temps de liaison produit une version annotée p_a de p. Dans p_a, la BTA a attaché à chaque sous-expression de p, une information précisant son caractère statique ou dynamique.

Une manière de représenter ces annotations consiste à utiliser un *langage à deux niveaux* ou *à deux étages* (GJ91; GM98; Mog92; MS97; NN92; TS97). Les expressions du premier niveau sont statiques tandis que celles du second niveau sont dynamiques. Syntaxiquement, on distingue les instructions du premier niveau de celles du second en soulignant ces dernières. Ainsi, un évaluateur partiel est un programme qui exécute les expressions du premier niveau d'un programme étagé et produit un programme résiduel (ne comportant plus qu'un seul niveau d'expressions) formé des expressions du second étage du programme initial.

a)

p_1 : $\texttt{let}^1$ foo = $\texttt{send}^2$ α ($\texttt{fun}$ x $\rightarrow$ x)^3
$\texttt{in}$ (($\underline{\texttt{receive}}^4$ α) $\underline{@}$ 1^5)^6

p_2 : $\texttt{let}^7$ f = $\texttt{receive}^8$ α
$\texttt{in}$ $\texttt{if}^9$ (f @ 0^{10})^{11} $=^{12}$ 0^{13} $\texttt{then}$
$\underline{\texttt{send}}^{14}$ α $\underline{\text{g}}$
$\texttt{else}$
$\underline{\texttt{send}}^{15}$ α $\underline{\text{h}}$

b)

p_1 : $\underline{\texttt{let}}^1$ foo = $\underline{\texttt{send}}^2$ α ($\texttt{fun}$ x $\rightarrow$ x)^3
$\underline{\texttt{in}}$ (($\underline{\texttt{receive}}^4$ α) $\underline{@}$ 1^5)^6

p_2 : $\underline{\texttt{let}}^7$ f = $\underline{\texttt{receive}}^8$ α
$\underline{\texttt{in}}$ $\underline{\texttt{if}}^9$ (f $\underline{@}$ 0^{10})^{11} $\underline{=}^{12}$ 0^{13} $\underline{\texttt{then}}$
$\underline{\texttt{send}}^{14}$ α g
$\texttt{else}$
$\underline{\texttt{send}}^{15}$ α h

FIG. 2.9 – **Versions étagées du programme de la figure 2.6.**

Des versions étagées du programme de la figure 2.6 sont données dans la figure 2.9. Elles correspondent à deux analyses des temps de liaison.

Soit $T = \{\mathrm{S}, \mathrm{D}\}$ le treillis à deux éléments ordonnés par $\mathrm{S} \trianglelefteq \mathrm{D}$, et soit $\widehat{\mathcal{B}}(\ell) \in T$ la valeur abstraite associée par la BTA au point d'étiquette ℓ du programme. Les valeurs abstraites associées aux points de réception se calculent à l'aide des équations suivantes dans lesquelles $\widehat{\mathcal{H}}(\alpha)$ décrit la valeur abstraite associée au canal α et $\widehat{\mathcal{C}}$ est le cache abstrait défini à la section 2.2.2.

$$(\texttt{send}^{\ell_0}\ e_1^{\ell_1}\ e_2^{\ell_2} \in \mathrm{PRG}) \ \Rightarrow\ \forall \alpha \in \widehat{\mathcal{C}}(\ell_1),\ \widehat{\mathcal{B}}(\ell_2) \trianglelefteq \widehat{\mathcal{H}}(\alpha) \tag{2.7}$$

$$(\texttt{receive}^{\ell_0}\ e^{\ell_1} \in \mathrm{PRG}) \ \Rightarrow\ \left(\bigvee_{\alpha \in \widehat{\mathcal{C}}(\ell_1)} \widehat{\mathcal{H}}(\alpha) \right) \trianglelefteq \widehat{\mathcal{B}}(\ell_0) \tag{2.8}$$

Pour toute émission et pour tout nom de canal α tel que $\alpha \in \widehat{\mathcal{C}}(\ell_1)$, l'équation (2.7) contraint la valeur de $\widehat{\mathcal{H}}(\alpha)$ à être au moins aussi grande que $\widehat{\mathcal{B}}(\ell_2)$, la valeur abstraite associée à l'objet de la communication.

Ensuite, l'équation 2.8 contraint la valeur abstraite d'une réception à être au moins aussi grande que le maximum des valeurs $\widehat{\mathcal{H}}(\alpha)$, pour tout nom de canal α pouvant être sujet de la communication.

Clairement, la qualité d'une analyse des temps de liaison fondée sur les équations (2.7) et (2.8) dépend de la précision du cache abstrait $\widehat{\mathcal{C}}$, c.à.d. de la précision de l'analyse de flot contrôle dont les résultats sont utilisés ici. Les annotations données dans les figures 2.9 a) et b) sont obtenues en utilisant les valeurs données dans les figures 2.7 et 2.8 pour la CFA.

2.3 Evaluation Partielle

L'évaluation partielle est une technique permettant d'optimiser un programme p par rapport à une partie de ses paramètres d'entrée. Etant donné un certain nombre de paramètres connus à l'avance (par exemple au moment de la compilation), on évalue les parties de p connues car dépendant uniquement des paramètres connus, et on génère un nouveau programme constitué des parties de p n'ayant pas pu être évaluées et dans lequel les parties connues ont été remplacées par le résultat des calculs correspondants.

Le programme obtenu est appelé *programme résiduel*, les paramètres connus fournis à l'évaluateur partiel sont dits *statiques* et les autres sont dits *dynamiques*. Par extension, une expression dont toutes les valeurs sont statiques est dite statique, on dit qu'elle est dynamique dans le cas contraire.

Le programme résiduel prend pour entrée les données non spécifiées au moment

de l'évaluation partielle, et produit le même résultat que le programme initial appliqué à l'ensemble des données. Ce programme permet d'obtenir les même résultats que le programme initial appelé avec des paramètres identiques à ceux fournis au moment de l'évaluation partielle, mais en réalisant moins de calculs. Il s'agit d'une version optimisée, mais d'une généralité moindre à celle du programme initial, puisque certains paramètres ont été fixés. On dit que le programme initial a été *spécialisé*.

Les données statiques peuvent être les données d'un programme connues au moment de la compilation. L'évaluation partielle permet alors d'optimiser le programme à compiler. Il peut aussi s'agir de données peu variables. Le but est alors de générer un programme optimisé par rapport à ces données stables, que l'on appliquera à divers jeux de données complémentaires. Par exemple, cette technique a été utilisée pour spécialiser un algorithme de lancer de rayons par rapport à une scène. Diverses vues peuvent ensuite être générées plus efficacement (And95).

Concernant les langages séquentiels, de nombreux travaux ont été consacrés à l'évaluation partielle de langages fonctionnels (CD93b; Dan96; GJ91; JGS93; Jon90; Kra88; NN92) et impératifs (CD93b; CHL+98; CMVR97; GZB94).

Concernant les langages parallèles, Hosoya et al. (HKY96) ont proposé un évaluateur partiel *en ligne* (CK93) pour le langage HACL (SK94). Marinescu et Goldberg ont étudié l'évaluation partielle de programmes écrits dans un sous-ensemble de CSP (Hoa78; Hoa85; MG97). Ils interdisent la création dynamique de noms de canaux et de processus, ainsi que la transmission de noms de canaux. Enfin, Concel et Danvy ont obtenu un générateur de compilateurs pour un langage fonctionnel fondé sur l'opérateur `future` (CD93a). Cette approche utilise une machine de type MIMD (LER92) à mémoire partagée. Le compilateur généré produit des programmes parallèles à partir de programmes sources séquentiels.

2.3.1 Définition formelle

Formellement, un langage de programmation $\mathcal{L}$ est spécifié par un ensemble $\mathcal{P}$ de $\mathcal{L}$-programmes, un ensemble Δ de $\mathcal{L}$-données, et une famille $\mathcal{S} = (\mathcal{S}_n)_{n\in\mathbb{N}}$, de relations $(n+2)$-aires (GJ91; Wan93). $\mathcal{S}$ est la sémantique de $\mathcal{L}$. Soit $\rho \in \mathcal{P}$, et $\delta_1, \ldots, \delta_n, \delta \in \Delta$. $(\rho, \delta_1, \ldots, \delta_n, \delta) \in \mathcal{S}_n$ si et seulement si δ est le résultat de l'application du programme ρ aux données $\delta_1 \ldots \delta_n$.

Un interpréteur est un programme qui utilise deux sortes de données, un programme ρ à interpréter et les données $\delta_1, \ldots, \delta_n$ de ρ. Un méta-interpréteur pour un langage $\mathcal{L}$ est un interpréteur pour $\mathcal{L}$ écrit en $\mathcal{L}$, qui peut ainsi s'appliquer à lui-même. Puisqu'un méta-interpréteur pour $\mathcal{L}$ est un programme `Eval` $\in \mathcal{P}$, ρ doit être codé sous la forme de

$\mathcal{L}$-données, pour être compris par `Eval`.

Définition 6 (Codage et méta-interprétation) *Un programme* `Eval` $\in \mathcal{P}$ *est un méta-interpréteur correspondant à une fonction de codage* $\ulcorner . \urcorner : \mathcal{P} \to \Delta$ *si et seulement si nous avons*

$$(\texttt{Eval}, \ulcorner \rho \urcorner, \delta_1, \ldots, \delta_n, \delta) \in \mathcal{S}_{n+1} \iff (\rho, \delta_1, \ldots, \delta_n, \delta) \in \mathcal{S}_n \tag{2.9}$$

pour tout $\rho \in \mathcal{P}$ *et pour tout* $\delta_1, \ldots \delta_n, \delta \in \Delta$. □

Un évaluateur partiel est un programme `Pev` qui spécialise un programme ρ par rapport à la partie statique de ses données. Il est toujours possible de concaténer toutes les données statiques, resp. dynamiques, utilisées par un programme ρ. Aussi, pouvons nous supposer qu'un programme ρ utilise seulement deux données δ_1 et δ_2, que l'on suppose respectivement statique et dynamique. Afin de déterminer les parties statiques d'un programme ρ, une analyse des temps de liaison $\Phi : \mathcal{P} \to \mathcal{P}_a$ est utilisée, qui produit un programme $\Phi(\rho) \in \mathcal{P}_a$ qui est une version annotée de ρ. Un terme annoté doit à son tour être codé sous la forme de $\mathcal{L}$-données, afin d'être compris par l'évaluateur partiel.

Définition 7 (Analyse et évaluation partielle) *Considérons un programme* $\rho \in \mathcal{P}$ *et des données* $\delta_1, \delta_2, \delta \in \Delta$, *telles que* $(\rho, \delta_1, \delta_2, \delta) \in \mathcal{S}_2$. *Un évaluateur partiel auto-applicable, correspondant à une fonction d'analyse* Φ *et à une fonction de codage* $\llcorner . \lrcorner : \mathcal{P}_a \to \Delta$ *est un programme* `Pev` $\in \mathcal{P}$ *qui produit un résultat* $\ulcorner \rho' \urcorner$, *avec* $\rho' \in \mathcal{P}$, *tel que*

$$(\texttt{Pev}, \llcorner \Phi(\rho) \lrcorner, \delta_1, \ulcorner \rho' \urcorner) \in \mathcal{S}_2 \Rightarrow (\texttt{Eval}, \ulcorner \rho' \urcorner, \delta_2, \delta) \in \mathcal{S}_2 \tag{2.10}$$

est satisfait. □

L'équation (2.10) indique qu'évaluer partiellement un programme ρ par rapport à la partie δ_1 de ses données produit le codage d'un programme ρ' tel que (`Eval` $\ulcorner \rho' \urcorner$ δ_2) = $\delta = (\rho\ \delta_1\ \delta_2)$. $(\texttt{Eval}, \ulcorner \rho' \urcorner, \delta_2, \delta) \in \mathcal{S}_2$ signifie, en utilisant l'équation (2.9), que $(\rho', \delta_2, \delta) \in \mathcal{S}_1$. Ainsi, pour tout $\delta_2, \delta \in \Delta$, nous avons

$$(\rho, \delta_1, \delta_2, \delta) \in \mathcal{S}_2 \Rightarrow (\rho', \delta_2, \delta) \in \mathcal{S}_1 \tag{2.11}$$

2.3.2 Projections de Futamura

Lorsque `Pev`$\in \mathcal{P}$, c.à.d. que `Pev` est un $\mathcal{L}$-programme, il est possible de l'appliquer à `Eval`, ce qui est appelé la *première projection de Futamura* (Fut71).

Tout d'abord, `Eval` est considéré comme un programme prenant pour entrée deux

données $\ulcorner\rho\urcorner$ et δ et renvoyant un résultat δ_r :

$$\texttt{Eval} : \ulcorner\rho\urcorner \times \delta \mapsto \delta_r \tag{2.12}$$

où $\ulcorner.\urcorner : \mathcal{P} \to \Delta$ est la fonction qui transforme les programmes en données. Lorsque l'on dispose d'un programme ρ sans connaître ses données, il est possible d'évaluer partiellement `Eval` par rapport à ρ . Le résultat est un programme ρ_c qui prend une donnée et renvoie un résultat. Ainsi, ρ a été compilé en ρ_c :

$$\ulcorner\rho_c\urcorner \widehat{=} (\texttt{Pev}\ \llcorner\Phi(\texttt{Eval})\ \ulcorner\rho\urcorner\lrcorner) \tag{2.13}$$

avec ρ_c tel que :

$$\mathcal{S}_1(\rho_c, \delta, \delta_r) \Rightarrow \mathcal{S}_2(\texttt{Eval}, \ulcorner\rho\urcorner, \delta, \delta_r) \tag{2.14}$$

De la même manière, plutôt que d'effectuer le calcul donné dans l'équation (2.13) pour différents programmes ρ, il est possible de spécialiser `Pev` par rapport à `Eval`. Ce calcul, appelé *seconde projection de Futamura*, consiste à calculer une fois pour toutes le terme

$$\texttt{Comp} \widehat{=} (\texttt{Pev}\ \llcorner\Phi(\texttt{Pev})\ \llcorner\Phi(\texttt{Eval})\lrcorner\lrcorner) \tag{2.15}$$

`Comp` est un programme qui prend pour entrée un programme ρ et produit une version compilée ρ_c de ρ : `Comp` est donc un compilateur au sens classique du terme.

$$\mathcal{S}_1(\texttt{Comp}, \rho, \rho_c) \Rightarrow \big(\mathcal{S}_1(\rho_c, \delta, \delta_r) \iff \mathcal{S}_2(\texttt{Eval}, \rho, \delta, \delta_r)\big) \tag{2.16}$$

Enfin, plutôt que de réitérer la démarche précédente pour différents interpréteurs, il est possible d'effectuer une fois pour toutes le calcul suivant, appelé *troisième projection de Futamura*.

$$\texttt{Cogen} \widehat{=} (\texttt{Pev}\ \llcorner\Phi(\texttt{Pev})\ \llcorner\Phi(\texttt{Pev})\lrcorner\lrcorner) \tag{2.17}$$

On produit ainsi un générateur de compilateurs, prenant pour entrée un interpréteur, et générant un compilateur.

De tels évaluateurs partiels, auto-applicables, ont été réalisés pour des langages fonctionnels non typés tels que Scheme (Con93; GJ91; JGB$^+$90) ou le $\boldsymbol{\lambda}$-calcul pur (Mog92), ou typés tels que ML (She97; SSPJ98; TS97).

Chapitre 3

Evaluation partielle pour le pi-calcul

3.1 Introduction

Dans ce chapitre, nous traitons de l'évaluation partielle du π-calcul (Mil93; San92). Ce langage, à la CCS (Mil89), permet de décrire l'exécution parallèle de processus mobiles interagissant par des communications explicites sur des canaux. Le π-calcul est un langage adapté tout autant à la modélisation des langages parallèles qu'à celle des langages orientés objet (Wal95).

Nous souhaitons montrer ici que l'évaluation partielle de langages parallèles à base de communications explicites est possible et formaliser celle-ci (GM97a; GM97b). Notre approche suit une méthodologie classique, en trois étapes, établie dans le cas de l'évaluation partielle du λ-calcul (GJ91; Mog94; Mog92; Pal93; Wan93). Cette approche est compatible avec les projections de Futamura (Fut71) et la génération automatique de compilateurs. Le fait que le méta-interpréteur soit réflexif est important pour l'auto-application de l'évaluateur partiel (Wan93). Cependant, comme discuté par Pfenning et Lee (PL91), il s'agit d'une propriété intéressante par elle-même, que l'on essaie généralement d'établir lors de l'étude d'un nouveau langage formel tel que le π-calcul. La méthodologie que nous suivons dans ce chapitre est la suivante.

(i) Nous définissons un méta-interpréteur pour le π-calcul et nous prouvons sa correction en utilisant la notion de congruence barbelée faible (Mil93), notée $\approx$. Nous introduisons une fonction $\ulcorner . \urcorner$ qui permet de coder les programmes dans le π-calcul, et nous présentons l'évaluateur `Eval` permettant d'exécuter les termes codés. La propriété de correction de l'évaluateur établit que l'application à `Eval` du code $\ulcorner P \urcorner$ d'un π-terme P est $\approx$-congruente à P.

(ii) Nous introduisons des annotations pour les π-termes en vue de leur évaluation partielle en définissant un langage à deux niveaux (NN92). Ces annotations in-

diquent à l'évaluateur partiel les communications qui peuvent être exécutées statiquement. Ensuite, nous caractérisons les termes bien annotés parmi l'ensemble des π-termes à deux niveaux. La notion de bonne annotation est nécessaire pour prouver la correction de l'évaluation partielle.

(iii) Enfin, nous définissons un évaluateur partiel `Pev`, prenant pour entrée un π-terme à deux niveaux codé par une fonction $\llcorner . \lrcorner$ et produisant un programme résiduel codé par la fonction $\ulcorner . \urcorner$. Les relations utilisées pour prouver la correction de l'évaluation partielle sont la congruence barbelée faible $\approx$ et la simulation faible par réduction $\prec$. Nous établissons que l'application d'un π-terme à deux niveaux bien annoté à `Pev` préserve la $\approx$-congruence ou la $\prec$-simulation selon les cas.

Par ailleurs, l'évaluation partielle de systèmes de processus mobiles permet d'exécuter statiquement certaines communications présentes dans les programmes fournis en entrée. Il est donc nécessaire d'étudier l'influence de cette transformation sur l'observabilité des programmes résiduels obtenus vis-à-vis de celle des programmes initiaux, et par conséquent sur leurs comportements respectifs lors de leurs exécutions dans un système plus large. A travers la notion de bonne annotation des termes, nous spécifions les conditions nécessaires pour qu'une communication puisse être exécutée au moment de l'évaluation partielle sans que le programme résiduel ait un comportement différent du programme initial d'un point de vue extérieur.

La section 3.2 présente les mécanismes utilisés dans le reste de ce chapitre en utilisant un langage plus familier, le λ-calcul (Bar84). Les sections 3.3, 3.4 et 3.5 traitent respectivement de l'interpréteur, des annotations et de l'évaluateur partiel.

3.2 Evaluation partielle pour le λ-calcul

Dans cette section, nous décrivons brièvement les travaux de Mogensen concernant l'évaluation partielle du λ-calcul pur (Mog94; Mog92) afin d'introduire la méthodologie relative à l'évaluation partielle d'un langage formel. La démarche présentée ici pour le λ-calcul sera celle que nous suivrons dans les sections 3.3, 3.4 et 3.5 pour traiter de l'évaluation partielle auto-applicable pour le π-calcul.

3.2.1 Méta-interprétation

Dans un premier temps, nous présentons un méta-interpréteur qui évalue des λ-termes codés par une fonction que nous explicitons. La seconde étape concerne la correction des annotations des programmes fournis à l'évaluateur partiel, et la dernière consiste

à définir une fonction de codage pour les termes annotés ainsi que l'évaluateur partiel permettant de les réduire.

Comme indiqué ci-dessous, les programmes dans le **λ**-calcul doivent être codés sous la forme de données, ce qui est réalisé grâce à la fonction $\ulcorner . \urcorner_\Lambda$ proposée par Mogensen (Mog94) et que nous donnons dans la figure 3.1. Cette fonction utilise les notions de représentation des signatures et de syntaxe abstraite d'ordre supérieur (PE88). Les variables a, b et c agissent comme des sélecteurs d'opérateurs et indiquent quel type d'opérateur est codé par le terme. a indique que t est une variable, et b et c indiquent respectivement l'occurrence d'une application et d'une abstraction.

$$
\begin{array}{rcl}
\ulcorner x \urcorner_\Lambda & \widehat{=} & \boldsymbol{\lambda} abc.a\ x \\
\ulcorner e_1\ e_2 \urcorner_\Lambda & \widehat{=} & \boldsymbol{\lambda} abc.b\ \ulcorner e_1 \urcorner_\Lambda\ \ulcorner e_2 \urcorner_\Lambda \\
\ulcorner \boldsymbol{\lambda} x.e \urcorner_\Lambda & \widehat{=} & \boldsymbol{\lambda} abc.c\ (\boldsymbol{\lambda} x.\ulcorner e \urcorner_\Lambda)
\end{array}
$$

$$
\begin{array}{rcll}
\mathtt{Eval}_\Lambda & \widehat{=} & \mathtt{Y}\ (\boldsymbol{\lambda} e.\boldsymbol{\lambda} m.m & (\boldsymbol{\lambda} x.x) \\
 & & & (\boldsymbol{\lambda} mn.(e\ m)(e\ n)) \\
 & & & (\boldsymbol{\lambda} m.\boldsymbol{\lambda} v.e\ (m\ v)))
\end{array}
$$

FIG. 3.1 – **Codage et méta-interprétation du λ-calcul.**

L'évaluateur $\mathtt{Eval}_\Lambda$, donné dans la figure 3.1, réduit des **λ**-termes codés par la fonction $\ulcorner . \urcorner_\Lambda$. L'opération correspondant à un des sélecteurs a, b ou c est appliquée à la variable x, aux termes $\ulcorner e_1 \urcorner_\Lambda$ et $\ulcorner e_2 \urcorner_\Lambda$ ou au terme $(\boldsymbol{\lambda} x.\ulcorner e \urcorner_\Lambda)$.

Y représente l'opérateur de point fixe, c.à.d.

$$\mathtt{Y} \widehat{=} \lambda h.(\lambda x.h\ (x\ x))\ (\lambda x.h\ (x\ x)) \tag{3.1}$$

On prouve la correction du méta-évaluateur en montrant que $\mathtt{Eval}_\Lambda$ appliqué à un programme $\ulcorner t \urcorner_\Lambda$ produit un terme sémantiquement équivalent à t. Dans le **λ**-calcul, ceci est fait en utilisant la relation de β-équivalence $=_\beta$, ce qui donne la propriété

$$\mathtt{Eval}_\Lambda\ \ulcorner t \urcorner_\Lambda\ =_\beta\ t \tag{3.2}$$

prouvée par Mogensen (Mog94).

3.2.2 Evaluation partielle

Un langage à deux niveaux est utilisé pour annoter les programmes. Un tel langage est construit en définissant deux versions syntaxiquement différentes des opérateurs (MS97; NN92). Les opérateurs du premier niveau ont la même sémantique et généra-

lement la même syntaxe que leurs équivalents dans le langage non étagé, et ceux du second décrivent, dans notre contexte, les opérations dynamiques dont les paramètres dépendent de l'exécution du programme. La syntaxe du $\boldsymbol{\lambda}$-calcul à deux niveaux est donnée ci-dessous.

Définition 8 ($\boldsymbol{\lambda}$-termes à deux niveaux) *Un $\boldsymbol{\lambda}$-terme à deux niveaux est un terme appartenant au langage décrit par la grammaire suivante.*

$$t ::= x \mid (t_1\ t_2) \mid \boldsymbol{\lambda} x.t \mid (t_1 \_ t_2) \mid \underline{\boldsymbol{\lambda}} x.t$$

□

x, $(t_1\ t_2)$, et $\boldsymbol{\lambda} x.t$ représentent des termes statiques qui seront réduits par l'évaluateur partiel. $(t_1 \_ t_2)$ et $\underline{\boldsymbol{\lambda}} x.t$ représentent des termes dynamiques pour lesquels l'évaluateur partiel évalue les sous-expressions et produit un programme résiduel r codé sous la forme de données. Des analyses des temps de liaison produisant des $\boldsymbol{\lambda}$-termes à deux niveaux sont décrites par Mogensen (Mog92), Palsberg (Pal93) et Wand (Wan93). Suivant Mogensen, nous codons les $\boldsymbol{\lambda}$-termes à deux niveaux en utilisant la fonction $\llcorner . \lrcorner_\Lambda$ définie dans la figure 3.2.

$$
\begin{array}{rcl}
\llcorner x \lrcorner_\Lambda & \widehat{=} & \boldsymbol{\lambda} abcde.a\ x \\
\llcorner e_1\ e_2 \lrcorner_\Lambda & \widehat{=} & \boldsymbol{\lambda} abcde.b\ \llcorner e_1 \lrcorner_\Lambda\ \llcorner e_2 \lrcorner_\Lambda \\
\llcorner \boldsymbol{\lambda} x.e \lrcorner_\Lambda & \widehat{=} & \boldsymbol{\lambda} abcde.c\ (\boldsymbol{\lambda} x.\llcorner e \lrcorner_\Lambda) \\
\llcorner e_1 \_ e_2 \lrcorner_\Lambda & \widehat{=} & \boldsymbol{\lambda} abcde.d\ \llcorner e_1 \lrcorner_\Lambda\ \llcorner e_2 \lrcorner_\Lambda \\
\llcorner \underline{\boldsymbol{\lambda}} x.e \lrcorner_\Lambda & \widehat{=} & \boldsymbol{\lambda} abcde.e\ (\boldsymbol{\lambda} x.\llcorner e \lrcorner_\Lambda)
\end{array}
$$

$$
\begin{array}{rcll}
\mathsf{Pev}_\Lambda & \widehat{=} & \mathsf{Y}\ (\boldsymbol{\lambda} p.\boldsymbol{\lambda} m.m & (\boldsymbol{\lambda} x.x) \\
 & & & (\boldsymbol{\lambda} mn.(p\ m)(p\ n)) \\
 & & & (\boldsymbol{\lambda} m.\boldsymbol{\lambda} v.p\ (m\ v)) \\
 & & & (\boldsymbol{\lambda} mn.\boldsymbol{\lambda} abc.b\ (p\ m)(p\ n)) \\
 & & & (\boldsymbol{\lambda} m.\boldsymbol{\lambda} abc.c\ (\boldsymbol{\lambda} v.p\ (m\ (\boldsymbol{\lambda} abc.a\ v)))))
\end{array}
$$

FIG. 3.2 – **Codage des $\boldsymbol{\lambda}$-termes à deux niveaux et évaluation partielle.**

$\llcorner . \lrcorner_\Lambda$ étend $\ulcorner . \urcorner_\Lambda$ afin de coder les opérations statiques et dynamiques. Cinq variables sont utilisées, a, b et c ayant la même signification qu'auparavant et d et e représentant respectivement une application et une abstraction dynamiques.

La figure 3.2 décrit Pev_Λ, l'évaluateur partiel pour le $\boldsymbol{\lambda}$-calcul qui interprète des $\boldsymbol{\lambda}$-termes à deux niveaux. Les termes statiques sont réduits par Pev_Λ, tandis que les opérateurs dynamiques sont partiellement évalués, ce qui donne pour résultat un programme résiduel codé par la fonction $\ulcorner . \urcorner_\Lambda$.

La correction de $\texttt{Pev}_\Lambda$ repose sur la propriété suivante. Soient t un $\boldsymbol{\lambda}$-terme, t_a une version bien annotée de celui-ci qui accepte un argument statique, et d un second $\boldsymbol{\lambda}$-terme. On montre que le terme $t' = \texttt{Pev}_\Lambda \llcorner t_a \lrcorner_\Lambda \; d$, s'il existe, est β-équivalent au terme $t\,d$. Les détails concernant cette propriété sont donnés par Mogensen (Mog92). Des conditions pour la correction de l'évaluation partielle dans le $\boldsymbol{\lambda}$-calcul ont aussi été données par Palsberg et par Wand (Pal93; Wan93).

3.3 Méta-évaluateur pour le π-calcul

Dans cette section, nous présentons un méta-évaluateur, noté `Eval`, pour le $\boldsymbol{\pi}$-calcul. Dans un premier temps, nous montrons comment coder les termes par l'intermédiaire d'une fonction $\ulcorner . \urcorner$, puis nous introduisons `Eval` et nous prouvons sa correction. Du fait que le $\boldsymbol{\pi}$-calcul est un langage fondé sur les communications, les programmes sont représentés par des processus qui envoient des messages à l'interpréteur. `Eval` reçoit et exécute les opérations codées.

3.3.1 Codage et évaluation

Afin de ne pas surcharger les notations et d'améliorer la lisibilité de l'interpréteur, nous supposons, sans perte de généralité, que toute somme non déterministe de processus $\sum_i \pi_i.P_i$ a une taille fixe n et peut se réécrire

$$\sum_{i=1}^{n/2} \alpha_i(x_i).P_i + \sum_{i=n/2+1}^{n} \overline{\alpha_i}[x_i].P_i \tag{3.3}$$

Cette hypothèse permet de donner une représentation plus compacte de `Eval`, mais peut être enlevée soit en utilisant des noms de canaux particuliers pour indiquer la taille des sommes, soit en définissant n comme étant l'arité maximale des sommes d'un programme et en surchargeant les autres sommes et en utilisant la propriété $\pi.P \approx \pi.P + \pi.P$ pour toute communication π et pour tout $\boldsymbol{\pi}$-terme P. Lorsque nous définissons l'interpréteur et l'évaluateur partiel dans la suite de cette section, nous nous affranchissons de cette restriction afin de donner des formes plus lisibles des programmes.

Comme indiqué à la section 2, les programmes dans le $\boldsymbol{\pi}$-calcul doivent être codés sous la forme de données, et, puisque le langage décrit uniquement des processus, ces données seront des processus. Les lettres a, b, c, d et e représentent des noms de canaux réservés. Un processus P est codé sur un canal γ_0 par la fonction $\ulcorner . \urcorner$ donnée dans la figure 3.3. γ_0 est le nom du canal sur lequel les données sont transmises à l'évaluateur. Toutes les noms choisis pour le codage sont des noms frais, n'ayant pas d'occurrence dans le terme P qui est codé.

$$
\begin{array}{rcl}
\ulcorner \mathbf{0}, \gamma_0 \urcorner & \widehat{=} & (\boldsymbol{\nu} abcde)\overline{\gamma_0}[abcde].\overline{a} \\
\ulcorner P_1 \| P_2, \gamma_0 \urcorner & \widehat{=} & (\boldsymbol{\nu} abcde)\overline{\gamma_0}[abcde].\overline{b}.(\boldsymbol{\nu}\gamma_1\gamma_2) \\
 & & \overline{\gamma_0}[\gamma_1\gamma_2].(\ulcorner P_1, \gamma_1 \urcorner \| \ulcorner P_2, \gamma_2 \urcorner) \\
\ulcorner !P, \gamma_0 \urcorner & \widehat{=} & (\boldsymbol{\nu} abcde)\overline{\gamma_0}[abcde].\overline{c}.!((\boldsymbol{\nu}\gamma)\overline{\gamma_0}[\gamma].\ulcorner P, \gamma \urcorner) \\
\ulcorner (\boldsymbol{\nu} x)P, \gamma_0 \urcorner & \widehat{=} & (\boldsymbol{\nu} abcde)\overline{\gamma_0}[abcde].\overline{d}.(\boldsymbol{\nu}\gamma_1\gamma_2)\overline{\gamma_0}[\gamma_1\gamma_2].\gamma_1(x).\ulcorner P, \gamma_2 \urcorner \\
 & & \\
\ulcorner \sum_{i=1}^{n} \pi_i.P_i, \gamma_0 \urcorner & \widehat{=} & (\boldsymbol{\nu} abcde)\overline{\gamma_0}[abcde].\overline{e}.(\boldsymbol{\nu}\gamma_1 \ldots \gamma_n\delta_1 \ldots \delta_n) \\
 & & \overline{\gamma_0}[\gamma_1 \ldots \gamma_n\delta_1 \ldots \delta_n\alpha_1 \ldots \alpha_n x_1 \ldots x_n]. \\
 & & \left(\sum_{i=1}^{n/2} \delta_i(x_i).\ulcorner P_i, \gamma_i \urcorner + \sum_{i=n/2+1}^{n} \delta_i.\ulcorner P_i, \gamma_i \urcorner\right) \\
 & & \\
 & & \text{avec} \quad \pi_i = \alpha_i(x_i),\ 1 \leq i \leq \frac{n}{2} \\
 & & \quad \text{et} \quad \pi_i = \overline{\alpha_i}[x_i],\ \frac{n}{2} + 1 \leq i \leq n
\end{array}
$$

FIG. 3.3 – **Codage des π-termes en vue de leur évaluation.**

Les noms a à e codent le type d'opération décrit par le terme. Le nom $\overline{a}$ code le processus inactif noté $\mathbf{0}$, $\overline{b}$ est utilisé pour représenter une composition parallèle. Dans ce cas deux nouveaux canaux γ_1 et γ_2 sont créés et leurs noms sont transmis sur γ_0. Ensuite, les processus P_1 et P_2 sont codés récursivement, sur γ_1 et γ_2 respectivement. Ces canaux seront utilisés pour les communications entre les codages des termes et leurs interpréteurs respectifs. Le nom $\overline{c}$ indique l'occurrence d'une réplication. Dans ce cas, de manière répétée, nous créons un nouveau canal γ que nous envoyons sur γ_0, puis nous codons sur γ le processus devant être répliqué.

Le message $\overline{d}$ indique l'occurrence d'une restriction et, d'un point de vue opérationnel, nous devons créer un nouveau canal pour x. Pour cela, deux canaux γ_1 et γ_2 sont créés et transmis sur γ_0. L'interprète retourne sur γ_1 un nouveau nom qui sera affecté à x. γ_2 est le canal pour les communications suivantes et $\ulcorner P, \gamma_2 \urcorner$ est récursivement codé. Finalement, pour coder la somme de n processus, les canaux $\gamma_1 \ldots \gamma_n$ et $\delta_1 \ldots \delta_n$ sont créés et envoyés sur γ_0. Les préfixes des termes P_i sont aussi transmis sur γ_0. Le codage d'une réception lit sur un canal δ la nouvelle valeur x, qui est substituée dans le codage du sous-terme. Pour les émissions, δ est uniquement utilisé comme un signal puisqu'aucune valeur n'est nécessaire pour coder le sous-terme.

L'interprète `Eval` qui évalue les programmes codés sur le canal γ_0 est donné dans la figure 3.4. Tout d'abord, `Eval` lit les noms $a \ldots e$ des sélecteurs sur γ_0. Ensuite, il reçoit la description de l'opération à exécuter, représentée par par un nom entre a et e. Le nom a indique la fin de l'exécution. Lorsque b est rencontré, l'interprète lit sur γ_0 les nouveaux noms γ_1 et γ_2 sur lesquels les sous-termes sont codés et exécute `Eval`(γ_1) parallèlement à `Eval`(γ_2). Remarquons que `Eval` est dupliqué, et `Eval`(γ_1), resp. `Eval`(γ_2), evalue $\ulcorner P_1, \gamma_1 \urcorner$, resp. $\ulcorner P_2, \gamma_2 \urcorner$. Le nom c indique que le processus doit être répliqué. Les

canaux γ des codages des copies du processus sont transmises sur γ_0 et $\texttt{Eval}(\gamma)$ est répliqué autant de fois que nécessaire. Le nom d indique l'occurrence d'une restriction et `Eval` lit γ_1 et γ_2 sur γ_0. Un nouveau nom x est créé et transmis au terme codé sur le canal γ_1, avant que $\texttt{Eval}(\gamma_2)$ ne soit exécuté. Enfin, la réception du nom e est suivie de la lecture sur γ_0 des noms des canaux de communication $\gamma_1, \ldots, \gamma_n$, des noms $\delta_1, \ldots, \delta_n$ et des préfixes des processus P_i. Pour $1 \leq i \leq n/2$, ce qui correspond aux réceptions, `Eval` reçoit la valeur v qui est substituée à x_i et la transmet au codage du processus sur le canal δ_i. On obtient ainsi $\ulcorner P_i\{x_i \leftarrow v\}, \gamma_i \urcorner \parallel \texttt{Eval}(\gamma_i)$. Concernant les émissions, $n/2+1 \leq i \leq n$, `Eval` émet la valeur et envoie un simple signal au processus codé dans le but de forcer les calcul suivants.

$$
\begin{aligned}
\texttt{Eval}(\gamma_0) \;\widehat{=}\; & \gamma_0(abcde).\\
& \big(\; a.\mathbf{0}\\
& + b.\gamma_0(\gamma_1\gamma_2).(\texttt{Eval}(\gamma_1) \parallel \texttt{Eval}(\gamma_2))\\
& + c.!(\gamma_0(\gamma).\texttt{Eval}(\gamma))\\
& + d.\gamma_0(\gamma_1\gamma_2).(\boldsymbol{\nu}\, x)\overline{\gamma_1}[x].\texttt{Eval}(\gamma_2)\\
& + e.\gamma_0(\gamma_1 \ldots \gamma_n\delta_1 \ldots \delta_n\alpha_1 \ldots \alpha_n x_1 \ldots x_n).\\
& \quad \Big(\textstyle\sum_{i=1}^{n/2} \alpha_i(x_i).\overline{\delta_i}[x_i].\texttt{Eval}(\gamma_i)\\
& \quad\; + \textstyle\sum_{i=n/2+1}^{n} \overline{\alpha_i}[x_i].\overline{\delta_i}.\texttt{Eval}(\gamma_i)\Big)\;\big)
\end{aligned}
$$

FIG. 3.4 – **Méta-évaluateur pour le π-calcul.**

Enfin, notons que `Eval` effectue des appels récursifs qui ne sont pas autorisés dans le π-calcul. Cependant, la récursion peut aisément être simulée sans incidence sur la congruence barbelée faible (Mil93). En fait, des communications sont utilisées au lieu des appels récursifs et la réplication permet de créer des nouvelles instances des termes récursifs.

3.3.2 Correction de `Eval`

Afin de prouver la correction de l'évaluateur, nous introduisons le terme $\mathbb{E}(P)$ défini à l'équation (3.4) et nous prouvons qu'il est $\approx$-congruent à P. Ainsi, ont obtient le même comportement en interprétant le terme codé qu'en exécutant le terme initial.

$$\mathbb{E}(P) \widehat{=} (\boldsymbol{\nu}\gamma_0)\big(\ulcorner P, \gamma_0 \urcorner \parallel \texttt{Eval}(\gamma_0)\big) \tag{3.4}$$

Le terme codé transmet le programme source à l'évaluateur sur le canal restreint γ_0. Les preuves de correction sont fondées sur la notion de congruence barbelée faible. Le choix de cette relation est motivé par le fait que, comme mentionné à la section 2.1.1, les termes

congruents par réduction faible ne peuvent pas être distingués d'un point de vue externe au système.

Lemme 9 (Compositionalité de $\mathbb{E}$) *Les propriétés suivantes sont vérifiées.*

(i) $\mathbb{E}(P_1 \| P_2) \approx \mathbb{E}(P_1) \| \mathbb{E}(P_2)$,

(ii) $\mathbb{E}((\boldsymbol{\nu} x)P) \approx (\boldsymbol{\nu} x)\mathbb{E}(P)$,

(iii) $\mathbb{E}(!P) \approx !\mathbb{E}(P)$,

(iv) $\mathbb{E}(\sum_{i=1}^{n} \pi_i.P_i) \approx \sum_{i=1}^{n} \pi_i.\mathbb{E}(P_i)$. □

PREUVE

Tout d'abord, notons qu'une simple induction sur la structure des termes permet d'établir que $\ulcorner P, \gamma \urcorner\{x \leftarrow y\} = \ulcorner P\{x \leftarrow y\}, \gamma \urcorner$, pourvu que les noms choisis pour le codage n'interfèrent pas avec les variables libres de P. Substituer un terme codé est donc équivalent à coder le terme substitué.

Dans chacun des cas et pour tout α, les termes dans la congruence ne sont pas observables en α avant réduction puisqu'ils communiquent sur le canal réservé γ_0. Nous devons montrer qu'ils sont toujours congruent après réduction. Dans tous les cas, le premier pas de réduction concerne des noms de canaux restreints et ne réalise aucune somme, réplication ou composition parallèle. La séquence des pas de réduction est par conséquence unique et maintient la $\approx$-congruence. La preuve utilise les propriétés usuelles de $\equiv$ données au chapitre 2.

Composition parallèle

$$\begin{aligned}
\mathbb{E}(P_1 \| P_2) \mathrel{\widehat{=}} \quad & (\boldsymbol{\nu}\gamma_0)\big(\mathtt{Eval}(\gamma_0) \parallel \\
& \qquad (\boldsymbol{\nu} abcde)(\overline{\gamma_0}[abcde].\overline{b}.(\boldsymbol{\nu}\gamma_1\gamma_2)\overline{\gamma_0}[\gamma_1\gamma_2].(\ulcorner P_1, \gamma_1 \urcorner \| \ulcorner P_2, \gamma_2 \urcorner))\big) \\
\approx \quad & \big((\boldsymbol{\nu}\gamma_1)(\ulcorner P_1, \gamma_1 \urcorner \parallel \mathtt{Eval}(\gamma_1))\big) \parallel \big((\boldsymbol{\nu}\gamma_2)(\ulcorner P_2, \gamma_2 \urcorner \parallel \mathtt{Eval}(\gamma_2))\big) \\
\mathrel{\widehat{=}} \quad & \mathbb{E}(P_1) \parallel \mathbb{E}(P_2)
\end{aligned}$$

Restriction

$$\begin{aligned}
\mathbb{E}((\boldsymbol{\nu} x)P) \mathrel{\widehat{=}} \quad & (\boldsymbol{\nu}\gamma_0)\big(\mathtt{Eval}(\gamma_0) \parallel \\
& \qquad (\boldsymbol{\nu} abcde)\overline{\gamma_0}[abcde].\overline{d}.(\boldsymbol{\nu}\gamma_1\gamma_2)\overline{\gamma_0}[\gamma_1\gamma_2].\gamma_1(x).\ulcorner P, \gamma_2 \urcorner\big)
\end{aligned}$$

$$
\begin{aligned}
&\equiv (\boldsymbol{\nu}\gamma_0\gamma_1\gamma_2)\big(\mathtt{Eval}(\gamma_0) \parallel \\
&\qquad\qquad (\boldsymbol{\nu}abcde)\overline{\gamma_0}[abcde].\overline{d}.\overline{\gamma_0}[\gamma_1\gamma_2].\gamma_1(x).\ulcorner P, \gamma_2 \urcorner\big) \\
&\approx (\boldsymbol{\nu}\gamma_1\gamma_2)\big((\boldsymbol{\nu}x')\overline{\gamma_1}[x'].\mathtt{Eval}(\gamma_2) \parallel \gamma_1(x).\ulcorner P, \gamma_2 \urcorner\big) \\
&\equiv (\boldsymbol{\nu}\gamma_1\gamma_2x')\big(\overline{\gamma_1}[x'].\mathtt{Eval}(\gamma_2) \parallel \gamma_1(x).\ulcorner P, \gamma_2 \urcorner\big) \\
&\approx (\boldsymbol{\nu}x')(\boldsymbol{\nu}\gamma_2)\big(\mathtt{Eval}(\gamma_2) \parallel \ulcorner P\{x \leftarrow x'\}, \gamma_2 \urcorner\big) \\
&\equiv (\boldsymbol{\nu}x)(\boldsymbol{\nu}\gamma_2)\big(\mathtt{Eval}(\gamma_2) \parallel \ulcorner P, \gamma_2 \urcorner\big) \\
&\widehat{=} (\boldsymbol{\nu}x)\mathbb{E}(P)
\end{aligned}
$$

Réplication

$$
\begin{aligned}
\mathbb{E}(!P) \widehat{=}\ & (\boldsymbol{\nu}\gamma_0)\big(\mathtt{Eval}(\gamma_0) \parallel (\boldsymbol{\nu}abcde)\overline{\gamma_0}[abcde].\overline{c}.!((\boldsymbol{\nu}\gamma)\overline{\gamma_0}[\gamma].\ulcorner P, \gamma \urcorner)\big) \\
\approx\ & \big(!(\gamma_0(\gamma).\mathtt{Eval}(\gamma)) \parallel !((\boldsymbol{\nu}\gamma)\overline{\gamma_0}[\gamma].\ulcorner P, \gamma \urcorner)\big) \\
\approx\ & \big((\boldsymbol{\nu}\gamma_1)(\mathtt{Eval}(\gamma_1) \parallel \ulcorner P, \gamma_1 \urcorner)\big) \parallel \big((\boldsymbol{\nu}\gamma_2)(\mathtt{Eval}(\gamma_2) \parallel \ulcorner P, \gamma_2 \urcorner)\big) \parallel \ldots \\
\widehat{=}\ & \mathbb{E}(P) \parallel \mathbb{E}(P) \parallel \ldots \widehat{=} !\mathbb{E}(P)
\end{aligned}
$$

Somme

$$
\begin{aligned}
\mathbb{E}(\textstyle\sum_{i=1}^{n} \pi_i.P_i) \approx\ & (\boldsymbol{\nu}\gamma_1 \ldots \gamma_n\delta_1 \ldots \delta_n) \\
& \big(\big(\textstyle\sum_{i=1}^{n/2} \alpha_i(x_i).\overline{\delta_i}[x_i].\mathtt{Eval}(\gamma_i) + \sum_{i=n/2+1}^{n} \overline{\alpha_i}[x_i].\overline{\delta_i}.\mathtt{Eval}(\gamma_i)\big) \\
& \parallel \big(\textstyle\sum_{i=1}^{n/2} \delta_i(x_i).\ulcorner P_i, \gamma_i \urcorner + \sum_{i=n/2+1}^{n} \delta_i.\ulcorner P_i, \gamma_i \urcorner\big)\big) \\
\widehat{=}\ & T
\end{aligned}
$$

$P = \sum_{i=1}^{n} \pi_i.P_i$ est observable exactement sur le même ensemble de canaux α_i, avec $1 \leq i \leq n$, et de même pour T. Si P communique sur α_i, il se réduit à P_i. Cette réduction est aussi possible pour T et nous obtenons $T \rightarrow^* (\boldsymbol{\nu}\gamma_i)(\mathtt{Eval}(\gamma_i) \parallel \ulcorner P_i, \gamma_i \urcorner)$. ainsi, $T \approx \sum_{i=1}^{n} \pi_i.\mathbb{E}(P_i)$, puisque les canaux δ_i sont restreints.

CQFD

Proposition 10 (Correction de l'évaluateur) *La propriété suivante est vérifiée pour tout $\boldsymbol{\pi}$-terme P.*

$$P \approx \mathbb{E}(P)$$

□

PREUVE

La preuve est une induction immédiate sur la structure de P, en utilisant le lemme 9.

– Si $P \widehat{=} \mathbf{0}$ alors $P \not\downarrow_\alpha$, pour tout α.

$$\mathbb{E}(\mathbf{0}) \widehat{=} (\boldsymbol{\nu}\gamma_0)\big(\mathtt{Eval}(\gamma_0) \parallel (\boldsymbol{\nu}abcde)(\overline{\gamma_0}[abcde].\overline{a})\big)$$

Clairement, $\mathbb{E}(\mathbf{0}) \not\downarrow_\alpha$ pour tout α, puisque toutes les variables sont restreintes. Ainsi,

$\mathbb{E}(\mathbf{0})$ se réduit à $\mathbf{0}$ sans être observable.

- Si $P \widehat{=} P_1 \parallel P_2$ alors $P{\downarrow_\alpha}$, ssi $P_1{\downarrow_\alpha}$ ou $P_2{\downarrow_\alpha}$. Maintenant, $\mathbb{E}(P_1 \parallel P_2) \approx \mathbb{E}(P_1) \parallel \mathbb{E}(P_2)$, par le lemme 9. Donc, $\mathbb{E}(P){\downarrow_\alpha}$, ssi $\mathbb{E}(P_1){\downarrow_\alpha}$ ou $\mathbb{E}(P_2){\downarrow_\alpha}$.
- Si $P \widehat{=} !P_1$ alors $P{\downarrow_\alpha}$, ssi $P_1{\downarrow_\alpha}$. D'après le lemme 9, $\mathbb{E}(!P) \approx !\mathbb{E}(P)$.
- Si $P \widehat{=} (\boldsymbol{\nu} x)P_1$ alors $P{\downarrow_\alpha}$, ssi $P_1{\downarrow_\alpha}$ et $\alpha \neq x$. $\mathbb{E}((\boldsymbol{\nu} x)P_1) \approx (\boldsymbol{\nu} x)\mathbb{E}(P_1)$, d'après le lemme 9.
- Si $P \widehat{=} \sum_{i=1}^{n} \pi_i.P_i$ alors P est observable exactement sur les sujets des communications π_i et P se réduit à P_i, ssi la communication π_i est exécutée. D'après le lemme 9, $\mathbb{E}(P) \approx \sum_{i=1}^{n} \pi_i.\mathbb{E}(P_i)$. Donc, $\mathbb{E}(P)$ est observable exactement sur les sujets des π_i et se réduit à $\mathbb{E}(P_i)$, ssi π_i est exécuté.

CQFD

3.4 Annotations

Dans cette section, nous présentons un système d'annotation des programmes permettant d'indiquer quelles sous-expressions d'un $\boldsymbol{\pi}$-terme sont statiques et lesquelles sont dynamiques. Ce système est donné sous la forme d'un $\boldsymbol{\pi}$-calcul à deux niveaux dans lequel les opérations du premier niveau sont statiques, tandis que celles du second sont dynamiques. Nous définissons ensuite l'ensemble des $\boldsymbol{\pi}$-termes bien annotés. La notion de bonne annotation permet d'affirmer que lorsqu'une communication est du premier niveau, elle peut être exécutée par l'évaluateur partiel de la section 3.5 sans invalider la correction du programme résiduel. Pour cela on impose notamment que le sujet d'une communication statique soit un nom défini localement dans le terme que nous traitons et qui n'est jamais exporté à l'extérieur de celui-ci. En conséquence, toute analyse des temps de liaison qui produit des termes bien annotés est compatible avec l'évaluateur partiel.

3.4.1 Le $\boldsymbol{\pi}$-calcul à deux niveaux

Un évaluateur partiel exécute les expressions d'un programme dépendant seulement des données connues statiquement. Nous commençons par préciser ce que cela signifie dans notre contexte. Soit $C[P]$ un système distribué représenté par un $\boldsymbol{\pi}$-terme P apparaissant dans un contexte $C[]$. P réalise des communications sur des canaux internes, inconnus de $C[]$, et des communications externes, c.à.d. des communications sur des canaux connus de $C[]$ et potentiellement partagés par P et Q pour un certain processus Q apparaissant dans $C[]$. Par conséquent, l'exécution de P dans l'environnement $C[]$ correspond à la réduction du terme $C[P]$.

Supposons maintenant que $C[]$ n'est pas exécuté parallèlement à P au moment de l'évaluation partielle, ce qui revient à considérer que $C[]$ est le processus vide $\mathbf{0}$. Evaluer

partiellement P en fonction de ses données connues consiste alors à réaliser les communications décrites par P qui utilisent des canaux locaux à P et à résidualiser les autres. Intuitivement, nous devons, d'une part, indiquer quelles communications dans P utilisent uniquement des noms de canaux locaux à P et, d'autre part, indiquer quelles parties de P ne peuvent pas être réduites en l'absence de $C[]$.

Annoter un $\boldsymbol{\pi}$-terme P en vue de son évaluation partielle consiste à indiquer quelles communications, c.à.d. quelles sommes de P, peuvent être réduites dans l'environnement partiel et lesquelles doivent être conservées dans le programme résiduel. Une communication est du *premier niveau*, ou *statique*, si elle fait intervenir des noms de canaux restreints. Une communication est du *second niveau*, ou *dynamique*, si elle fait éventuellement intervenir des noms de canaux connus à l'extérieur de P. L'évaluateur partiel présenté dans la section 3.5 réduit les communications du premier niveau et résidualise celles du second niveau. Les opérateurs pour la composition parallèle de termes et la réplication décrivent la structure du programme et sont toujours résidualisés. Les créations de nouveaux noms de canaux enrichissent l'environnement statique et sont toujours exécutées par l'évaluateur partiel. Aussi, les annotations ne concernent que l'opérateur somme.

Définition 11 ($\boldsymbol{\pi}$-termes à deux niveaux) *Un $\boldsymbol{\pi}$-terme à deux niveaux est une expression générée par la grammaire suivante.*

$$\omega \ ::= \ \sum_{i=0}^{n} \pi_i.\omega_i \ \mid \ \underline{\sum}_{i=1}^{n} \pi_i.\omega_i \ \mid \ \omega_1 \| \omega_2 \ \mid \ !\omega \ \mid \ (\boldsymbol{\nu} x)\,\omega$$

avec $\pi \ ::= \ \overline{x}[y] \ \mid \ x(y)$. □

Une somme soulignée indique un comportement dynamique. Lorsqu'une somme unaire est écrite comme une communication, nous soulignons cette dernière. Si aucun des préfixes π_i ne peut communiquer, la somme est évidemment dynamique. A l'opposé, si tous les préfixes π_i peuvent communiquer, la somme est statique. Un problème survient pour annoter une somme contenant à la fois des communications statiques et des communications dynamiques. Puisqu'aucun $\boldsymbol{\pi}$-terme à deux niveaux ne les décrit, nous devons donner une approximation sous la forme d'un $\boldsymbol{\pi}$-terme à deux niveaux pur, c.à.d. sous la forme d'une somme dont les préfixes sont soit tous statiques, soit tous dynamiques. Dans ce qui suit, nous considérons deux manières de réaliser cette approximation.

3.4.2 Exemples d'annotations

Comme indiqué précédemment, annoter un $\boldsymbol{\pi}$-terme P consiste à partitionner les communications en deux ensembles, selon qu'elles utilisent des canaux connus localement dans P ou qu'elles utilisent des canaux éventuellement connus hors de P. La notion

de bonne annotation est formellement définie dans la section 3.4.3. Ici, nous motivons notre choix pour ce critère à travers quelques exemples. Considérons le jugement

$$\models (\boldsymbol{\nu} ab)\ \left(a(x).\overline{c}[x].\mathbf{0} \parallel \overline{a}[b].\mathbf{0}\right) \ \triangleright\ (\boldsymbol{\nu} ab)\ \left(a(x).\underline{\overline{c}[x]}.\mathbf{0} \parallel \overline{a}[b].\mathbf{0}\right) \tag{3.5}$$

L'assertion de l'équation (3.5) indique que le terme $P \mathrel{\widehat{=}} (\boldsymbol{\nu} ab)\left(a(x).\overline{c}[x].\mathbf{0} \parallel \overline{a}[b].\mathbf{0}\right)$ est bien annoté par le terme à deux niveaux $\omega \mathrel{\widehat{=}} (\boldsymbol{\nu} ab)\left(a(x).\underline{\overline{c}[x]}.\mathbf{0} \parallel \overline{a}[b].\mathbf{0}\right)$ dans lequel la seule communication dynamique est $\underline{\overline{c}[x]}$. Ceci implique que $a(x)$ et $\overline{a}[b]$ seront réduits à l'évaluation partielle, et que l'on obtiendra le terme résiduel $P_r \mathrel{\widehat{=}} (\boldsymbol{\nu} b)\overline{c}[b].\mathbf{0}$. Ceci est correct, puisque $P \approx P_r$. Substituer P_r à P et exécuter le système $C[P_r]$ au lieu du système initial $C[P]$ ne modifiera pas le comportement du système global quel que soit le contexte $C[]$, et un processus Q présent dans $C[]$ ne pourra pas observer de différence entre les comportements de P et P_r. Le fait que les noms a et b soient restreints dans P autorise les annotations données dans l'équation (3.5). L'équation (3.6) montre les conséquences de la suppression de ces restrictions.

$$\models \left(a(x).\overline{c}[x].\mathbf{0} \parallel \overline{a}[b].\mathbf{0}\right) \ \triangleright\ \left(\underline{a(x)}.\underline{\overline{c}[x]}.\mathbf{0} \parallel \underline{\overline{a}[b]}.\mathbf{0}\right) \tag{3.6}$$

Dans le terme $P \mathrel{\widehat{=}} (a(x).\overline{c}[x].\mathbf{0} \parallel \overline{a}[b].\mathbf{0})$, a est un nom de canal non restreint, qui, par conséquent, peut être connu hors de P. Ainsi, pour un contexte $C[]$ inconnu au moment de l'évaluation partielle, nous ne pouvons pas déterminer si, lors de l'exécution de $C[P]$, la réception $a(x)$ présente dans P se synchronisera avec l'émission $\overline{a}[b]$ présente dans ce même terme ou avec une autre émission $\overline{a}[z]$ présente dans un processus Q de $C[]$.

Par ailleurs, nous acceptons le jugement de l'équation (3.7) qui décrit une variante de celui de l'équation (3.5) dans laquelle l'ordre des communications sur les canaux a et c a été inversé.

$$\models (\boldsymbol{\nu} ab)\left(c(y).a(x).\mathbf{0} \parallel \overline{a}[b].\mathbf{0}\right) \ \triangleright\ (\boldsymbol{\nu} ab)\left(\underline{c(y)}.a(x).\mathbf{0} \parallel \overline{a}[b].\mathbf{0}\right) \tag{3.7}$$

Le jugement de l'équation (3.7) est correct car la valeur reçue par $c(y)$ n'est pas utilisée dans la communication suivante. Un évaluateur partiel peut exécuter $a(x)$ sans connaître la valeur de y et le programme résiduel $P_r \mathrel{\widehat{=}} c(y).\mathbf{0}$ est correct.

S'il est clair que les communications utilisant des noms de canaux libres dans P sont dynamiques, celles utilisant des noms locaux à P doivent aussi parfois être annotées comme étant dynamiques. Considérons par exemple le jugement suivant.

$$\models (\boldsymbol{\nu} a)\left(b(y).\overline{y}[a].\mathbf{0}\right) \ \triangleright\ (\boldsymbol{\nu} a)\left(\underline{b(y)}.\underline{\overline{y}[a]}.\mathbf{0}\right) \tag{3.8}$$

Le nom y est local à P mais reçoit une valeur à travers une réception dynamique sur

le canal b. En conséquence, l'émission $\overline{y}[a]$ devient aussi dynamique dans l'équation (3.8). Ainsi, un nom local qui reçoit éventuellement, directement ou indirectement, une valeur qui est un nom connu hors de P est aussi dynamique. Un autre exemple est donné par l'équation (3.9).

$$\models (\boldsymbol{\nu} a)\left(\overline{b}[a].a(x).\mathbf{0}\right) \ \triangleright\ (\boldsymbol{\nu} a)\left(\underline{\overline{b}[a]}.\underline{a(x)}.\mathbf{0}\right) \tag{3.9}$$

Le nom de canal a est créé localement mais peut être transmis par l'instruction $\overline{b}[a]$ à n'importe quel processus hors du terme P. La réception $a(x)$ ne peut pas être considérée comme étant statique, car nous ne sommes pas assurés que cette communication a lieu avec un autre terme connu au moment de l'évaluation partielle.

En résumé, les noms locaux à P susceptibles "d'importer" des noms externes à P, ou susceptibles d'être "exportés" hors de P, doivent être dynamiques. Ainsi, une communication est statique seulement si son sujet est un nom local à P, qui ne peut ni prendre une valeur connue hors de P, ni être exporté hors de P. Une autre condition doit cependant être ajoutée aux précédentes, que nous illustrons à travers les exemples des équations (3.10) et (3.11).

$$\models (\boldsymbol{\nu} a)\left(\overline{a}[b].\mathbf{0} \parallel a(x).\overline{x}[c].\mathbf{0}\right) \quad \triangleright \quad (\boldsymbol{\nu} a)\left(\overline{a}[b].\mathbf{0} \parallel a(x).\underline{\overline{x}[c]}.\mathbf{0}\right) \tag{3.10}$$

$$\models (\boldsymbol{\nu} a)\left(c(b).\overline{a}[b].\mathbf{0} \parallel a(x).\overline{x}[c].\mathbf{0}\right) \quad \triangleright \quad (\boldsymbol{\nu} a)\left(\underline{c(b)}.\underline{\overline{a}[b]}.\mathbf{0} \parallel \underline{a(x)}.\underline{\overline{x}[c]}.\mathbf{0}\right) \tag{3.11}$$

Dans les deux cas, le nom a est local au terme. Dans l'exemple (3.10), l'émission $\overline{a}[b]$ est statique et transmet le nom libre b. Le programme résiduel est simplement $\overline{b}[c].\mathbf{0}$. Le second exemple diffère au sens où la valeur transmise par $\overline{a}[b]$ est un nom lié qui ne peut pas recevoir la véritable valeur au moment de l'évaluation partielle. Si nous avions annoté le terme par $(\boldsymbol{\nu} a)(\underline{c(b)}.\overline{a}[b].\mathbf{0} \parallel a(x).\underline{\overline{x}[c]}.\mathbf{0})$, nous aurions obtenu le programme résiduel $(\boldsymbol{\nu} a)(c(b).\mathbf{0} \parallel \overline{b}[c].\mathbf{0})$. Clairement, ceci n'est pas correct car le nom b se retrouve hors de la portée de son lieur $c(b)$. Les restrictions que nous déduisons de ce cas d'étude sont immédiates. Une émission dont le sujet est a priori statique peut être réduite à l'évaluation partielle seulement si la valeur transmise n'est pas un nom lié recevant sa valeur à travers une réception dynamique.

3.4.3 Bonne annotation des π-termes à deux niveaux

La notion de bonne annotation conservatrice et celle de bonne annotation optimisée des $\boldsymbol{\pi}$-termes à deux niveaux ont pour but de garantir que les programmes initiaux et résiduels seront $\approx$-congruents ou $\prec$-simulables, selon le point de vue que l'on adopte. Pour cela, toute communication observable à l'extérieur de P doit être résidualisée. Les

seules communications statiques sont celles dont le sujet n'est pas connu hors de P. Autrement dit, les communications statiques dans un terme P sont celles qui ne sont pas observables par un terme Q présent dans un contexte $C[]$ lors d'une exécution de $C[P]$. Naturellement, il est de plus nécessaire que l'objet de l'émission soit une valeur connue.

Parce que les noms sont substitués pendant la réduction de P, nous devons connaître l'ensemble de tous les noms pouvant être utilisés en tant que sujet d'une communication π au cours de n'importe quelle exécution. Aussi, nous introduisons les annotations $\mathcal{V}(x)$, pour toute occurrence d'un nom x dans P. $\mathcal{V}(x)$ est clairement une information dynamique dont nous devrons calculer statiquement une approximation conservatrice afin de calculer effectivement une analyse. Nous précisons ce point dans la section 3.4.4.

Définition 12 (Annotation des noms de canaux) *Soit x une occurrence d'un nom de canal dans un $\boldsymbol{\pi}$-terme P. $\mathcal{V}(x)$ représente l'ensemble des noms de canaux pouvant être éventuellement substitués à x pendant la réduction de $C[P]$, pour tout contexte $C[]$. Dans tous les cas on a $\{x\} \subseteq \mathcal{V}(x)$.* □

Connaissant l'ensemble $\mathcal{V}(x)$ des noms pouvant être substitués à une occurrence de x, nous devons partitionner l'ensemble des canaux en deux catégories, selon qu'une communication sur un canal donné soit autorisée à l'évaluation partielle ou pas. Nous considérons trois sortes de noms pour un terme P.

- $\mathcal{B}(P)$ est l'ensemble des noms liés dans P, c.à.d. tout nom x se trouvant sous la portée d'une restriction $(\boldsymbol{\nu} x)$ ou d'une réception $y(x)$.
- $\mathcal{F}(P)$ est l'ensemble des noms libres dans P, c.à.d. les noms apparaissant dans P mais qui n'appartiennent pas à l'ensemble $\mathcal{B}(P)$. Implicitement, nous supposons que les ensembles $\mathcal{B}(P)$ et $\mathcal{F}(P)$ sont disjoints, ce qui peut toujours être obtenu par renommage des noms liés.
- $\mathcal{O}(P)$ est l'ensemble des noms n'apparaissant pas dans P.

Dans un premier temps, nous faisons une distinction entre les annotations d'un nom de canal et les annotations de ses occurrences. Un nom de canal est *statique*, resp. *dynamique*, si toutes ses occurrences sont statiques, resp. dynamiques. Dans un second temps, nous reportons les annotations d'un noms à celles de ses occurrences. Nous illustrons ceci sur le jugement de l'équation (3.12) dans lequel les occurrences du nom a sont indexées pour les besoins de l'explication. L'occurrence a_1 est dynamique car le nom peut être envoyé hors du terme sur le nom de canal libre c. En conséquence, le nom a devient dynamique, c.à.d. le lieur $(\boldsymbol{\nu} a_0)$ devient dynamique. Finalement, l'occurrence a_2 est aussi annotée dynamique afin d'obtenir un état cohérent.

$$\models (\boldsymbol{\nu} a_0 b)\left(\overline{c}[a_1].\mathbf{0} \parallel a_2(b).\mathbf{0}\right) \triangleright (\boldsymbol{\nu} a_0 b)\left(\underline{\overline{c}[a_1]}.\mathbf{0} \parallel \underline{a_2(b)}.\mathbf{0}\right) \tag{3.12}$$

Nous définissons maintenant les annotations pour une occurrence d'un nom. Soit x le sujet d'une communication, qu'il s'agisse d'une émission ou d'une réception. Pour que cette occurrence soit statique il est nécessaire que $\mathcal{V}(x) \subseteq \mathcal{B}(P)$, ce qui signifie que toutes les valeurs que x peut prendre sont des noms liés dans P. Cependant, cette condition n'est pas suffisante car, comme nous l'avons vu, une communication ne peut demeurer statique que si nous pouvons garantir que les noms pouvant être affectés à x ne sont pas connus hors de P. Aussi, pour tout $\alpha \in \mathcal{V}(x)$, nous devons vérifier qu'il n'y a pas d'émission $\overline{z}[t]$ dans P telle que $\alpha \in \mathcal{V}(t)$ et $\mathcal{V}(z) \nsubseteq \mathcal{B}(P)$. En effet, une telle émission peut communiquer avec un processus défini hors de P, car $\mathcal{V}(z) \nsubseteq \mathcal{B}(P)$, et transmettre le nom α, car $\alpha \in \mathcal{V}(t)$, hors du terme P.

Définition 13 (Réceptions statiques) *Soit P un $\boldsymbol{\pi}$-terme et $x(y)$ l'occurrence d'une réception dans P. $x(y)$ est statique si les conditions suivantes sont vérifiées.*

(i) $\mathcal{V}(x) \subseteq \mathcal{B}(P)$.

(ii) $\forall \alpha \in \mathcal{V}(x),\ \nexists \overline{z}[t]$ *dans* $P\ :\ \alpha \in \mathcal{V}(t)$ *et* $\mathcal{V}(z) \nsubseteq \mathcal{B}(P)$. □

Pour les réceptions, ces conditions sont suffisantes. Pour qu'une émission $\overline{x}[y]$ soit statique, nous devons de plus vérifier certaines propriétés sur les noms pouvant être affectés à l'objet y, comme le montrent les exemples des équation (3.10) et (3.11). Clairement, si $\mathcal{V}(y) \nsubseteq \mathcal{B}(P) \cup \mathcal{F}(P)$, la valeur de y peut provenir de l'extérieur du terme P. Par conséquent, l'émission $\overline{x}[y]$ devient dynamique. Cependant, même lorsque cette condition est vérifiée, comme dans l'équation (3.11), nous attachons l'annotation dynamique au point d'émission si, pour un $z \in \mathcal{V}(y)$, il existe une réception dynamique $u(v)$ dans P telle que $z \in \mathcal{V}(v)$. Cela signifie que la valeur z que le nom y peut prendre ($z \in \mathcal{V}(y)$) a éventuellement été reçue ($z \in \mathcal{V}(v)$) au point de réception dynamique $u(v)$.

Définition 14 (Emissions statiques) *Soit P un $\boldsymbol{\pi}$-terme et $\overline{x}[y]$ l'occurrence d'une émission dans P. $\overline{x}[y]$ est statique si les conditions suivantes sont vérifiées.*

(i) $\mathcal{V}(x) \subseteq \mathcal{B}(P)$.

(ii) $\forall \alpha \in \mathcal{V}(x),\ \nexists \overline{z}[t]$ *dans* $P\ :\ \alpha \in \mathcal{V}(t)$ *et* $\mathcal{V}(z) \nsubseteq \mathcal{B}(P)$.

(iii) $\mathcal{V}(y) \subseteq \mathcal{B}(P) \cup \mathcal{F}(P)$.

(iv) $\forall z \in \mathcal{V}(y),\ \forall u(v)$ *dans* $P,\ z \notin \mathcal{V}(v)$. □

Jusqu'à présent nous avons seulement considéré de simples occurrences de communications, c.à.d. des sommes unaires. Lorsque les sommes ont plus d'un élément, les différents membres s'influencent mutuellement. Si tous les préfixes d'une somme sont statiques ou dynamiques, la somme prend cette même annotation. Un problème survient lorsque nous devons définir la notion de bonne annotation pour une somme composée

de communications statiques et dynamiques. La première façon d'aborder ce problème est dite conservatrice, car une somme devient dynamique dès que l'un de ses préfixes l'est. Cela signifie que tous ses préfixes sont annotés dynamiques. En conséquence, aucun préfixe ne peut être réduit statiquement et la congruence barbelée faible $\approx$ est conservée entre le terme initial et le terme résiduel calculé par l'évaluateur partiel décrit à la section 3.5. Les deux termes sont observables strictement sur les mêmes canaux.

Définition 15 (Bonne annotation conservatrice) *Un π-terme P et bien annoté de manière conservatrice par ω, ce qui est noté $\models_C P \triangleright \omega$, si ω est un terme à deux niveaux ayant exactement la même structure que P et tel que les préfixes de toutes les sommes statiques, resp. dynamiques, sont statiques, resp. dynamiques.* □

Aussitôt qu'un des termes de la somme est dynamique, la somme entière doit être annotée dynamique. Ceci est illustré par l'assertion de l'équation (3.13).

$$\models_C (\boldsymbol{\nu} a)\left(a(x) + b(y) \parallel \overline{a}[c]\right) \triangleright (\boldsymbol{\nu} a)\left(a(x) \underline{+} b(y) \parallel \underline{\overline{a}[c]}\right) \tag{3.13}$$

Il est possible d'obtenir des programmes résiduels plus optimisés en autorisant l'évaluateur partiel à résoudre autant de non-déterminisme que possible. Cette seconde annotation est obtenue en posant qu'une somme est statique aussitôt qu'un de ses préfixes l'est. De cette manière, l'évaluateur partiel choisit seulement parmi les communications statiques et supprime celles qui sont dynamiques. Ainsi, il adopte un des comportements possibles pour le programme initial et perd connaissance des autres. Les programmes résiduels obtenus ne sont pas observables sur tous les canaux sur lesquels le programme initial était observable. Cependant, cette approche est acceptable car le programme résiduel adopte un des comportements que le programme original aurait pu adopter.

Formellement, une seconde définition de la notion de bonne annotation est donnée ci-dessous. Une somme est statique dès qu'au moins l'un de ses préfixes l'est. Les notions de bonne annotation conservatrice et optimisée sont liées. S'il est possible de supprimer certains, mais pas tous, membres d'une somme et d'obtenir ainsi une somme statique au sens de la notion de bonne annotation conservatrice, alors le terme inital est bien annoté comme étant statique au sens de la notion de bonne annotation optimisée.

Définition 16 (Bonne annotation optimisée) *Soit P un π-terme et ω un π-terme à deux niveaux. ω est une version bien annotée de P, ce qui est noté $\models_O P \triangleright \omega$, si ω est un terme qui n'a pas nécessairement la même structure que P, mais vérifie les conditions suivantes. Soit P' un terme obtenu à partir de P en supprimant dans les sommes certains, mais pas tous, membres, de telle manière que P' et ω ont la même structure. Pour le terme P' obtenu de cette manière nous avons $\models_C P' \triangleright \omega$.* □

Cette nouvelle définition permet le jugement suivant pour le terme de l'équation (3.13), indiquant que la communication sur a sera privilégiée à celle sur b.

$$\models_O (\boldsymbol{\nu} a)\left(a(x) + b(y) \parallel \overline{a}[c]\right) \triangleright (\boldsymbol{\nu} a)\left(a(x) \parallel \overline{a}[c]\right) \tag{3.14}$$

Les notions de bonne annotation ont pour but d'assurer la correction d'un évaluateur partiel qui exécute les communications du premier niveau et résidualise celles du second. Intuitivement, les définitions 15 et 16 sont correctes pour les raisons suivantes. Toute exécution de $C[P]$ qui atteint un point de communication statique π aura pour sujet un certain canal a, restreint dans P, connu au moment de l'évaluation partielle et inconnu hors de P. Par conséquent, l'interprétation de π par l'évaluateur partiel est indépendante du contexte $C[]$. Cependant, remarquons que l'évaluation partielle de termes bien annotés peut conduire à des situations de blocage. Les jugements $\models_{C,O} (\boldsymbol{\nu} a)a(x).\mathbf{0} \triangleright (\boldsymbol{\nu} a)a(x).\mathbf{0}$ considèrent que la réception sur a est statique. Cependant, a étant un nom restreint, l'évaluateur partiel bloquera. Ce comportement est acceptable car l'évaluation du terme original bloque aussi. Des situations comparables sont présentes dans l'évaluation partielle de langages séquentiels, par exemple lors de l'exécution d'une boucle statique infinie.

Nous achevons cette section en présentant une propriété vérifiée par les termes bien annotés, de manière conservatrice ou optimisée. Cette propriété établit que si $y \in \mathcal{V}(x)$, alors substituer y à x dans ω préserve les propriétés de bonne annotation.

Proposition 17 (substitution) *Soit P un $\boldsymbol{\pi}$-terme et x un nom apparaissant dans P. La propriété suivante est vérifiée*

$$\models P \triangleright \omega \text{ et } y \in \mathcal{V}(x) \Rightarrow \models P\{x \leftarrow y\} \triangleright \omega\{x \leftarrow y\}$$

où $\models$ représente $\models_C$ ou $\models_O$. □

3.4.4 Analyse de flot de contrôle

Les annotations que nous utilisons sont fondées sur l'ensemble $\mathcal{V}(x)$ associé à une occurrence du nom x dans un terme P. $\mathcal{V}(x)$ est un sur-ensemble des valeurs pouvant être substituées à x dans une exécution quelconque de $C[P]$, pour un contexte $C[]$ quelconque. Comme le calcul exact de cet ensemble n'est pas possible en général, il est nécessaire d'utiliser une analyse statique afin de calculer une approximation conservatrice de cet ensemble. Naturellement, nous souhaitons que cette approximation soit aussi précise que possible.

Le calcul des ensembles $\mathcal{V}(x)$ est un problème de flot de contrôle. Comme discuté au chapitre 2, des CFA pour des systèmes de processus mobiles ont été proposées par Sol-

berg et al. (SNN97) et par Bodei et al. (BDNN98). Bodei et al. ont proposé une CFA pour le π-calcul pur. Pour un terme P, cette analyse consiste tout d'abord à relier chaque nom lié de P à son lieur. Ensuite, elle associe à chaque nom x apparaissant syntaxiquement dans un terme P, une approximation conservatrice $\mathcal{V}^R(x)$ de l'ensemble $\mathcal{V}(x)$ des noms pouvant être substitués à x durant l'exécution de P. De plus, cette CFA calcule, pour tout nom de canal x, l'ensemble $\mathcal{K}(x)$ des noms de canaux pouvant être communiqués sur x. Ainsi, pour chaque émission $\overline{x}[y]$ telle que $\alpha \in \mathcal{V}^R(x)$, $\mathcal{V}^R(y)$ est ajouté à $\mathcal{K}(\alpha)$. Réciproquement, pour chaque réception $x(y)$ telle que $\alpha \in \mathcal{V}^R(x)$, $\mathcal{K}(\alpha)$ est ajouté à $\mathcal{V}^R(y)$.

Nous présentons, au chapitre 4, une autre analyse de flot de contrôle permettant d'obtenir des annotations plus précises, et, par conséquent, de reconnaître plus souvent que des communications sont statiques.

3.5 Evaluation partielle

Dans cette section, nous montrons comment évaluer des π-termes à deux niveaux. Tout d'abord, nous codons ces derniers en utilisant une fonction $\llcorner . \lrcorner$. Ensuite, nous présentons l'extension naturelle de `Eval` qui donne un évaluateur partiel et nous prouvons sa correction par rapport aux notions de $\approx$-congruence et de $\prec$-simulation. Cet évaluateur partiel prend pour argument un programme bien annoté et produit un programme résiduel codé par la fonction $\ulcorner . \urcorner$.

3.5.1 L'évaluateur partiel

Comme `Eval`, l'évaluateur partiel `Pev` lit un programme d'entrée sur un canal réservé μ_0. De plus, `Pev` produit un programme résiduel codé par la fonction $\ulcorner . \urcorner$, qui est transmis sur un canal de sortie γ_0. Ainsi $\texttt{Pev}(\mu_0, \gamma_0)$ lit un programme d'entrée sur le canal μ_0 et produit un programme résiduel codé sur le canal γ_0.

Le codage des termes annotés s'effectue par la fonction $\llcorner . \lrcorner$ qui diffère seulement de $\ulcorner . \urcorner$ dans le traitement des sommes. Cette fonction est donnée dans la figure 3.5, où ω, ω_1, etc. sont des expressions à deux niveaux.

Comme nous l'avons déjà précisé, deux cas doivent être distingués lorsque nous codons les sommes. Le message $\overline{v}$ indique que $\sum_{i=1}^{n} \pi_i.\omega_i$ est statique, c.à.d. que les π_i peuvent communiquer. Le message $\overline{w}$ est envoyé lorsque les communications ne peuvent pas avoir lieu au moment de l'évaluation partielle. Lorsqu'une communication π est annotée statique, π est exécutée. Sinon `Pev` calcule le code du programme résiduel et évalue partiellement, de façon récursive, les sous-expressions du programme. Comme à la section 3.3, la récursion est laissée implicite. L'évaluateur partiel a la forme générale donnée dans la figure 3.6. Les termes `A` à `F` correspondent respectivement au processus nul, à

$$
\begin{array}{rcl}
\llcorner \mathbf{0}, \mu_0 \lrcorner & \widehat{=} & (\boldsymbol{\nu} rstuvw)\overline{\mu_0}[rstuvw].\overline{r} \\
\llcorner \omega_1 \| \omega_2, \mu_0 \lrcorner & \widehat{=} & (\boldsymbol{\nu} rstuvw)\overline{\mu_0}[rstuvw].\overline{s}.(\boldsymbol{\nu}\mu_1\mu_2) \\
 & & \overline{\mu_0}[\mu_1\mu_2].(\llcorner \omega_1, \mu_1 \lrcorner \| \llcorner \omega_2, \mu_2 \lrcorner) \\
\llcorner !\omega, \mu_0 \lrcorner & \widehat{=} & (\boldsymbol{\nu} rstuvw)\overline{\mu_0}[rstuvw].\overline{t}.!((\boldsymbol{\nu}\mu)\overline{\mu_0}[\mu].\llcorner \omega, \mu \lrcorner) \\
\llcorner (\boldsymbol{\nu} x)\omega, \mu_0 \lrcorner & \widehat{=} & (\boldsymbol{\nu} rstuvw)\overline{\mu_0}[rstuvw].\overline{u}. \\
 & & (\boldsymbol{\nu}\mu_1\mu_2)\overline{\mu_0}[\mu_1\mu_2].\mu_1(x).\llcorner \omega, \mu_2 \lrcorner \\
\llcorner \sum_{i=1}^{n} \pi_i.\omega_i, \mu_0 \lrcorner & \widehat{=} & (\boldsymbol{\nu} rstuvw)\overline{\mu_0}[rstuvw].\overline{v}.(\boldsymbol{\nu}\mu_1 \ldots \mu_n\eta_1 \ldots \eta_n) \\
 & & \overline{\mu_0}[\mu_1 \ldots \mu_n\eta_1 \ldots \eta_n\alpha_1 \ldots \alpha_n x_1 \ldots x_n]. \\
 & & \quad \big(\sum_{i=1}^{n/2} \eta_i(x_i).\llcorner \omega_i, \mu_i \lrcorner + \sum_{i=n/2+1}^{n} \eta_i.\llcorner \omega_i, \mu_i \lrcorner \big) \\
 & & \text{avec} \quad \pi_i = \alpha_i(x_i) \ \text{ ou } \ \pi_i = \overline{\alpha_i}[x_i] \\
\llcorner \underline{\sum}_{i=1}^{n} \pi_i.\omega_i, \mu_0 \lrcorner & \widehat{=} & (\boldsymbol{\nu} rstuvw)\overline{\mu_0}[rstuvw].\overline{w}.(\boldsymbol{\nu}\mu_1 \ldots \mu_n\eta_1 \ldots \eta_n) \\
 & & \overline{\mu_0}[\mu_1 \ldots \mu_n\eta_1 \ldots \eta_n\alpha_1 \ldots \alpha_n x_1 \ldots x_n]. \\
 & & \quad \big(\sum_{i=1}^{n/2} \eta_i(x_i).\llcorner \omega_i, \mu_i \lrcorner + \sum_{i=n/2+1}^{n} \eta_i.\llcorner \omega_i, \mu_i \lrcorner \big) \\
 & & \text{avec} \quad \pi_i = \alpha_i(x_i) \ \text{ ou } \ \pi_i = \overline{\alpha_i}[x_i]
\end{array}
$$

FIG. 3.5 – **Codage des π-termes à deux niveaux.**

la composition parallèle, à la réplication, à la restriction, aux sommes statiques et dynamiques.

Soit $\llcorner \omega, \mu_0 \lrcorner$ le code d'un $\boldsymbol{\pi}$-terme à deux niveaux à évaluer partiellement. $\mathsf{Pev}(\mu_0, \gamma_0)$ lit $\llcorner \omega, \mu_0 \lrcorner$ sur μ_0, exécute les communications statiques de ω et produit le code $\ulcorner R, \gamma_0 \urcorner$ du programme résiduel R sur γ_0.

Lorsque le message r est rencontré, le programme résiduel généré sur γ_0 est le code du processus nul. Lors de l'évaluation partielle de la composition parallèle de ω_1 et ω_2, Pev reçoit sur μ_0 les noms μ_1 et μ_2 des sous-expressions, crée deux nouveaux canaux γ_1 et γ_2, ainsi que des canaux a, b, c, d et e. Il envoie sur γ_0 le nom b décrivant le type de l'opération codée avant d'exécuter $\mathsf{Pev}(\omega_1, \gamma_1)$ en parallèle à $\mathsf{Pev}(\omega_2, \gamma_2)$. Le code d'une réplication amène Pev à produire une variante de la fonction de codage $\ulcorner . \urcorner$, dans laquelle le processus devant être répliqué est partiellement évalué. Lorsque le code de $(\boldsymbol{\nu} x)$ est rencontré, Pev commence par lire sur μ_0 deux nouveaux noms de canaux μ_1 et μ_2. Ensuite, le nom x est créé par l'évaluateur partiel et transmis au codage sur μ_1. L'évaluation partielle du reste du terme est alors exécutée par $\mathsf{Pev}(\mu_2, \gamma_0)$. Notons que x est lié dans le terme D.

Pour les sommes, nous avons deux cas. Pour une somme statique, la communication est exécutée et le reste du terme est évalué partiellement. Pour une somme dynamique, ce qui correspond au terme F, nous procédons de la manière suivante. Pev reçoit les canaux μ_i sur lesquels les communications avec le terme codé prendront place au moment de l'évaluation partielle. Il reçoit de plus les noms η_i, sur lesquels il transmettra au terme

$$\mathsf{Pev}(\mu_0,\gamma_0) \mathrel{\widehat{=}} \mu_0(rstuvw).(\mathsf{A}+\mathsf{B}+\mathsf{C}+\mathsf{D}+\mathsf{E}+\mathsf{F})$$

$$\mathsf{A} \mathrel{\widehat{=}} r.\ulcorner \mathbf{0}, \gamma_0 \urcorner \mathrel{\widehat{=}} r.(\boldsymbol{\nu} abcde)\overline{\gamma_0}[abcde].\overline{a}$$

$$\mathsf{B} \mathrel{\widehat{=}} s.\mu_0(\mu_1\mu_2).\underbrace{(\boldsymbol{\nu} abcde)\overline{\gamma_0}[abcde].\overline{b}.(\boldsymbol{\nu}\gamma_1\gamma_2)\overline{\gamma_0}[\gamma_1\gamma_2].}_{\text{début du codage de } \ulcorner P_1 \| P_2, \gamma_0 \urcorner}\big(\mathsf{Pev}(\mu_1,\gamma_1) \parallel \mathsf{Pev}(\mu_2,\gamma_2)\big)$$

$$\mathsf{C} \mathrel{\widehat{=}} t.\underbrace{(\boldsymbol{\nu} abcde).\overline{\gamma_0}[abcde].\overline{c}.!((\boldsymbol{\nu}\gamma)\overline{\gamma_0}[\gamma]}_{\text{début du codage de } \ulcorner !P, \gamma_0 \urcorner}.\mu_0(\mu).\mathsf{Pev}(\mu,\gamma))$$

$$\mathsf{D} \mathrel{\widehat{=}} u.\mu_0(\mu_1\mu_2).\underbrace{(\boldsymbol{\nu} x)\overline{\mu_1}[x]}_{\text{nouveau canal créé par } \mathsf{Pev}}.\mathsf{Pev}(\mu_2,\gamma_0)$$

$$\mathsf{E} \mathrel{\widehat{=}} v.\mu_0(\mu_1\ldots\mu_n\eta_1\ldots\eta_n\alpha_1\ldots\alpha_n x_1\ldots x_n).\Big(\textstyle\sum_{i=1}^{n/2}\alpha_i(x_i).\overline{\eta_i}[x_i].\mathsf{Pev}(\mu_i,\gamma_0) + \sum_{i=n/2+1}^{n}\overline{\alpha_i}[x_i].\overline{\eta_i}.\mathsf{Pev}(\mu_i,\gamma_0)\Big)$$

$$\mathsf{F} \mathrel{\widehat{=}} w.\mu_0(\mu_1\ldots\mu_n\eta_1\ldots\eta_n\alpha_1\ldots\alpha_n x_1\ldots x_n).\underbrace{(\boldsymbol{\nu} abcde)\overline{\gamma_0}[abcde].\overline{e}.(\boldsymbol{\nu}\gamma_1\ldots\gamma_n\delta_1\ldots\delta_n)\overline{\gamma_0}[\gamma_1\ldots\gamma_n\delta_1\ldots\delta_n\alpha_1\ldots\alpha_n x_1\ldots x_n].}_{\text{début du codage de } \ulcorner \sum_{i=1}^{n}\pi_i.P_i,\gamma_0 \urcorner \text{ avec } \pi_i=\alpha_i(x_i) \text{ or } \pi_i=\overline{a_i}[x_i]}\big(\textstyle\sum_{i=1}^{n/2}\delta_i(x_i).\overline{\eta_i}[x_i].\mathsf{Pev}(\mu_i,\gamma_i) + \sum_{i=n/2+1}^{n}\delta_i.\overline{\eta_i}.\mathsf{Pev}(\mu_i,\gamma_i)\big)$$

FIG. 3.6 – **Evaluateur partiel π-calcul.**

codé le résultat du choix non-déterministe, ainsi que les noms α_i et x_i. Pev commence par coder la somme résiduelle, ce qui consiste à créer les canaux γ_i et δ_i et à les transmettre, ainsi que les noms α_i et x_i, à l'interpréteur. Ce dernier exécutera les communications et retournera les résultats, ou de simples signaux sur les canaux δ_i. Ces résultats et signaux sont transmis au terme codé sur les canaux η_i, afin d'indiquer le résultat du choix.

3.5.2 Correction

Dans cette section, nous prouvons la correction de l'évaluateur partiel, lorsque celui-ci est appliqué à des termes bien annotés, comme définis à la section 3.4. Avant de présenter les propriétés de correction satisfaites par Pev, nous introduisons les notations suivantes dans lesquelles ω représente un $\boldsymbol{\pi}$-terme à deux niveaux.

$$\mathbb{T}(\omega,\gamma_0) \mathrel{\widehat{=}} (\boldsymbol{\nu}\mu_0)\big(\llcorner\omega, \mu_0\lrcorner \parallel \mathsf{Pev}(\mu_0,\gamma_0)\big) \tag{3.15}$$

$$\mathbb{P}(\omega) \mathrel{\widehat{=}} (\boldsymbol{\nu}\gamma_0)\big(\mathbb{T}(\omega,\gamma_0) \parallel \mathtt{Eval}(\gamma_0)\big) \tag{3.16}$$

Le terme $\mathbb{T}(\omega, \gamma_0)$ est la composition parallèle du codage du terme annoté ω sur le canal μ_0, et du terme $\texttt{Pev}(\mu_0, \gamma_0)$. $\mathbb{T}(\omega, \gamma_0)$ représente l'évaluateur partiel appliqué à une version annotée ω d'un terme P. Le programme résiduel est transmis sur le canal γ_0. $\mathbb{P}(\omega)$ est la composition parallèle de $\mathbb{T}(\omega, \gamma_0)$ et de $\texttt{Eval}(\gamma_0)$, c.à.d. l'exécution parallèle de trois termes, le codage, l'évaluateur partiel et le méta-interpréteur. Nous prouvons la correction de l'évaluateur partiel en comparant les comportements de $\mathbb{P}(\omega)$ et P. Le terme $\mathbb{P}(\omega)$ correspond en fait à un *calcul étagé*, puisqu'il fait intervenir à la fois `Eval` et `Pev` qui interagissent ensemble pour calculer le résultat. Le $\boldsymbol{\pi}$-calcul permet seulement de réduire les communications de tête d'un terme P, comme le décrit la règle de l'équation (2.2). Par conséquent, la réduction des communications sous une somme dynamique ne peut être décrite, à moins d'ajouter un autre niveau de codage et d'interprétation.

Lemme 18 (Compositionalité de $\mathbb{P}$) *Soit P un $\boldsymbol{\pi}$-terme et ω une version bien annotée de P, au sens de la notion de bonne annotation conservatrice. Nous avons les propriétés suivantes pour le terme $\mathbb{P}(\omega)$. Un résultat similaire peut être développé pour les notions de bonne annotation optimisée et de simulation barbelée faible en substituant $\prec$ à $\approx$.*

(i) $\omega \mathrel{\widehat{=}} \omega_1 \parallel \omega_2 \Rightarrow \mathbb{P}(\omega) \;\approx\; \mathbb{P}(\omega_1) \parallel \mathbb{P}(\omega_2)$,

(ii) $\omega \mathrel{\widehat{=}} (\boldsymbol{\nu} x)\omega_1 \Rightarrow \mathbb{P}(\omega) \;\approx\; (\boldsymbol{\nu} x)\mathbb{P}(\omega_1)$,

(iii) $\omega \mathrel{\widehat{=}} !\omega_1 \Rightarrow \mathbb{P}(\omega) \;\approx !\mathbb{P}(\omega_1)$,

(iv) $\omega \mathrel{\widehat{=}} \sum_{i=1}^{n} \pi_i.P_i \Rightarrow \mathbb{P}(\omega) \;\approx\; \sum_{i=1}^{n} \pi_i.\mathbb{P}(\omega_i)$, *pour les sommes statiques et dynamiques.* □

PREUVE

Une simple induction sur la structure des termes permet d'établir que $\llcorner \omega, \mu \lrcorner \{x \leftarrow y\} = \llcorner \omega\{x \leftarrow y\}, \mu \lrcorner$, pourvu que les noms choisis pour le codage n'interfèrent pas avec les variables libres de ω.

La preuve est par induction sur la structure du terme ω. Dans tous les cas, les premiers pas de réduction concernent des communications déterministes sur des canaux restreints. La séquence de réduction est donc unique et préserve la $\approx$-congruence. Nous utilisons les propriétés usuelles de $\equiv$ données dans le chapitre 2.

Composition parallèle

$$\begin{aligned}
\mathbb{P}(\omega_1 \parallel \omega_2) \mathrel{\widehat{=}} & \ (\boldsymbol{\nu}\gamma_0)\big(\mathtt{Eval}(\gamma_0) \parallel (\boldsymbol{\nu}\mu_0)(\llcorner\omega_1 \parallel \omega_2, \mu_0\lrcorner \parallel \mathtt{Pev}(\mu_0,\gamma_0))\big) \\
\approx & \ (\boldsymbol{\nu}\gamma_0)\big(\mathtt{Eval}(\gamma_0) \parallel \\
& \qquad (\boldsymbol{\nu}\mu_1\mu_2)(\llcorner\omega_1,\mu_1\lrcorner \parallel \llcorner\omega_2,\mu_2\lrcorner \parallel \\
& \qquad\qquad (\boldsymbol{\nu}abcde)\ldots(\mathtt{Pev}(\mu_1,\gamma_1) \parallel \mathtt{Pev}(\mu_2,\gamma_2)))\big) \\
\approx & \ (\boldsymbol{\nu}\mu_1\mu_2\gamma_1\gamma_2)(\llcorner\omega_1,\mu_1\lrcorner \parallel \llcorner\omega_2,\mu_2\lrcorner \parallel \\
& \qquad\qquad \mathtt{Pev}(\mu_1,\gamma_1) \parallel \mathtt{Pev}(\mu_2,\gamma_2) \parallel \mathtt{Eval}(\gamma_1) \parallel \mathtt{Eval}(\gamma_2)) \\
\equiv & \ (\boldsymbol{\nu}\gamma_1)\big(\mathtt{Eval}(\gamma_1) \parallel (\boldsymbol{\nu}\mu_1)(\llcorner\omega_1,\mu_1\lrcorner \parallel \mathtt{Pev}(\mu_1,\gamma_1))\big) \\
& \ \parallel (\boldsymbol{\nu}\gamma_2)\big(\mathtt{Eval}(\gamma_2) \parallel (\boldsymbol{\nu}\mu_2)(\llcorner\omega_2,\mu_2\lrcorner \parallel \mathtt{Pev}(\mu_2,\gamma_2))\big) \\
\mathrel{\widehat{=}} & \ \mathbb{P}(\omega_1) \parallel \mathbb{P}(\omega_2)
\end{aligned}$$

Restriction

$$\begin{aligned}
\mathbb{P}((\boldsymbol{\nu}x)\,\omega) \approx & \ (\boldsymbol{\nu}\gamma_0)\big(\mathtt{Eval}(\gamma_0) \parallel \\
& \qquad (\boldsymbol{\nu}\mu_1\mu_2)(\mu_1(x).\llcorner\omega,\mu_2\lrcorner \parallel (\boldsymbol{\nu}abcde)\ldots(\boldsymbol{\nu}x)\overline{\mu_1}[x].\mathtt{Pev}(\mu_2,\gamma_0))\big) \\
\approx & \ (\boldsymbol{\nu}\gamma_0\mu_1\mu_2x')\big(\mathtt{Eval}(\gamma_0) \parallel \\
& \qquad\qquad \overline{\mu_1}[x'].\mathtt{Pev}(\mu_2,\gamma_0) \parallel \mu_1(x).\llcorner\omega,\mu_2\lrcorner\big) \\
\approx & \ (\boldsymbol{\nu}x')(\boldsymbol{\nu}\gamma_0)\big(\mathtt{Eval}(\gamma_0) \parallel (\boldsymbol{\nu}\mu_2)(\llcorner\omega\{x \leftarrow x'\},\mu_2\lrcorner \parallel \mathtt{Pev}(\mu_2,\gamma_0))\big) \\
\mathrel{\widehat{=}} & \ (\boldsymbol{\nu}x)\mathbb{P}(\omega)
\end{aligned}$$

Réplication

$$\begin{aligned}
\mathbb{P}(!\omega) \approx & \ (\boldsymbol{\nu}\gamma_0\mu_0)\big(!(\gamma_0(\gamma).\mathtt{Eval}(\gamma)) \parallel \\
& \qquad !((\boldsymbol{\nu}\gamma)\overline{\gamma_0}[\gamma].\mu_0(\mu).\mathtt{Pev}(\mu,\gamma)) \parallel !((\boldsymbol{\nu}\mu)\overline{\mu_0}[\mu].\llcorner\omega,\mu\lrcorner)\big) \\
\approx & \ (\boldsymbol{\nu}\gamma_1)\big(\mathtt{Eval}(\gamma_1) \parallel (\boldsymbol{\nu}\mu_1)(\mathtt{Pev}(\mu_1,\gamma_1) \parallel \llcorner\omega,\mu_1\lrcorner)\big) \parallel \ldots \\
\mathrel{\widehat{=}} & \ \mathbb{P}(\omega) \parallel \mathbb{P}(\omega) \parallel \ldots \mathrel{\widehat{=}} !\mathbb{P}(\omega)
\end{aligned}$$

Somme statique

$$\begin{aligned}
\mathbb{P}(\textstyle\sum_{i=1}^n \pi_i.\omega_i) \approx & \ (\boldsymbol{\nu}\gamma_0\mu_1\ldots\mu_n\eta_1\ldots\eta_n) \\
& \ \big(\mathtt{Eval}(\gamma_0) \parallel \\
& \ \big(\textstyle\sum_{i=1}^{n/2}\alpha_i(x_i).\overline{\eta_i}[x_i].\mathtt{Pev}(\mu_i,\gamma_0) + \sum_{i=n/2+1}^{n}\overline{\alpha_i}[x_i].\overline{\eta_i}.\mathtt{Pev}(\mu_i,\gamma_0)\big) \\
& \ \parallel \big(\textstyle\sum_{i=1}^{n/2}\eta_i(x_i).\llcorner\omega_i,\mu_i\lrcorner + \sum_{i=n/2+1}^{n}\eta_i.\llcorner\omega_i,\mu_i\lrcorner\big)\big) \\
& \ \mathrel{\widehat{=}} \mathsf{T}
\end{aligned}$$

Le terme T est observable exactement sur les canaux α_i, avec $1 \leq i \leq n$, tout comme le terme $\sum_{i=1}^n \pi_i.\mathbb{P}(\omega_i)$. Aussi, $\mathbb{P}(\sum_{i=1}^n \pi_i.\omega_i) \approx \sum_{i=1}^n \pi_i.\mathbb{P}(\omega_i)$, pourvu que cette propriété, ainsi que celle de bonne annotation soit préservées par réduction. Supposons que le préfixe

π_i, avec $i \leq n/2$, soit choisi et que nous commençons par réaliser l'émission $\overline{\alpha_i}[y]$. Le cas $i > n/2$ est similaire mais plus simple. T se réduit de manière unique comme suit.

$$\mathsf{T} \rightarrow^* (\boldsymbol{\nu}\gamma_0\mu_k)(\mathtt{Eval}(\gamma_0) \parallel \mathtt{Pev}(\mu_k, \gamma_0) \parallel \llcorner\omega_k\{x_k \leftarrow y\}, \mu_k\lrcorner) \mathrel{\widehat{=}} \mathbb{P}(\omega_k\{x_k \leftarrow y\})$$

Somme dynamique

Pour une somme dynamique, la situation est similaire et nous avons la séquence de réduction ci-dessous. Comme auparavant, le terme T que nous obtenons est observable exactement sur les canaux α_i, avec $1 \leq i \leq n$. A nouveau, $\mathbb{P}(\sum_{i=1}^n \pi_i.\omega_i) \approx \sum_{i=1}^n \pi_i.\mathbb{P}(\omega_i)$, pourvu que cette propriété, ainsi que celle de bonne annotation soit préservée par réduction.

$$\begin{array}{rl}
\mathbb{P}(\sum_{i=1}^n \pi_i.\omega_i) \approx & (\boldsymbol{\nu}\gamma_0\mu_1 \ldots \mu_n\eta_1 \ldots \eta_n) \\
& \big(\mathtt{Eval}(\gamma_0) \parallel \\
& (\boldsymbol{\nu}abcde) \ldots (\sum_{i=1}^{n/2} \delta_i(x_i).\overline{\eta_i}[x_i].\mathtt{Pev}(\mu_i, \gamma_i) \\
& \qquad + \sum_{i=n/2+1}^n \delta_i.\overline{\eta_i}.\mathtt{Pev}(\mu_i, \gamma_i)) \\
& \parallel (\sum_{i=1}^{n/2} \eta_i(x_i).\llcorner\omega_i, \mu_i\lrcorner + \sum_{i=n/2+1}^n \eta_i.\llcorner\omega_i, \mu_i\lrcorner)\big) \\
\approx & (\boldsymbol{\nu}\gamma_1 \ldots \gamma_n\delta_1 \ldots \delta_n\mu_1 \ldots \mu_n\eta_1 \ldots \eta_n) \\
& \big((\sum_{i=1}^{n/2} \alpha_i(x_i).\overline{\delta_i}[x_i].\mathtt{Eval}(\gamma_i) + \sum_{i=n/2+1}^n \overline{\alpha_i}[x_i].\overline{\delta_i}.\mathtt{Eval}(\gamma_i)) \\
& \parallel (\sum_{i=1}^{n/2} \delta_i(x_i).\overline{\eta_i}[x_i].\mathtt{Pev}(\mu_i, \gamma_i) + \sum_{i=n/2+1}^n \delta_i.\overline{\eta_i}.\mathtt{Pev}(\mu_i, \gamma_i)) \\
& \parallel (\sum_{i=1}^{n/2} \eta_i(x_i).\llcorner\omega_i, \mu_i\lrcorner + \sum_{i=n/2+1}^n \eta_i.\llcorner\omega_i, \mu_i\lrcorner)\big) \\
\mathrel{\widehat{=}} & \mathsf{T}
\end{array}$$

Comme la somme est dynamique, nous ne savons pas avec quel terme le processus T communique. Soit E un terme représentant le contexte dans lequel T est exécuté. T peut communiquer avec E sur un des canaux α_i qui est observable pour E. Supposons qu'une communication soit possible sur α_k, avec $1 \leq k \leq n/2$. Cela signifie que T reçoit une valeur y sur le canal α_k. Le cas $k > n/2$, c.à.d. lorsque T émet, est similaire mais plus simple. Nous avons la séquence de réduction suivante.

$$\begin{array}{rcl}
\mathsf{T} \parallel E & \rightarrow^* & (\boldsymbol{\nu}\gamma_k\mu_k)\big(\mathtt{Eval}(\gamma_k) \parallel \mathtt{Pev}(\mu_k, \gamma_k) \parallel \llcorner\omega_k\{x_k \leftarrow y\}, \mu_k\lrcorner\big) \parallel E' \\
& \mathrel{\widehat{=}} & \mathbb{P}(\omega_k\{x_k \leftarrow y\}) \parallel E'
\end{array}$$

Dans le même contexte E, le terme $\sum_{i=1}^n \pi_i.\mathbb{P}(\omega_i)$ se réduit immédiatement au terme

$$\sum_{i=1}^n \pi_i.\mathbb{P}(\omega_i) \parallel E \rightarrow \mathbb{P}(\omega_k\{x_k \leftarrow y\}) \parallel E'$$

Aussi, $\mathbb{P}(\sum_{i=1}^n \pi_i.\omega_i)$ et $\sum_{i=1}^n \pi_i.\mathbb{P}(\omega_i)$ demeurent $\approx$-congruents après réduction.

En résumé, nous avons montré que l'évaluation partielle préservait la $\approx$-congruence et que la bonne annotation des termes était préservée par réduction.

CQFD

Proposition 19 (Correction de l'évaluation partielle) *Pour tout $\boldsymbol{\pi}$-terme P, les propriétés suivante sont vérifiées.*

$$\begin{aligned} &\models_C P : \omega \Rightarrow \mathbb{P}(\omega) \;\approx\; P \\ &\models_O P : \omega \Rightarrow \mathbb{P}(\omega) \;\prec\; P \end{aligned} \tag{3.17}$$

□

PREUVE

La preuve est une induction sur la structure de P et est similaire à celle de la proposition 10. L'hypothèse $\models_C P \triangleright \omega$ permet l'utilisation du lemme 18. Nous utilisons la notation $\omega(P)$ pour représenter la version bien annotée du terme P.

– Si $P \widehat{=} \mathbf{0}$, alors P n'est observable sur aucun canal α.

$$\mathbb{P}(\mathbf{0}) \widehat{=} (\boldsymbol{\nu}\gamma_0)\big(\mathtt{Eval}(\gamma_0) \parallel (\boldsymbol{\nu}\mu_0)(\mathtt{Pev}(\mu_0,\gamma_0) \parallel (\boldsymbol{\nu}rstuvw)\overline{\mu_0}[rstuvw].\overline{r})\big)$$

  Clairement, $\mathbb{P}(\mathbf{0})\not\downarrow_\alpha$ pour tout α, puisque toutes les variables sont restreintes. Ainsi, $\mathbb{P}(\mathbf{0})$ se réduit à $\mathbf{0}$ sans être observable.

– Si $P \widehat{=} P_1 \parallel P_2$ alors $P{\downarrow_\alpha}$, si et seulement si $P_1{\downarrow_\alpha}$ ou $P_2{\downarrow_\alpha}$. Par ailleurs,

$$\mathbb{P}(P_1 \parallel P_2) \;\approx\; \mathbb{P}(P_1) \parallel \mathbb{P}(P_2)$$

  d'après le lemme 18. Par conséquent, $\mathbb{P}(P){\downarrow_\alpha}$, si et seulement si $\mathbb{P}(P_1){\downarrow_\alpha}$ ou $\mathbb{P}(P_2){\downarrow_\alpha}$.

– Si $P \widehat{=} !P_1$ alors $P{\downarrow_\alpha}$, si et seulement si $P_1{\downarrow_\alpha}$. D'après le lemme 18, $\mathbb{P}(!P) \;\approx\; !\mathbb{P}(P)$.

– Si $P \widehat{=} (\boldsymbol{\nu}x)P_1$ alors $P{\downarrow_\alpha}$, si et seulement si $P_1{\downarrow_\alpha}$ et $\alpha \neq x$. $\mathbb{P}((\boldsymbol{\nu}x)P_1) \;\approx\; (\boldsymbol{\nu}x)\mathbb{P}(P_1)$, d'après le lemme 18.

– Si $P \equiv \sum_{i=1}^{n} \pi_i.P_i$ alors P est observable exactement sur les les canaux des sujets des π_i et P se réduit à P_i, si et seulement si la communication π_i est exécutée. Par le lemme 18, $\mathbb{P}(\omega(P)) \;\approx\; \sum_{i=1}^{n} \pi_i.\mathbb{P}(\omega(P_i))$. Aussi, $\mathbb{P}(\omega(P))$ est observable exactement sur les sujets des π_i et se réduit à $\mathbb{P}(\omega(P_i))$, si et seulement π_i est exécuté.

CQFD

La proposition 19 s'applique à la notion de bonne annotation conservatrice des programmes. Pour la notion de bonne annotation optimisée, la discussion concernant les sommes statiques change. Du fait qu'une somme est statique dès que l'un de ses préfixes l'est, le terme $\mathbb{P}(\sum_{i=1}^{n} \pi_i.P_i)$ est moins déterministe que le terme $\sum_{i=1}^{n} \pi_i.\mathbb{P}(P_i)$. Aussi, nous avons $\sum_{i=1}^{n} \pi_i.\mathbb{P}(P_i) \prec \mathbb{P}(\sum_{i=1}^{n} \pi_i.P_i)$ au lieu de la $\approx$-congruence. Ce cas force toutes

les $\approx$-congruences du lemme 18 à devenir des simulations. La preuve de la proposition19 demeure la même, avec $\approx$ changé en $\prec$.

3.6 Conclusion

Dans ce chapitre, nous nous sommes intéressés à l'évaluation partielle du π-calcul, langage qui permet de modéliser des systèmes de processus mobiles. Nous avons utilisé une méthodologie en trois étapes qui consiste à écrire un méta-interpréteur, définir des annotations correctes pour les termes et à étendre le méta-interpréteur afin d'obtenir un évaluateur partiel. Nous avons prouvé la correction de l'évaluateur partiel en utilisant les notions de congruence barbelée faible et de simulation barbelée faible.

Nous pouvons remarquer que tous les opérateurs du π-calcul sont nécessaires pour obtenir la réflexivité du langage. Un méta-interpréteur ne peut pas être écrit pour un sous-ensemble strict du π-calcul. En effet, la composition parallèle est nécessaire pour représenter les différents processus, la réplication permet de coder la récursion et la restriction est utile pour garantir que les programmes interprétés seront observables strictement sur les mêmes canaux que les termes initiaux. Les sommes permettent de sélectionner parmi plusieurs alternatives.

Les notions de bonne annotation des π-termes à deux niveaux permettent de définir les cas dans lesquels une communication peut être exécutée lors de l'évaluation partielle ainsi que l'incidence de cette optimisation sur les programmes résiduels obtenus. Nous avons vu que pour qu'une communication soit statique, il est nécessaire qu'elle ne puisse pas être observable par un processus extérieur au terme que nous traitons. Pour cela, son sujet doit être un nom de canal défini à l'intérieur du terme à évaluer partiellement, qui de plus ne peut pas être transmis hors de ce terme durant l'exécution. Concernant les sommes, nous avons proposé deux manières de les annoter, reliées à deux critères de correction pour l'évaluateur partiel.

Afin de vérifier la bonne annotation d'un programme, il est nécessaire de disposer d'informations sur les lieux possibles d'utilisation d'un nom de canal. Ces informations sont fournies par une analyse de flot de contrôle, et de la précision de cette dernière dépend la qualité des annotations qu'il est possible de déduire. Nous présentons au chapitre 4 une telle analyse qui calcule une approximation de la topologie des communications réalisées par les programmes afin d'accroître sa précision.

L'évaluation partielle du π-calcul telle que nous l'avons présentée dans ce chapitre permet de mettre en évidence qu'il est possible d'écrire un interpréteur et un évaluateur partiel auto-applicable pour des langages parallèles utilisant des communications synchrones, et montre le principe de fonctionnement de ces derniers. Cependant, `Pev` ne

permet pas, par auto-application d'obtenir un compilateur et un générateur de compilateur, comme le prédisent les projections de Futamura. Ceci est du au fait qu'il n'est pas possible dans le π-calcul de réduire une communication qui n'est pas en tête du terme. Plus précisément, lorsque `Pev` rencontre une somme dynamique (terme `F` de la figure 3.6), il code cette somme à la manière de la fonction $\ulcorner . \urcorner$ et la transmet à `Eval` sur un canal γ_0. Comme expliqué à la section 3.5.2, il s'agit là d'un calcul étagé, l'interprète exécutant la somme et transmettant le résultat à `Pev` sur l'un des canaux δ_i, $1 \leq i \leq n$. Considérons par exemple la troisième projection de Futamura. L'évaluateur partiel le plus interne prend pour entrée un interpréteur codé sur un canal γ_0 et pour lequel nous souhaitons construire un compilateur. Le code de cet interpréteur n'étant pas connu au moment de l'auto-application, les communications sur γ_0 seront dynamiques, ce qui entrainera un blocage des versions les plus externes de `Pev` dans l'équation correspondant à cette projection. En revanche, la structure de l'évaluateur partiel que nous avons présentée peut être reprise pour écrire un évaluateur partiel pour un langage permettant ce type de réduction. Nous montrons, au chapitre 5, qu'il est possible d'obtenir, par auto-application, un générateur de compilateur pour un noyau non typé du langage Concurrent ML.

Chapitre 4

Analyse de flot de contrôle

4.1 Introduction

Dans ce chapitre, nous définissons une analyse de flot de contrôle pour Concurrent ML. Cette analyse calcule une approximation de la topologie des communications réalisées par les programmes (MG00a; MG00b; MG00d). Comme nous l'avons souligné au chapitre 2, la précision de la CFA influence sensiblement la façon dont les programmes sont annotés pour l'évaluation partielle. Par ailleurs, nous montrons à la section 4.5 des applications directes de cette analyse pour la vérification de programmes. Le calcul de la topologie des communications s'effectue de la manière suivante.

(i) Ordonner les points de synchronisation de chaque processus. La topologie des communications d'un programme distribué dépend de l'ordre dans lequel sont exécutées les instructions invoquant des synchronisations dans les différents processus. Cet ordre dépend du flot de contrôle sur les parties séquentielles du programme mais aussi des synchronisations précédentes. Dans un langage d'ordre supérieur, des fonctions contenant des communications peuvent être transmises et appliquées dans le processus qui les reçoit. Pour chaque processus p, nous construisons un automate d'états finis $\widehat{\mathcal{A}}_p$ qui indique l'ordre de succession des points de synchronisation dans p. A chaque sous-expression du programme correspond un automate et les transitions entre différents automates indiquent l'ordre d'évaluation des expressions associées. Les transitions qui correspondent à des pas de réduction séquentielle sont étiquetées par ε alors qu'une ℓ-transition indique l'occurrence d'un point de communication d'étiquette ℓ. Le résultat est une approximation de l'ordre des synchronisations réalisées par chaque processus.

(ii) Déterminer les interactions entre les processus. Les points de synchronisation de chaque processus étant ordonnés, il reste à déterminer une approximation de leurs in-

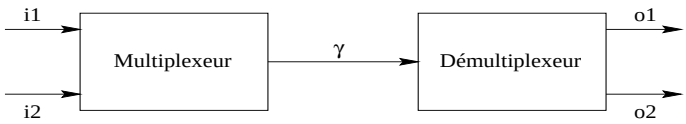
teractions. Une première manière d'aborder ce problème consiste à construire le produit des automates définis dans la phase précédente. A partir d'une collection $\widehat{\mathcal{G}} = (\widehat{\mathcal{A}}_p)_{p \in \text{Proc(PRG)}}$ d'automates d'états finis correspondant aux différents processus du programme PRG, nous montrons que l'automate produit $\widehat{\mathcal{A}}^S_\otimes(\widehat{\mathcal{G}})$ décrit une approximation correcte des communications que le programme peut réaliser. Cependant, d'une part la taille de $\widehat{\mathcal{A}}^S_\otimes(\widehat{\mathcal{G}})$ peut être exponentielle en la taille du programme, tandis que d'autre part, il renferme plus d'informations que véritablement nécessaire pour les besoins de l'analyse. Aussi, nous introduisons un *automate produit réduit* $\widehat{\mathcal{A}}^R_\otimes(\widehat{\mathcal{G}})$, de taille polynômiale en la taille du programme à analyser, et nous prouvons que $\widehat{\mathcal{A}}^R_\otimes(\widehat{\mathcal{G}})$ est une approximation correcte de $\widehat{\mathcal{A}}^S_\otimes(\widehat{\mathcal{G}})$. Nous déduisons de $\widehat{\mathcal{A}}^S_\otimes(\widehat{\mathcal{G}})$ un sur-ensemble des synchronisations possibles entre processus.

Nous examinons ci-dessous les approximations réalisées par notre analyse. La valeur abstraite attachée à un point de réception est l'union de toutes les valeurs abstraites envoyées par les émetteurs potentiels lors des différentes exécutions non déterministes du programme. Il s'agit là d'une approximation importante mais les valeurs abstraites collectées de cette manière correspondent toutes à une valeur concrète résultant d'une des exécutions possibles.

D'autres approximations sont réalisées pendant l'analyse des boucles. Les différents noms de canaux créés par la même instruction `channel()`à l'intérieur d'une boucle devront être identifiés. Ainsi, nous supposons que certaines communications sont possibles même si l'émetteur et le récepteur utilisent des noms différents, créés par deux instances de la même instruction.

Par ailleurs, un processus p peut communiquer avec les points se trouvant dans le corps d'une boucle b aussi longtemps que p suppose que la boucle n'a pas terminé son exécution. Cependant, grâce à l'utilisation des automates, l'analyse garde une trace de l'ordre des communications dans la boucle. Ainsi, les seules séquences de communications autorisées sont celles qui correspondent à un déroulement de b.

Par exemple, considérons le programme de la figure 4.1, décrivant un multiplexeur-démultiplexeur de canaux. Ce programme est composé de deux processus p_1 et p_2 écrits en Concurrent ML. Chaque instruction est annotée par une étiquette unique. p_1 reçoit des données sur les canaux ι_1 et ι_2 successivement. Ces données sont transmises dans l'ordre sur le canal γ. p_2 reçoit sur γ les données multiplexées par p_1 et les transmet sur o_1 et o_2. On suppose que le canal γ est partagé par p_1 et p_2 uniquement. Ainsi les données envoyées sur o_1 (resp. o_2) sont celles que le multiplexeur a reçu sur ι_1 (resp. ι_2). Enfin, nous définissons un processus p_3 qui a pour rôle d'envoyer des nouveaux noms de canaux au multiplexeur sur ι_1 et ι_2.



p_1 : rec^{1} multiplex $\iota_1\ \iota_2\ \gamma$ =
let^{2} c_1= send^{3} γ (receive^{4} ι_1)
and^{5} c_2= send^{6} γ (receive^{7} ι_2)
in (multiplex $\iota_1\ \iota_2\ \gamma$)^{8}

p_2 : rec^{9} demult $o_1\ o_2\ \gamma$ =
let^{10} c_1= send^{11} o_1 (receive^{12} γ)
and^{13} c_2= send^{14} o_2 (receive^{15} γ)
in (demult $o_1\ o_2\ \gamma$)^{16}

p_3 : rec^{17} emitter $\iota_1\ \iota_2$ =
let^{18} e_1= send^{19} ι_1 channel()^{20}
and^{21} e_2= send^{22} ι_2 channel()^{23}
in (emitter $\iota_1\ \iota_2$)^{24}

FIG. 4.1 – **Multiplexeur - Démultiplexeur écrits en Concurrent ML.**

Notre implantation de l'analyse décrite dans ce chapitre établit que la valeur abstraite émise au point 11 (resp. 14) est le singleton contenant l'étiquette 20 (resp. 23). Comme indiqué auparavant, les différents noms créés au même point ne sont pas distingués. Cependant, l'analyse détecte que les instances de la première réception sur γ (point 12) peuvent seulement recevoir une valeur émise par une instance de la première émission (point 3). Il en va de même pour la seconde réception (point 15) vis à vis de la seconde émission sur γ (point 6).

Dans la section 4.2, nous décrivons la manière dont les processus sont analysés séparément. La section 4.3 décrit la construction des automates produits et réduits et définit l'analyse d'un groupe de processus. La section 4.4 montre comment calculer effectivement une CFA pour un groupe de processus. Enfin, nous illustrons le fonctionnement de notre analyse à la section 4.5. Les preuves relatives aux propriétés énoncées dans ce chapitre sont données dans l'annexe A.

4.2 Analyse des expressions séquentielles

4.2.1 Spécification

Dans cette section, nous donnons une spécification de l'analyse des expressions séquentielles dont nous prouvons la correction par une preuve par réduction (NNH99), ce qui correspond à la première étape de la méthodologie décrite à la Section 4.1. Il s'agit d'analyser séparément les différents processus composant un programme distribué afin d'ordonner leurs points de synchronisation. On obtient ainsi une collection d'automates.

Etant donnés LAB l'ensemble des étiquettes apparaissant dans e^l et ID celui des va-

$$\widehat{\mathcal{C}},\widehat{\mathcal{E}},\widehat{\mathcal{A}} \vdash c^l \iff [\![\mathrm{B}_l \overset{\varepsilon}{\multimap} \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}}$$

$$\widehat{\mathcal{C}},\widehat{\mathcal{E}},\widehat{\mathcal{A}} \vdash x^l \iff \widehat{\mathcal{E}}(x) \subseteq \widehat{\mathcal{C}}(l),\ [\![\mathrm{B}_l \overset{\varepsilon}{\multimap} \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}}$$

$$\widehat{\mathcal{C}},\widehat{\mathcal{E}},\widehat{\mathcal{A}} \vdash (\texttt{fun}\ x^{l_1} \texttt{=>}\ e_0^{l_0})^l \iff \begin{cases} l \in \widehat{\mathcal{C}}(l),\ \widehat{\mathcal{C}},\widehat{\mathcal{E}}[x \mapsto \widehat{\mathcal{C}}(l_1)],\widehat{\mathcal{A}}_0 \vdash e_0^{l_0} \\ \widehat{\mathcal{A}}_0 \sqsubseteq \widehat{\mathcal{A}},\ [\![\mathrm{B}_l \overset{\varepsilon}{\multimap} \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}} \\ [\![\mathrm{B}_l \overset{\mu}{\multimap} \mathrm{B}_{l_0}]\!] \sqsubseteq \widehat{\mathcal{A}},\ [\![\mathrm{E}_{l_0} \overset{\mu}{\multimap} \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}} \end{cases}$$

$$\widehat{\mathcal{C}},\widehat{\mathcal{E}},\widehat{\mathcal{A}} \vdash (\texttt{rec}\ f^{l_1}\ x^{l_2} \texttt{=>}\ e_0^{l_0})^l \iff \begin{cases} l \in \widehat{\mathcal{C}}(l),\ \widehat{\mathcal{C}}(l) \subseteq \widehat{\mathcal{C}}(l_1) \\ \widehat{\mathcal{C}},\widehat{\mathcal{E}}[f \mapsto \widehat{\mathcal{C}}(l_1)][x \mapsto \widehat{\mathcal{C}}(l_2)],\widehat{\mathcal{A}}_0 \vdash e_0^{l_0} \\ \widehat{\mathcal{A}}_0 \sqsubseteq \widehat{\mathcal{A}},\ [\![\mathrm{B}_l \overset{\varepsilon}{\multimap} \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}} \\ [\![\mathrm{B}_l \overset{\mu}{\multimap} \mathrm{B}_{l_0}]\!] \sqsubseteq \widehat{\mathcal{A}},\ [\![\mathrm{E}_{l_0} \overset{\mu}{\multimap} \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}} \end{cases}$$

$$\widehat{\mathcal{C}},\widehat{\mathcal{E}},\widehat{\mathcal{A}} \vdash (e_0^{l_0}\ e_1^{l_1})^l \iff \begin{cases} \widehat{\mathcal{C}},\widehat{\mathcal{E}},\widehat{\mathcal{A}}_0 \vdash e_0^{l_0},\ \widehat{\mathcal{C}},\widehat{\mathcal{E}},\widehat{\mathcal{A}}_1 \vdash e_1^{l_1} \\ \widehat{\mathcal{A}}_0 \sqsubseteq \widehat{\mathcal{A}},\ \widehat{\mathcal{A}}_1 \sqsubseteq \widehat{\mathcal{A}},\ [\![\mathrm{B}_l \overset{\varepsilon}{\multimap} \mathrm{B}_{l_0}]\!] \sqsubseteq \widehat{\mathcal{A}},\ [\![\mathrm{E}_{l_0} \overset{\varepsilon}{\multimap} \mathrm{B}_{l_1}]\!] \sqsubseteq \widehat{\mathcal{A}} \\ \forall l_2 \in \widehat{\mathcal{C}}(l_0)\ :\ (\texttt{fun}\ x^{l_4} \texttt{=>}\ e_3^{l_3})^{l_2} \in \mathrm{prg}, \\ \quad \widehat{\mathcal{C}}(l_1) \subseteq \widehat{\mathcal{C}}(l_4),\ \widehat{\mathcal{C}}(l_3) \subseteq \widehat{\mathcal{C}}(l),\ [\![\mathrm{E}_{l_1} \overset{\varepsilon}{\multimap} \mathrm{B}_{l_3}]\!] \sqsubseteq \widehat{\mathcal{A}},\ [\![\mathrm{E}_{l_3} \overset{\varepsilon}{\multimap} \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}} \\ \forall l_2 \in \widehat{\mathcal{C}}(l_0)\ :\ (\texttt{rec}\ f^{l_4}\ x^{l_5} \texttt{=>}\ e_3^{l_3})^{l_2} \in \mathrm{prg}, \\ \quad \widehat{\mathcal{C}}(l_1) \subseteq \widehat{\mathcal{C}}(l_5),\ \widehat{\mathcal{C}}(l_3) \subseteq \widehat{\mathcal{C}}(l) \\ \quad [\![\mathrm{E}_{l_1} \overset{\varepsilon}{\multimap} \mathrm{B}_{l_3}]\!] \sqsubseteq \widehat{\mathcal{A}},\ [\![\mathrm{E}_{l_3} \overset{\varepsilon}{\multimap} \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}} \\ \quad [\![\mathrm{E}_{l_1} \overset{\varepsilon}{\multimap} \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}},\ [\![\mathrm{E}_{l_3} \overset{\varepsilon}{\multimap} \mathrm{B}_{l_3}]\!] \sqsubseteq \widehat{\mathcal{A}} \end{cases}$$

$$\widehat{\mathcal{C}},\widehat{\mathcal{E}},\widehat{\mathcal{A}} \vdash (\texttt{if}\ e_0^{l_0}\ e_1^{l_1}\ e_2^{l_2})^l \iff \begin{cases} \widehat{\mathcal{C}},\widehat{\mathcal{E}},\widehat{\mathcal{A}} \vdash e_i^{l_i},\ 0 \leq i \leq 2,\ \widehat{\mathcal{C}}(l_1) \subseteq \widehat{\mathcal{C}}(l),\ \widehat{\mathcal{C}}(l_2) \subseteq \widehat{\mathcal{C}}(l) \\ \widehat{\mathcal{A}}_0 \sqsubseteq \widehat{\mathcal{A}},\ \widehat{\mathcal{A}}_1 \sqsubseteq \widehat{\mathcal{A}},\ \widehat{\mathcal{A}}_2 \sqsubseteq \widehat{\mathcal{A}} \\ [\![\mathrm{B}_l \overset{\varepsilon}{\multimap} \mathrm{B}_{l_0}]\!] \sqsubseteq \widehat{\mathcal{A}},\ [\![\mathrm{E}_{l_0} \overset{\varepsilon}{\multimap} \mathrm{B}_{l_1}]\!] \sqsubseteq \widehat{\mathcal{A}} \\ [\![\mathrm{E}_l \overset{\varepsilon}{\multimap} \mathrm{B}_{l_2}]\!] \sqsubseteq \widehat{\mathcal{A}},\ [\![\mathrm{E}_{l_1} \overset{\varepsilon}{\multimap} \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}},\ [\![\mathrm{E}_{l_2} \overset{\varepsilon}{\multimap} \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}} \end{cases}$$

$$\widehat{\mathcal{C}},\widehat{\mathcal{E}},\widehat{\mathcal{A}} \vdash (\texttt{let}\ x^{l_2} = e_0^{l_0}\ \texttt{in}\ e_1^{l_1})^l \iff \begin{cases} \widehat{\mathcal{C}},\widehat{\mathcal{E}},\widehat{\mathcal{A}}_0 \vdash e_0^{l_0},\ \widehat{\mathcal{C}},\widehat{\mathcal{E}}[x \mapsto \widehat{\mathcal{C}}(l_2)],\widehat{\mathcal{A}}_1 \vdash e_1^{l_1} \\ \widehat{\mathcal{C}}(l_0) \subseteq \widehat{\mathcal{C}}(l_2),\ \widehat{\mathcal{C}}(l_1) \subseteq \widehat{\mathcal{C}}(l),\ \widehat{\mathcal{A}}_0 \sqsubseteq \widehat{\mathcal{A}},\ \widehat{\mathcal{A}}_1 \sqsubseteq \widehat{\mathcal{A}} \\ [\![\mathrm{B}_l \overset{\varepsilon}{\multimap} \mathrm{B}_{l_0}]\!] \sqsubseteq \widehat{\mathcal{A}},\ [\![\mathrm{E}_{l_0} \overset{\varepsilon}{\multimap} \mathrm{B}_{l_1}]\!] \sqsubseteq \widehat{\mathcal{A}},\ [\![\mathrm{E}_{l_1} \overset{\varepsilon}{\multimap} \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}} \end{cases}$$

FIG. 4.2 – **Spécification de l'analyse des expressions séquentielles - Partie 1.**

riables de la même expression, le résultat de l'analyse d'une expression e^l est un triplet $(\widehat{\mathcal{C}},\widehat{\mathcal{E}},\widehat{\mathcal{A}})$ dans lequel $\widehat{\mathcal{C}}$, $\widehat{\mathcal{E}}$ et $\widehat{\mathcal{A}}$ sont définis comme suit.

- $\widehat{\mathcal{C}}$: LAB $\to \wp$(LAB) est le *cache abstrait* qui associe à chaque sous-expression e^l du programme un ensemble $\widehat{\mathcal{C}}(l)$ de valeurs abstraites en lesquelles e^l peut s'évaluer. Ces valeurs abstraites sont soit des fonctions identifiées par leurs étiquettes, soit des noms de canaux identifiés par l'étiquette de l'instruction `channel()` qui les a créés. Le typage garantit qu'aucune confusion n'est faite entre ces deux sortes de valeurs et évite le recours à deux caches distincts.
- $\widehat{\mathcal{E}}$: ID $\to \wp$(LAB) est l'environnement abstrait qui associe une valeur abstraite à chaque variable libre pendant l'analyse.
- $\widehat{\mathcal{A}} = (\Sigma, \mathrm{B}, \mathbb{Q}, \mathbb{Q}_f, \delta)$ est un automate d'états finis qui indique comment les points

$$\widehat{\mathcal{C}},\widehat{\mathcal{E}},\widehat{\mathcal{A}} \vdash \texttt{channel()}^l \iff l \in \widehat{\mathcal{C}}(l),\ [\![\mathrm{B}_l \xrightarrow{\varepsilon} \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}}$$

$$\widehat{\mathcal{C}},\widehat{\mathcal{E}},\widehat{\mathcal{A}} \vdash k^l \iff l \in \widehat{\mathcal{C}}(l),\ [\![\mathrm{B}_l \xrightarrow{\varepsilon} \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}}$$

$$\widehat{\mathcal{C}},\widehat{\mathcal{E}},\widehat{\mathcal{A}} \vdash (\texttt{fork}\ e_0^{l_0})^l \iff \widehat{\mathcal{C}},\widehat{\mathcal{E}},\widehat{\mathcal{A}}_0 \vdash e_0^{l_0},\ [\![\mathrm{B}_l \xrightarrow{l} \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}},\ \widehat{\mathcal{A}}_0 \sqsubseteq \widehat{\mathcal{A}},\ [\![\mathrm{B}_l \xrightarrow{l} \mathrm{B}_{l_0}]\!] \sqsubseteq \widehat{\mathcal{A}}$$

$$\widehat{\mathcal{C}},\widehat{\mathcal{E}},\widehat{\mathcal{A}} \vdash (\texttt{send}\ e_0^{l_0}\ e_1^{l_1})^l \iff \begin{cases} \widehat{\mathcal{C}},\widehat{\mathcal{E}},\widehat{\mathcal{A}}_0 \vdash e_0^{l_0},\ \widehat{\mathcal{C}},\widehat{\mathcal{E}},\widehat{\mathcal{A}}_1 \vdash e_1^{l_1},\ \widehat{\mathcal{C}}(l_1) \subseteq \widehat{\mathcal{C}}(l) \\ \widehat{\mathcal{A}}_0 \sqsubseteq \widehat{\mathcal{A}},\ \widehat{\mathcal{A}}_1 \sqsubseteq \widehat{\mathcal{A}},\ [\![\mathrm{B}_l \xrightarrow{\varepsilon} \mathrm{B}_{l_0}]\!] \sqsubseteq \widehat{\mathcal{A}} \\ [\![\mathrm{E}_{l_0} \xrightarrow{\varepsilon} \mathrm{B}_{l_1}]\!] \sqsubseteq \widehat{\mathcal{A}},\ [\![\mathrm{E}_{l_1} \xrightarrow{l} \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}} \end{cases}$$

$$\widehat{\mathcal{C}},\widehat{\mathcal{E}},\widehat{\mathcal{A}} \vdash (\texttt{receive}\ e_0^{l_0})^l \iff \begin{cases} \widehat{\mathcal{C}},\widehat{\mathcal{E}},\widehat{\mathcal{A}}_0 \vdash e_0^{l_0} \\ \widehat{\mathcal{A}}_0 \sqsubseteq \widehat{\mathcal{A}},\ [\![\mathrm{B}_l \xrightarrow{\varepsilon} \mathrm{B}_{l_0}]\!] \sqsubseteq \widehat{\mathcal{A}},\ [\![\mathrm{E}_{l_0} \xrightarrow{l} \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}} \end{cases}$$

FIG. 4.3 – **Spécification de l'analyse des expressions séquentielles - Partie 2.**

de synchronisation sont ordonnés au sein de l'expression séquentielle que nous analysons. $\Sigma = \mathrm{Lab} \cup \{\varepsilon, \mu\}$ est l'alphabet. μ est un symbole particulier utilisé dans les automates décrivant des fonctions. $\mathbb{Q}$ est l'ensemble des états, $\mathrm{B} \in \mathbb{Q}$ est l'état initial, $\mathbb{Q}_f = \{\mathrm{E}\} \subseteq \mathbb{Q}$ contient un état final unique. $\delta \in (\mathbb{Q} \times \Sigma) \to \wp(\mathbb{Q})$ est la fonction de transition.

Dans les figures 4.2 et 4.3, nous définissons, inductivement sur la structure des termes, les contraintes qu'un triplet $(\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}})$ doit satisfaire afin de constituer une analyse correcte pour une expression e^l, ce qui est noté

$$\widehat{\mathcal{C}},\widehat{\mathcal{E}},\widehat{\mathcal{A}} \vdash e^l \tag{4.1}$$

Nous utilisons les notations suivantes. $[\![\mathrm{Q} \xrightarrow{\ell} \mathrm{Q}']\!]$ définit un automate dont l'état initial et l'état final sont respectivement Q et Q′ et tel que $\mathbb{Q} = \{\mathrm{Q}, \mathrm{Q}'\}$ et $\mathrm{Q}' \in \delta(\mathrm{Q}, \ell)$. $\widehat{\mathcal{A}} \sqsubseteq \widehat{\mathcal{A}}'$ indique que $\widehat{\mathcal{A}}$ est un sous-automate de $\widehat{\mathcal{A}}'$, c.à.d. que $\mathbb{Q} \subseteq \mathbb{Q}'$ et $\delta \subseteq \delta'$. Ainsi, $[\![\mathrm{Q} \xrightarrow{\ell} \mathrm{Q}']\!] \sqsubseteq \widehat{\mathcal{A}}$ indique qu'il y a une ℓ-transition entre Q et Q′ dans $\widehat{\mathcal{A}}$.

L'analyse construit, pour chaque expression e^l, un automate $\widehat{\mathcal{A}}$ dont l'état initial et l'état final sont respectivement $\textsc{Start}(\widehat{\mathcal{A}}) = \mathrm{B}_l$ et $\textsc{End}(\widehat{\mathcal{A}}) = \{\mathrm{E}_l\}$. Ces automates sont définis inductivement sur la structure des termes. Leur représentation graphique est donnée dans la figure 4.4.

Pour une expression séquentielle, notre analyse réalise une 0-CFA classique (NNH99) qui, de plus, construit un automate $\widehat{\mathcal{A}}$ indiquant comment les points de synchronisation sont ordonnés dans e^l. Les ε-transitions correspondent à des pas de réduction séquentielle.

Parce que l'évaluation d'une constante du premier ordre c^l n'engendre pas de synchronisation, l'automate associé à cette expression est composé d'une ε-transition allant

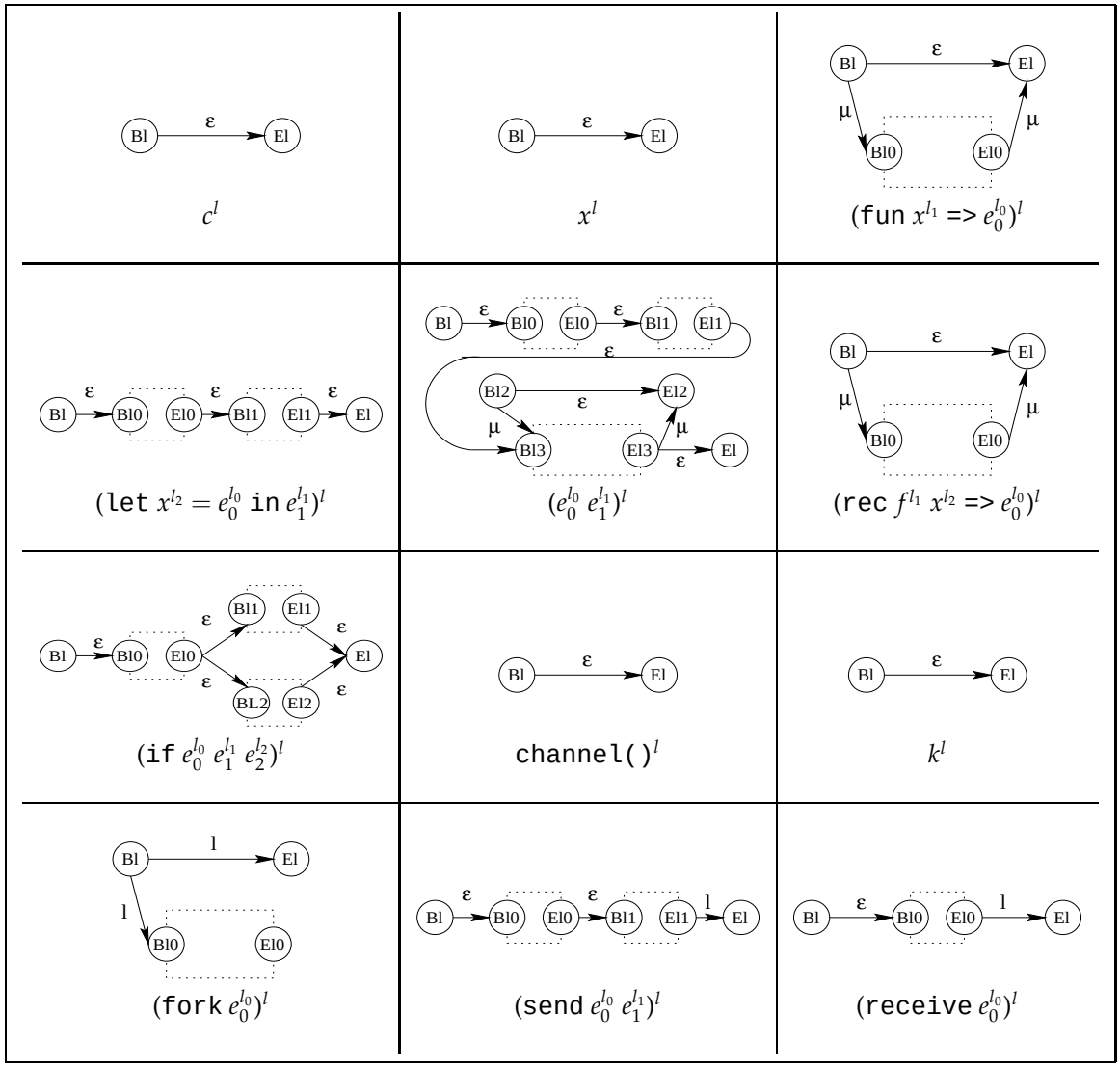


FIG. 4.4 – **Automates construits durant l'analyse.**

de l'état initial B_l à l'état final E_l, comme indiqué à la figure 4.4. Ceci est imposé par la contrainte $[\![\mathrm{B}_l \xrightarrow{\varepsilon} \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}}$ dans la spécification de la figure 4.2.

Pour une variable x^l, $\widehat{\mathcal{E}}(x)$ est inclus dans $\widehat{\mathcal{C}}(l)$, indiquant que la valeur abstraite du point l dépend de la valeur abstraite de x dans l'environnement. De plus, comme aucune réduction ne peut provenir de l'évaluation d'une variable, l'automate associé à x^l est constitué d'une ε-transition entre les états B_l et E_l. A nouveau, ceci est assuré par la contrainte $[\![\mathrm{B}_l \xrightarrow{\varepsilon} \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}}$.

(fun x^{l_1} => $e_0^{l_0}$)^l et (rec f x^{l_1} => $e_0^{l_0}$)^l sont des valeurs. Leur évaluation ne génère donc pas de calcul. Aussi, tout comme pour les constantes du premier ordre, l'état initial B_l de l'automate est relié à l'état final E_l par une ε-transition. De plus, le corps $e_0^{l_0}$ est analysé, ce qui donne un automate $\widehat{\mathcal{A}}_0$ avec B_{l_0} pour état initial et E_{l_0} pour état final. Comme décrit

dans la figure 4.4, nous relions $\widehat{\mathcal{A}}_0$ à $\widehat{\mathcal{A}}$ par des μ-transitions grâce aux contraintes $[\![\mathrm{B}_l \xrightarrow{\mu} \mathrm{B}_{l_0}]\!] \sqsubseteq \widehat{\mathcal{A}}$ et $[\![\mathrm{E}_{l_0} \xrightarrow{\mu} \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}}$. Par convention, les μ-transitions sont uniquement utilisées pour relier les deux parties de l'automate et ne définissent jamais un chemin valide entre des états. Enfin, l est inclus dans $\widehat{C}(l)$, afin d'indiquer que la fonction en question peut apparaître à ce point.

Analyser une application $(e_0^{l_0}\ e_1^{l_1})^l$ consiste à analyser $e_0^{l_0}$ et $e_1^{l_1}$ puis à calculer le résultat de l'application. L'analyse de $e_0^{l_0}$ et $e_1^{l_1}$ produit deux automates $\widehat{\mathcal{A}}_0$ et $\widehat{\mathcal{A}}_1$ qui indiquent comment les synchronisations sont ordonnées dans $e_0^{l_0}$ et $e_1^{l_1}$. A cause de l'ordre d'évaluation, les synchronisations réalisées par $e_0^{l_0}$ précèdent celles réalisées par $e_1^{l_1}$. Par ailleurs, si l'on suppose que $e_0^{l_0}$ s'évalue en la fonction $(\texttt{fun}\ x^{l_4}\ \texttt{=>}\ e_3^{l_3})^{l_2}$, les synchronisations de $e_1^{l_1}$ précèdent celles résultant de l'évaluation de $e_3^{l_3}$, avec la bonne valeur pour x. Aussi construisons nous l'automate suivant. B_l est relié à l'état initial de $\widehat{\mathcal{A}}_0$, et l'état final de $\widehat{\mathcal{A}}_0$ est relié à l'état initial de $\widehat{\mathcal{A}}_1$ par des ε-transitions. Ceci correspond aux contraintes $\widehat{\mathcal{A}}_0 \sqsubseteq \widehat{\mathcal{A}}$, $\widehat{\mathcal{A}}_1 \sqsubseteq \widehat{\mathcal{A}}$, $[\![\mathrm{B}_l \xrightarrow{\varepsilon} \mathrm{B}_{l_0}]\!] \sqsubseteq \widehat{\mathcal{A}}$ et $[\![\mathrm{E}_{l_0} \xrightarrow{\varepsilon} \mathrm{B}_{l_1}]\!] \sqsubseteq \widehat{\mathcal{A}}$.

Ensuite, $\widehat{C}(l_0)$ contient les étiquettes des fonctions en lesquelles $e_0^{l_0}$ peut s'évaluer. Pour chaque fonction $(\texttt{fun}\ x^{l_4}\ \texttt{=>}\ e_3^{l_3})^{l_2}$ du programme telle que $l_2 \in \widehat{C}(l_0)$, nous indiquons que les synchronisations dans le corps $e_3^{l_3}$ suivent celles de $e_1^{l_1}$ en imposant $[\![\mathrm{E}_{l_1} \xrightarrow{\varepsilon} \mathrm{B}_{l_3}]\!] \sqsubseteq \widehat{\mathcal{A}}$. Enfin, les états finaux de e^l et de $e_3^{l_3}$ sont reliés par $[\![\mathrm{E}_{l_3} \xrightarrow{\varepsilon} \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}}$. PRG est le programme que nous analysons. Dans la figure 4.4, nous montrons l'automate résultant de l'analyse d'une expression $(e_0^{l_0}\ e_1^{l_1})^l$, en supposant que $\widehat{C}(l_0) = \{l_2\}$ et que $(\texttt{fun}\ x^{l_4}\ \texttt{=>}\ e_3^{l_3})^{l_2} \in \mathrm{PRG}$.

Le cas de l'application d'une fonction récursive avec un argument est similaire. Comme le corps de la fonction peut être exécuté zéro ou plusieurs fois, les contraintes $[\![\mathrm{B}_l \xrightarrow{\varepsilon} \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}}$ et $[\![\mathrm{E}_{l_4} \xrightarrow{\varepsilon} \mathrm{B}_{l_4}]\!] \sqsubseteq \widehat{\mathcal{A}}$ sont ajoutées, où l_4 est l'étiquette du corps de la fonction récursive.

Les canaux sont identifiés à leur point de création. `channel()` étant un appel de fonction, pour une occurrence de $\texttt{channel()}^l$ dans le programme, nous ajoutons l à $\widehat{C}(l)$. Ainsi, aucune distinction n'est faite entre différents noms de canaux générés en un point d'une fonction récursive.

Nous considérons qu'une instruction `fork` induit une synchronisation entre les processus père et fils. Cela nous permet de traiter les instructions de création de processus de la même manière que les communications. Pour une expression $(\texttt{fork}\ e_0^{l_0})^l$, $e_0^{l_0}$ est analysé, donnant un automate $\widehat{\mathcal{A}}_0$. D'une part $[\![\mathrm{B}_l \xrightarrow{l} \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}}$ indique que l'exécution du `fork` consiste à faire une synchronisation nommée l. D'autre part, l'automate $\widehat{\mathcal{A}}_0$ décrivant les synchronisations réalisées par $e_0^{l_0}$ est inclus dans $\widehat{\mathcal{A}}$. Il est relié à l'état initial B_l par $[\![\mathrm{B}_l \xrightarrow{l} \mathrm{B}_{l_0}]\!] \sqsubseteq \widehat{\mathcal{A}}$. Dans la section 4.3.2, lorsque nous nous intéresserons à l'automate produit, $e_0^{l_0}$ reste gelé jusqu'à qu'il puisse se synchroniser sur l.

Lorsqu'une émission $(\texttt{send}\ e_0^{l_0}\ e_1^{l_1})^l$ est rencontrée, $e_0^{l_0}$ et $e_1^{l_1}$ sont analysés. $[\![\mathrm{E}_{l_0} \overset{\varepsilon}{\multimap} \mathrm{E}_{l_1}]\!] \sqsubseteq \widehat{\mathcal{A}}$ indique que l'exécution de $e_0^{l_0}$ précède celle de $e_1^{l_1}$ et $[\![\mathrm{E}_{l_1} \overset{l}{\multimap} \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}}$ précise que l est le dernier point de synchronisation dans $(\texttt{send}\ e_0^{l_0}\ e_1^{l_1})^l$.

Les réceptions de la forme $(\texttt{receive}\ e_0^{l_0})^l$ sont traitées similairement. $[\![\mathrm{E}_{l_0} \overset{l}{\multimap} \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}}$ indique que les synchronisations dans $e_0^{l_0}$ précèdent celle correspondant à la réception. Notons qu'à ce stade de l'analyse, aucune contrainte n'est introduite pour spécifier la valeur abstraite reçue. Ceci provient du fait que, à ce stade de la CFA, nous analysons les différents processus séquentiels sans tenir compte de leurs interactions. Cependant, dans une exécution parallèle, une réception $(\texttt{receive}\ e_0^{l_0})^l$ peut recevoir une valeur du premier ordre, une fonction ou encore un nom de canal, selon son type. Dans les sections 4.3.1 et 4.3.2, lorsque nous nous intéresserons aux interactions entre processus, nous ajouterons des contraintes indiquant la valeur associée à $\widehat{\mathcal{C}}(l)$. Ces contraintes sont spécifiées dans les définitions 28 et 33.

4.2.2 Correction

Nous présentons ici des résultats concernant la correction de l'analyse définie dans les figures 4.2 et 4.3. Les preuves relatives aux propriétés énoncées ci-dessous sont présentées dans l'annexe A. Dans un premier temps, nous prouvons l'existence d'une meilleure analyse pour une expression e^l. En effet, il existe en général plusieurs analyses correctes pour une même expression, mais dont la précision varie (par exemple, l'analyse associant, à chaque expression de type fonction, l'ensemble de toutes les fonctions du programme est toujours correcte mais très imprécise). Il est possible de classer ces analyses en utilisant une relation d'ordre que nous donnons, et nous montrons qu'il existe toujours un plus petit élément dans l'ensemble des analyses correctes.

Ensuite, nous nous consacrons à la correction de la CFA. Pour cela, nous montrons, par une propriété de réduction du sujet, qu'une analyse correcte pour une expression e^l, le demeure pour l'expression e'^l obtenue en un pas de réduction à partir de e^l. Ces résultats sont utilisés dans la suite de ce chapitre pour prouver la correction de l'analyse d'un groupe de processus.

Définition 20 *Soient* $(\widehat{\mathcal{C}}_1, \widehat{\mathcal{E}}_1, \widehat{\mathcal{A}}_1)$ *et* $(\widehat{\mathcal{C}}_2, \widehat{\mathcal{E}}_2, \widehat{\mathcal{A}}_2)$ *deux analyses pour une expression* e^l*. On dit que* $(\widehat{\mathcal{C}}_1, \widehat{\mathcal{E}}_1, \widehat{\mathcal{A}}_1)$ *est plus précise que* $(\widehat{\mathcal{C}}_2, \widehat{\mathcal{E}}_2, \widehat{\mathcal{A}}_2)$ *et on note* $(\widehat{\mathcal{C}}_1, \widehat{\mathcal{E}}_1, \widehat{\mathcal{A}}_1) \prec (\widehat{\mathcal{C}}_2, \widehat{\mathcal{E}}_2, \widehat{\mathcal{A}}_2)$ *si et seulement si*

$$\begin{cases} \forall l \in Lab,\ \widehat{\mathcal{C}}_1(l) \subseteq \widehat{\mathcal{C}}_2(l) \\ \forall x \in Id,\ \widehat{\mathcal{E}}_1(x) \subseteq \widehat{\mathcal{E}}_2(x) \\ \widehat{\mathcal{A}}_1 \sqsubseteq \widehat{\mathcal{A}}_2 \end{cases} \tag{4.2}$$

□

L'existence d'une plus petite analyse, au sens de $\prec$, provient du fait que l'ensemble des analyses correctes pour une expression est une *famille de Moore* (NNH99), c.à.d. un sous-ensemble X d'un treillis complet tel que toute partie Y de X possède un plus petit élément.

Définition 21 (Famille de Moore) *Un sous-ensemble X d'un treillis complet $(T, \leq)$ est une famille de Moore si et seulement si pour tout ensemble $Y \subseteq X$, $(\sqcap Y) \in X$.* □

Notons que la famille de Moore X n'est jamais vide et admet un plus petit élément puisque $(\sqcap X) \in X$. Ainsi, si l'ensemble des analyses correctes est une famille de Moore, nous sommes assurés de l'existence d'une meilleure analyse.

Proposition 22 *Soit e^l une expression étiquetée. L'ensemble $\{(\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}) : \widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}} \vdash e^l\}$ est une famille de Moore.* □

La principale propriété concernant la correction de l'analyse des expressions séquentielles établit que si un triplet $(\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}})$ est une analyse correcte pour une expression e^l, alors elle le demeure après un pas de réduction séquentielle $\hookrightarrow$ (Section 2.1.2). Tout d'abord, nous présentons le lemme suivant utilisé dans la preuve de la Proposition 26. $\mathbb{SG}(\widehat{\mathcal{A}}, x)$ décrit le sous-automate de $\widehat{\mathcal{A}}$ correspondant aux occurrences x^l de la variable x, pour toute étiquette l. Ces sous-expressions sont aisément identifiables dans $\widehat{\mathcal{A}}$ puisqu'elles correspondent aux sous-automates de la forme $[\![\mathrm{B}_l \xrightarrow{\varepsilon} \mathrm{E}_l]\!]$, où l est l'étiquette d'une occurrence de x dans l'expression. $\widehat{\mathcal{A}}\{\mathbb{SG}(\widehat{\mathcal{A}}, x) \leftarrow \widehat{\mathcal{A}}_0\}$ définit la substitution dans $\widehat{\mathcal{A}}$ des sous-automates correspondant aux occurrences de la variable x par $\widehat{\mathcal{A}}_0$.

Lemme 23 (Substitution) *Si $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}[x \mapsto \widehat{\mathcal{C}}(l_1)], \widehat{\mathcal{A}} \vdash e^l$ et $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_0 \vdash v^{l_0}$ et $\widehat{\mathcal{C}}(l_0) \subseteq \widehat{\mathcal{C}}(l_1)$ alors on a $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}\{\mathbb{SG}(\widehat{\mathcal{A}}, x) \leftarrow \widehat{\mathcal{A}}_0\} \vdash e^l\{x \leftarrow v^{l_0}\}$.* □

Le Lemme 23 indique que l'analyse d'une expression e^l reste correcte après substitution d'une variable x de e^l par une valeur v^{l_0}. Par hypothèse, on suppose que $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_0 \vdash v^{l_0}$ et $\widehat{\mathcal{C}}(l_0) \subseteq \widehat{\mathcal{C}}(l_1)$.

De plus, nous utilisons le lemme suivant qui indique que la réduction préserve la précision des annotations d'une expression e^l.

Lemme 24 (Monotonie) *Si $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}} \vdash e^l$ et $e^l \hookrightarrow e'^{l'}$ alors $\widehat{\mathcal{C}}(l') \subseteq \widehat{\mathcal{C}}(l)$.* □

La propriété principale concernant les pas de réduction séquentiels indique que l'analyse d'une expression reste valide par réduction (*subject reduction*). Elle dépend des lemmes 23 et 24. Avant de l'énoncer, nous définissons la relation d'ordre $\preccurlyeq$ sur l'ensemble des automates.

Définition 25 *Soient $\widehat{\mathcal{A}}$ et $\widehat{\mathcal{A}}'$ deux automates. $\widehat{\mathcal{A}} \lessdot \widehat{\mathcal{A}}'$ si et seulement si tout chemin étiqueté $\ell_1 \dots \ell_n$ dans $\widehat{\mathcal{A}}$ est un chemin dans $\widehat{\mathcal{A}}'$.* □

Intuitivement, les étiquettes des transitions d'un automate $\widehat{\mathcal{A}}$ construit pendant l'analyse d'une expression e^l correspondent aux synchronisations réalisées durant l'exécution de e^l. Ainsi, un chemin dans $\widehat{\mathcal{A}}$ décrit une séquence de synchronisations possible, résultant d'une exécution de e^l.

Aucune synchronisation n'est réalisée lors d'un pas de réduction séquentiel mais certaines séquences de synchronisations peuvent être éliminées. C'est le cas, par exemple, lors de l'exécution d'une instruction conditionnelle. Aussi, si $e^l \hookrightarrow e'^{l'}$, toute séquence de synchronisations possible pour $e'^{l'}$ est une séquence possible pour e^l. Si l'on considère les automates $\widehat{\mathcal{A}}$ et $\widehat{\mathcal{A}}'$ construits durant les analyses de e^l et de $e'^{l'}$, nous avons $\widehat{\mathcal{A}}' \lessdot \widehat{\mathcal{A}}$. Ceci est résumé par la proposition suivante.

Proposition 26 (Réduction du sujet) *Si $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}} \vdash e^l$ et $e^l \hookrightarrow e'^{l'}$ alors $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}' \vdash e'^{l'}$ pour un certain $\widehat{\mathcal{A}}'$ tel que $\widehat{\mathcal{A}}' \lessdot \widehat{\mathcal{A}}$.* □

Analyser séparément les processus d'un programme nous permet d'ordonner les points de synchronisation dans chacun des composants de celui-ci. Dans la suite de ce chapitre, nous nous intéressons au produit des automates définis ci-dessus, dans le but de déterminer une approximation de la topologie des communications.

4.3 Analyse de groupes de processus

Dans cette section, nous nous intéressons à l'analyse d'un groupe de processus. La section 4.3.1 définit l'automate produit $\mathrm{A}^S_\otimes(\widehat{\mathcal{G}})$ correspondant à une collection $\widehat{\mathcal{G}}$ d'automates construits par l'analyse de la section 4.2.1. Nous présentons aussi une analyse fondée sur $\mathrm{A}^S_\otimes(\widehat{\mathcal{G}})$ pour un groupe de processus. Cependant, $\mathrm{A}^S_\otimes(\widehat{\mathcal{G}})$ peut avoir un nombre d'états exponentiel en la taille du programme à analyser. Dans la section 4.3.2, nous définissons un automate produit réduit, $\mathrm{A}^R_\otimes(\widehat{\mathcal{G}})$ dont la taille est polynômiale en celle du programme d'entrée et nous prouvons que l'on obtient une analyse correcte en substituant $\mathrm{A}^R_\otimes(\widehat{\mathcal{G}})$ à $\mathrm{A}^S_\otimes(\widehat{\mathcal{G}})$. Les preuves relatives aux propriétés énoncées dans cette section sont données dans l'annexe A.

4.3.1 Analyse utilisant l'automate produit

Soit $\widehat{\mathcal{G}} = (\widehat{\mathcal{A}}_p)_{p \in \mathrm{Dom}(P)}$ une collection d'automates. Les états de l'automate produit $\mathrm{A}^S_\otimes(\widehat{\mathcal{G}})$ sont les produits des états des automates de $\widehat{\mathcal{G}}$. Les ε-transitions d'un automate $\widehat{\mathcal{A}}_p$

correspondent à des pas de réduction séquentiels et sont conservés dans $\mathrm{A}^S_\otimes(\widehat{\mathcal{G}})$. Une l-transition dans $\widehat{\mathcal{A}}_p$ décrit un point de synchronisation. Nous ajoutons des (l, l')-transitions dans $\mathrm{A}^S_\otimes(\widehat{\mathcal{G}})$ si l et l' sont des transitions dans deux automates de $\widehat{\mathcal{G}}$ telles que les instructions étiquetées l et l' peuvent interagir ensemble.

Définition 27 (Automate produit) *Soit $\widehat{\mathcal{G}}$ une collection de n automates. L'automate produit $\mathrm{A}^S_\otimes(\widehat{\mathcal{G}})$ des automates de $\widehat{\mathcal{G}}$ est un quadruplet $(\Sigma^S, \mathrm{Q}^S_0, \mathbb{Q}^S, \delta^S)$. L'alphabet est $\Sigma^S \subseteq Lab^2 \cup \{\varepsilon, \mu\}$. $\mathbb{Q}^S$ est composé de k-uplets $(\mathrm{Q}_1, \ldots, \mathrm{Q}_k)$ dans lesquels Q_i, $1 \leq i \leq k$, décrit l'avancement du i^{ieme} automate. Q^S_0 est l'état initial et $\delta^S \in (\mathbb{Q}^S \times \Sigma^S) \to \wp(\mathbb{Q}^S)$ est la fonction de transition. $\mathrm{A}^S_\otimes(\widehat{\mathcal{G}})$ est construit de la manière suivante.*

(i) L'état initial $\mathrm{Q}^S_0 \in \mathbb{Q}^S$ est le produit des états initiaux $\textsc{Start}(\widehat{\mathcal{A}}_p)$ des automates $\widehat{\mathcal{A}}_p \in \widehat{\mathcal{G}}$, c.à.d.

$$\mathrm{Q}^S_0 \widehat{=} \prod_{\widehat{\mathcal{A}}_p \in \widehat{\mathcal{G}}} \textsc{Start}(\widehat{\mathcal{A}}_p) \tag{4.3}$$

(ii) Pour tout $\mathrm{Q}^S \in \mathbb{Q}^S$ tel que $\mathrm{Q}^S = (\mathrm{Q}_1, \ldots, \mathrm{Q}_k)$,

(a) Pour tout i, $1 \leq i \leq k$, et $\ell \in \{\varepsilon, \mu\}$,

$$\left(\exists \widehat{\mathcal{A}}_p \in \widehat{\mathcal{G}} \ : \ [\![\mathrm{Q}_i \overset{\ell}{\multimap} \mathrm{Q}'_i]\!] \sqsubseteq \widehat{\mathcal{A}}_p\right) \ \Rightarrow \ [\![\mathrm{Q}^S \overset{\ell}{\multimap} (\mathrm{Q}_1, \ldots, \mathrm{Q}'_i, \ldots, \mathrm{Q}_k)]\!] \sqsubseteq \mathrm{A}^S_\otimes(\widehat{\mathcal{G}}) \tag{4.4}$$

(b) Pour tout i et j, $1 \leq i, j \leq k$, $i \neq j$,

$$\left(\exists \widehat{\mathcal{A}}_p, \widehat{\mathcal{A}}_{p'} \in \widehat{\mathcal{G}} \ : \ \left| \begin{array}{l} [\![\mathrm{Q}_i \overset{l_s}{\multimap} \mathrm{Q}_{l_s}]\!] \sqsubseteq \widehat{\mathcal{A}}_p \\ [\![\mathrm{Q}_j \overset{l_r}{\multimap} \mathrm{Q}_{l_r}]\!] \sqsubseteq \widehat{\mathcal{A}}_{p'} \\ (\texttt{send}\ e^{l_0}_0\ e^{l_1}_1)^{l_s} \in \mathrm{PRG} \\ (\texttt{receive}\ e^{l_2}_2)^{l_r} \in \mathrm{PRG} \end{array} \right.\right) \tag{4.5}$$

$$\Longrightarrow [\![\mathrm{Q}^S \overset{l_s, l_r}{\multimap} (\mathrm{Q}_1, \ldots, \mathrm{Q}_{l_s}, \ldots, \mathrm{Q}_{l_r}, \ldots, \mathrm{Q}_k)]\!] \sqsubseteq \mathrm{A}^S_\otimes(\widehat{\mathcal{G}})$$

(c) Pour tout i, $1 \leq i \leq k$,

$$\left(\exists \widehat{\mathcal{A}}_p \in \widehat{\mathcal{G}} \ : \ \left| \begin{array}{l} [\![\mathrm{Q}_i \overset{l_f}{\multimap} \mathrm{Q}_{l_f}]\!] \sqsubseteq \widehat{\mathcal{A}}_p \\ [\![\mathrm{Q}_i \overset{l_f}{\multimap} \mathrm{Q}_{l_0}]\!] \sqsubseteq \widehat{\mathcal{A}}_p \\ (\texttt{fork}\ e^{l_0}_0)^{l_f} \in \mathrm{PRG} \end{array} \right.\right) \tag{4.6}$$

$$\Longrightarrow [\![\mathrm{Q}^S \overset{l_f, l_f}{\multimap} (\mathrm{Q}_1, \ldots, \mathrm{Q}_{l_f}, \ldots, \mathrm{Q}_k, \mathrm{Q}_{l_0})]\!] \sqsubseteq \mathrm{A}^S_\otimes(\widehat{\mathcal{G}})$$

□

L'automate produit est construit incrémentalement comme indiqué ci-après.

(i) Nous commençons avec un unique état Q_0^S. Tout état $\mathrm{Q}^S = (\mathrm{Q}_1, \ldots, \mathrm{Q}_k)$ dans $\mathrm{A}_\otimes^S(\widehat{\mathcal{G}})$ décrit un des avancements possibles de l'exécution du groupe de processus. Nous ajoutons une ε-transition partant de Q^S pour chaque ε-transition présente dans l'un des automates $\widehat{\mathcal{A}}_p$ de $\widehat{\mathcal{G}}$ et partant de Q_i, $1 \leq i \leq k$. On obtient ainsi un nouvel état qui décrit aussi un avancement possible de l'exécution du groupe.

(ii) Une l-transition correspond à un point de synchronisation. Deux processus peuvent communiquer entre eux s'ils possèdent des points de synchronisation compatibles l_s et l_r correspondant à une émission et une réception éventuellement actives au même moment. Dans ce cas, il y a un état $\mathrm{Q}^S = (\mathrm{Q}_1, \ldots, \mathrm{Q}_k)$ et deux automates $\widehat{\mathcal{A}}_p$ et $\widehat{\mathcal{A}}_{p'}$ dans $\widehat{\mathcal{G}}$ tels que $[\![\mathrm{Q}_s \overset{l_s}{\multimap} \mathrm{Q}'_s]\!] \sqsubseteq \widehat{\mathcal{A}}_p$ et $[\![\mathrm{Q}_r \overset{l_r}{\multimap} \mathrm{Q}'_r]\!] \sqsubseteq \widehat{\mathcal{A}}_{p'}$, $1 \leq s, r \leq k$, $s \neq r$. Nous ajoutons alors la transition $\mathrm{Q}^S \overset{l_s, l_r}{\multimap} \mathrm{Q}^{S'}$ à l'automate produit où $\mathrm{Q}^{S'}$ est le nouvel état obtenu en substituant Q'_s et Q'_r à Q_s et Q_r dans Q^S.
Remarquons que l'automate produit autorise une communication entre n'importe quelle paire (s, r) d'émetteur et de récepteur qui peuvent être actifs au même moment, indépendamment du canal utilisé. Dans la définition 28, la valeur abstraite émise par s est ajoutée à la valeur abstraite attachée à r si et seulement si s et r communiquent peut-être sur le même canal. Une plus grande précision serait obtenue en interdisant ces communications au moment de la construction de $\mathrm{A}_\otimes^S(\widehat{\mathcal{G}})$. Cependant, l'automate produit ne serait plus défini indépendemment de l'analyse, ce qui surchargerait les notations.

(iii) Un `fork` d'étiquette l_f crée un nouveau processus. Pour toute transition $[\![\mathrm{Q}_i \overset{l_f}{\multimap} \mathrm{Q}_{l_f}]\!] \sqsubseteq \widehat{\mathcal{A}}_p$ telle qu'il existe un état $\mathrm{Q}^S = (\mathrm{Q}_1, \ldots, \mathrm{Q}_i, \ldots, \mathrm{Q}_k)$ dans $\mathbb{Q}^S$, l'état $(\mathrm{Q}_1, \ldots, \mathrm{Q}_{l_f}, \ldots, \mathrm{Q}_k, \mathrm{Q}_{l_0})$ est ajouté à $\mathbb{Q}^S$. Cet état décrit l'avancement du programme après création du nouveau processus. Il contient une composante supplémentaire Q_{l_0} correspondant à l'avancement du nouveau processus. Concernant le processus père, l'état correspondant à son avancement est mis à jour afin d'indiquer que la création a eu lieu.

Dans la suite de cette section, nous donnons les conditions qu'un triplet $(\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_\otimes^S)$ doit satisfaire pour analyser correctement un groupe de processus. $\widehat{\mathcal{C}}$ et $\widehat{\mathcal{E}}$ sont définis de la même manière qu'à la section 4.2.1 et $\widehat{\mathcal{A}}_\otimes^S$ est le produit des automates $\widehat{\mathcal{A}}_p$ construits durant l'analyse des processus séquentiels de P. De plus, nous définissons une nouvelle relation d'ordre étendant $\lessdot$ afin d'exprimer comment les automates produit analysant P sont reliés lorsque ce dernier est réduit.

La notation $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_\otimes^S \models^S P$ signifie que le triplet $(\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_\otimes^S)$ est une analyse correcte pour le groupe de processus P, $\models^S$ faisant l'objet de la définition 28. Intuitivement, nous

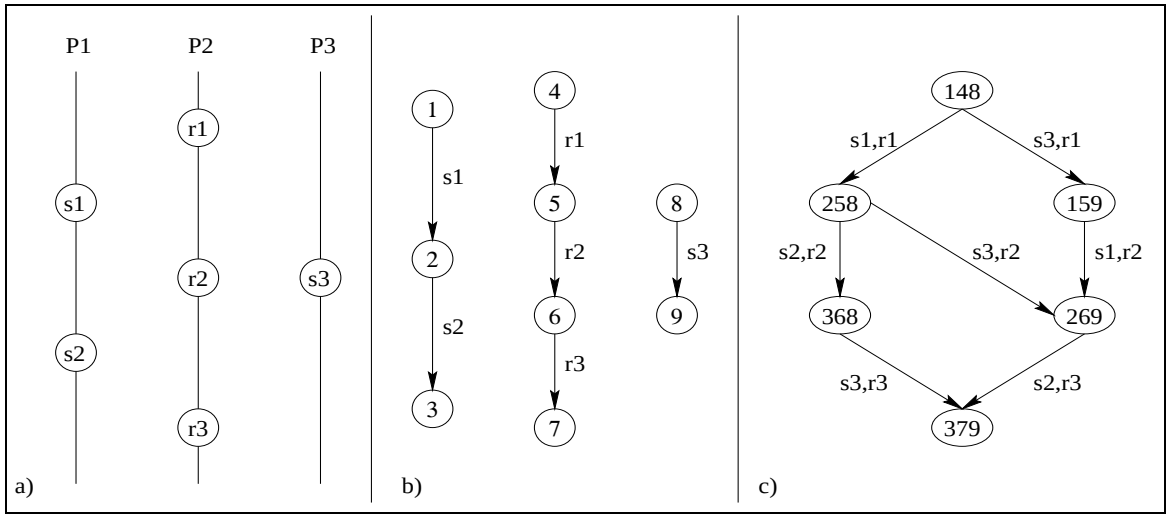


FIG. 4.5 – **a) Un programme composé de trois processus P_1, P_2 et P_3. s_1, s_2 et s_3 sont des émissions et r_1, r_2 et r_3 sont des réceptions. b) Collection d'automates correspondant au programme précédent. c) Automate produit relatif à ce programme.**

demandons que toutes les expressions du groupe soient correctement analysées en utilisant les automates de la collection $\widehat{\mathcal{G}}$, et que, pour toute communication autorisée par l'automate produit correspondant à $\widehat{\mathcal{G}}$, la valeur abstraite émise soit ajoutée à la valeur du point de réception.

Définition 28 *On dit qu'un triplet $(\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}^S_\otimes)$ est une analyse correcte pour un groupe de processus P, et on note $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}^S_\otimes \models^S P$ si et seulement si*

(i) $\widehat{\mathcal{A}}^S_\otimes = \mathrm{A}^S_\otimes(\widehat{\mathcal{G}})$ *pour une collection* $\widehat{\mathcal{G}}$ *d'automates tels que* $\widehat{\mathcal{G}} = (\widehat{\mathcal{A}}_p)_{p \in Dom(P)}$ *et* $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_p \vdash e^l$ *pour tout* $\langle p : e^l \rangle \in P$.

(ii) Pour tout $[\![q \overset{l_s, l_r}{\multimap} q']\!] \sqsubseteq \widehat{\mathcal{A}}^S_\otimes$ *t.q.* $\begin{cases} (\texttt{send}\ e_0^{l_0}\ e_1^{l_1})^{l'_i} \in \mathrm{PRG} \\ (\texttt{receive}\ e_2^{l_2})^{l'_j} \in \mathrm{PRG} \end{cases}$, $\widehat{\mathcal{C}}(l_0) \cap \widehat{\mathcal{C}}(l_2) \neq \varnothing$ *implique* $\widehat{\mathcal{C}}(l_1) \subseteq \widehat{\mathcal{C}}(l_r)$. □

Comme indiqué à la section 4.2.1, le cache $\widehat{\mathcal{C}}$ contient des valeurs abstraites représentant indifféremment des fonctions ou des noms de canaux. Dans la définition 28, $e_0^{l_0}$ et $e_2^{l_2}$ sont de type canal. Par conséquence, la condition $\widehat{\mathcal{C}}(l_0) \cap \widehat{\mathcal{C}}(l_2) \neq \varnothing$ fait référence à des valeurs abstraites correspondant à des noms de canaux afin de déterminer si l'émetteur et le récepteur peuvent communiquer ensemble. Lorsque cette condition est vérifiée, on ajoute la contrainte $\widehat{\mathcal{C}}(l_1) \subseteq \widehat{\mathcal{C}}(l_r)$ qui fait référence à des valeurs abstraites relatives à des noms canaux ou des fonctions, selon le type de $e_1^{l_1}$.

Comme à la section 4.2.1, l'existence d'une plus petite analyse pour un groupe de processus P est due au fait que les analyses correctes pour P sont ordonnées par la relation $\prec$ donnée à l'équation (4.2) et définissent une famille de Moore.

Proposition 29 (Famille de Moore) *Soit P un groupe de processus. L'ensemble $\{(\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}^S_\otimes) : \widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}^S_\otimes \models^S P\}$ des analyses correctes pour P est une famille de Moore.* □

Dans le but de prouver que la correction d'une analyse pour un groupe de processus est préservée par réduction, nous utilisons une nouvelle relation $\overset{\ell}{\lessdot}$ étendant $\lessdot$, où ℓ est une étiquette du programme. Un chemin dans $\widehat{\mathcal{A}}^S_\otimes$ décrit une séquence de synchronisations correspondant à une exécution possible du programme. $\overset{\ell}{\lessdot}$ nous permet d'affirmer que si $K, P \xrightarrow{[\ell]} K', P'$ alors l'automate produit $\widehat{\mathcal{A}}^{S'}_\otimes$ relatif à P' contient tous les chemins de $\widehat{\mathcal{A}}^S_\otimes$ partant de $\mathrm{Q}^{S'}$, où $\mathrm{Q}^{S'}$ est un état accessible à partir de Q^S_0 par une ℓ-transition. Ainsi, toute séquence de synchronisations dans $\widehat{\mathcal{A}}^{S'}_\otimes$ est aussi une séquence de synchronisations dans $\widehat{\mathcal{A}}^S_\otimes$ commençant immédiatement après la synchronisation décrite par ℓ.

Définition 30 *Soient $\widehat{\mathcal{A}}^S_\otimes$ et $\widehat{\mathcal{A}}^{S'}_\otimes$ deux automates produits et $\ell \in \Sigma$ un symbole de l'alphabet. $\widehat{\mathcal{A}}^S_\otimes \overset{\ell}{\lessdot} \widehat{\mathcal{A}}^{S'}_\otimes$ si et seulement si pour tout chemin étiqueté $\ell_1 \dots \ell_n$ dans $\widehat{\mathcal{A}}^S_\otimes$, il existe un chemin étiqueté $\ell.\ell_1 \dots \ell_n$ dans $\widehat{\mathcal{A}}^{S'}_\otimes$.* □

Remarquons que $\widehat{\mathcal{A}}^S_\otimes \overset{\varepsilon}{\lessdot} \widehat{\mathcal{A}}^{S'}_\otimes$ si et seulement si $\widehat{\mathcal{A}}^S_\otimes \lessdot \widehat{\mathcal{A}}^{S'}_\otimes$. La propriété suivante indique comment les résultats de l'analyse évoluent par réduction.

Proposition 31 (Preuve par réduction) *Soit P un groupe de processus tel que $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}^S_\otimes \models^S P$. Si $K, P \xrightarrow{[\ell]} K, P'$ alors nous avons $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}^{S'}_\otimes \models^S P'$ pour un automate produit $\widehat{\mathcal{A}}^{S'}_\otimes$ tel que $\widehat{\mathcal{A}}^{S'}_\otimes \overset{\ell}{\lessdot} \widehat{\mathcal{A}}^S_\otimes$.* □

L'automate produit $\mathrm{A}^S_\otimes(\widehat{\mathcal{G}})$ présenté dans cette section nous a permis d'établir la correction de l'analyse. Cependant, sa taille est potentiellement exponentielle en celle du programme. Dans la section suivante, nous présentons une version réduite $\mathrm{A}^R_\otimes(\widehat{\mathcal{G}})$ de $\mathrm{A}^S_\otimes(\widehat{\mathcal{G}})$, dont la taille est polynômiale, et qui est une approximation correcte de $\mathrm{A}^S_\otimes(\widehat{\mathcal{G}})$.

4.3.2 Analyse utilisant l'automate produit réduit

Dans cette section, nous présentons un automate $\mathrm{A}^R_\otimes(\widehat{\mathcal{G}})$ qui est une version réduite de $\mathrm{A}^S_\otimes(\widehat{\mathcal{G}})$. Tandis que la taille de l'automate produit est potentiellement exponentielle en la taille n du programme, $\mathrm{A}^R_\otimes(\widehat{\mathcal{G}})$ a pour taille $O(n^4)$. Nous montrons que $\mathrm{A}^R_\otimes(\widehat{\mathcal{G}})$ peut être utilisé au lieu de $\mathrm{A}^S_\otimes(\widehat{\mathcal{G}})$ pour l'analyse.

Intuitivement, les seules informations véritablement utiles pour l'analyse sont les paires de points de synchronisation pouvant interagir ensemble. Par exemple, nous souhaitons connaître l'ensemble des émetteurs potentiels pour une réception donnée, indépendemment de toute notion d'ordre, c.à.d. de la place de cette communication dans une trace de l'exécution du programme.

L'automate produit $\mathrm{A}^{S}_{\otimes}(\widehat{\mathcal{G}})$ contient toutes les séquences de synchronisations pour toutes les exécutions possibles, ce qui est trop précis dans notre cas. Aussi, nous réduisons sa taille en supprimant ces informations inutiles, tout en gardant assez de précision pour éliminer des synchronisations impossibles.

Ce compromis est obtenu dans $\mathrm{A}^{R}_{\otimes}(\widehat{\mathcal{G}})$ par la construction d'un automate permettant de répondre à la question : "quels sont les points de synchronisation susceptibles de succéder à une synchronisation donnée ?". Soient l_0 et l_1 deux étiquettes correspondant à des points de synchronisation compatibles dans le programme. Nous introduisons un état Q_{l_0,l_1} qui décrit cette synchronisation dans $\mathrm{A}^{R}_{\otimes}(\widehat{\mathcal{G}})$ et l'ensemble des points de synchronisation pouvant suivre cette interaction est contenu dans un ensemble appelé $\mathbb{L}(\mathrm{Q}_{l_0,l_1})$.

Une nouvelle synchronisation entre des points l_2 et l_3 est autorisée si l_2 et l_3 appartiennent tous les deux à $\mathbb{L}(\mathrm{Q}_{l,l'})$, pour une synchronisation précédente correspondant à l'état $\mathrm{Q}_{l,l'}$. Dans ce cas, nous ajoutons une (l_2, l_3)-transition entre $\mathrm{Q}_{l,l'}$ et Q_{l_2,l_3}, et $\mathbb{L}(\mathrm{Q}_{l_2,l_3})$ est mis à jour.

Soit S l'ensemble des états accessibles par une (l_s, l_r)-transition dans $\mathrm{A}^{S}_{\otimes}(\widehat{\mathcal{G}})$. S est approché dans $\mathrm{A}^{R}_{\otimes}(\widehat{\mathcal{G}})$ par un seul état Q_{l_s,l_r}. Ainsi, toute (l_s, l_r)-transition dans $\mathrm{A}^{R}_{\otimes}(\widehat{\mathcal{G}})$ aboutit à Q_{l_s,l_r}. De plus, $\mathrm{A}^{R}_{\otimes}(\widehat{\mathcal{G}})$ garde trace de l'avancement du calcul séquentiel à l'intérieur des processus après une synchronisation décrite pas (l_s, l_r) via $\mathbb{L}(\mathrm{Q}_{l_s,l_r})$ qui contient tous les états Q tels que Q apparaît dans un état Q^S de l'automate produit. Remarquons que $\mathbb{L}(\mathrm{Q}_{l_s,l_r})$ est un ensemble non ordonné, contenant dans le pire des cas tous les états de tous les automates, sa complexité en taille est donc de l'ordre de $O(n)$. Nous présentons maintenant la définition formelle de $\mathrm{A}^{R}_{\otimes}(\widehat{\mathcal{G}})$.

Définition 32 (Automate produit réduit) *Soit $\widehat{\mathcal{G}}$ une collection de n automates. L'automate produit réduit $\mathrm{A}^{R}_{\otimes}(\widehat{\mathcal{G}})$ des automates de $\widehat{\mathcal{G}}$ est un quintuplet $(\Sigma^R, \mathrm{Q}^R_0, \mathbb{Q}^R, \delta^R, \mathbb{L})$. L'alphabet est $\Sigma^R = Lab^2$. Les états appartiennent à $\mathbb{Q}^R = \{\mathrm{Q}^R_{l,l'} \ : \ l, l' \in Lab\} \cup \{\mathrm{Q}^R_0\}$ où Q^R_0 est un nouvel état intial. $\delta^R \in (\mathbb{Q}^R \times \Sigma^R) \to \wp(\mathbb{Q}^R)$ est la fonction de transition. $\mathbb{L} \ : \ \mathbb{Q}^R \to \wp(\mathbb{Q}_P)$, avec $\mathbb{Q}_P = \cup_{\widehat{\mathcal{A}}_p \in \widehat{\mathcal{G}}} \textsc{State}(\widehat{\mathcal{A}}_p)$, associe à chaque état $\mathrm{Q}^R_{l,l'}$ l'ensemble des états potentiellement actifs dans les automates $\widehat{\mathcal{A}}_p \in \widehat{\mathcal{G}}$ une fois que les points l et l' se sont synchronisés. $\mathrm{A}^{R}_{\otimes}(\widehat{\mathcal{G}})$ vérifie les conditions suivantes.*

(i) $\mathrm{Q}^R_0 \in \mathbb{Q}^R$ *et* $\mathbb{L}(\mathrm{Q}^R_0) = \cup_{\widehat{\mathcal{A}}_p \in \widehat{\mathcal{G}}} \textsc{Start}(\widehat{\mathcal{A}}_p)$.

(ii) Pour tout $\mathrm{Q}^R \in \mathbb{Q}^R$,

(a) Pour tout $\mathrm{Q} \in \mathbb{L}(\mathrm{Q}^R)$,

$$\left(\exists \widehat{\mathcal{A}}_p \in \widehat{\mathcal{G}} \ : \ [\![\mathrm{Q} \stackrel{\varepsilon}{\multimap} \mathrm{Q}']\!] \sqsubseteq \widehat{\mathcal{A}}_p\right) \Rightarrow \{\mathrm{Q}'\} \subseteq \mathbb{L}(\mathrm{Q}^R) \tag{4.7}$$

(b) Pour tout $\mathrm{Q}_s \in \mathbb{L}(\mathrm{Q}^R)$, $\mathrm{Q}_r \in \mathbb{L}(\mathrm{Q}^R)$,

$$\left(\exists \widehat{\mathcal{A}}_p, \widehat{\mathcal{A}}_{p'} \in \widehat{\mathcal{G}} \ : \ \left\{\begin{array}{l} [\![\mathrm{E}_{l_1} \overset{l_s}{\multimap} \mathrm{Q}_{l_s}]\!] \sqsubseteq \widehat{\mathcal{A}}_p \\ [\![\mathrm{E}_{l_2} \overset{l_r}{\multimap} \mathrm{Q}_{l_r}]\!] \sqsubseteq \widehat{\mathcal{A}}_{p'} \\ (\texttt{send}\ e_0^{l_0}\ e_1^{l_1})^{l_s} \in \mathrm{PRG} \\ (\texttt{receive}\ e_2^{l_2})^{l_r} \in \mathrm{PRG} \end{array}\right.\right) \tag{4.8}$$

$$\Longrightarrow \left\{\begin{array}{l} [\![\mathrm{Q}^R \overset{l_s,l_r}{\multimap} \mathrm{Q}^R_{l_s,l_r}]\!] \sqsubseteq \mathrm{A}^R_{\otimes}(\widehat{\mathcal{G}}) \\ \left(\mathbb{L}(\mathrm{Q}^R) \setminus \{\mathrm{E}_{l_1}, \mathrm{E}_{l_2}\}\right) \cup \{\mathrm{Q}_{l_s}, \mathrm{Q}_{l_r}\} \subseteq \mathbb{L}(\mathrm{Q}^R_{l_s,l_r}) \end{array}\right.$$

(c) Pour tout $\mathrm{Q}_f \in \mathbb{L}(\mathrm{Q}^R)$,

$$\left(\exists \widehat{\mathcal{A}}_p \in \widehat{\mathcal{G}} \ : \ \left\{\begin{array}{l} [\![\mathrm{B}_{l_f} \overset{l_f}{\multimap} \mathrm{Q}_{l_f}]\!] \sqsubseteq \widehat{\mathcal{A}}_p \\ [\![\mathrm{B}_{l_f} \overset{l_f}{\multimap} \mathrm{Q}_{l_0}]\!] \sqsubseteq \widehat{\mathcal{A}}_p \\ (\texttt{fork}\ e_0^{l_0})^{l_f} \in \mathrm{PRG} \end{array}\right.\right) \tag{4.9}$$

$$\Longrightarrow \left\{\begin{array}{l} [\![\mathrm{Q}^R \overset{l_f,l_f}{\multimap} \mathrm{Q}^R_{l_f,l_f}]\!] \sqsubseteq \mathrm{A}^R_{\otimes}(\widehat{\mathcal{G}}) \\ \left(\mathbb{L}(\mathrm{Q}^R) \setminus \{\mathrm{B}_{l_f}\}\right) \cup \{\mathrm{Q}_{l_f}, \mathrm{Q}_{l_0}\} \subseteq \mathbb{L}(\mathrm{Q}^R_{l_f,l_f}) \end{array}\right.$$

Nous construisons $\mathrm{A}^R_{\otimes}(\widehat{\mathcal{G}})$ incrémentalement.

(i) Nous commençons avec un nouvel état initial Q^R_0. $\mathbb{L}(\mathrm{Q}^R_0)$ contient les états initiaux des automates présents dans $\widehat{\mathcal{G}}$. Ceci indique que les points de départ des processus séquentiels sont les points actifs du programme distribué au début de l'exécution. Une ε-transition dans un automate $\widehat{\mathcal{A}}_p \in \widehat{\mathcal{G}}$ décrit un pas de réduction interne au processus p. Pour tout $\mathrm{Q} \in \mathbb{L}(\mathrm{Q}^R)$ et pour toute transition $[\![\mathrm{Q} \overset{\varepsilon}{\multimap} \mathrm{Q}']\!] \sqsubseteq \widehat{\mathcal{A}}_p$ telle que $\mathrm{Q} \in \mathbb{Q}_P$ et $\mathrm{Q}^R \in \mathbb{Q}^R$, Q' est ajouté à $\mathbb{L}(\mathrm{Q})$ indiquant que Q' est aussi un des points éventuellement actifs après exécution de la synchronisation décrite par Q^R et avant toute autre synchronisation. Ceci correspond à l'équation (4.7).

(ii) Considérons les transitions $[\![\mathrm{E}_{l_1} \overset{l_s}{\multimap} \mathrm{Q}_{l_s}]\!]$ et $[\![\mathrm{E}_{l_2} \overset{l_r}{\multimap} \mathrm{Q}_{l_r}]\!]$ dans deux automates $\widehat{\mathcal{A}}_p$ et $\widehat{\mathcal{A}}_{p'}$. La communication est possible si E_{l_1} et E_{l_2} sont éventuellement actifs au même moment, c.à.d. si E_{l_1} et E_{l_2} appartiennent tous les deux à $\mathbb{L}(\mathrm{Q}^R)$ pour un certain $\mathrm{Q}^R \in Q^R$. Dans ce cas, la transition $[\![q \overset{l_s,l_r}{\multimap} q_{l_s,l_r}]\!]$ est ajoutée à $\mathrm{A}^R_{\otimes}(\widehat{\mathcal{G}})$. De plus, $\mathbb{L}(\mathrm{Q}^R_{l_s,l_r})$ doit contenir les points actifs après réalisation de la synchronisation. $\mathbb{L}(\mathrm{Q}^R)$ contient les points actifs avant la synchronisation sauf E_{l_1} et E_{l_2}, plus les points suivant la communication, c.à.d. Q_{l_s} et Q_{l_r}. Ceci correspond à l'équation (4.8). Les `fork` sont

traités de manière similaire dans l'équation (4.9). Un nouvel état $Q^R_{l_f,l_f}$ est ajouté à Q^R et $\mathbb{L}(Q^R_{l_f,l_f})$ est mis à jour de la même façon que pour les communications.

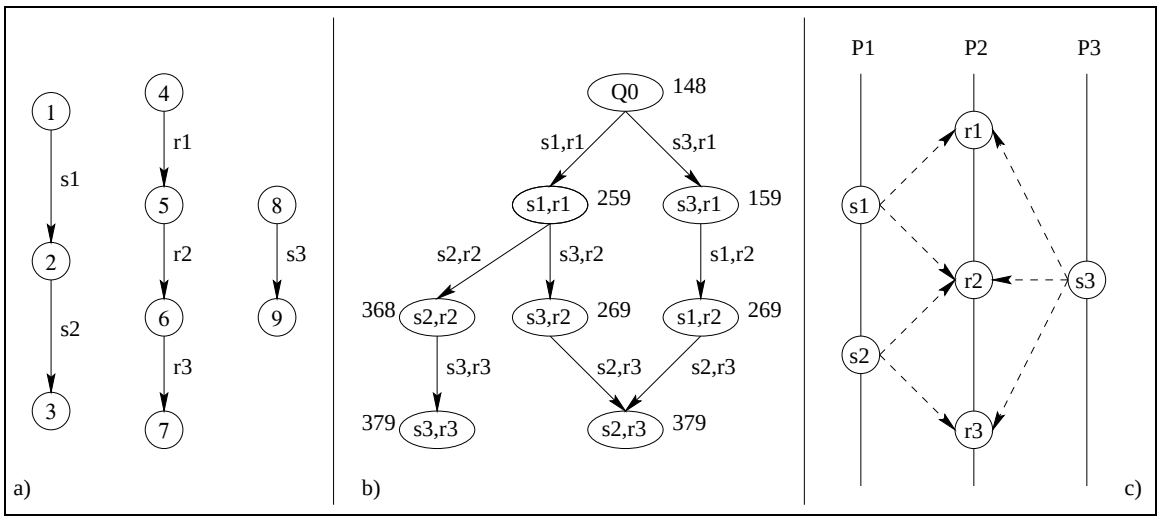


FIG. 4.6 – **a) Collection d'automates pour le programme de la figure 4.5. b) Automate produit réduit correspondant à la collection précédente. c) Approximation de la topologie produite par notre analyse pour le programme de la figure 4.5.**

Les figures 4.6 et 4.7 donnent des exemples d'automates produits réduits. La figure 4.6 montre l'automate produit réduit correspondant au programme de la figure 4.5. Dans cet exemple, nous pouvons remarquer qu'il y a une correspondance exacte entre les états de l'automate produit et les valeurs de la fonction $\mathbb{L}$. Ici, l'automate réduit est aussi précis que l'automate produit. La figure 4.7 montre les automates obtenus pour des programme comportant un haut degré de non-déterminisme.

Nous utilisons $\mathrm{A}^R_\otimes(\widehat{\mathcal{G}})$ pour définir une nouvelle analyse $\models^R$ pour un groupe de processus P. A nouveau, nous utilisons les ensembles $\widehat{\mathcal{C}}$, $\widehat{\mathcal{E}}$ définis à la section 4.2 ainsi que l'automate produit réduit noté $\widehat{\mathcal{A}}^R_\otimes$. $\models^R$ est obtenu en remplaçant l'automate produit par l'automate produit réduit dans la définition de $\models^S$.

Définition 33 *On dit qu'un triplet $(\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}^R_\otimes)$ définit une analyse correcte pour un groupe de processus P, et on note $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}^R_\otimes \models^R P$ si et seulement si*

(i) $\widehat{\mathcal{A}}^R_\otimes = \mathrm{A}^R_\otimes(\widehat{\mathcal{G}})$ *pour une collection* $\widehat{\mathcal{G}}$ *d'automates tels que* $\widehat{\mathcal{G}} = (\widehat{\mathcal{A}}_p)_{p\in Dom(P)}$ *et* $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_p \vdash e^l$ *pour tout* $\langle p : e^l \rangle \in P$.

(ii) Pour tout $[\![q \overset{l_s,l_r}{\multimap} q']\!] \sqsubseteq \mathrm{A}^R_\otimes(\widehat{\mathcal{G}})$ *t.q.* $\begin{cases} (\texttt{send}\ e_0^{l_0}\ e_1^{l_1})^{l_s} \in \mathrm{PRG} \\ (\texttt{receive}\ e_2^{l_2})^{l_r} \in \mathrm{PRG} \end{cases}$, $\widehat{\mathcal{C}}(l_0) \cap \widehat{\mathcal{C}}(l_2) \neq \varnothing$ *implique* $\widehat{\mathcal{C}}(l_1) \subseteq \widehat{\mathcal{C}}(l_r)$. □

A nouveau, l'existence d'une plus petite analyse au sens de $\prec$ est garantie par le fait que l'ensemble des analyses d'un groupe de processus forme une famille de Moore.

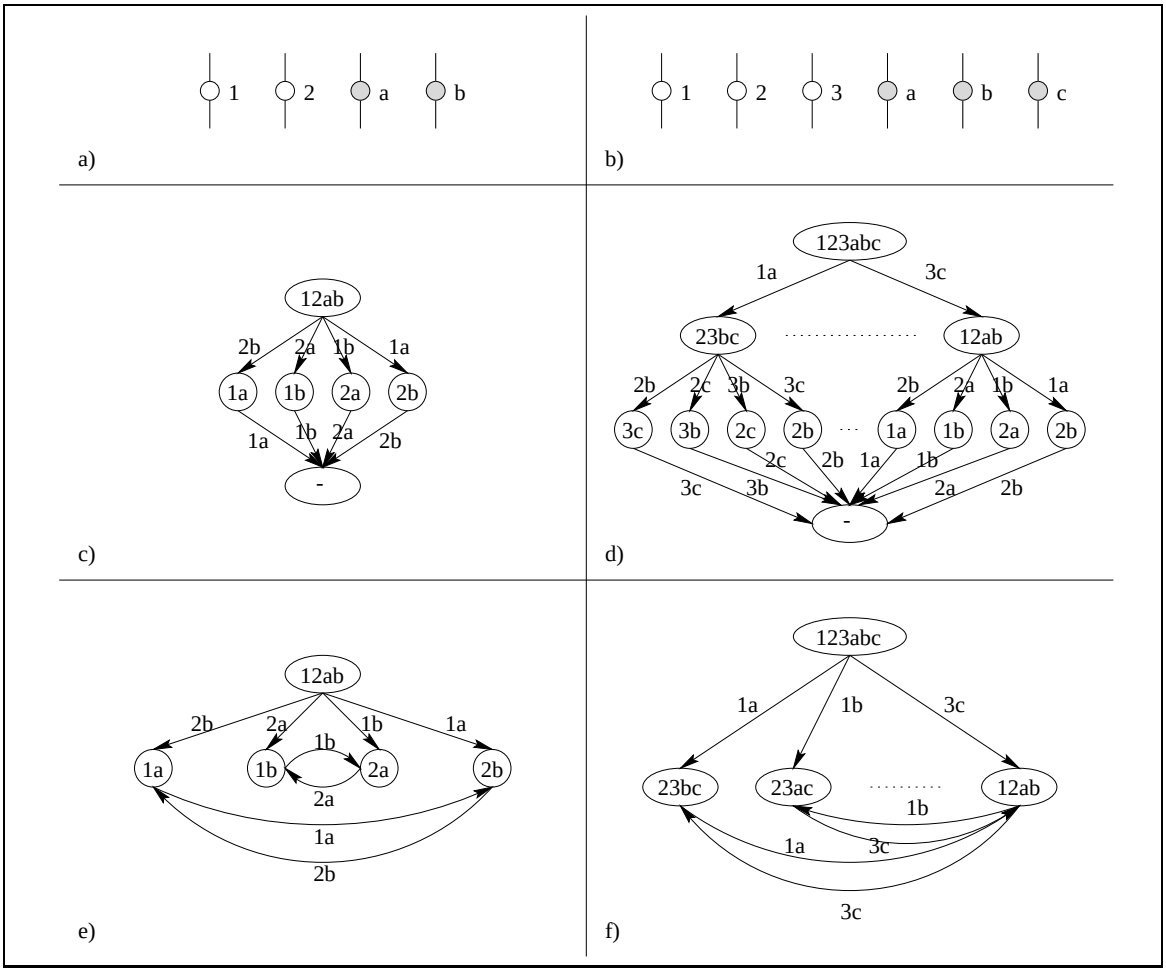


FIG. 4.7 – **Comparaison des automates produit et des automates réduits pour deux programmes composé de deux (resp. trois) émissions et de deux (resp. trois) réceptions exécutées en parallèle par quatre (resp. six) processus différents, donnés dans les schémas a) et b). Les schémas c) et d) montrent les automates produits obtenus. Les schémas e) et f) montrent les automates réduits correspondants. Les valeurs dans les états correspondent à $\mathbb{L}(\mathrm{Q})$.**

Proposition 34 (Famille de Moore) *Soit P un groupe de processus. L'ensemble des analyses $\{(\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}^R_\otimes) \,:\, \widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}^R_\otimes \models^R P\}$ est une famille de Moore.* □

L'automate produit réduit $\mathrm{A}^R_\otimes(\widehat{\mathcal{G}})$ fusionne les états de l'automate produit $\mathrm{A}^S_\otimes(\widehat{\mathcal{G}})$ suivant une synchronisation donnée. Par conséquent, toute transition sortant d'un état q^S dans $\mathrm{A}^S_\otimes(\widehat{\mathcal{G}})$ correspond à une transition sortant d'un état q^R dans $\mathrm{A}^R_\otimes(\widehat{\mathcal{G}})$, où q^R est l'état approximant q^S. Ainsi, tout chemin de $\mathrm{A}^S_\otimes(\widehat{\mathcal{G}})$ est aussi un chemin dans $\mathrm{A}^R_\otimes(\widehat{\mathcal{G}})$. Nous utilisons cette observation afin de démontrer que $\mathrm{A}^R_\otimes(\widehat{\mathcal{G}})$ peut être substitué à $\mathrm{A}^S_\otimes(\widehat{\mathcal{G}})$. Ainsi, nous établissons que l'automate réduit contient toutes les séquences de synchronisation de l'automate produit.

Proposition 35 (Relation entre les automates produits et réduits) *Soit P un groupe de processus et $\widehat{\mathcal{G}} = (\widehat{\mathcal{A}}_p)_{p \in Dom(P)}$ une collection d'automates tels que pour tout $\langle p : e^l \rangle \in P$, $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_p \vdash e^l$. La propriété*

$$\mathrm{A}^S_\otimes(\widehat{\mathcal{G}}) \preccurlyeq \mathrm{A}^R_\otimes(\widehat{\mathcal{G}}) \tag{4.10}$$

est satisfaite. □

Ainsi, $\models^R$ peut être utilisé pour l'analyse au lieu de $\models^S$ sans que les propriétés établies dans la section 4.3.1 soit invalidées.

Corollaire 36 *Soit P un groupe de processus. $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}^R_\otimes \models^R P$ implique $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}^S_\otimes \models^S P$ pour un automate $\widehat{\mathcal{A}}^S_\otimes$ tel que $\widehat{\mathcal{A}}^S_\otimes \preccurlyeq \widehat{\mathcal{A}}^R_\otimes$.* □

Nous concluons cette section en présentant la propriété suivante concernant la taille de l'automate produit réduit.

Proposition 37 (Taille de l'automate produit réduit) *Soit $\widehat{\mathcal{G}}$ une collection de k automates de taille $O(m)$ et soit $n = km$. L'automate produit réduit $\mathrm{A}^R_\otimes(\widehat{\mathcal{G}})$ a pour taille $O(n^4)$.* □

La Proposition 37 repose sur l'observation suivante. Il y a au plus $O(n)$ points de synchronisation dans le programme. Donc, le nombre d'états dans $\mathrm{A}^R_\otimes(\widehat{\mathcal{G}})$ est $O(n^2)$ et la fonction de transition a pour taille $O(n^4)$. Pour un état donné $\mathrm{Q} \in Q^R$, $\mathbb{L}(\mathrm{Q})$ contient, dans le pire des cas, l'ensemble des étiquettes du programme, c.à.d. que $\mathbb{L}(\mathrm{Q})$ a pour taille $O(n)$. donc $\mathbb{L}$ a pour taille $O(n^3)$ et la taille totale de l'automate est majorée par $O(n^4)$, à cause de la fonction de transition.

4.4 Calcul effectif des solutions

Dans cette section, nous montrons comment calculer une analyse $(\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_\otimes)$ pour un groupe de processus P. Nous générons un ensemble $\mathrm{C}[P]$ de contraintes tel qu'une solution $(\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_\otimes)$ à $\mathrm{C}[P]$ satisfait $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_\otimes \models^R P$. Le calcul de la plus petite solution s'effectue en un temps polynômial.

4.4.1 Génération de contraintes

Les contraintes c que nous utilisons appartiennent au langage défini par la grammaire suivante.

$$\mathrm{C}[c^l] = \{\mathrm{E}_l \in \overline{\delta}(\mathrm{B}_l, \varepsilon)\}$$

$$\mathrm{C}[x^l] = \{\overline{\mathcal{E}}(x) \subseteq \overline{\mathcal{C}}(l),\ \mathrm{E}_l \in \overline{\delta}(\mathrm{B}_l, \varepsilon)\}$$

$$\mathrm{C}[(\texttt{fun}\ x^{l_1}\ \texttt{=>}\ e_0^{l_0})^l]\,\overline{\mathcal{E}} = \left| \begin{array}{l} \mathrm{C}[e_0^{l_0}] \cup \{l \in \overline{\mathcal{C}}(l),\ \overline{\mathcal{C}}(l_1) \subseteq \overline{\mathcal{E}}(x)\} \cup \\ \{\mathrm{E}_l \in \overline{\delta}(\mathrm{B}_l, \varepsilon),\ \mathrm{B}_{l_0} \in \overline{\delta}(\mathrm{B}_l, \mu),\ \mathrm{E}_l \in \overline{\delta}(\mathrm{E}_{l_0}, \mu)\} \end{array} \right.$$

$$\mathrm{C}[(\texttt{rec}\ f^{l_1}\ x^{l_2}\ \texttt{=>}\ e_0^{l_0})^l] = \left| \begin{array}{l} \mathrm{C}[e_0^{l_0}] \cup \{l \in \overline{\mathcal{C}}(l),\ \overline{\mathcal{C}}(l) \subseteq \overline{\mathcal{C}}(l_1) \\ \overline{\mathcal{C}}(l_1) \subseteq \overline{\mathcal{E}}(f),\ \overline{\mathcal{C}}(l_2) \subseteq \overline{\mathcal{E}}(x)\} \cup \\ \{\mathrm{E}_l \in \overline{\delta}(\mathrm{B}_l, \varepsilon),\ \mathrm{B}_{l_0} \in \overline{\delta}(\mathrm{B}_l, \mu),\ \mathrm{E}_l \in \overline{\delta}(\mathrm{E}_{l_0}, \mu)\} \end{array} \right.$$

$$\mathrm{C}[(e_0^{l_0}\ e_1^{l_1})^l] = \left| \begin{array}{l} \mathrm{C}[e_0^{l_0}] \cup \mathrm{C}[e_1^{l_1}] \cup \{\mathrm{B}_{l_0} \in \overline{\delta}(\mathrm{B}_l, \varepsilon),\ \mathrm{B}_{l_1} \in \overline{\delta}(\mathrm{E}_{l_0}, \varepsilon)\} \cup \\ \quad \bigcup_{(\texttt{fun}\ x^{l_4}\ \texttt{=>}\ e_3^{l_3})^{l_2} \in \mathrm{prg}} \left\{ \begin{array}{l} \left(l_2 \in \overline{\mathcal{C}}(l_0) \Rightarrow \overline{\mathcal{C}}(l_1) \subseteq \overline{\mathcal{C}}(l_4)\right) \\ \left(l_2 \in \overline{\mathcal{C}}(l_0) \Rightarrow \overline{\mathcal{C}}(l_3) \subseteq \overline{\mathcal{C}}(l)\right) \\ \left(l_2 \in \overline{\mathcal{C}}(l_0) \Rightarrow \mathrm{B}_{l_3} \in \overline{\delta}(\mathrm{E}_{l_1}, \varepsilon)\right) \\ \left(l_2 \in \overline{\mathcal{C}}(l_0) \Rightarrow \mathrm{E}_l \in \overline{\delta}(\mathrm{E}_{l_3}, \varepsilon)\right) \end{array} \right\} \cup \\ \quad \bigcup_{(\texttt{rec}\ f^{l_4}\ x^{l_5}\ \texttt{=>}\ e_3^{l_3})^{l_2} \in \mathrm{prg}} \left\{ \begin{array}{l} \left(l_2 \in \overline{\mathcal{C}}(l_0) \Rightarrow \overline{\mathcal{C}}(l_1) \subseteq \overline{\mathcal{C}}(l_5)\right) \\ \left(l_2 \in \overline{\mathcal{C}}(l_0) \Rightarrow \overline{\mathcal{C}}(l_2) \subseteq \overline{\mathcal{C}}(l_4)\right) \\ \left(l_2 \in \overline{\mathcal{C}}(l_0) \Rightarrow \overline{\mathcal{C}}(l_3) \subseteq \overline{\mathcal{C}}(l)\right) \\ \left(l_2 \in \overline{\mathcal{C}}(l_0) \Rightarrow \mathrm{B}_{l_3} \in \overline{\delta}(\mathrm{E}_{l_1}, \varepsilon)\right) \\ \left(l_2 \in \overline{\mathcal{C}}(l_0) \Rightarrow \mathrm{E}_l \in \overline{\delta}(\mathrm{E}_{l_3}, \varepsilon)\right) \\ \left(l_2 \in \overline{\mathcal{C}}(l_0) \Rightarrow \mathrm{E}_l \in \overline{\delta}(\mathrm{E}_{l_1}, \varepsilon)\right) \\ \left(l_2 \in \overline{\mathcal{C}}(l_0) \Rightarrow \mathrm{B}_{l_3} \in \overline{\delta}(\mathrm{E}_{l_3}, \varepsilon)\right) \end{array} \right\} \end{array} \right.$$

FIG. 4.8 – **Contraintes générées pour une expression e^l - Partie 1.**

$$\begin{array}{rcl} c & ::= & \mathrm{Set} \subseteq v \mid v_1 \subseteq v_2 \mid \mathrm{Set} \subseteq v \Rightarrow c \\ & & \mid v_1 \cap v_2 \neq \varnothing \Rightarrow c \mid v_1 \setminus \mathrm{Set} \subseteq v_2 \\ v & ::= & \overline{\mathcal{C}} \mid \overline{\mathcal{E}} \mid \overline{\Delta} \mid \overline{\Delta}_\otimes \mid \overline{\mathbb{L}} \end{array} \qquad \begin{array}{rcl} \overline{\mathcal{C}} & ::= & \overline{\mathcal{C}}(l_1) \mid \overline{\mathcal{C}}(l_2) \mid \ldots \\ \overline{\mathcal{E}} & ::= & \overline{\mathcal{E}}(x) \mid \overline{\mathcal{E}}(y) \mid \ldots \\ \overline{\Delta} & ::= & \overline{\delta}(\nu, l) \mid \ldots \\ \overline{\Delta}_\otimes & ::= & \overline{\delta}_\otimes(q, (l, l')) \mid \ldots \\ \overline{\mathbb{L}} & ::= & \overline{\mathbb{L}}(q) \mid \ldots \end{array}$$

Les contraintes sont définies entre des éléments de $\overline{\mathcal{C}}, \overline{\mathcal{E}}, \overline{\Delta}, \overline{\Delta}_\otimes$ et $\overline{\mathbb{L}}$. $\overline{\mathcal{C}} : \mathrm{Lab} \to \wp(\mathrm{Lab})$ est le cache abstrait, $\overline{\mathcal{E}} : \mathrm{Id} \to \wp(\mathrm{Lab})$ est l'environnement abstrait, $\overline{\Delta}$ est l'union des domaines des fonctions de transition des automates décrivant les processus séquentiels et $\overline{\Delta}_\otimes$ et $\overline{\mathbb{L}}$ sont les domaines des fonctions $\overline{\delta}_\otimes$ et $\overline{\mathbb{L}}$ définissant l'automate produit réduit $\overline{\mathcal{A}}_\otimes = (\overline{\Sigma}_\otimes, \overline{q}_0, \overline{Q}_\otimes, \overline{\delta}_\otimes, \overline{\mathbb{L}})$.

Remarquons que cette grammaire définit des contraintes *monotones*. Plus précisément, il est toujours possible de satisfaire une contrainte c en ajoutant des éléments dans un des ensembles. Ceci est évident pour les contraintes d'inclusion. Quant aux implications, leur

$$\mathrm{C}[(\texttt{if}\ e_0^{l_0}\ e_1^{l_1}\ e_2^{l_2})^l] = \left| \begin{array}{l} \mathrm{C}[e_0^{l_0}] \cup \mathrm{C}[e_1^{l_1}] \cup \mathrm{C}[e_2^{l_2}] \cup \{\overline{C}(l_1) \subseteq \overline{C}(l),\ \overline{C}(l_2) \subseteq \overline{C}(l)\} \cup \\ \{\mathrm{B}_{l_0} \in \overline{\delta}(\mathrm{B}_l, \varepsilon),\ \mathrm{B}_{l_1} \in \overline{\delta}(\mathrm{E}_{l_0}, \varepsilon)\} \cup \\ \{\mathrm{B}_{l_2} \in \overline{\delta}(\mathrm{E}_{l_0}, \varepsilon),\ \mathrm{E}_l \in \overline{\delta}(\mathrm{E}_{l_1}, \varepsilon),\ \mathrm{E}_l \in \overline{\delta}(\mathrm{E}_{l_2}, \varepsilon)\} \end{array} \right.$$

$$\mathrm{C}[(\texttt{let}\ x^{l_2} = e_0^{l_0}\ \texttt{in}\ e_1^{l_1})^l] = \left| \begin{array}{l} \mathrm{C}[e_0^{l_0}] \cup \mathrm{C}[e_1^{l_1}] \cup \{\overline{C}(l_0) \subseteq \overline{C}(l_2) \\ \overline{C}(l_2) \subseteq \overline{\mathcal{E}}(x),\ \overline{C}(l_1) \subseteq \overline{C}(l)\} \cup \\ \{\mathrm{B}_{l_0} \in \overline{\delta}(\mathrm{B}_l, \varepsilon),\ \mathrm{B}_{l_1} \in \overline{\delta}(\mathrm{E}_{l_0}, \varepsilon),\ \mathrm{E}_l \in \overline{\delta}(\mathrm{E}_{l_1}, \varepsilon)\} \end{array} \right.$$

$$\mathrm{C}[\texttt{channel()}^l] = \{l \in \overline{C}(l),\ \mathrm{E}_l \in \overline{\delta}(\mathrm{B}_l, \varepsilon)\}$$

$$\mathrm{C}[(\texttt{fork}\ e_0^{l_0})^l] = \mathrm{C}[e_0^{l_0}] \cup \{\mathrm{E}_l \in \overline{\delta}(\mathrm{B}_l, l),\ \mathrm{B}_{l_0} \in \overline{\delta}(\mathrm{B}_l, l)\}$$

$$\mathrm{C}[(\texttt{send}\ e_0^{l_0}\ e_1^{l_1})^l] = \left| \begin{array}{l} \mathrm{C}[e_0^{l_0}] \cup \mathrm{C}[e_1^{l_1}] \cup \{\overline{C}(l_1) \subseteq \overline{C}(l)\} \cup \\ \{\mathrm{B}_{l_0} \in \overline{\delta}(\mathrm{B}_l, \varepsilon),\ \mathrm{B}_{l_1} \in \overline{\delta}(\mathrm{E}_{l_0}, \varepsilon),\ \mathrm{E}_l \in \overline{\delta}(\mathrm{E}_{l_1}, l)\} \end{array} \right.$$

$$\mathrm{C}[(\texttt{receive}\ e_0^{l_0})^l] = \mathrm{C}[e_0^{l_0}] \cup \{\mathrm{B}_{l_0} \in \overline{\delta}(\mathrm{B}_l, \varepsilon),\ \mathrm{E}_l \in \overline{\delta}(\mathrm{E}_{l_0}, l)\}$$

FIG. 4.9 – **Contraintes générées pour une expression e^l - Partie 2.**

partie droite est toujours une inclusion. Par conséquent, aucun ensemble n'est modifié si la condition est fausse, et nous avons une inclusion dans le cas contraire. De plus, les conditions dans la partie gauche sont toujours de la forme Set$\subseteq v$ où Set est un ensemble statique (jamais modifié). Ainsi, lorsqu'une implication est vérifiée, elle ne peut plus être invalidée puisque nous n'enlevons pas d'éléments dans les ensembles. Le fait que les contraintes sont monotones permet de résoudre les systèmes obtenus en utilisant des algorithmes itératifs.

Nous générons un ensemble $\mathrm{C}[e^l]$ de contraintes sur $\overline{C}$, $\overline{\mathcal{E}}$ et $\overline{\Delta}$ pour chaque processus $\langle p : e^l \rangle$ du groupe. Ces contraintes sont décrites dans les figures 4.8 et 4.9. Il s'agit d'une version dirigée par la syntaxe des contraintes données pour la spécification de $\vdash$ dans les figures 4.2 et 4.3. De plus, une solution $(\widehat{C}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}})$ à $\mathrm{C}[e^l]$ est une analyse correcte de l'expression e^l, comme indiqué par la propriété suivante.

Proposition 38 *Si $(\widehat{C}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}})$ est une solution à $\mathrm{C}[e^l]$, alors $\widehat{C}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}} \vdash e^l$.* □

Pour un groupe de processus P, outre les contraintes correspondant aux différents processus pris individuellement, nous générons les contraintes pour l'automate produit réduit de la section 4.3.2. Ces contraintes sont données à la figure 4.10.

Soit P le groupe de processus à analyser. Les contraintes définies dans l'équation (4.11) imposent que $\widehat{C}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}} \vdash e^l$ pour tout $\langle p : e^l \rangle \in P$. Celles définies dans les équations (4.12), (4.13) et (4.15) correspondent aux conditions données à la définition 32 pour la construction de l'automate produit réduit. LAB_S est l'ensemble des étiquettes attachées

$$\mathrm{C}[P] = \bigcup_{\langle p : e^l \rangle \in P} \mathrm{C}[e^l] \tag{4.11}$$

$$\bigcup \left\{ \{\nu\} \subseteq \overline{L}(q) \Rightarrow (\overline{\delta}(\nu, \varepsilon) \subseteq \overline{L}(q)) \ : \ q \in \overline{Q}_{\otimes},\ \{\nu\} \subseteq \overline{Q} \right\} \tag{4.12}$$

$$\bigcup \left\{ \left(\begin{array}{l|l|l} \{\mathrm{E}_{l_1}, \mathrm{E}_{l_2}\} \subseteq \overline{L}(q) & \Rightarrow & \{q_{l_s,l_r}\} \subseteq \overline{\delta}_{\otimes}(q, (l_s, l_r)) \\ \{\mathrm{E}_{l_s}\} \subseteq \overline{\delta}(\mathrm{E}_{l_1}, l_s) & & \overline{L}(q) \cap \mathrm{Lab}_S \setminus \{\mathrm{E}_{l_1}, \mathrm{E}_{l_2}\} \\ \{\mathrm{E}_{l_r}\} \subseteq \overline{\delta}(\mathrm{E}_{l_2}, l_r) & & \quad \subseteq \overline{L}(q_{l_s,l_r}) \\ & & \{\mathrm{E}_{l_s}, \mathrm{E}_{l_r}\} \subseteq \overline{L}(q_{l_s,l_r}) \end{array} \right) : \left| \begin{array}{l} (\texttt{send}\ e_0^{l_0}\ e_1^{l_1})^{l_s} \in \mathrm{prg} \\ (\texttt{receive}\ e_2^{l_2})^{l_r} \in \mathrm{prg} \\ q \in \overline{Q}_{\otimes} \end{array} \right. \right\} \tag{4.13}$$

$$\bigcup \left\{ \left(\begin{array}{c} \overline{\delta}_{\otimes}(q, (l_s, l_r)) \neq \varnothing \Rightarrow \\ \left((\widehat{C}(l_0) \cap \widehat{C}(l_2) \neq \varnothing) \Rightarrow \widehat{C}(l_1) \subseteq \widehat{C}(l_r) \right) \end{array} \right) : \left| \begin{array}{l} (\texttt{send}\ e_0^{l_0}\ e_1^{l_1})^{l_s} \in \mathrm{prg} \\ (\texttt{receive}\ e_2^{l_2})^{l_r} \in \mathrm{prg} \\ q \in \overline{Q}_{\otimes} \end{array} \right. \right\} \tag{4.14}$$

$$\bigcup \left\{ \left(\begin{array}{l|l|l} \{\mathrm{B}_{l_f}\} \subseteq \overline{L}(q) & \Rightarrow & \{q_{l_f,l_f}\} \subseteq \overline{\delta}_{\otimes}(q, (l_f, l_f)) \\ \{\mathrm{E}_{l_f}\} \subseteq \overline{\delta}(\mathrm{B}_{l_f}, l_f) & & \overline{L}(q) \cap \mathrm{Lab}_S \setminus \{\mathrm{B}_{l_f}\} \cup \{\mathrm{B}_{l_0}, \mathrm{E}_{l_f}\} \\ \{\mathrm{B}_{l_0}\} \subseteq \overline{\delta}(\mathrm{B}_{l_f}, l_f) & & \quad \subseteq \overline{L}(q_{l_f,l_f}) \end{array} \right) : \left| \begin{array}{l} (\texttt{fork}\ e_0^{l_0})^{l_f} \in \mathrm{prg} \\ q \in \overline{Q}_{\otimes} \end{array} \right. \right\} \tag{4.15}$$

FIG. 4.10 – **Constraintes générées pour un groupe de processus P.**

à des points de synchronisation. Enfin, les contraintes données à l'équation (4.14) définissent la manière dont les valeurs sont affectées aux caches abstraits correspondant aux points de réception. Ceci correspond aux conditions données dans la définition 33 pour spécifier $\models^R$.

$\mathrm{C}[P]$ étant une version dirigée par la syntaxe de la spécification développée au cours de la section 4.3.2, les solutions à ce système sont des analyses correctes pour P.

Proposition 39 *Soit P un groupe de processus et soit $(\widehat{C}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_{\otimes})$ une solution à $\mathrm{C}[P]$. Alors $\widehat{C}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_{\otimes} \models^R P$.* □

4.4.2 Résolution

Plusieurs méthodes sont possibles pour résoudre les systèmes de contraintes générés par les règles des figures 4.8 et 4.9, dont les méthodes de calcul de points fixe et celles utilisant une formulation sous forme de graphe des contraintes, que nous utilisons (NNH99; SNN97).

Intuitivement, dans ces graphes, chaque nœud v correspond à une valeur abstraite et un arc (v, v') décrit une dépendance entre v et v'. Cette dépendance indique que si la valeur abstraite décrite par v est modifiée, alors celle correspondant à v' doit l'être aussi. Par exemple, soient v et v' deux nœuds correspondant à $\overline{C}(l)$ et $\overline{C}(l')$ et soit $\overline{C}(l) \subseteq \overline{C}(l')$ une contrainte du système. L'arc (v, v') indique que si $\overline{C}(l)$ est modifié, alors $\overline{C}(l')$ doit aussi l'être.

La résolution s'effectue en deux temps. Tout d'abord, nous construisons le graphe de

dépendances entre les contraintes du système. Ensuite, nous résolvons le système en suivant les chemins dans le graphe et en modifiant les valeurs attachées aux nœuds lorsque cela est nécessaire.

Soit

$$V_{CE} \mathrel{\widehat{=}} \{\overline{\mathcal{C}}(l) \ : \ l \in \textsc{Lab}\} \ \cup \ \{\overline{\mathcal{E}}(x) \ : \ x \in \textsc{Id}\} \tag{4.16}$$

Un nœud $v \in V_{CE}$ correspond à la valeur en un point des fonctions $\overline{\mathcal{C}}$ ou $\overline{\mathcal{E}}$.

Afin de construire les automates pour les différents processus du programme ainsi que l'automate produit réduit, nous construisons aussi les ensembles de nœuds V_{AS} et V_{AP}.

$$\begin{array}{rcl} V_{AS} & \mathrel{\widehat{=}} & \left\{\overline{\delta}(\textsc{b}_{l_1}, l_2) \ : \ l_1 \in \textsc{Lab},\ l_2 \in \textsc{Lab} \cup \{\varepsilon\}\right\} \\ & \cup & \left\{\overline{\delta}(\textsc{e}_{l_1}, l_2) \ : \ l_1 \in \textsc{Lab},\ l_2 \in \textsc{Lab} \cup \{\varepsilon\}\right\} \end{array} \tag{4.17}$$

$$\begin{array}{rcl} V_{AP} & \mathrel{\widehat{=}} & \{\overline{\delta}_{\otimes}(\overline{q}_0, (l_1, l_2)) \ : \ l_1, l_2 \in \textsc{Lab}\} \\ & \cup & \{\overline{\delta}_{\otimes}(q_{l_1,l_2}, (l_3, l_4)) \ : \ l_i \in \textsc{Lab},\ 1 \leq i \leq 4\} \\ & \cup & \{\overline{\mathbb{L}}(q_{l_0,l_1}) \ : \ l_0, l_1 \in \textsc{Lab}\} \end{array} \tag{4.18}$$

De plus, chaque nœud v dans l'un des ensembles V_{CE}, V_{AS} ou V_{AP} est relié à une valeur par les fonctions $\mathbb{F}_{CE} : V_{CE} \to \wp(\text{Lab})$, $\mathbb{F}_{AS} : V_{AS} \to \wp(\overline{Q}_P)$ et $\mathbb{F}_{AP} : V_{AP} \to \wp(\overline{Q}_{\otimes})$ où $\overline{Q}_P \mathrel{\widehat{=}} \{\textsc{b}_l, \textsc{e}_l,\ l \in \textsc{Lab}\}$. Intuitivement, $\mathbb{F}_{CE}(v)$ est un environnement qui indique la valeur associée durant la résolution à la variable décrite par le noeud v. $\mathbb{F}_{AS}(v)$ et $\mathbb{F}_{AP}(v)$ sont utilisés de la même manière pour des noeuds correspondant à des ensembles de nature différente. Au début de la résolution, $\mathbb{F}_{CE}(v)$, $\mathbb{F}_{AS}(v)$ et $\mathbb{F}_{AP}(v)$ sont initialisés à partir des contraintes dont la partie gauche est un ensemble statique (contraintes de la forme Set$\subseteq v$ par exemple). Ces contraintes ne pouvant pas être invalidées une fois qu'elles sont vérifiées, nous ne les considérons plus lors de la phase itérative de la résolution. Ainsi nous avons initialement

$$\mathbb{F}_{CE}(v) = \{l \ : \ (l \in v) \in \textsc{c}[P]\} \tag{4.19}$$

$$\mathbb{F}_{AS}(v) = \{\nu' \ : \ (\nu' \in \overline{\delta}(\nu, l)) \in \textsc{c}[P],\ v = \overline{\delta}(\nu, l)\} \tag{4.20}$$

$$\mathbb{F}_{AP}(v) = \begin{cases} \varnothing \text{ si } v \neq \overline{\mathbb{L}}(\overline{q}_0) \\ \{\textsc{b}_l \ : \ \langle p : e^l \rangle \in P\} \text{ si } v = \overline{\mathbb{L}}(\overline{q}_0) \end{cases} \tag{4.21}$$

Pour résoudre $\textsc{c}[P]$, nous considérons le graphe $G = (V_{CE} \cup V_{AS} \cup V_{AP}, E)$. Les arcs de E correspondent aux contraintes $c \in \textsc{c}[P]$. La valeur attachée à un nœud v est donnée par $\mathbb{F}(v)$ qui est $\mathbb{F}_{CE}(v)$, $\mathbb{F}_{AS}(v)$ ou $\mathbb{F}_{AP}(v)$ selon la nature de v. Un arc (v, v') étiqueté c entre deux sommets v et v' indique que si $\mathbb{F}(v)$ est modifié durant la résolution, alors $\mathbb{F}(v')$ doit aussi être modifié afin de satisfaire la contrainte c.

Soit RSH(c) la fonction donnant la partie droite d'une contrainte c, c.à.d.

$$\text{RHS}(c) = \begin{cases} v_2 \text{ si } c = (v_1 \subseteq v_2) \text{ ou } c = (v_1 \setminus \text{Set} \subseteq v_2) \\ \text{RSH}(c') \text{ si } c = (\text{Set} \subseteq v_1 \Rightarrow c') \\ \text{RSH}(c') \text{ si } c = (v_1 \cap v_2 \neq \varnothing \Rightarrow c') \end{cases} \tag{4.22}$$

Les arcs sont construits selon le type de contrainte rencontrée. Leur sommet destination indique que la valeur abstraite correspondante doit être modifiée lorsque celle attachée au sommet source l'est.

(i) $(v_1, v_2) \in E$ si $(v_1 \subseteq v_2) \in \text{C}[P]$.

(ii) $(v_1, v_2) \in E$ si $(v_1 \setminus \text{Set} \subseteq v_2) \in \text{C}[P]$.

(iii) $(v_1, v_2) \in E$ si $(\text{Set} \subseteq v_1 \Rightarrow c) \in \text{C}[P]$ et $v_2 = \text{RHS}(c)$.

(iv) $(v_1, v) \in E$ et $(v_2, v) \in E$ si $(v_1 \cap v_2 \neq \varnothing \Rightarrow c) \in \text{C}[P]$ et $v = \text{RHS}(c)$.

L'algorithme utilisé pour la résolution est donné dans les figures 4.11 et 4.12. [] est la liste vide. M est la liste des sommets v dont la valeur $\mathbb{F}(v)$ a été modifiée lors de l'itération précédente. La phase de construction du graphe construit les arcs du graphe en fonction des contraintes. Pour chaque sommet v, nous ajoutons $E(v)$ à l'ensemble des contraintes devant être examinées si $\mathbb{F}(v)$ est modifié. Cette phase résout aussi les contraintes immédiates de la forme Set $\subseteq v$ et affecte à M la liste des sommets à examiner lors de la première itération de la phase suivante. Finalement, dans la phase d'itération, les valeurs sont propagées jusqu'à ce que toutes les contraintes soient satisfaites.

Proposition 40 *Soit P un groupe de processus. L'algorithme des figures 4.11 et 4.12 appliqué à* $\text{C}[P]$ *calcule le plus petit triplet* $(\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}})$ *au sens de* $\prec$ *tel que* $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}} \models^R P$. □

La complexité de la résolution de $\text{C}[P]$ pour un groupe de processus P correspondant à un programme de taille $O(n)$ provient des observations suivantes. Par définition de V_{CE}, V_{AS} et V_{AP}, le graphe $G = (V, E)$ a au plus $O(n^4)$ sommets. Chaque arc correspond à une contrainte et le nombre de celles-ci dans $\text{C}[P]$ est majoré par le $O(n^4)$ à cause des équations (4.13) et (4.14). En conséquence, il y a au plus $O(n^4)$ arcs dans G. Pour un sommet donné v, la taille de $\mathbb{F}(v)$ est au plus $O(n^2)$, ce qui implique de $\mathbb{F}(v)$ est modifié au plus $O(n^2)$ fois. Ainsi, il y a $O(n^4)$ arcs qui sont examinés au plus $O(n^2)$ fois. La complexité globale de l'algorithme est $O(n^6)$.

4.5 Applications

Dans cette section, nous examinons les résultats fournis par une implantation de l'analyse de flot de contrôle décrite dans ce chapitre. Nous montrons comment cette

```
Main : /* Initialization */
       M ← []
       F(q̄_0) ← {B_l : ⟨p : e^l⟩ ∈ P}
       for each v ∈ V \ {q̄_0} do
           F(v) ← ∅
           E(v) ← []
       /* Graph Construction */
       for each c ∈ C[P] do
           match c with
                 Set ⊆ v : Add (v,Set)
                 _       : for each v ∈ Mod(c) do
                               E(v) ← c :: E(v)
       /* Iteration */
       while M ≠ [] do
             v ← head (M)
             M ← tail (M)
             for each c ∈ E(v) do
                 solve (c)
```

FIG. 4.11 – **Algorithme pour la résolution des contraintes - Partie 1.**

analyse peut être utilisée afin de vérifier la sûreté d'une implantation d'un mécanisme d'allocation de circuits virtuels similaire à celui utilisé dans ATM (All95; Tan96), et pour établir que des propriétés de confidentialité sont vérifiées par une application distribuée de vente aux enchères.

4.5.1 Vérification d'un mécanisme d'allocation de circuits

La création d'un circuit virtuel se déroule selon le schéma de la figure 4.13. Les différents nœuds du réseau sont reliés à leurs voisins par des canaux de contrôle. Ces canaux sont utilisés afin de propager un message de demande de création de circuit jusqu'au destinataire. Ce dernier établit un lien avec son voisin qui procède de même avec son autre voisin, jusqu'à ce que le nœud source soit atteint (toujours en utilisant les canaux de contrôle). Chaque nœud intermédiaire se charge ensuite de propager les données arrivant sur son lien d'entrée vers son lien de sortie.

Une implantation complète de ce mécanisme a été réalisée en Concurrent ML, le langage traité par l'analyse. Pour des raisons de clarté, nous en donnons une version simplifiée, écrite dans un pseudo-langage impératif (voir figure 4.14). Par ailleurs, un seul nœud intermédiaire est présent entre la source et la destination.

Les trois processus p_1, p_2 et p_3 sont supposés s'exécuter sur des noeuds différents. #ctrl_12 (resp. #ctrl_23) est un canal de contrôle bidirectionnel reliant les nœuds correspondant à p_1 et p_2 (resp. p_2 et p_3). De plus, comme p_1 crée deux circuits différents a et b,

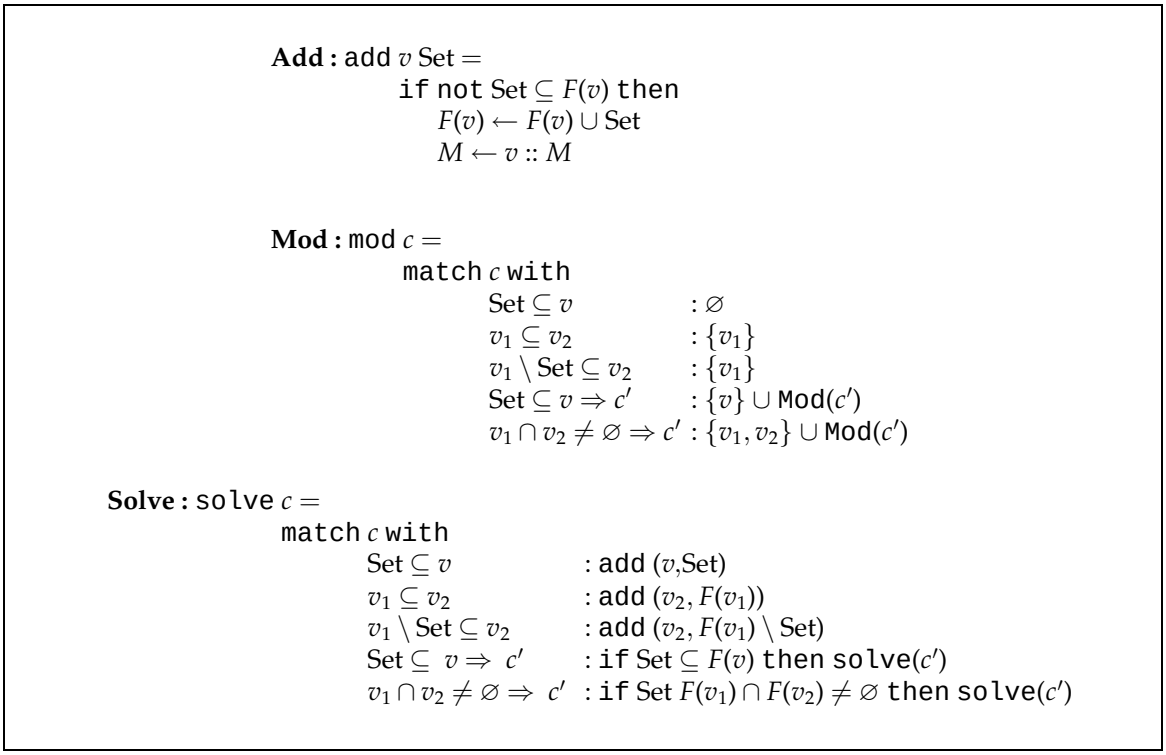


FIG. 4.12 – **Algorithme pour la résolution des contraintes - Partie 2.**

les morceaux de code donnés pour les processus p_2 sont exécutés deux fois. Dans notre implantation, nous utilisons une analyse de flot de contrôle très simple pour les parties séquentielles du programme (0-CFA (NN97)). Ceci nous oblige à dupliquer ces morceaux de code afin de ne pas perdre en précision. Cependant ces duplications pourraient être évitées par l'utilisation d'une analyse plus élaborée (k-CFA). Ainsi, dans la figure 4.14, les deux étiquettes annotant certaines instructions font références aux étiquettes uniques de deux copies du morceau de code en question.

p_1, p_2 et p_3 fonctionnent comme suit. p_1 demande successivement la création des deux circuits. La fonction `connect` envoie un message d'initialisation sur le canal de contrôle `#ctrl_12` et affecte le nom du canal à utiliser pour a à la variable `virt_ch_a` lorsque celui-ci est reçu sur `#ctrl_12`. L'opération est répétée pour la création de b. Enfin, des données sont transmises sur `virt_ch_a` et `virt_ch_b`.

p_2 correspond au nœud situé entre la source et la destination. Lorsqu'une demande d'allocation de circuit est reçue sur `#ctrl_12`, elle est transmise à la destination via le canal de contrôle `#ctrl_23`. p_2 reçoit ensuite le nom du canal de sortie à utiliser, qu'il affecte à la variable `ch_out` et crée un nouveau canal `ch_in` (instruction `channel()`) qu'il transmet au processus source. Enfin, un nouveau processus est créé qui retransmet sur `ch_out` les valeurs reçues sur `ch_in`. L'opération est répétée pour b.

Enfin, p_3 correspond au nœud destination. A l'arrivée des messages de contrôle, p_3

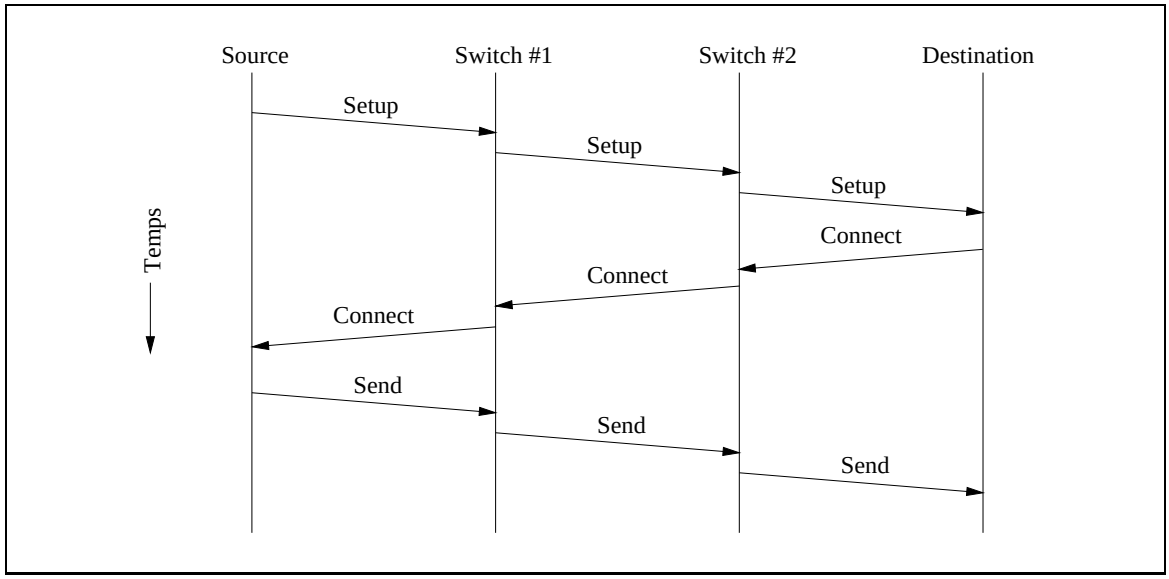


FIG. 4.13 – **Principe de fonctionnement du mécanisme de création de circuit virtuel.**

crée un nouveau canal qui est affecté à la variable `ch_in` et transmise à p_2 via le canal de contrôle `#ctrl_23`. Une fois les circuits créés, p_3 reçoit des données sur les canaux correspondant à a et b.

Nous avons utilisé l'analyse décrite dans ce chapitre afin de vérifier le bon fonctionnement d'une implantation complète du programme. Nous présentons ici les résultats obtenus qui ont été transposés au programme de la figure 4.14. Les tableaux ci-dessous donnent les valeurs abstraites obtenues pour les principaux points du programme ainsi que la valeur abstraite associée à certaines variables.

Comme indiqué précédemment, les morceaux de code correspondant à p_2 et p_3 sont exécutés deux fois. Dans le tableau de la figure 4.15 donnant les valeurs abstraites des variables, nous indiquons à laquelle des deux exécutions il est fait référence par (a) et (b).

La principale observation concerne les valeurs abstraites reçues par p_3 aux points 132 et 134. Ces valeurs, $\{30\}$ et $\{35\}$, sont celles émises par p_1 sur les circuits a et b. Ceci valide le mécanisme d'allocation des circuits. Les valeurs reçues sur a (resp. b) sont celles qui ont été envoyées sur ce circuit et seulement celles-ci. En outre, cela nous assure que le processus p_2 n'intervertit pas les valeurs reçues sur ses liens d'entrée et retransmises sur ses liens de sortie.

Concernant la création du circuit a, il est possible de vérifier que que les variables `virt_ch_a` et `ch_dest` (a) qui correspondent aux deux extrémités du circuit ont pour valeurs abstraites $\{58\}$ et $\{124\}$, ce qui correspond aux canaux créés au points d'étiquettes correspondantes. Le nouveau processus créé par p_2 retransmet sur le canal créé en 124 les valeurs arrivant sur le canal créé en 58 (variables `ch_in` (a) et `ch_out` (a)).

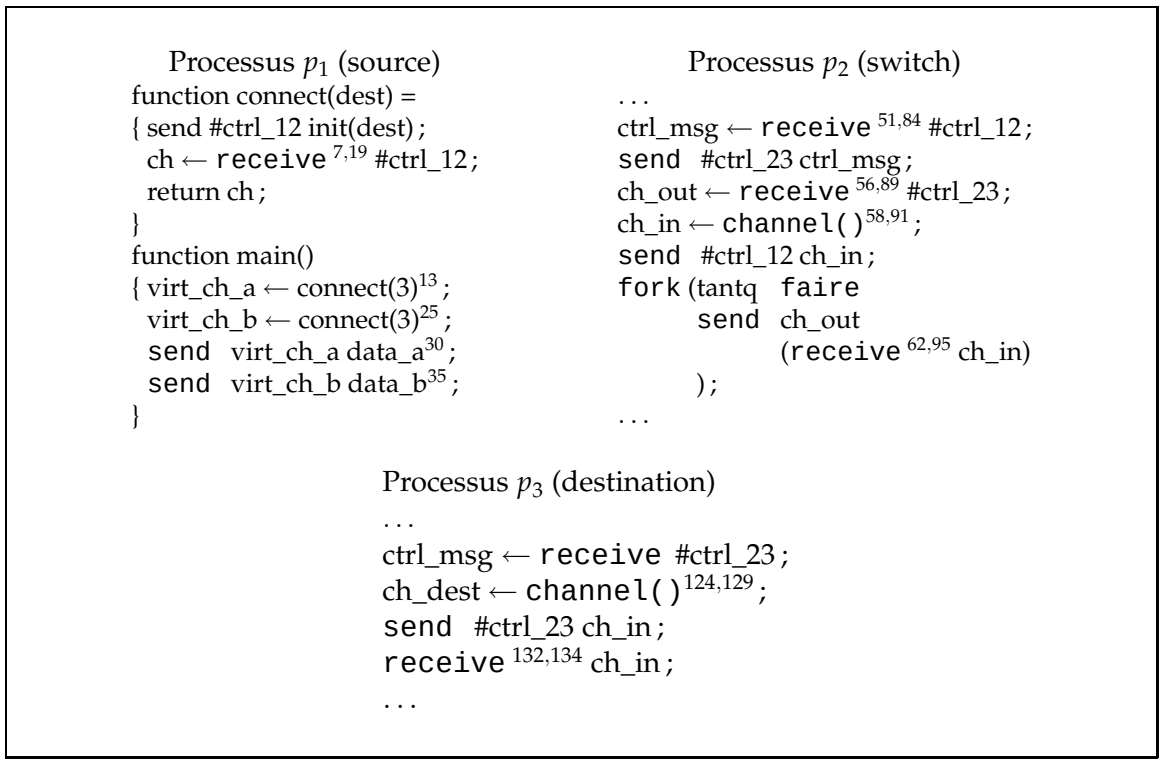


FIG. 4.14 – **Implémentation du mécanisme de création de circuits virtuels. La procédure connect ainsi que les morceaux de code des processus p_2 et p_3 sont exécutés deux fois.**

La transmission des données de la source vers la destination est donc assurée. De plus, il serait possible de vérifier que les canaux de valeurs abstraites {58} et {124} ne sont pas utilisés en d'autres points du programme. Ainsi, les données envoyées sur le circuit virtuel *a* sont uniquement transmises au destinataire. Les mêmes constatations peuvent être faites pour le circuit *b*.

Enfin, notons que pour obtenir ces résultats, il est nécessaire d'utiliser une analyse construisant une approximation de la topologie des communications réalisées par le programme. En effet, les mêmes canaux de contrôle `#ctrl_12` et `#ctrl_23` sont utilisés pour la création des deux circuits virtuels. En outre, l'analyse décrite par les équa-

Etiquette	7	13	19	25	30	35	51	56	58
Valeur	{58}	{13}	{91}	{25}	{30}	{35}	{13}	{124}	{58}

Etiquette	62	84	89	91	95	124	129	132	134
Valeur	{30}	{25}	{129}	{91}	{35}	{124}	{129}	{30}	{35}

Variable	#ctrl_12	#ctrl_23	virt_ch_a	virt_ch_b	ctrl_msg (*a*)	ctrl_msg (*b*)
Valeur	{1}	{2}	{58}	{91}	{13}	{25}

Variable	ch_in (*a*)	ch_in (*b*)	ch_out (*a*)	ch_out (*b*)	ch_dest (*a*)	ch_dest (*b*)
Valeur	{58}	{91}	{124}	{129}	{124}	{129}

FIG. 4.15 – **Valeurs abstraites attachées au programme de la figure 4.14.**

tions (2.5) et (2.6) serait insuffisante et nous obtiendrions par exemple `virt_ch_a` = `virt_ch_b` = $\{58, 91\}$, ce qui ne permet pas de vérifier qu'aucune confusion n'est faite entre les valeurs envoyées sur les deux circuits.

4.5.2 Vérification d'une application de vente aux enchères

Dans cette section, nous montrons comment utiliser notre CFA pour vérifier des propriétés liées à la sécurité d'une application distribuée de vente aux enchères.

Cette application est décrite par un programme distribué composé d'un serveur et de plusieurs clients. Les clients envoient au serveur des données publiques (par exemple le montant de leur enchère ou leur pseudonyme) ainsi que des données privées (par exemple leur numéro de carte de crédit). Chaque client est relié au serveur par un canal de communication. Le serveur diffuse à tous les clients les informations publiques qu'il reçoit et conserve les informations privées. Notre but est de montrer que, dans notre implantation de cette application, seule les données publiques sont effectivement diffusées.

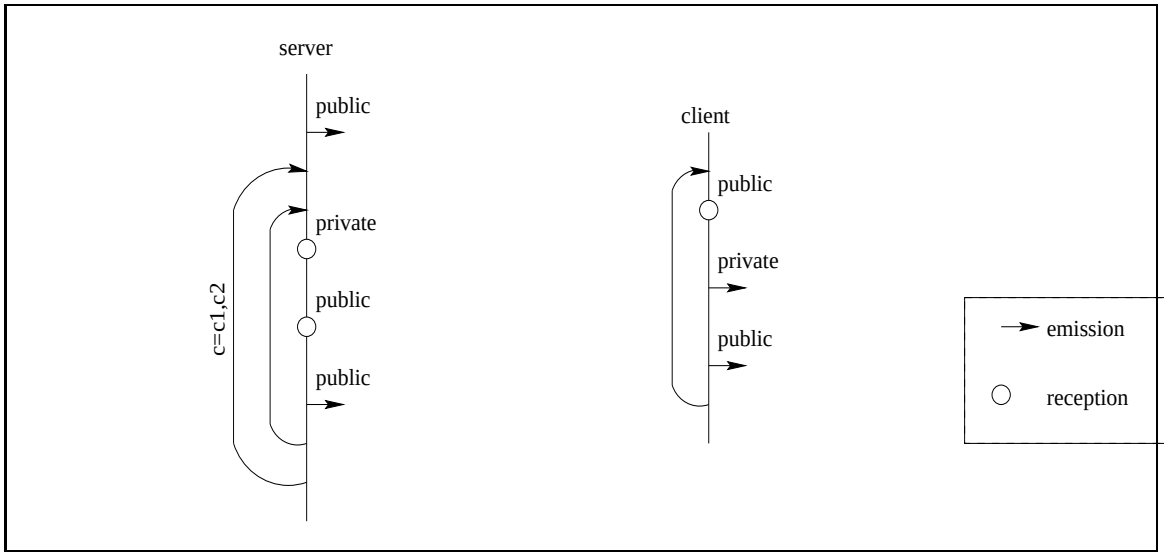


FIG. 4.16 – **Processus clients et serveur de l'application de vente aux enchères.**

Dans notre implantation, décrite à la figure 4.16, un client réalise les communications suivantes de manière répétée. Tout d'abord, il reçoit des données publiques, par exemple le montant de la dernière enchère. Ensuite, il envoie des données privées puis publiques. Dans notre implantation, ces trois étapes sont modélisées par trois communications sur le même canal.

Le serveur initialise le mécanisme en envoyant des données publiques initiales au premier client (par exemple le montant initial de l'enchère). Ensuite, chaque client est successivement consulté et le processus est répété jusqu'à ce que les enchères soient closes. Consulter un client consiste à recevoir ses données privées et publiques et à diffuser ces

dernières aux autres clients. Notre implantation décrit un programme distribué composé d'un serveur et de deux clients communicant avec le serveur sur des canaux γ_1 et γ_2 respectivement.

Tout comme pour le mécanisme d'allocation de circuits virtuels, cette application a été écrite en Concurrent ML et nous présentons les résultats obtenus pour une version réécrite de ce programme donnée à la figure 4.17. Dans le code du client, deux étiquettes sont attachées aux instructions. Elles correspondent aux deux instances de ce processus. Dans le code du serveur, les deux étiquettes attachées aux instructions correspondent aux deux instances de la même instruction (pour les mêmes raisons que dans le mécanisme de création de circuit, nous devons dupliquer les instructions se trouvant dans la boucle la plus interne du code du serveur).

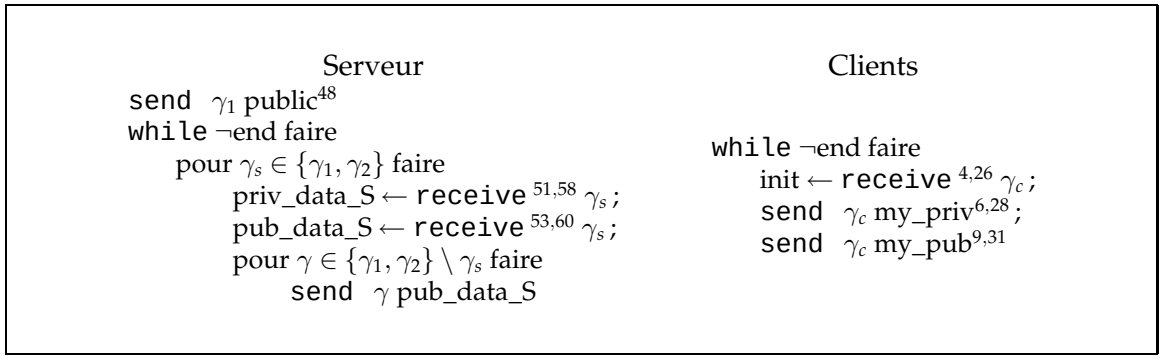


FIG. 4.17 – **Implementation de l'application de vente aux enchères.**

Les valeurs abstraites calculées par l'analyse sont données dans la figure 4.18. Le résultat principal concerne les valeurs abstraites obtenues pour les points 4 et 26 correspondant aux valeurs reçues par les deux instances du processus client. La valeur abstraite attachée au point 4 est $\{31, 48\}$. Cela signifie que le premier client reçoit soit la valeur émise durant la phase d'initialisation du serveur (valeur abstraite $\{48\}$), soit une valeur abstraite émise au point 31 par le second client. L'analyse ne distingue pas les valeurs émises à différentes itérations de la boucle du code du client, mais l'analyse détecte qu'une ins-

Etiquette	4	26	48	51	53	58	60
Valeur	$\{31, 48\}$	$\{9\}$	$\{48\}$	$\{6\}$	$\{9\}$	$\{28\}$	$\{31\}$

Variable	priv_data_S (1)	priv_data_S (2)	γ_s (1)	γ_s (2)	γ_1
Valeur	$\{6\}$	$\{28\}$	$\{2\}$	$\{1\}$	$\{2\}$
Variable	pub_data_S (1)	pub_data_S (2)	init (1)	init (2)	γ_c (1)
Valeur	$\{9\}$	$\{31\}$	$\{31, 48\}$	$\{9\}$	$\{2\}$
Variable	my_priv (1)	my_priv (2)	my_pub (1)	my_pub (2)	γ_c (2)
Valeur	$\{6\}$	$\{28\}$	$\{9\}$	$\{31\}$	$\{1\}$

FIG. 4.18 – **Valeurs abstraites attachées au programme de la figure 4.17.**

tance de la réception étiquetée 4 reçoit seulement des valeurs émises par une instance de l'instruction (send γ_c my_pub^{31}). Puisqu'uniquement des valeurs publiques sont émises aux points 31 et 48, cela garantit que le premier client ne reçoit pas des données privées qui ne le concerne pas.

De manière similaire, la valeur abstraite attachée au point de réception 26 dans le code du second client est $\{9\}$, ce qui correspond à l'une des données publiques émises par le premier client. Par conséquence, le second client ne reçoit pas non plus des données privées qui ne lui sont pas destinées.

Plus généralement, nous pouvons vérifier que les valeurs abstraites correspondant aux données privées émises par un client apparaissent seulement dans les valeurs attachées aux points du code du serveur et du client lui-même. Cela garantit la sécurité de notre implantation de cette application.

A nouveau, pour obtenir ces résultats, nous avons besoin d'une CFA qui construit une approximation de la topologie des communications de l'application. Du fait que les clients communiquent toutes leurs données sur le même canal, une analyse fondée sur les équations (2.5) et (2.6) mélange les données publiques et privées et il devient impossible de distinguer celles qui apparaissent à un point donné.

4.6 Conclusion

Dans ce chapitre, nous avons défini une analyse de flot de contrôle qui construit une approximation de la topologie des synchronisations d'un programme distribué écrit en Concurrent ML. Les aspects dynamiques des primitives de gestion du parallélisme sont pris en compte, tels que la création dynamique de processus ou de noms de canaux, ainsi que les transmission de fonctions. Nous avons implanté cette analyse et nous avons montré les résultats obtenus dans la section 4.5.

L'automate produit réduit nous permet de calculer une approximation fine de la topologie des synchronisations. Ceci permet de minimiser la valeur abstraite attachée aux points de réception. D'une part, cela accroît la précision de l'analyse sur les parties séquentielles du programme utilisant les valeurs reçues. D'autre part, le fait que les noms de canaux peuvent être communiqués permet d'approcher plus précisément les paires d'émetteurs et de récepteurs pour les communications suivantes du programme.

Nous pouvons constater que les gains dus à l'utilisation d'informations topologiques dans la CFA améliorent sensiblement la qualité de l'analyse pour de nombreuses applications. Pour des schémas de communication déterministes et très classiques, tels qu'une séquence de communications sur le même canal entre deux processus, cela nous permet de distinguer les valeurs échangées à chaque communication. Pour des schémas de

communication plus compliqués, il peut y avoir des pertes de précision sur certaines parties du programme, par exemple à cause des alternatives, des boucles ou du non-déterminisme. Cependant, la précision augmentera à nouveau lorsque les ambiguïtés disparaîtront, par exemple après une synchronisation globale.

Chapitre 5

Evaluation partielle de langages fonctionnels concurrents

5.1 Introduction

Dans ce chapitre, nous nous intéressons à l'évaluation partielle de programmes écrits dans un sous-ensemble non typé du langage Concurrent ML (MG00c). Nous définissons un évaluateur partiel nommé `Pev` et une analyse des temps de liaison permettant d'annoter les programmes. L'évaluateur partiel repose sur un méta-interpréteur `Eval` que nous définissons et qui est auto-applicable. La BTA utilise les résultats d'une analyse de flot de contrôle. Bien que n'importe quelle CFA puisse être utilisée, la précision de cette dernière influence la précision de la BTA, et nous utilisons ici la CFA décrite au chapitre 4.

L'évaluateur partiel, tout comme les analyses de flot de contrôle et des temps de liaison, ont été implantés et nous décrivons les résultats obtenus pour différents exemples. `Pev` est compatible avec les projections de Futamura (Fut71) et nous montrons les compilateurs et les générateurs de compilateurs que nous avons obtenus.

L'évaluation partielle de programmes écrits dans un langage fonctionnel concurrent permet d'exécuter à la compilation les communications statiques d'une application distribuée. Cela permet d'optimiser cette dernière en fonction d'un contexte d'exécution particulier. Dans la suite de ce chapitre, nous montrons comment cette technique peut être utilisée pour spécialiser des protocoles de communication.

Afin d'illustrer le principe de fonctionnement de notre évaluateur partiel, considérons le programme de la Figure 5.1, dans lequel deux processus p_1 et p_2 établissent un lien de communication entre eux par l'intermédiaire d'un serveur s. p_1 et p_2 sont connectés au serveur par les canaux γ_1 et γ_2. Le canal pour les communications entre p_1 et p_2 est fourni

par le serveur et est nommé α_1 dans p_1 et α_2 dans p_2.

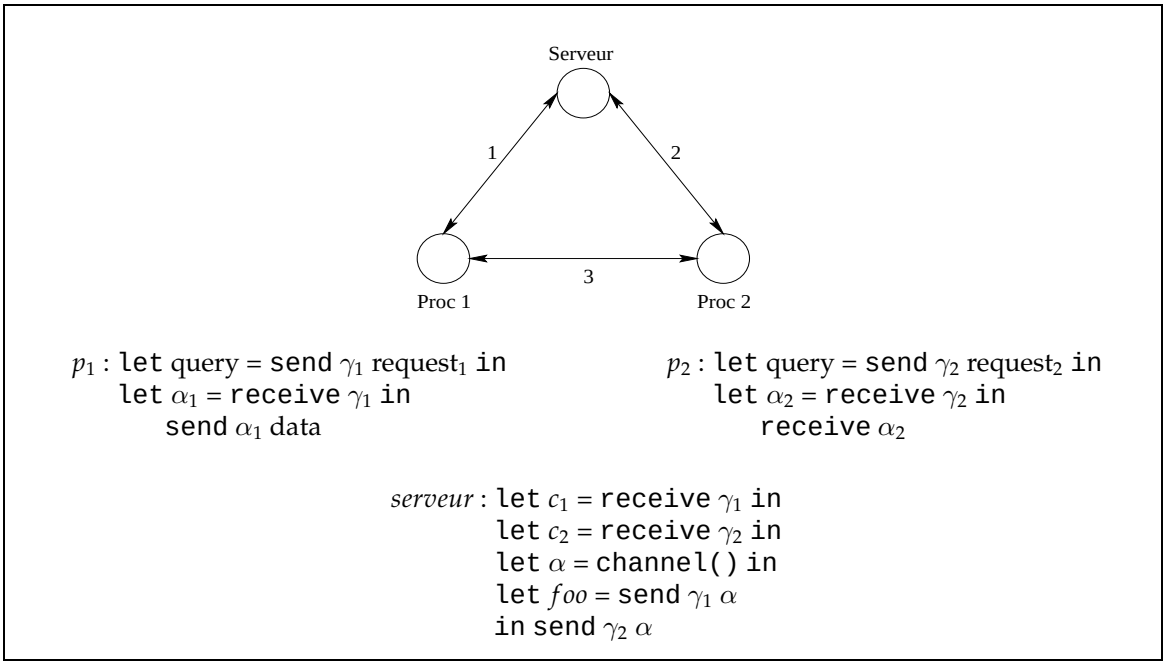


FIG. 5.1 – **Création d'un lien de communication entre deux processus par l'intermédiaire d'un serveur.**

Nous montrons ci-dessous comment cette application peut être spécialisée pour un réseau particulier, c.à.d. dans le cas où les canaux qui permettent d'accéder au serveur sont connus statiquement et où celui-ci est capable de déterminer statiquement le lien devant être utilisé pour les communications entre p_1 et p_2. Par contre, les données transmises entre p_1 et p_2 sont supposées inconnues au moment de la spécialisation.

Nous modélisons cette application par le programme suivant dans lequel les canaux de communication entre les deux processus et le serveur sont des paramètres d'entrée.

```
define p1-p2 =
  fun g1 g2 data ->
    let p1 = fork (let query = send g1 p2Id in
                   let a1 = receive g1 in
                   _send a1 data)
    in (* p2 *)   (let query = send g2 p1Id in
                   let a2 = receive g2 in
                   _receive a2) ;
```

La variable `data` est supposée dynamique. Aussi, la communication entre p_1 et p_2 ne peut être réalisée au moment de l'évaluation partielle. Ceci est indiqué dans le programme ci-dessus par les espaces blancs soulignés précédant les instructions de communication correspondantes. Ces annotations peuvent être obtenues automatiquement en

utilisant l'analyse des temps de liaison présentée à la section 5.3.

Nous spécialisons ce programme dans un contexte où les canaux γ_1 et γ_2 sont connus et où le serveur détermine le lien à utiliser pour les communications entre p_1 et p_2. Pour cela, γ_1 et γ_2 sont définis et le programme est codé sous la forme de données à fournir à l'évaluateur partiel. Nous indiquons de plus que la variable `data` est inconnue. Parallèlement à l'évaluation partielle du programme décrivant les processus p_1 et p_2, nous exécutons le programme correspondant au serveur afin que les communications statiques puissent avoir lieu. Ceci correspond à la séquence de commandes suivante (réécrite pour une plus grande lisibilité).

```
> define ch1 = channel() ;
> define ch2 = channel() ;

> define p1-p2-encoded = encode p1-p2 with g1=ch1, g2=ch2, data=? ;

> run {let server = (fork (let c1 = receive ch1 in
                           let c2 = receive ch2 in
                           let p1p2Ch = channel in
                           let foo1 = send ch1 p1p2Ch
                           in send ch2 p1p2Ch)
                    )
       in Pev @ p1-p2-encoded} ;
```

Lors de la spécialisation, les deux processus p_1 et p_2 sont créés, et les communications avec le serveur sont réalisées. Ainsi, le nom de canal effectif utilisé comme lien entre p_1 et p_2, qui est noté `#ch`, est inséré dans le code de ces processus. On obtient pour résultat le code de deux processus résiduels correspondant aux versions réduites de p_1 et p_2 et que nous présentons ci-dessous.

```
{ send #ch data } | { receive #ch }
```

La suite de ce chapitre est organisée de la manière suivante. La section 5.2 décrit le fonctionnement de notre évaluateur partiel et celui du méta-interpréteur sur lequel il est fondé. La section 5.3 définit l'analyse des temps de liaison que nous utilisons pour annoter les programmes. Les exemples présentés dans ces deux sections ont été testés grâce à nos implantations de `Pev` et de la BTA. Enfin, nous montrons, à la section 5.4, les résultats relatifs aux projections de Futamura, obtenus par auto-application de notre évaluateur partiel.

5.2 Méta-interprétation et évaluation partielle

Dans cette section, nous décrivons le principe de fonctionnement de notre l'évaluateur partiel, appelé `Pev`, qui prend en entrée un programme écrit dans un langage fonctionnel concurrent à deux niveaux (GM98; NN92; TS97) dont nous donnons la définition. `Pev` est construit à partir d'un méta-interpréteur `Eval` dont nous donnons la définition.

5.2.1 Méta-interprétation

Nous présentons ici le méta-interpréteur `Eval` qui évalue les termes appartenant au langage décrit par la grammaire donnée ci-dessous.

$$\begin{aligned} e \quad ::= \quad & v \mid x \mid \texttt{fun}\ x\ \texttt{=>}\ e_0 \mid \texttt{rec}\ f\ x\ \texttt{=>}\ e_0 \mid e_0\ \texttt{@}\ e_1 \mid \texttt{if}\ e_0\ e_1\ e_2 \\ & \mid \texttt{channel()} \mid \texttt{fork}\ e_0 \mid \texttt{send}\ e_0\ e_1 \mid \texttt{receive}\ e_0 \end{aligned} \tag{5.1}$$

Afin d'être reconnus par `Eval`, les programmes construits à partir de cette grammaire sont codés par une fonction $\ulcorner . \urcorner$. Dans notre implantation, $\ulcorner e \urcorner$ est l'arbre syntaxique correspondant à l'expression e. L'évaluation d'une expression e est alors définie par

$$\texttt{Eval}\ \ulcorner e \urcorner\ \rho \tag{5.2}$$

où ρ est un environnement contenant des variables globales (du premier ordre et d'ordre supérieur), accessibles par tous les processus. Lorsque `Eval` rencontre une variable libre dans le code d'un programme, il recherche sa valeur dans ρ, qui peut être vu comme une zone de mémoire partagée par les processus que l'on utilise pour définir les fonctions auxiliaires appelées par l'interpréteur.

Lorsque `Eval` rencontre une instruction $\ulcorner \texttt{fork}\ e \urcorner$ dans le programme qu'il interprète, il crée un nouveau processus dans lequel une copie `Eval'` de l'interprète est appliquée à $\ulcorner e \urcorner$. Les programmes `Eval` et `Eval'` sont identiques et font appel aux mêmes fonctions auxiliaires que l'on définit dans ρ et que l'on souhaite donc accessibles par tous les processus.

En revanche, lors de l'application d'une fonction `fun` x `=>` e, le paramètre effectif est directement substitué à x dans e et ρ demeure inchangé. Cette approche permet d'éviter les problèmes liés aux variables de même nom ayant des valeurs différentes dans différents processus ainsi que les problèmes de fermeture lors de la transmission d'une fonction au cours d'une communication.

La figure 5.2 décrit le fonctionnement de `Eval`. Les règles utilisées pour l'évaluation des expressions séquentielles sont classiques (GJ91). L'évaluation du code $\ulcorner \texttt{fun}\ x\ \texttt{=>}\ e \urcorner$ d'une fonction dans l'environnement ρ produit une nouvelle fonction $\texttt{fun}\ v\ \texttt{=>}\ \texttt{Eval}\ \ulcorner e\{x \leftarrow$

$v\}\urcorner\rho$ dans laquelle $\ulcorner e\{x \leftarrow v\}\urcorner$ est le code de e dans lequel la variable v a été substituée à la variable x.

`Eval` $\ulcorner$`channel()`$\urcorner$ ρ crée un nouveau nom de canal. L'évaluation de $\ulcorner$`fork` $e\urcorner$ crée un nouveau processus qui évalue $\ulcorner e\urcorner$ dans ρ. Lorsqu'une réception $\ulcorner$`receive` $e\urcorner$ est rencontrée, $\ulcorner e\urcorner$ est évalué dans ρ produisant un résultat α, puis la communication `receive` α est réalisée. De même, pour une émission $\ulcorner$`send` $e_0\ e_1\urcorner$, les expressions $\ulcorner e_0\urcorner$ et $\ulcorner e_1\urcorner$ sont évaluées et les résultats sont utilisés pour réaliser la communication.

$$
\begin{array}{rcl}
\texttt{Eval}\ulcorner c\urcorner\,\rho & = & c \\
\texttt{Eval}\ulcorner x\urcorner\,\rho & = & \rho(x) \\
\texttt{Eval}\ulcorner\texttt{fun}\ x\ \texttt{=>}\ e\urcorner\,\rho & = & \texttt{fun}\ v\ \texttt{=>}\ \texttt{Eval}\ulcorner e\{x \leftarrow v\}\urcorner\,\rho \\
\texttt{Eval}\ulcorner\texttt{rec}\ f\ x\ \texttt{=>}\ e\urcorner\,\rho & = & \texttt{fun}\ v\ \texttt{=>}\ \texttt{Eval}\ulcorner e\{x \leftarrow v\}\{f \leftarrow \texttt{rec}\ f\ x\ \texttt{=>}\ e\}\urcorner\,\rho \\
\texttt{Eval}\ulcorner(e_0\ \texttt{@}\ e_1)\urcorner\,\rho & = & (\texttt{Eval}\ulcorner e_0\urcorner\,\rho)\ \texttt{@}\ (\texttt{Eval}\ulcorner e_1\urcorner\,\rho) \\
\texttt{Eval}\ulcorner\texttt{if}\ e_0\ e_1\ e_2\urcorner\,\rho & = & \texttt{if}\ (\texttt{Eval}\ulcorner e_0\urcorner\,\rho)\ (\texttt{Eval}\ulcorner e_1\urcorner\,\rho)\ (\texttt{Eval}\ulcorner e_2\urcorner\,\rho) \\
\texttt{Eval}\ulcorner\texttt{channel()}\urcorner\,\rho & = & \texttt{channel()} \\
\texttt{Eval}\ulcorner\texttt{fork}\ e_0\urcorner\,\rho & = & \texttt{fork}\ (\texttt{Eval}\ulcorner e_0\urcorner\,\rho) \\
\texttt{Eval}\ulcorner\texttt{receive}\ e_0\urcorner\,\rho & = & \texttt{receive}\ (\texttt{Eval}\ulcorner e_0\urcorner\,\rho) \\
\texttt{Eval}\ulcorner\texttt{send}\ e_0\ e_1\urcorner\,\rho & = & \texttt{send}\ (\texttt{Eval}\ulcorner e_0\urcorner\,\rho)\ (\texttt{Eval}\ulcorner e_1\urcorner\,\rho)
\end{array}
$$

FIG. 5.2 – **Principe de fonctionnement du méta-évaluateur.**

5.2.2 Evaluation partielle

Afin d'annoter les programmes en vue de leur évaluation partielle, nous construisons un langage à deux niveaux à partir de celui de l'équation (5.1). Ce langage est défini par la grammaire de l'équation (5.3).

$$
\begin{array}{rcl}
e & ::= & c \mid x \mid \texttt{fun}\ x\ \texttt{=>}\ e_0 \mid \texttt{rec}\ f\ x\ \texttt{=>}\ e_0 \mid e_0\ \texttt{@}\ e_1 \mid \texttt{if}\ e_0\ e_1\ e_2 \\
 & & \mid \texttt{channel()} \mid \texttt{fork}\ e_0 \mid \texttt{send}\ e_0\ e_1 \mid \texttt{receive}\ e_0 \\
 & & \mid \underline{c} \mid \underline{\texttt{fun}}\ x\ \texttt{=>}\ e_0 \mid \underline{\texttt{rec}}\ f\ x\ \texttt{=>}\ e_0 \mid e_0\ \underline{\texttt{@}}\ e_1 \mid \underline{\texttt{if}}\ e_0\ e_1\ e_2 \\
 & & \mid \underline{\texttt{channel()}} \mid \underline{\texttt{fork}}\ e_0 \mid \underline{\texttt{send}}\ e_0\ e_1 \mid \underline{\texttt{receive}}\ e_0 \mid \texttt{lift}\ e
\end{array}
\tag{5.3}
$$

Les expressions soulignées sont dynamiques, ou du second niveau, tandis que les autres sont statiques, ou du premier niveau. `lift` c transforme une constante du premier ordre c du premier niveau en une constante du second niveau. Afin d'être reconnus par notre évaluateur partiel, nommé `Pev`, ces termes doivent être codé par une fonction $\llcorner.\lrcorner$ qui construit l'arbre syntaxique correspondant à un terme à deux niveaux. L'évaluation partielle d'une expression e est alors définie par

$$\texttt{Pev}\ \llcorner e\lrcorner\ \rho \tag{5.4}$$

$$
\begin{array}{rcl}
\texttt{Pev}\ \llcorner c \lrcorner\ \rho & = & c \\
\texttt{Pev}\ \llcorner x \lrcorner\ \rho & = & \rho(x) \\
\texttt{Pev}\ \llcorner \texttt{fun}\ x\ \texttt{=>}\ e \lrcorner\ \rho & = & \texttt{fun}\ v\ \texttt{=>}\ \texttt{Pev}\ \llcorner e\{x \leftarrow v\} \lrcorner\ \rho \\
\texttt{Pev}\ \llcorner \texttt{rec}\ f\ x\ \texttt{=>}\ e \lrcorner\ \rho & = & \texttt{fun}\ v\ \texttt{=>}\ \texttt{Pev}\ \llcorner e\{x \leftarrow v\}\{f \leftarrow \texttt{rec}\ f\ x\ \texttt{=>}\ e\} \lrcorner\ \rho \\
\texttt{Pev}\ \llcorner (e_0\ \texttt{@}\ e_1) \lrcorner\ \rho & = & (\texttt{Pev}\ \llcorner e_0 \lrcorner\ \rho)\ \texttt{@}\ (\texttt{Pev}\ \llcorner e_1 \lrcorner\ \rho) \\
\texttt{Pev}\ \llcorner \texttt{if}\ e_0\ e_1\ e_2 \lrcorner\ \rho & = & \texttt{if}\ (\texttt{Pev}\ \llcorner e_0 \lrcorner\ \rho)\ (\texttt{Pev}\ \llcorner e_1 \lrcorner\ \rho)\ (\texttt{Pev}\ \llcorner e_2 \lrcorner\ \rho) \\
\texttt{Pev}\ \llcorner \texttt{lift}\ c \lrcorner\ \rho & = & \ulcorner c \urcorner \\
\texttt{Pev}\ \llcorner \texttt{channel()} \lrcorner\ \rho & = & \texttt{channel()} \\
\texttt{Pev}\ \llcorner \texttt{fork}\ e_0 \lrcorner\ \rho & = & \texttt{fork}\ (\texttt{Pev}\ \llcorner e_0 \lrcorner\ \rho) \\
\texttt{Pev}\ \llcorner \texttt{receive}\ e_0 \lrcorner\ \rho & = & \texttt{receive}\ (\texttt{Pev}\ \llcorner e_0 \lrcorner\ \rho) \\
\texttt{Pev}\ \llcorner \texttt{send}\ e_0\ e_1 \lrcorner\ \rho & = & \texttt{send}\ (\texttt{Pev}\ \llcorner e_0 \lrcorner\ \rho)\ (\texttt{Pev}\ \llcorner e_1 \lrcorner\ \rho) \\
 & & \\
\texttt{Pev}\ \llcorner \underline{c} \lrcorner\ \rho & = & \ulcorner c \urcorner \\
\texttt{Pev}\ \llcorner \underline{\texttt{fun}}\ x\ \texttt{=>}\ e \lrcorner\ \rho & = & \texttt{let}\ nvar = \text{NEWVAR}\ \texttt{in} \\
 & & \quad \text{BUILD-FUN}(nvar, \texttt{Pev}\ \llcorner e\{x \leftarrow nvar\} \lrcorner\ \rho) \\
\texttt{Pev}\ \llcorner \underline{\texttt{rec}}\ f\ x\ \texttt{=>}\ e \lrcorner\ \rho & = & \text{BUILD-REC}(f, x, \texttt{Pev}\ \llcorner e \lrcorner\ \rho) \\
\texttt{Pev}\ \llcorner (e_0\ \underline{\texttt{@}}\ e_1) \lrcorner\ \rho & = & \text{BUILD-APP}(\texttt{Pev}\ \llcorner e_0 \lrcorner\ \rho, \texttt{Pev}\ \llcorner e_1 \lrcorner\ \rho) \\
\texttt{Pev}\ \llcorner \underline{\texttt{if}}\ e_0\ e_1\ e_2 \lrcorner\ \rho & = & \text{BUILD-IF}(\texttt{Pev}\ \llcorner e_0 \lrcorner\ \rho, \texttt{Pev}\ \llcorner e_1 \lrcorner\ \rho, \texttt{Pev}\ \llcorner e_2 \lrcorner\ \rho) \\
\texttt{Pev}\ \llcorner \underline{\texttt{channel()}} \lrcorner\ \rho & = & \ulcorner \texttt{channel()} \urcorner \\
\texttt{Pev}\ \llcorner \underline{\texttt{fork}}\ e_0 \lrcorner\ \rho & = & \text{BUILD-FORK}(\texttt{Pev}\ \llcorner e_0 \lrcorner\ \rho) \\
\texttt{Pev}\ \llcorner \underline{\texttt{receive}}\ e_0 \lrcorner\ \rho & = & \text{BUILD-RCV}(\texttt{Pev}\ \llcorner e_0 \lrcorner\ \rho) \\
\texttt{Pev}\ \llcorner \underline{\texttt{send}}\ e_0\ e_1 \lrcorner\ \rho & = & \text{BUILD-SEND}(\texttt{Pev}\ \llcorner e_0 \lrcorner\ \rho, \texttt{Pev}\ \llcorner e_1 \lrcorner\ \rho)
\end{array}
$$

FIG. 5.3 – **Principe de fonctionnement de l'évaluateur partiel.**

où ρ est un environnement accessible par tous les processus, semblable à celui utilisé par le méta-évaluateur `Eval`. Notons que nous ne définissons qu'un seul niveau de variables. Afin d'assurer le bon fonctionnement de `Pev` lorsqu'une variable x de valeur inconnue est rencontrée, nous enrichissons l'environnement ρ par $\rho(x) = \ulcorner x \urcorner$. De cette manière, `Pev` construit un morceau de code résiduel pour les variables dont la valeur n'est pas connue au moment de l'évaluation partielle.

Le fonctionnement de l'évaluateur partiel est décrit par les règles de la figure 5.3 et est fondé sur une sémantique par substitution identique à celle décrite pour le méta-interpréteur. Pour les expressions du premier niveau, `Pev` effectue les mêmes opérations que `Eval`, et pour celles du second, les sous-expressions sont évaluées partiellement et un terme résiduel est construit.

Les fonctions BUILD-FUN, etc, sont des fonctions auxiliaires utilisées pour la construction des termes résiduels. Elles sont définies dans l'environnement ρ.

`Pev` permet d'exécuter au moment de l'évaluation partielle les communications statiques d'un programme. Celles-ci sont déterminées par une analyse des temps de liaison respectant les conditions établies dans le chapitre 3. Nous présentons une BTA pour Concurrent ML dans la section 5.3. Auparavant, nous achevons cette section en illustrant le fonctionnement de `Pev` sur deux programmes faisant intervenir des communications

statiques et dynamiques.

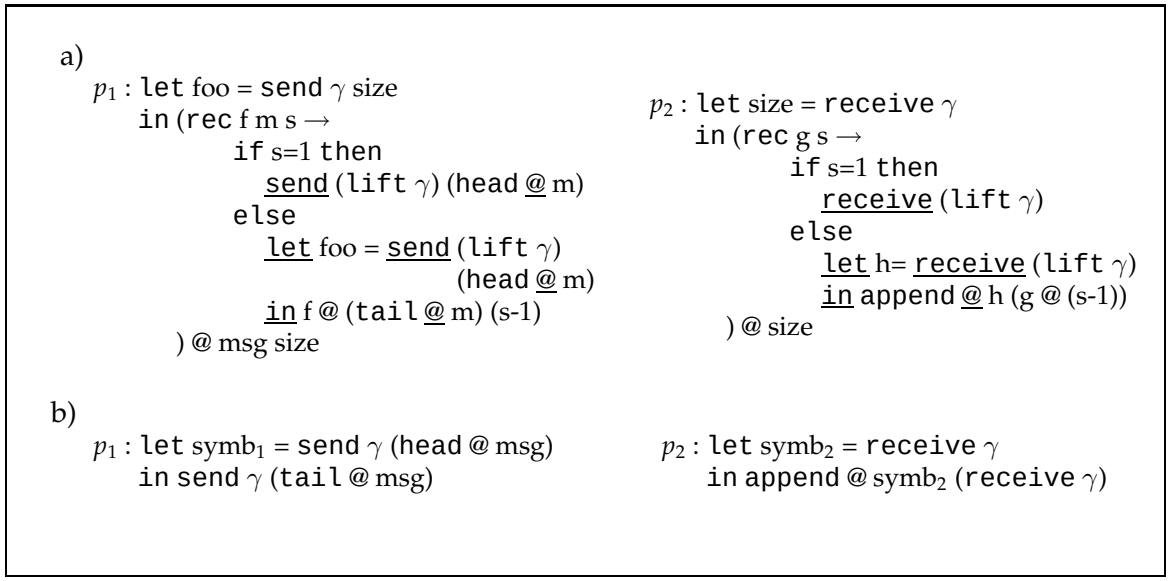


FIG. 5.4 – **Découpage en paquets de messages de taille connue et de contenu inconnu. a) Version annotée du programme. b) Résultat de la spécialisation pour size= 2.**

Le programme de la figure 5.4 décrit un système composé de deux processus se transmettant un message découpé en paquets. La taille des paquets est supposée constante et leur nombre dépend de la taille du message. Le processus p_1 transmet tout d'abord au processus p_2 la taille du message (supposée, par simplicité, égale au nombre de paquets) puis construit et envoie les différents paquets. Le processus p_2 reçoit au cours d'une première communication la taille du message, réalise le nombre correspondant de réceptions de paquets, et enfin reconstitue le message.

Nous souhaitons spécialiser ce programme pour le cas particulier dans lequel la taille du message est connue au moment de l'évaluation partielle mais pas son contenu. Les annotations correspondantes du programme sont données dans la figure 5.4 a) et nous supposons que la taille du message est égale à 2. La figure 5.4 b) montre le programme résiduel produit par notre évaluateur partiel. La première communication concernant la taille du message a été exécutée et les boucles réalisant les émissions et les réceptions de paquets ont été déroulées.

Notre second exemple concerne le programme de la figure 5.5 qui décrit un système composé de deux processus échangeant un message. Lorsque le message est reçu, une vérification est effectuée afin de détecter si celui-ci est cohérent. Lorsque ce test échoue, ce qui correspond à une erreur de transmission, un message est envoyé à l'émetteur pour lui indiquer de recommencer la transmission. Dans le cas contraire, un acquittement est envoyé à l'émetteur et les deux processus terminent leur exécution.

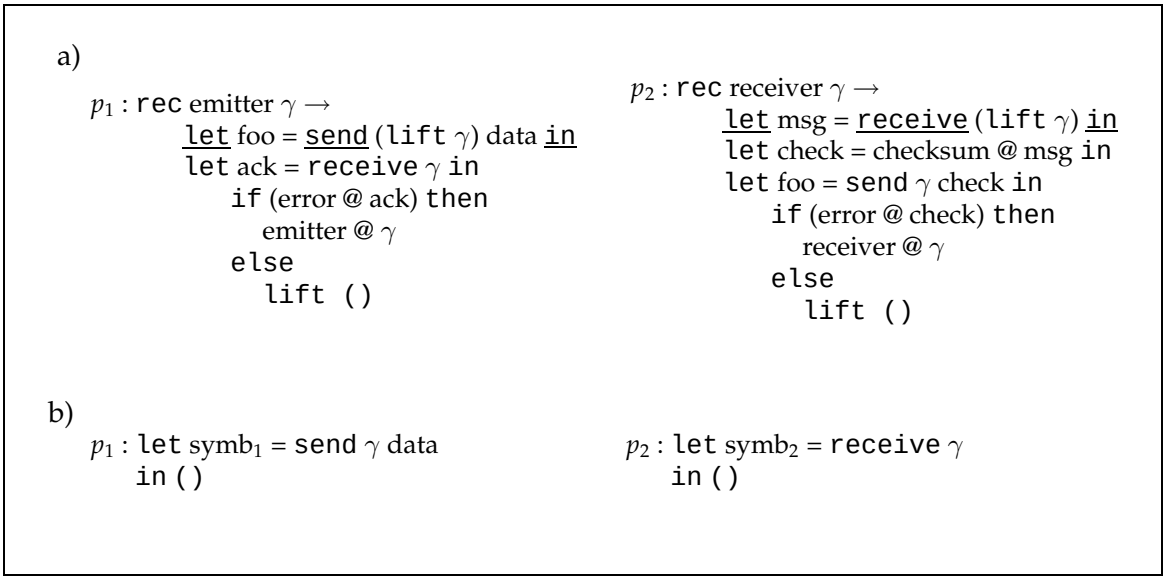


FIG. 5.5 – **Spécialisation d'un protocole de communication avec gestion des erreurs de transmission . a) Version annotée du programme. b) Résultat de la spécialisation dans le cas où l'on suppose qu'il n'y a pas d'erreur lors des transmissions.**

Nous souhaitons spécialiser ce protocole pour un réseau particulier dans lequel nous supposons qu'il n'y a pas d'erreur lors des transmissions. Pour cela, nous supposons que la fonction `checksum`, qui vérifie la cohérence des messages reçus, est statique est indique toujours que le message est correct. Par évaluation partielle, nous obtenons un programme dans lequel les communications relatives aux acquittements ont été supprimées. La figure 5.5 a) présente une version annotée du programme, et la figure 5.5 b) montre le programme spécialisé obtenu.

5.3 Analyse des temps de liaisons

Dans cette section, nous présentons une analyse des temps de liaison permettant d'étager des programmes écrits dans le langage défini par l'équation (5.1). Cette BTA utilise le cache abstrait $\widehat{C}$ résultant d'une analyse de flot de contrôle de e ainsi qu'une fonction $\widehat{\mathcal{T}}$, calculée à partir des résultats de la CFA, et indiquant la topologie des communications. La BTA produit une version à deux niveaux des programmes fournis en entrée, écrite dans le langage de l'équation (5.3).

5.3.1 Analyse des expressions séquentielles

L'analyse des expressions séquentielles est similaire à celle que Bondorf et Jorgensen ont proposé en se fondant sur les résultats d'une CFA (BJ93).

Comme au chapitre 4, une étiquette unique est attachée à chaque instruction de l'expression e à analyser. La grammaire de l'équation (5.1) se réécrit alors

$$\begin{aligned} e^l \quad ::= \quad & v^l \mid x^l \mid (\texttt{fun}\ x^{l_1} \texttt{=>}\ e_0^{l_0})^l \mid (\texttt{rec}\ f^{l_1}\ x^{l_2} \texttt{=>}\ e_0^{l_0})^l \mid (e_0^{l_0} \texttt{@}\ e_1^{l_1})^l \mid (\texttt{if}\ e_0^{l_0}\ e_1^{l_1}\ e_2^{l_2})^l \\ & \mid \texttt{channel()}^l \mid (\texttt{fork}\ e_0^{l_0})^l \mid (\texttt{send}\ e_0^{l_0}\ e_1^{l_1})^l \mid (\texttt{receive}\ e_0^{l_0})^l \end{aligned} \tag{5.5}$$

Le treillis T utilisé pour l'analyse, dont les éléments sont ordonnés par $\sqsubseteq$, est défini à la figure 5.6. Il correspond à celui utilisé par Bondorf et Jorgensen (BJ93). S est la valeur abstraite associée à une valeur statique du premier ordre, $(\text{CL}_n)_n$ est une famille de valeurs abstraites dans laquelle CL_n correspond à la valeur abstraite associée à une fonction statique d'arité n, et D est la valeur abstraite associée aux expressions dynamiques.

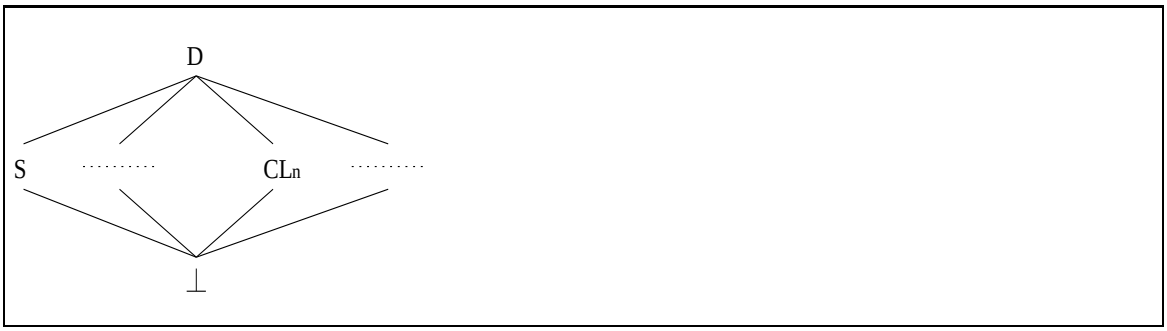


FIG. 5.6 – **Treillis T utilisé pour l'analyse des temps de liaison.**

Nous définissons d'abord les conditions qu'un quadruplet $(\widehat{\mathcal{B}}, \widehat{\mathcal{X}}, \widehat{\mathcal{C}}, \widehat{\mathcal{E}})$ doit satisfaire pour définir une analyse des temps de liaison correcte pour une expression e^l, puis nous nous intéressons à l'analyse d'un groupe de processus. $\widehat{\mathcal{B}}$, $\widehat{\mathcal{X}}$, $\widehat{\mathcal{C}}$ et $\widehat{\mathcal{E}}$ sont définis comme suit.

- $\widehat{\mathcal{B}}$: LAB $\rightarrow$ T est un cache qui associe à chaque étiquette l du programme un temps de liaison *interne*, représenté par une valeur du treillis T, de manière à indiquer si *le constructeur* d'étiquette l est statique ou dynamique,
- $\widehat{\mathcal{X}}$: LAB $\rightarrow$ T associe à chaque étiquette l un temps de liaison *externe*, représenté par une valeur du treillis T, indiquant si *le résultat* de l'évaluation partielle de l'expression e^l est statique ou dynamique,
- $\widehat{\mathcal{C}}$: LAB $\rightarrow \wp$(LAB) est le cache abstrait résultant d'une analyse de flot de contrôle de l'expression. Dans ce qui suit, nous utilisons le cache $\widehat{\mathcal{C}}$ produit par l'analyse décrite au chapitre 4, mais les résultats d'une autre analyse, telle que celle décrite à la section 2.2.2, peuvent être utilisés. La précision de l'analyse des temps de liaison sera cependant dépendante de celle utilisée pour le calcul du flot de contrôle,
- $\widehat{\mathcal{E}}$: ID $\rightarrow$ T est un environnement qui associe aux variables d'une expression une valeur abstraite appartenant au treillis T.

$\widehat{\mathcal{B}}(l)$ et $\widehat{\mathcal{X}}(l)$ sont deux temps de liaison associés au même point d'étiquette l. Cette distinction, utilisée par Henglein (Hen91) et par Bondorf et Jorgensen (BJ93) est nécessaire pour différencier les expressions représentant une opération statique mais dont le résultat est dynamique. Dans ce cas, $\widehat{\mathcal{X}}(l)$ sera dynamique mais pas $\widehat{\mathcal{B}}(l)$. Lorsque nous étageons un programme à partir des résultats de son analyse des temps de liaison, nous prenons la version dynamique de l'opérateur le plus externe d'une expression e^l lorsque $\widehat{\mathcal{B}}(l)$ est dynamique et nous prenons la version statique dans les autres cas.

Dans la définition de l'analyse, les relations devant être satisfaites entre les éléments de $\widehat{\mathcal{B}}$, $\widehat{\mathcal{X}}$ et $\widehat{\mathcal{E}}$ sont exprimées à l'aide des opérateurs suivants.

- $v_1 \sqsubseteq v_2$ indique que la valeur abstraite v_1 doit être inférieure à la valeur abstraite v_2 dans le treillis T,
- $v_1 = v_2$ indique que les valeurs abstraites de v_1 et v_2 doivent être égales dans le treillis T,
- $v_1 \triangleright v_2$ indique que si la valeur abstraite v_1 est dynamique alors v_2 doit l'être aussi. On a $(v_1 \triangleright v_2) \iff (v_1 = \mathrm{D} \Rightarrow v_2 = \mathrm{D})$,
- $v_1 \rightsquigarrow v_2$ est une contrainte pour les "lift". On a $(v_1 \rightsquigarrow v_2) \iff (v_1 = \bot) \vee (v_1 = \mathrm{S} \wedge v_2 = \mathrm{D}) \vee (v_1 = v_2)$. $v_1 \rightsquigarrow v_2$ est satisfaite si v_1 et v_2 sont égaux, ou si $v_1 = \bot$, ou enfin si $v_1 = \mathrm{S}$ et $v_2 = \mathrm{D}$. Ce dernier cas correspond à une constante du premier ordre statique devant être étagée au second niveau pour produire un morceau de programme résiduel en utilisant l'opérateur `lift`. Par contre, ce mécanisme n'étant pas utilisé pour les constantes d'ordre supérieur, $\mathrm{CL}_n \rightsquigarrow \mathrm{D}$ est toujours faux.

Dans la figure 5.7, nous donnons les conditions qu'un quadruplet $(\widehat{\mathcal{B}}, \widehat{\mathcal{X}}, \widehat{\mathcal{C}}, \widehat{\mathcal{E}})$ doit satisfaire pour analyser correctement une expression e^l, ce qui est noté

$$\widehat{\mathcal{B}}, \widehat{\mathcal{X}}, \widehat{\mathcal{C}}, \widehat{\mathcal{E}} \vdash_{\mathrm{BTA}} e^l \tag{5.6}$$

5.3.2 Analyse des synchronisations

Ces contraintes doivent être complétées pour tenir compte des valeurs abstraites des paramètres d'entrée, du fait que la spécialisation doit toujours produire un programme résiduel et pour associer des valeurs abstraites correctes aux points de réception.

Tout d'abord, nous introduisons une fonction

$$\widehat{\mathcal{T}} \ : \ \mathrm{LAB} \rightarrow \wp(\mathrm{LAB}) \tag{5.7}$$

indiquant l'ensemble des points d'émission pouvant éventuellement se synchroniser avec une réception donnée et, réciproquement, l'ensemble des points de réception pouvant éventuellement se synchroniser avec une émission donnée. Cette fonction est calculée à

$$\widehat{\mathcal{B}},\widehat{\mathcal{X}},\widehat{\mathcal{C}},\widehat{\mathcal{E}} \vdash_{\text{BTA}} c^l \iff \text{S} \sqsubseteq \widehat{\mathcal{B}}(l) \qquad \widehat{\mathcal{B}},\widehat{\mathcal{X}},\widehat{\mathcal{C}},\widehat{\mathcal{E}} \vdash_{\text{BTA}} x^l \iff \widehat{\mathcal{E}}(x) = \widehat{\mathcal{B}}(l)$$

$$\widehat{\mathcal{B}},\widehat{\mathcal{X}},\widehat{\mathcal{C}},\widehat{\mathcal{E}} \vdash_{\text{BTA}} (\texttt{fun}\ x^{l_1}\ \texttt{=>}\ e_0^{l_0})^l \iff \begin{cases} \text{CL}_1 \sqsubseteq \widehat{\mathcal{B}}(l),\ \widehat{\mathcal{B}}(l) \triangleright \widehat{\mathcal{X}}(l_0), \widehat{\mathcal{B}}(l) \triangleright \widehat{\mathcal{B}}(l_1) \\ \widehat{\mathcal{B}},\widehat{\mathcal{X}},\widehat{\mathcal{C}},\widehat{\mathcal{E}}[x \mapsto \widehat{\mathcal{B}}(l_1))] \vdash_{\text{BTA}} e_0^{l_0} \end{cases}$$

$$\widehat{\mathcal{B}},\widehat{\mathcal{X}},\widehat{\mathcal{C}},\widehat{\mathcal{E}} \vdash_{\text{BTA}} (\texttt{rec}\ f^{l_1}\ x^{l_2}\ \texttt{=>}\ e_0^{l_0})^l \iff \begin{cases} \text{CL}_1 \sqsubseteq \widehat{\mathcal{B}}(l),\ \widehat{\mathcal{B}}(l) \triangleright \widehat{\mathcal{X}}(l_0), \widehat{\mathcal{B}}(l) \triangleright \widehat{\mathcal{B}}(l_2),\ \widehat{\mathcal{B}}(l) = \widehat{\mathcal{B}}(l_1) \\ \widehat{\mathcal{B}},\widehat{\mathcal{X}},\widehat{\mathcal{C}},\widehat{\mathcal{E}}[f \mapsto \widehat{\mathcal{B}}(l_1), x \mapsto \widehat{\mathcal{B}}(l_2)] \vdash_{\text{BTA}} e_0^{l_0} \end{cases}$$

$$\widehat{\mathcal{B}},\widehat{\mathcal{X}},\widehat{\mathcal{C}},\widehat{\mathcal{E}} \vdash_{\text{BTA}} (e_0^{l_0}\ e_1^{l_1})^l \iff \begin{cases} \widehat{\mathcal{B}},\widehat{\mathcal{X}},\widehat{\mathcal{C}},\widehat{\mathcal{E}} \vdash_{\text{BTA}} e_0^{l_0},\ \widehat{\mathcal{B}},\widehat{\mathcal{X}},\widehat{\mathcal{C}},\widehat{\mathcal{E}} \vdash_{\text{BTA}} e_1^{l_1} \\ \text{CL}_1 \sqsubseteq \widehat{\mathcal{X}}(l_0),\ \widehat{\mathcal{X}}(l_0) \triangleright \widehat{\mathcal{B}}(l) \\ \forall l_2 \in \widehat{\mathcal{C}}(l_0)\ :\ (\texttt{fun}\ x^{l_4}\ \texttt{=>}\ e_3^{l_3})^{l_2} \in \text{prg}, \\ \quad \widehat{\mathcal{X}}(l_3) = \widehat{\mathcal{X}}(l),\ \widehat{\mathcal{X}}(l_2) = \widehat{\mathcal{X}}(l_1) \\ \forall l_2 \in \widehat{\mathcal{C}}(l_0)\ :\ (\texttt{rec}\ f^{l_4}\ x^{l_5}\ \texttt{=>}\ e_3^{l_3})^{l_2} \in \text{prg}, \\ \quad \widehat{\mathcal{X}}(l_3) = \widehat{\mathcal{X}}(l),\ \widehat{\mathcal{X}}(l_3) = \widehat{\mathcal{X}}(l_1) \end{cases}$$

$$\widehat{\mathcal{B}},\widehat{\mathcal{X}},\widehat{\mathcal{C}},\widehat{\mathcal{E}} \vdash_{\text{BTA}} (\texttt{if}\ e_0^{l_0}\ e_1^{l_1}\ e_2^{l_2})^l \iff \begin{cases} \widehat{\mathcal{B}},\widehat{\mathcal{X}},\widehat{\mathcal{C}},\widehat{\mathcal{E}} \vdash_{\text{BTA}} e_i^{l_i},\ 0 \leq i \leq 2 \\ \widehat{\mathcal{X}}(l) = \widehat{\mathcal{X}}(l_1),\ \widehat{\mathcal{X}}(l) = \widehat{\mathcal{X}}(l_2),\ \widehat{\mathcal{X}}(l_0) \triangleright \widehat{\mathcal{B}}(l),\ \text{S} \sqsubseteq \widehat{\mathcal{X}}(l_0) \end{cases}$$

$$\widehat{\mathcal{B}},\widehat{\mathcal{X}},\widehat{\mathcal{C}},\widehat{\mathcal{E}} \vdash_{\text{BTA}} \texttt{channel()}^l \iff \text{S} \sqsubseteq \widehat{\mathcal{B}}(l) \qquad \widehat{\mathcal{B}},\widehat{\mathcal{X}},\widehat{\mathcal{C}},\widehat{\mathcal{E}} \vdash_{\text{BTA}} k^l \iff \text{S} \sqsubseteq \widehat{\mathcal{B}}(l)$$

$$\widehat{\mathcal{B}},\widehat{\mathcal{X}},\widehat{\mathcal{C}},\widehat{\mathcal{E}} \vdash_{\text{BTA}} (\texttt{fork}\ e_0^{l_0})^l \iff \widehat{\mathcal{B}},\widehat{\mathcal{X}},\widehat{\mathcal{C}},\widehat{\mathcal{E}} \vdash_{\text{BTA}} e_0^{l_0},\ \text{S} \sqsubseteq \widehat{\mathcal{B}}(l),\ \widehat{\mathcal{X}}(l) \triangleright \widehat{\mathcal{X}}(l_0),\ \widehat{\mathcal{B}}(l) = \widehat{\mathcal{X}}(l)$$

$$\widehat{\mathcal{B}},\widehat{\mathcal{X}},\widehat{\mathcal{C}},\widehat{\mathcal{E}} \vdash_{\text{BTA}} (\texttt{send}\ e_0^{l_0}\ e_1^{l_1})^l \iff \begin{cases} \widehat{\mathcal{B}},\widehat{\mathcal{X}},\widehat{\mathcal{C}},\widehat{\mathcal{E}} \vdash_{\text{BTA}} e_0^{l_0},\ \widehat{\mathcal{B}},\widehat{\mathcal{X}},\widehat{\mathcal{C}},\widehat{\mathcal{E}} \vdash_{\text{BTA}} e_1^{l_1} \\ \widehat{\mathcal{B}}(l_0) \sqsubseteq \widehat{\mathcal{B}}(l),\ \widehat{\mathcal{B}}(l_1) \sqsubseteq \widehat{\mathcal{B}}(l),\ \widehat{\mathcal{B}}(l) = \widehat{\mathcal{X}}(l) \end{cases}$$

$$\widehat{\mathcal{B}},\widehat{\mathcal{X}},\widehat{\mathcal{C}},\widehat{\mathcal{E}} \vdash_{\text{BTA}} (\texttt{receive}\ e_0^{l_0})^l \iff \left\{ \widehat{\mathcal{B}},\widehat{\mathcal{X}},\widehat{\mathcal{C}},\widehat{\mathcal{E}} \vdash_{\text{BTA}} e_0^{l_0},\ \widehat{\mathcal{B}}(l_0) \sqsubseteq \widehat{\mathcal{B}}(l),\ \widehat{\mathcal{B}}(l) = \widehat{\mathcal{X}}(l) \right.$$

FIG. 5.7 – **Spécification de l'analyse des temps de liaison pour les expressions séquentielles.**

partir des résultats de l'analyse de flot de contrôle. Dans la suite de ce chapitre, nous la calculons à partir des résultats de la CFA décrite au chapitre 4, en utilisant les noms des états produits de l'automate réduit[1]. Cependant, il serait par exemple possible d'utiliser les résultats de la CFA décrite au chapitre 2, même si plus d'instructions seraient annotées dynamiques dans ce dernier cas.

Nous traitons les communications comme suit. Une émission $s = (\texttt{send}\ e_0^{l_0}\ e_1^{l_1})^{l_s}$ est statique si son sujet $e_0^{l_0}$ et son objet $e_1^{l_1}$ le sont, et si les réceptions susceptibles de recevoir la valeur transmise par s, dont la liste est fournie par $\widehat{\mathcal{T}}$, le sont. Elle est dynamique dans les autres cas. Naturellement, une réception $r = (\texttt{receive}\ e_2)^{l_2}$ est dynamique lorsque son sujet $e_2^{l_2}$ l'est. Dans le cas contraire, r est statique si toutes les émissions s susceptibles de se synchroniser avec r sont statiques.

[1]Plus précisément, les communications sont données par les points du domaine de définition de la fonction $\mathbb{L}$.

Définition 41 (Analyse des temps de liaison) *Un quintuplet* $(\widehat{\mathcal{B}}, \widehat{\mathcal{X}}, \widehat{\mathcal{C}}, \widehat{\mathcal{T}}, \widehat{\mathcal{E}})$ *est une analyse des temps de liaison correcte pour une expression* e^l*, ce qui est noté*

$$\widehat{\mathcal{B}}, \widehat{\mathcal{X}}, \widehat{\mathcal{C}}, \widehat{\mathcal{T}}, \widehat{\mathcal{E}} \models_{\text{BTA}} e^l$$

si $\widehat{\mathcal{C}}$ *est un cache abstrait correct produit par une analyse de flot de contrôle de* e^l*, si* $\widehat{\mathcal{T}}$ *est une approximation correcte des communications réalisées par* e^l*, et si les conditions suivantes sont vérifiée.*

(i) On a la propriété $\widehat{\mathcal{B}}, \widehat{\mathcal{X}}, \widehat{\mathcal{C}}, \widehat{\mathcal{E}} \vdash_{\text{BTA}} e^l$*.*

(ii) Pour chaque variable d'entrée x*, on a une contrainte* $\text{S} \sqsubseteq \widehat{\mathcal{E}}(x)$ *ou* $\text{D} \sqsubseteq \widehat{\mathcal{E}}(x)$*, selon que* x *est statique ou dynamique.*

(iii) Pour toute sous-expression $e_0^{l_0}$ *du programme, on a la contrainte* $\widehat{\mathcal{B}}(l_0) \rightsquigarrow \widehat{\mathcal{X}}(l_0)$ *permettant de résidualiser les constantes du premier ordre lorsque cela est nécessaire.*

(iv) On a la contrainte $\text{D} \sqsubseteq \widehat{\mathcal{X}}(l)$ *indiquant que le résultat global de l'évaluation partielle doit être un morceau de code.*

(v) Pour toute réception $(\texttt{receive}\ e_0^{l_0})^{l_r}$ *et pour toute émission d'étiquette* l_s *telle que* $l_s \in \widehat{\mathcal{T}}(l_r)$ *on a* $\widehat{\mathcal{B}}(l_s) = \widehat{\mathcal{B}}(l_r)$*.* □

Nous concluons cette section en effectuant quelques remarques concernant le fonctionnement de cette analyse. Tout d'abord, comme nous l'avons déjà indiqué, la précision de la BTA dépend de celle de la fonction $\widehat{\mathcal{T}}$ calculée à partir des résultats de la CFA. Considérons le programme de la figure 5.8 a) qui représente deux processus réalisant successivement une communication statique et une communication dynamique sur le même canal γ.

a)

$$\texttt{let}\ p_1 = \texttt{fork} \left(\begin{array}{l} \texttt{let}\ s_1 = \texttt{send}\ \gamma\ S\ \texttt{in} \\ \texttt{let}\ s_2 = \texttt{send}\ \gamma\ D\ \texttt{in} \\ \ldots \end{array} \right) \texttt{in} \left(\begin{array}{l} \texttt{let}\ r_1 = \texttt{receive}\ \gamma\ \texttt{in} \\ \texttt{let}\ r_2 = \texttt{receive}\ \gamma\ \texttt{in} \\ \ldots \end{array} \right)$$

b)

$$\texttt{let}\ p_1 = \texttt{fork} \left(\begin{array}{l} \texttt{let}\ s_1 = \texttt{send}\ \gamma\ S\ \texttt{in} \\ \texttt{let}\ s_2 = \underline{\texttt{send}}\ \gamma\ D\ \texttt{in} \\ \ldots \end{array} \right) \texttt{in} \left(\begin{array}{l} \texttt{let}\ r_1 = \texttt{receive}\ \gamma\ \texttt{in} \\ \texttt{let}\ r_2 = \underline{\texttt{receive}}\ \gamma\ \texttt{in} \\ \ldots \end{array} \right)$$

FIG. 5.8 – **a) Programme réalisant deux communications sur le canal γ. La première valeur envoyée est supposée statique et la seconde dynamique. b) Annotations produites par la BTA pour ce programme.**

Les résultats produits par notre BTA pour ce programme sont donnés dans la figure

5.8 b). Nous pouvons constater que seule la seconde communication est annotée dynamique. Ceci est dû au fait que les informations topologiques permettent, dans ce cas, de déterminer exactement les paires d'émetteurs et de récepteurs.

Notre seconde remarque concerne les points d'émission et de réception pour lesquels le sujet et l'objet (dans le cas des réceptions) sont statiques mais qui n'ont pas de correspondant dans le programme. Par exemple, considérons une émission $(\texttt{send}\ e_0^{l_0}\ e_1^{l_1})^{l_s}$ telle que e_0 et e_1 sont statiques et $\widehat{\mathcal{T}}(l_s) = \varnothing$. De telles instructions de communication seront annotées statiques par l'analyse, induisant un blocage de l'évaluateur partiel. Ceci est acceptable car le programme en question aurait été bloqué de la même manière lors d'une exécution classique. En effet, dans notre langage, les communications avec un processus non défini dans le programme ne sont possibles, puisque le seul moyen de définir un canal est l'instruction `channel()` qui crée un nouveau nom inconnu de l'extérieur.

Cette restriction pourrait être levée en autorisant la définition de valeurs de type canal, éventuellement connues hors du programme que nous traitons. Dans ce cas il serait nécessaire que la valeur abstraite produite par la CFA pour de tels canaux soit l'ensemble de tous les canaux, et que la BTA associe toujours une valeur dynamique aux communications utilisant de tels canaux.

Ensuite, remarquons que, parmi les contraintes énumérées dans la figure 5.7, nous avons $\widehat{\mathcal{B}}(l_s) = \widehat{\mathcal{X}}(l_s)$ et $\widehat{\mathcal{B}}(l_r) = \widehat{\mathcal{X}}(l_r)$ pour les émissions et réceptions d'étiquettes respectives l_s et l_r. Ces contraintes ont pour but de rendre une instruction de communication dynamique lorsqu'elle apparaît dans un contexte dynamique, même si le sujet et l'objet sont statiques. Par exemple, considérons le programme de la figure 5.9 a), dans lequel une réception se synchronise avec un émission parmi deux possibles, en fonction d'une condition supposée dynamique. Le sujet et l'objet des émissions sont statiques, mais il est nécessaire de reporter l'exécution de la communication afin de connaître la valeur de la condition. L'analyse décrite dans cette section produit sur cet exemple les annotations données dans la figure 5.9 b).

Les créations dynamiques de processus sont traitées de manière similaire aux instructions de communication. Un `fork` est statique sauf s'il apparaît dans un contexte dynamique (contraintes $\mathrm{S} \sqsubseteq \widehat{\mathcal{B}}(l)$ et $\widehat{\mathcal{B}}(l) = \widehat{\mathcal{X}}(l)$). Cela permet de geler la création de processus lorsque celle-ci dépend d'une condition dynamique (situation similaire aux émissions présentes dans le programme de la figure 5.9) ou apparaissent dans une boucle dynamique. Par ailleurs, les instructions de communication présentes dans le code d'un processus créé par un `fork` dynamique doivent être dynamiques. Ceci est assuré par le fait que le contexte de ces dernières est dynamique (contrainte $\widehat{\mathcal{X}}(l) \rhd \widehat{\mathcal{X}}(l_0)$).

Enfin, remarquons que, comme cela a été fait pour la CFA du chapitre 4, il est nécessaire de générer les contraintes relatives à la BTA à partir de la syntaxe d'un programme

a)

$$\texttt{let } p_1 = \texttt{fork} \left(\begin{array}{l} \texttt{if } \textit{cond} \texttt{ then send } \gamma\ 0 \\ \texttt{else send } \gamma\ 1 \end{array} \right) \texttt{ in receive } \gamma$$

b)

$$\texttt{let } p_1 = \texttt{fork} \left(\begin{array}{l} \underline{\texttt{if}}\ \textit{cond}\ \underline{\texttt{then}}\ \underline{\texttt{send}}\ \gamma\ 0 \\ \underline{\texttt{else}}\ \underline{\texttt{send}}\ \gamma\ 1 \end{array} \right) \texttt{ in } \underline{\texttt{receive}}\ \gamma$$

FIG. 5.9 – **a) Programme réalisant une communication parmi deux possibles, en fonction d'une condition dynamique. b) Annotations produites par la BTA pour ce programme.**

et de résoudre le système correspondant. Tout comme au chapitre 4, les contraintes définies pour l'analyse des temps de liaison sont monotones et il est possible de trouver la plus petite solution à de tels systèmes en utilisant un algorithme similaire à celui donné à la section 4.4. Ces mécanismes sont présents dans notre implantation de l'analyse.

5.4 Auto-application

Dans cette section, nous décrivons les résultats obtenus par auto-application de l'évaluateur partiel `Pev` décrit au début de ce chapitre. `Pev` est écrit dans le sous-ensemble non typé du langage Concurrent ML défini par l'équation (5.1) pour lequel nous avons écrit un interpréteur dans le langage CAML light (WL99). `Pev` prend en entrée des expressions à deux niveaux codées par la fonction $\llcorner . \lrcorner$. Ces expressions sont obtenues à partir des analyses de flot de contrôle et des temps de liaison décrites au chapitre 4 et à la section 5.3 respectivement. Ces analyses ont aussi été implémentées et permettent en particulier d'annoter correctement `Pev` en vue de son auto-application.

5.4.1 Première projection

La première projection de Futamura consiste à évaluer partiellement, avec `Pev`, un interpréteur `Eval`, avec pour paramètre connu le programme p devant être interprété. On obtient ainsi un nouveau programme dans lequel les calculs liés à l'interprétation des instructions ont été supprimés. Il s'agit d'une version compilée de p. `Eval` doit être annoté afin d'indiquer à l'évaluateur partiel les instructions qui sont statiques, car dépendantes de p, et celles qui sont dynamiques, car dépendantes des paramètres de p. Nous donnons dans la figure 5.10 la version `Eval_a` annotée en vue de la première projection de Futamura du méta-interpréteur `Eval` présenté à la figure 5.2. Les opérations dynamiques sont précédées du caractère _.

```
define eval = rec eval e ->
  if (isVar? @ e) then (getEnv _@ e) else
  if (isLambda? @ e) then
    (_fun value -> (eval @ (substitute @ (fetchLambdaBody @ e)
                                         value
                                         (fetchLambdaId @ e)))) else
  if (isRec? @ e) then
    (_fun rvalue -> (eval @ (substitute @ (substitute @ (fetchRecBody @ e)
                                                        rvalue
                                                        (fetchRecArg @ e))
                                          e
                                          (fetchRecId @ e)))) else
  if (isApp? @ e) then
    ((eval @ (fetchAppFun @ e)) _@ (eval @ (fetchAppArg @ e))) else
  if (isCond? @ e) then
    _if (getBase @ (eval @ (fetchCond @ e))) _then
       (eval @ (fetchThen @ e))
    _else
      (eval @ (fetchElse @ e)) else
  if (isConst? @ e) then lift (fetchConst @ e) else
  if (isChannel? @ e) then (createNewChan _@ ()) else
  if (isFork? @ e) then (_fork (eval @ (fetchChild @ e))) else
  if (isReceive? @ e) then _receive (eval @ (fetchRecChan @ e)) else
  if (isSend? @ e) then _send (eval @ (fetchSendChan @ e))
                              (eval @ (fetchSendVal @ e))
  else buildError @ e                                                    ;
```

FIG. 5.10 – **Version annotée du méta-interpréteur.**

Pour l'évaluateur partiel `Pev`, dont le principe de fonctionnement est donné dans la figure 5.3, pour la version annoté `Eval_a` de notre méta-interpréteur, et pour le code $\ulcorner p \urcorner$ d'un programme p, nous obtenons via la première projection de Futamura une version compilée p_c de $\ulcorner p \urcorner$, syntaxiquement égale à p, à l'exception des noms de variables qui ont été redéfinis. Par exemple pour

$$p = \texttt{(fun x => (fun y => send x y))} \tag{5.8}$$

nous avons

$$\ulcorner p \urcorner = \texttt{['fun','x',['fun','y',['send',['var','x'],['var','y']]]]} \tag{5.9}$$

et

$$p_c = \texttt{(fun symb\_1 => (fun symb\_2 => send symb\_1 symb\_2))} \tag{5.10}$$

```
...
if (isLambda? @ e) then
  (_fun value -> (Pev @ (substitute @ (fetchLambdaBody @ e)
                                      value
                                      (fetchLambdaId @ e)))) else
if (isApp? @ e) then
  ((Pev @ (fetchAppFun @ e)) _@ (Pev @ (fetchAppArg @ e))) else
if (isChannel? @ e) then (createNewChan _@ ()) else
if (isFork? @ e) then (_fork (Pev @ (fetchChild @ e))) else
if (isReceive? @ e) then _receive (Pev @ (fetchRecChan @ e)) else
if (isSend? @ e) then _send (Pev @ (fetchSendChan @ e))
                            (Pev @ (fetchSendVal @ e))

if (isDLambda? @ e) then
 (_fun newVar ->
     (buildLambda _@ newVar (Pev @ (substitute @ (fetchLambdaBody @ e)
                                                 newVar
                                                 (fetchLambdaId @ e))))
 ) _@ (genSymb @ ()) else
if (isDApp? @ e) then
  (buildApp _@ (Pev @ (fetchAppFun @ e)) (Pev @ (fetchAppArg @ e))) else
if (isDChannel? @ e) then dynChanCode else
if (isDFork? @ e) then (buildFork _@ (Pev @ (fetchChild @ e))) else
if (isDReceive? @ e) then
  (buildReceive _@ (Pev @ (fetchRecChan @ e))) else
if (isDSend? @ e) then
  (buildSend _@ (Pev @ (fetchSendChan @ e))
                (Pev @ (fetchSendVal @ e)))
                                                                    ...
```

FIG. 5.11 – **Version annotée de l'évaluateur partiel.**

5.4.2 Seconde projection

La seconde projection de Futamura consiste à appliquer l'évaluateur partiel à lui-même et à un interpréteur afin d'obtenir un compilateur nommé `Comp`. L'évaluateur partiel et l'interpréteur passés en arguments doivent être annotés et codés sous forme de données, comme décrit par l'équation suivante dans laquelle `Pev_a` et `Eval_a` sont des versions annotées de l'évaluateur partiel et du méta-interpréteur.

$$\texttt{Pev}\,\llcorner\texttt{Pev\_a}\lrcorner\,\llcorner\texttt{Eval\_a}\lrcorner = \texttt{Comp} \tag{5.11}$$

`Pev_a` est donné dans la figure 5.11 et `Eval_a` est annoté de manière similaire. Une version complète de `Pev_a` est par ailleurs donnée dans l'annexe B.1. A nouveau les opérations dynamiques sont précédées du symbole _. Le compilateur `Comp` obtenu est donné dans la figure 5.12, dans laquelle certains noms de variables ont été renommés. Appliqué au code $\llcorner p \lrcorner$ d'un programme p, `Comp` produit une version compilée de $\llcorner p \lrcorner$, syntaxiquement équivalente à p, à l'exception des noms de variables qui sont modifiés.

```
define Comp = rec Comp e ->
  if (isVar? @ e) then (getEnv @ e) else
  if (isLambda? @ e) then
    ((fun Symb2 ->
       ((buildLambda @ Symb2)
                    @ (Comp @ (((substitute @ (fetchLambdaBody @ e))
                                            @ Symb2)
                                            @ (fetchLambdaId @ e)))
     )) @ (genSymb @ ())) else
  if (isRec? @ e) then
    ((fun Symb3 ->
       ((buildLambda @ Symb3)
                    @ (Comp @ (((substitute
                                  @ (((substitute @ (fetchRecBody @ e))
                                                  @ Symb3)
                                                  @ (fetchRecArg @ e)))
                                  @ e)
                                  @ (fetchRecArg @ e)))
       )) @ (genSymb @ ())) else
  if (isApp? @ e) then
    ((buildApp @ (Comp @ (fetchAppFun @ e)))
               @ (Comp @ (fetchAppArg @ e))) else
  if (isCond? @ e) then
    (((buildCond @ (Comp @ (fetchCond @ e)))
                 @ (Comp @ (fetchThen @ e)))
                 @ (Comp @ (fetchElse @ e))) else
  if (isConst? @ e) then e else
  if (isChannel? @ e) then
    ((buildApp @ createNewChan)
               @ ()) else
  if (isFork? @ e) then
    (buildFork @ (Comp @ (fetchChild @ e))) else
  if (isReceive? @ e) then (buildReceive @ (Comp @ (fetchRecChan @ e))) else
  if (isSend? @ e) then ((buildSend @ (Comp @ (fetchSendChan @ e)))
                                    @ (Comp @ (fetchSendVal @ e)))
  else (buildError @ e)                                                    ;
```

FIG. 5.12 – **Compilateur obtenu par la seconde projection de Futamura.**

Par exemple, appliqué au programme de l'équation (5.9), `Comp` produit le programme compilé p_c donné dans l'équation (5.10).

5.4.3 Troisième projection

Enfin, la troisième projection de Futamura consiste à évaluer partiellement `Pev` avec pour argument statique `Pev` lui-même, afin d'obtenir un générateur de compilateurs. Naturellement, les deux versions de l'évaluateur partiel fournies en arguments doivent être annotées comme à la figure 5.11. Ceci est résumé par l'équation (5.12).

$$\texttt{Pev}\,\llcorner\texttt{Pev\_a}\lrcorner\,\llcorner\texttt{Pev\_a}\lrcorner = \texttt{Cogen} \tag{5.12}$$

Des extraits du générateur de compilateurs obtenu sont donnés dans les figures 5.13 et 5.14. Une version complète de celui-ci est aussi présentée dans l'annexe B.2.

```
...
 if (isLambda? @ e) then
   ((fun Symb2 ->
      ((buildLambda @ Symb2)
                  @ (Cogen @ (((substitute @ (fetchLambdaBody @ e))
                                           @ Symb2)
                                           @ (fetchLambdaId @ e)))
    )) @ (genSymb @ ())) else
 if (isApp? @ e) then
   ((buildApp @ (Cogen @ (fetchAppFun @ e)))
              @ (Cogen @ (fetchAppArg @ e))) else
 if (isChannel? @ e) then ((buildApp @ createNewChan)
                                    @ ()) else
 if (isFork? @ e) then
   (buildFork @ (Cogen @ (fetchChild @ e))) else
 if (isReceive? @ e) then
   (buildReceive @ (Cogen @ (fetchRecChan @ e))) else
 if (isSend? @ e) then
   ((buildSend @ (Cogen @ (fetchSendChan @ e)))
               @ (Cogen @ (fetchSendVal @ e))) else
                                                                    ...
```

FIG. 5.13 – **Générateur de compilateurs obtenu - Partie 1.**

5.5 Conclusion

Dans ce chapitre, nous nous sommes intéressés à l'évaluation partielle et à l'analyse des temps de liaison de programmes écrits dans un noyau non typé du langage Concurrent ML. Tous deux ont été implantés et nous décrivons les résultats obtenus sur différents exemples.

La BTA utilise des informations concernant la topologie des communications réalisées par les programmes. Ces informations sont extraites des résultats produits par une analyse de flot de contrôle. Bien que n'importe quelle CFA puisse être utilisée, la précision de cette dernière influence sensiblement celle de la BTA. Nous avons supposé dans ce chapitre que la BTA utilisait les résultats produit par la CFA décrite au chapitre 4.

Par ailleurs, pour être correcte, l'analyse des temps de liaison doit respecter les critères de correction énoncés au chapitre 3 dans le cadre du π-calcul. Ces critères, qui ont pour but d'assurer l'équivalence observationelle des programmes initiaux et résiduels, interdisent l'évaluation partielle d'instructions de communication utilisant des canaux connus à l'extérieur du système de processus que l'on traite. Ces conditions sont immédiatement vérifiées pour le langage que nous considérons, car le seul moyen de définir

```
...
 if (isDLambda? @ e) then
   ((buildApp @
      ((fun Symb4 ->
         ((buildLambda @ Symb4)
                       @ ((buildApp @ ((buildApp @ buildLambda)
                                                  @ Symb4))
                                     @ (Cogen @ (((substitute
                                                    @ (fetchLambdaBody @ e))
                                                    @ Symb4)
                                                    @ (fetchLambdaId @ e))))
                 )) @ (genSymb @ ()))) @ ((buildApp @ genSymb)
                                                    @ ())) else
 if (isDApp? @ e) then
   ((buildApp @ ((buildApp @ buildApp)
                            @ (Cogen @ (fetchAppFun @ e))))
              @ (Cogen @ (fetchAppArg @ e))) else
 if (isDChannel? @ e) then buildChanCode else
 if (isDFork? @ e) then
   ((buildApp @ buildFork)
              @ (Cogen @ (fetchChild @ e))) else
 if (isDReceive? @ e) then
   ((buildApp @ buildReceive)
              @ (Cogen @ (fetchRecChan@ e))) else
 if (isDSend? @ e) then
   ((buildApp @ ((buildApp @ buildSend)
                            @ (Cogen @ (fetchSendChan @ e))))
              @ (Cogen @ (fetchSendVal @ e)))
                                                                          ...
```

FIG. 5.14 – **Générateur de compilateurs obtenu - Partie 2.**

un nom de canal consiste à utiliser l'instruction `channel()` qui crée un nouveau canal inconnu du reste du système.

L'évaluation partielle de systèmes de processus communicants est utile pour spécialiser des applications en fonction de certains contextes d'utilisation, comme le montrent les différents exemples que nous avons présentés : connaissance statique de la topologie du réseau, hypothèses sur le fonctionnement de celui-ci, ou encore informations sur la nature des messages transmis. Concernant ce dernier point, nous avons montré comment spécialiser un protocole de transmission de messages décomposés en paquets lorsque la taille du message est connue statiquement. Ceci reprend l'idée avancée par Muller et al. qui ont utilisé un évaluateur partiel pour spécialiser le protocole RPC en fonction du type des données transmises (MVM97). Cependant, le fait que l'évaluateur partiel qu'ils utilisent réduit seulement les parties séquentielles des programmes, ne permet pas d'évaluer partiellement des instructions de communication statiques (la taille du message pour reprendre l'exemple de la figure 5.4) et nécessite l'utilisation de techniques de spécialisation à l'exécution (run-time specialisation) (CN96; LL93). L'évaluation partielle

des communications telle que nous la proposons, permet d'effectuer des spécialisations similaires entièrement statiquement.

Enfin, nous avons montré que notre évaluateur partiel était compatible avec les projections de Futamura et plus généralement qu'il était possible de générer automatiquement des compilateurs et des générateurs de compilateurs à partir d'un évaluateur partiel pour des langages à base de communications synchrones. L'évaluateur partiel a été automatiquement annoté en vue de son auto-application par l'analyse des temps de liaison présentée à la section 5.3. L'implantation de langage de programmation de haut niveau est généralement une tâche complexe et la possibilité d'utiliser des générateurs de compilateurs dans ce contexte semble être une perspective intéressante.

Chapitre 6

Conclusion

Dans ce document, nous nous sommes intéressés à l'analyse statique et à l'évaluation partielle de systèmes de processus mobiles. Nous avons défini des analyses de flot de contrôle et des temps de liaison pour un langage fonctionnel concurrent ainsi que les conditions nécessaires pour garantir la correction de l'évaluation partielle des communications à travers la notion de bonne annotation des programmes écrits dans le π-calcul. Par ailleurs, nous avons présenté un méta-interpréteur et un évaluateur partiel pour le π-calcul. Bien que cet évaluateur partiel soit auto-applicable, il n'est pas compatible avec les projections de Futamura, les communications étant bloquantes dans le π-calcul. Cependant, il permet de mettre en évidence les principes inhérents à l'évaluation partielle de systèmes de processus mobiles, qui peuvent être réutilisés dans le cas d'autres langages possédant plus de constructeurs, comme nous l'avons montré pour Concurrent ML.

L'analyse de flot de contrôle présentée au chapitre 4 calcule une approximation de la topologie des communications réalisées par les programmes. Ces informations peuvent être utilisées pour améliorer la précision d'autres analyses statiques, comme nous le faisons pour l'analyse des temps de liaison définie au chapitre 5. Ces résultats pourraient être utilisés par d'autres analyses statiques, comme, par exemple, pour introduire du sous-typage dans un système de type comme celui de Concurrent ML.

Par ailleurs, cette CFA est utile par elle-même pour vérifier des propriétés de sûreté et de sécurité pour des applications distribuées, comme nous le montrons à la section 4.5. Concernant les propriétés de sécurité, il serait intéressant de reprendre l'approche proposée par Bodei et al. (BDNN98) pour le π-calcul, qui consiste à distinguer des canaux publics et des canaux privés et à définir des critères de sécurité à partir des lieux d'utilisation de ceux-ci. Ces travaux sont aussi fondés sur une CFA dont la précision est inférieure à celle de la CFA définie au chapitre 4.

Aussi, nous pensons qu'il serait possible de définir une CFA fondée sur un automate produit réduit de taille inférieure à celle de l'automate réduit présenté à la section 4.3.2.

Cette analyse serait moins précise que celle décrite dans ce document mais aussi de complexité moindre, tout en restant précise pour des schémas de communication simples.

L'analyse des temps de liaison définie au chapitre 5 utilise les résultats d'une CFA. Bien que n'importe quelle CFA puisse être utilisée, la précision de cette dernière influence celle de la BTA. Cette dernière est compatible avec les critères de bonne annotation des programmes donnés au chapitre 3 pour le π-calcul. Cela est dû au fait que seuls des noms de canaux inconnus de l'extérieur du système que nous traitons peuvent être définis dans le langage utilisé. Cette restriction pourrait être levée en ajoutant au langage un mécanisme de définition de noms de canaux partagés. Parce qu'il n'est pas possible de prédire l'utilisation faite de tels canaux à l'extérieur du système, il faudrait alors modifier la BTA de manière à ce que le temps de liaison associé à de tels noms soit toujours dynamique. Par ailleurs, la CFA devrait aussi être modifiée de manière à indiquer que n'importe quelle fonction ou nom de canal connu de l'extérieur peut être reçu lors d'une communication entre le système que nous analysons et l'extérieur.

L'évaluateur partiel présenté au chapitre 5 traite un sous-ensemble non typé du langage Concurrent ML. Celui-ci pourrait être typé en utilisant les notions de termes à plusieurs niveaux (She97; TS97) et de typage dynamique (SSPJ98), comme cela est fait dans le langage Méta ML (MS97). Nous souhaitons étendre le langage que nous avons considéré de manière à prendre en compte les autres instructions pour la gestion du parallélisme présent dans Concurrent ML, tel que l'opérateur `wrap`, similaire à la somme du π-calcul.

Les résultats présentés pour Concurrent ML sont transposables à tout autre langage utilisant le même modèle de parallélisme. Nous souhaiterions réutiliser les méthodes décrites dans ce document pour définir un évaluateur partiel capable de traiter les primitives de communication offertes par des bibliothèques telles que PVM (GBD$^+$94).

Enfin, nous avons montré qu'il était possible de générer automatiquement un compilateur et un générateur de compilateurs par auto-application de l'évaluateur partiel. Un tel générateur de compilateurs peut être utilisé de manière classique pour obtenir un compilateur pour un langage parallèle à partir d'un interpréteur. Il semble aussi possible de lui fournir en entrée un évaluateur donnant une sémantique parallèle d'un langage séquentiel. Cependant, il conviendrait de déterminer les cas dans lesquels cette technique doit être employée pour garantir l'efficacité de la parallélisation, peut-être en définissant des analyses statiques adaptées.

Bibliographie

Jean-Raymond Abrial. *The B Book - Assigning Programs to Meanings*. Cambridge University Press, August 1996.

Antony Alles. ATM Internetworking. Technical report, CISCO Systems Inc., 1995.

Peter Holst Andersen. Partial evaluation applied to ray tracing. DIKU Research Report 95/2, University of Copenhagen, 1995.

T Amtoft, Flemming Nielson, and Hanne Riis Nielson. Behaviour analysis and safety conditions : a case study in CML. In *FASE'98*, number 1382 in Lecture Notes in Computer Science, pages 255–269. Springer-Verlag, 1998.

H. P. Barendregt. *The Lambda Calculus : Its Syntax and Semantics*. Elsevier, 1984.

Patrick Behm, Paul Benoit, Alain Faivre, and Jean-Marc Meynadier. Meteor : A successful application of b in a large project. In *FM'99*, Lecture Notes in Computer Science, pages 369–387. Springer-Verlag, 1999.

Dragan Bosnacki, Dennis Dams, and Leszek Holenderski. Symmetric spin. In *7th international SPIN Workshop on Model Checking and Software Verification*, volume 1885 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.

Chiara Bodei, Pierpaolo Degano, Flemming Nielson, and Hanne Riis Nielson. Control flow analysis for the pi-calculus. In *Concur'98*, number 1466 in Lecture Notes in Computer Science, pages 84–98. Springer-Verlag, 1998.

Chiara Bodei, Pierpaolo Degano, Flemming Nielson, and Hanne Riis Nielson. Static analysis of processes for no read-up and no write-down. In *FOSSAC'99*, number 1578 in Lecture Notes in Computer Science, pages 120–134. Springer-Verlag, 1999.

Anders Bondorf and Jesper Jorgensen. Efficient analyses for realistic off-line parial evaluation : Extended version. DIKU Research Report 93/4, University of Copenhagen, 1993.

Dave Berry, Robin Milner, and David N. Turner. A semantics for ML concurrency primitives. In *Proceedings of the ACM-SIGPLAN Symposium on Principles of Programming Languages POPL'92*. ACM, 1992.

Gérard Boudol. Asynchrony and the π-calculus (note). Rapport de Recherche 1702, INRIA Sophia-Antipolis, may 1992.

Michele Boreale and Davide Sangiorgi. Some congruence properties for π-calculus bisimilarities. *Theoretical Computer Science*, 198 :159–176, 1998.

Lars Birkedal and Morten Welinder. Binding-time analysis for Standard ML. In *ACM-SIGPLAN Symposium on Partial Evaluation and Semantics Based Program Manipulations, PEPM'94*, pages 61–71, 1994.

Patrick Cousot and Radhia Cousot. Abstract interpretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the ACM-SIGPLAN Symposium on Principles of Programming Languages, POPL'77*, pages 238–252, 1977.

Patrick Cousot and Radhia Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4) :511–547, 1992.

Patrick Cousot and Radhia Cousot. Higher-order abstract interpretation (and application to comportment analysis generalizing strictness, termination, projection and per analysis of functional languages). In *International Conference on Computer Languages*, pages 95–112. IEEE Computer Society Press, May 1994.

Charles Consel and Olivier Danvy. Partial evaluation in parallel. *Lisp and Symbolic Computation*, 5 :327–342, 1993.

Charles Consel and Olivier Danvy. Partial evaluation : principles and perspectives. In *Proceedings of the ACM-SIGPLAN Symposium on Principles of Programming Languages, POPL'93*, 1993.

Charles Consel, Luke Hornof, Julia Lawall, Renaud Marlet, Gilles Muller, Jacques Noyé, Scott Thibault, and Eugen-Nicolae Volanschi. Tempo : Specializing systems applications and beyond. *ACM Computing Surveys, Symposium on Partial Evaluation (SOPE '98)*, 30(3), 1998.

Charles Consel and Siau Cheng Khoo. On-line and off-line partial evaluation : Semantic specifications and correctness proofs. Technical report, Department of Computer Science, Yale University, 1993.

Ludovic Casset and Jean-Louis Lanet. How to formally specify the java byte code semantics using the b method. In *ECOOP'99*, volume 1743 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.

Charles Consel, Gilles Muller, Eugen-Nicolae Volanschi, and Marlet Renaud. Adaptabilité et évaluation partielle : Tempo, un spécialiseur pour le langage C. Technical report, IRISA, 1997.

Charles Consel and François Noël. A general approach for run-time specialisation and its application to c. In *Proceedings of the ACM-SIGPLAN Symposium on Principles of Programming Languages POPL'96*. ACM, 1996.

Christopher Colby. Analyzing the communication topology of concurrent programs. In *Proceedings of the ACM-SIGPLAN Symposium on Partial Evaluation and semantic based program manipulations, PEPM'95*, pages 202–213. ACM, 1995.

Jean-Louis Colaço. *Analyses Statiques d'un Calcul d'Acteurs par Typage*. PhD thesis, Université Paul Sabatier - Toulouse, 1997.

Charles Consel. A tour of schism : A partial evaluation system for higher-order applicative languages. In *Proceedings of the ACM-SIGPLAN Symposium on Partial Evaluation and semantic based program manipulations, PEPM'93*. ACM, 1993.

Olivier Danvy. Type-directed partial evaluation. In *Proceedings of the ACM-SIGPLAN Symposium on Principles of Programming Languages POPL'96*. ACM, 1996.

R. De Nicola, G. Ferrari, and R. Pugliese. Types as specifications of access policies. In *Secure Internet Programming : Security Issues for Distributed and Mobile Objects, Lectures Notes in Computer Science, Springer*, volume 1603, 1999.

R. De Nicola, G. Ferrari, R. Pugliese, and B. Venneri. Types for access control. *Journal of Theoretical Computer Science*, 2000. à paraître.

Cristian Ene and Traian Muntean. Expressiveness of point-to-point versus broadcast communications. In *Fundamentals of Computation Theory, FTC'99*, volume 1684 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.

William Ferreira, Matthew Hennessy, and Alan Jeffrey. A theory of weak bisimulation for core CML. Technical Report 05/95, University of Sussex, School of Cognitive and Computer Sciences, 1995.

Yoshihito Futamura. Partial evaluation of computation process - an approach to a compiler-compiler. *Systems, Computers, Controls*, 2(5) :49–50, 1971.

A. Geist, A. Beguelin, J. Dongarra, R. Manchek, W. Jiang, and V. Sunderam. *PVM : A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press, 1994.

Carsten K. Gomard and Neil D. Jones. A partial evaluator for the untyped lambda-calculus. *Journal of Functional Programming*, 1(1) :21–69, January 1991.

Marc Gengler and Matthieu Martel. Self-applicable partial evaluation for the pi-calculus. In *Partial Evaluation and Semantics-Based Program Manipulation, PEPM'97*, pages 36–46, 1997.

Marc Gengler and Matthieu Martel. Self-applicable partial evaluation for the pi-calculus. Technical Report RR-9702, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, 1997.

Marc Gengler and Matthieu Martel. Des étages en Concurrent ML. In *Rencontres Francophones du Parallélisme, Renpar'10*, 1998.

Robert Glück, Robert Zöchling, and Richard Baier. Partial evaluation of numerical programs in Fortran. In *ACM-SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation, PEPM'94*, pages 119–132, 1994.

Luke Hornof, Charles Consel, and Jacques Noyé. Effective specialization of realistic programs via use sensitivity. In *Static Analysis Symposium, SAS'97*, volume 1302 of *Lecture Notes in Computer Science*, pages 448–472. Springer-Verlag, 1997.

Fritz Henglein. Efficient type inference for higher-order binding-time analysis. In *Conference on Functional Programming and Computer Architecture*, volume 523 of *Lecture Notes in Computer Science*, pages 448–472. Springer-Verlag, 1991.

Haruo Hosoya, Naoki Kobayashi, and Akinori Yonezawa. Partial evaluation scheme for concurrent languages and its correctness. In *Euro-par'96*, volume 1123 of *Lecture Notes in Computer Science*, pages 625–632. Springer-Verlag, 1996.

Robert Harper and Jhon C. Mitchell. The essence of ml. In *Proceedings of the ACM-SIGPLAN Symposium on Principles of Programming Languages POPL'88*, pages 332–345. ACM, 1988.

Fritz Henglein and Christian Mossin. Polymorphic binding-time analysis. In D. Sannella, editor, *Programming Languages and Systems, ESOP'94. 5th European Symposium on Programming*, volume 788 of *Lecture Notes in Computer Science*, pages 287–301. Springer-Verlag, April 1994.

Luke Hornof and Jacques Noyé. Accurate binding-time analysis for imperative languages : Flow, context and return sensitivity. In *ACM-SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulations, PEPM'97*, pages 63–73, 1997.

C.A.R Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8), 1978.

C.A.R Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

Gerard J. Holzmann. The model checker spin. *IEEE transactions on software engineering*, 23(5) :279–295, 1997.

Gerard J. Holzmann. The engineering of a model checker : the gnu i-protocol case study revisited. In *Practical Aspects of Spin Model-checking*, volume 1680 of *Lecture Notes in computer Science*. Springer-Verlag, 1999.

Sebastian Hunt and David Sands. Binding time analysis : A new PERspective. In *ACM-SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulations, PEPM'91*, pages 154–165, 1991.

Fritz Henglein and David Sands. A semantic model of binding times for safe partial evaluation. In *Proceedings of Programming Languages : Implementations, Logics and Programs (PLILP)*, Lecture Notes in Computer Science. Springer-Verlag, September 1995.

Kohei Honda and Mario Tokoro. On asynchronous communication semantics. In *Lecture Notes in computer Science*, volume 612, pages 21–51. Springer-Verlag, 1991.

Sebastian Hunt. *Abstract Interpretation of Functional Languages : From Theory to Practice*. PhD thesis, University of London, Imperial College of Science, Technology and Medecine, Department of Computing, October 1991.

Suresh Jagannathan. Locality abstractions for parallel and distributed computing. In *International Conference on Theory and Practice of Parallel Programming*, number 907 in Lecture Notes in Computer Science. Springer-Verlag, 1994.

N. D. Jones, C. K. Gomard, A. Bondorf, O. Danvy, and T. Mogensen. A self-applicable partial evaluator for the lambda calculus. In *IEEE International Conference on Computer Languages*, pages 49–58, 1990.

Neil D. Jones, Carsten K. Gomard, and Peter Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice Hall International, International Series in Computer Science, June 1993.

Neil D. Jones. Partial evaluation, self-application and types. In M.S. Paterson, editor, *Automata, Languages and Programming. Lecture Notes in Computer Science*, volume 443, pages 639–659. Springer-Verlag, 1990.

Naoki Kobayashi, Motoki Nakade, and Akinori Yonezawa. Static analysis of communications for asynchronous concurrent programming languages. In *SAS'95*, volume 983 of *Lecture Notes in Computer Science*, pages 225–242. Springer-Verlag, 1995.

David Andrew Kranz. *ORBIT : An Optimizing Compiler for Scheme*. PhD thesis, Graduate School of Yale University, February 1988.

John Launchbury. *Projection Factorisations in Partial Evaluation*. Cambridge University Press, 1991.

T.G. Lewis and H. El-Rewini. *Introduction to parallel computing*. Prentice-Hall international Editions, 1992.

P Lacan, Monfort J.N., L.V.Q. Ribal, and Gonthier G Deutsch A. The software reliability verification process : The ariane 5 example. In *Proceedings DASIA'98, Data Systems In Aerospace, ESA Publications, May 1998*, 1998.

Peter Lee and Mark Leone. Deferred compilation : The automation of run-time code generation. Technical report, School of Computer Science, Carnegie Mellon University, Pittsburgh, December 1993.

Mihnea Marinescu and Benjamin Goldberg. Partial evaluation techniques for concurrent programs. In *Proceedings of the ACM-SIGPLAN Symposium on Partial Evaluation and Semantic based Program Manipulations, PEPM'97*, pages 47–62. ACM, 1997.

Matthieu Martel and Marc Gengler. Analyse statique pour la sûreté des applications distribuées. In *Rencontres Francophones du Parallélisme, Renpar'12*, 2000.

Matthieu Martel and Marc Gengler. Communication topology analysis for concurrent programs. In *SPIN'2000*, volume 1885 of *Lecture Notes in Computer Science*. Springer-Verlag, August 2000.

Matthieu Martel and Marc Gengler. Partial evaluation of concurrent functional languages. Soumis, 2000.

Matthieu Martel and Marc Gengler. Static analysis of the communication topology of concurrent programs. Technical Report TR-354, Laboratoire d'Informatique de Marseille, April 2000.

Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.

Robin Milner. The polyadic π-calculus : a tutorial. In *Logic and Compositionality*. Springer, 1993.

Robin Milner. *Communicating and Mobile Systems : the pi-Calculus*. Cambridge University Press, May 1999.

Torben Æ. Mogensen. Binding time analysis for polymorphically typed higher order languages. In J. Diaz and F. Orejas, editors, *TAPSOFT '89. Proc. Int. Conf. Theory and Practice of Software Development*, volume 352 of *Lecture Notes in Computer Science*, pages 298–312. Springer-Verlag, March 1989.

Torben Æ. Mogensen. Self-applicable partial evaluation for pure lambda calculus. In *Proceedings of the ACM-SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation, PEPM'92*, pages 116–121, 1992.

Torben Æ. Mogensen. Efficient self-interpretation in lambda calculus. *Journal of Functional Programming*, 2(3) :345–364, July 1994.

Robin Milner and Davide Sangiorgi. Barbed bisimulation. In *ICALP'92*, number 623 in Lecture Notes in Computer Science, pages 685–695. Springer-Verlag, 1992.

Matthieu Martel and Tim Sheard. Introduction to multi-stage programming using MetaML. Technical report, Oregon Graduate Institute of Science and Technology, 1997.

Robin Milner, Mads Tofte, Robert Harper, and David MacQueen. *The Definition of Standard ML (Revised)*. MIT-Press, 1997.

Gilles Muller, Eugen-Nicolae Volanschi, and Renaud Marlet. Scaling up partial evaluation for optimizing the sun commercial rpc protocol. In *Proceedings of the ACM-SIGPLAN Symposium on Partial Evaluation and semantic based program manipulations, PEPM'97, Amsterdam*, pages 101–111. ACM, 1997.

Flemming Nielson and Hanne Riis Nielson. *Two-level functional languages*. Cambridge university press, 1992.

Flemming Nielson and Hanne Riis Nielson. Constraints for polymorphics behaviours of Concurrent ML. In *Constraints in Computational Logics*, number 845 in Lecture Notes in computer Science, pages 73–88. Springer-Verlag, 1994.

Flemming Nielson and Hanne Riis Nielson. Higher-order concurrent programs with finite communication topology. In *Proceedings of the ACM-SIGPLAN Symposium on Principles of Programming Languages POPL'94*, pages 84–97. ACM, 1994.

Flemming Nielson and Hanne Riis Nielson. Static and dynamic processor allocation for higher-order concurrent languages. In *TAPSOFT'95*, number 915 in Lecture Notes in Computer Science, pages 590–604. Springer-Verlag, 1995.

Flemming Nielson and Hanne Riis Nielson. Infinitary control flow analysis : a collecting semantics for closure analysis. In *Proceedings of the ACM-SIGPLAN Symposium on Principles of Programming Languages POPL'97*, pages 332–345. ACM, 1997.

Flemming Nielson and Hanne Riis Nielson, editors. *ML with Concurrency*. Monograph in Computer Science. Springer, 1999.

Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer, 1999.

Jens Palsberg. Correctness of binding-time analysis. *Journal of Functional Programming*, 3(3) :347–363, July 1993.

Frank Pfenning and Conal Elliot. Higher-order abstract syntax. In *Programming Language Design and Implementation, Atlanta, Georgia*, 1988.

Frank Pfenning and Peter Lee. Metacircularity in the polymorphic λ-calculus. *Theoretical Computer Science*, 89 :137–159, 1991.

Jens Palsberg and Michael I. Schwartzbach. Binding-time analysis : Abstract interpretation versus type inference. In *International Conference on Computer Languages, Toulouse, France*, pages 289–298. IEEE Computer Society Press, May 1994.

Benjamin C. Pierce and David N. Turner. *Pict Language Definition, Version 3.9d*, 1996.

John H. Reppy. An operational semantics of first-class synchronous operations. Technical Report TR-91-1232, Department of Computer Science, Cornell University, Ithaca, NY 14853, 1991.

John H. Reppy. *Higher-order Concurrency*. PhD thesis, Department of Computer Science, Cornell University, Ithaca, NY 14853, 1992.

John H. Reppy. *Concurrent Programming in ML*. Cambridge University Press, 1999.

A.W. Roscoe. Model checking csp. In *A Classical Mind : Essays in Honour of C.A.R. Hoare*, pages 353–378. Prentice-Hall, 1994.

Davide Sangiorgi. *Expressing Mobility in Process Algebras : First-Order and Higher-Order Paradigms*. PhD thesis, University of Edinburgh, 1992.

Tim Sheard. A type-directed, on-line partial evaluator for a polymorphic language. In *Proceedings of the ACM-SIGPLAN Symposium on Partial Evaluation and semantic based program manipulations, PEPM'97, Amsterdam*, pages 22–35. ACM, 1997.

Olin Shivers. Control flow analysis in scheme. In *Proceedings of the ACM-SIGPLAN Conference on Programming Language Design and Implementation, PLDI'88*, pages 164–174. ACM, 1988.

Olin Shivers. *Control Flow Analysis of Higher Order Languages*. PhD thesis, Carnegie Mellon University, Scholl of Computer Science, 1991. Technical Report CMU-CS-91-145.

Toshihiro Shimizu and Naoki Kobayashi. HACL user's manual. Technical report, Department of Information Science, University of Tokyo, 1994.

Kirsten L. Solberg, Flemming Nielson, and Hanne Riis Nielson. Systematic realisation of control flow analyses for CML. In *Proceedings of the ACM-SIGPLAN International Conference on Functional Programming, ICFP'97*, pages 38–51. ACM, 1997.

Mark Shields, Tim Sheard, and Simon Peyton Jones. Dynamic typing as staged type inference. In *Proceedings of the ACM-SIGPLAN Symposium on Principles of Programming Languages POPL'98*, 1998.

Andrew S. Tanenbaum. *Computer Networks, Third Edition*. Prentice Hall, 1996.

Walid Taha and Tim Sheard. Multi-stage programming with explicit annotations. In *Proceedings of the ACM-SIGPLAN Symposium on Partial Evaluation and Semantic Based Program Manipulations, PEPM'97*, pages 203–217. ACM, 1997.

Arnaud Venet. Abstract interpretation of the pi-calculus. In *LOMAPS'96*, volume 1192 of *Lecture Notes in Computer Science*, pages 51–75. Springer-Verlag, 1996.

David Walker. Objects and the pi-calculus. *Information and Computation*, 1995.

Mitchell Wand. Specifying the correctness of binding-time analysis. *Journal of Functional Programming*, 3(3) :365–387, July 1993.

Andrew K. Wright and Matthias Felleisen. A syntactic approach to type soundness. *Information and Computation*, 115 :38–94, 1994.

Pierre Weis and Xavier Leroy. *Le Langage CAML, Seconde Edition*. Dunod, 1999.

Annexe A

Correction de la CFA

Nous développons ici les preuves relatives aux propriétés énoncées dans le chapitre 4 concernant l'analyse de flot de contrôle.

A.1 Familles de Moore

Lemme 42 *Si* $\widehat{\mathcal{C}}_1, \widehat{\mathcal{E}}_1, \widehat{\mathcal{A}}_1 \vdash e^l$ *et* $\widehat{\mathcal{C}}_2, \widehat{\mathcal{E}}_2, \widehat{\mathcal{A}}_2 \vdash e^l$ *alors* $\left((\widehat{\mathcal{C}}_1, \widehat{\mathcal{E}}_1, \widehat{\mathcal{A}}_1) \sqcap (\widehat{\mathcal{C}}_2, \widehat{\mathcal{E}}_2, \widehat{\mathcal{A}}_2)\right) \vdash e^l$. □

PREUVE

Simple induction sur la structure de e^l.

CQFD

Proposition 22 *Soit* e^l *une expression étiquetée. L'ensemble* $\{(\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}) \ : \ \widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}} \vdash e^l\}$ *est une famille de Moore.* □

PREUVE

Soit $Y \subseteq \{(\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}) \ : \ \widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}} \vdash e^l\}$. Y peut s'écrire

$$\{(\widehat{\mathcal{C}}_i, \widehat{\mathcal{E}}_i, \widehat{\mathcal{A}}_i),\ 1 \leq i \leq n\}$$

et

$$\sqcap Y = (\widehat{\mathcal{C}}_1, \widehat{\mathcal{E}}_1, \widehat{\mathcal{A}}_1) \sqcap \ldots \sqcap (\widehat{\mathcal{C}}_n, \widehat{\mathcal{E}}_n, \widehat{\mathcal{A}}_n)$$

et par le lemme 42, $\sqcap Y \vdash e^l$.

CQFD

A.2 Lemme de substitution

Lemme 23 (Substitution) *Si* $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}[x \mapsto \widehat{\mathcal{C}}(l_1)], \widehat{\mathcal{A}} \vdash e^l$ *et* $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_0 \vdash v^{l_0}$ *et* $\widehat{\mathcal{C}}(l_0) \subseteq \widehat{\mathcal{C}}(l_1)$ *alors on a* $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}\{\mathbb{SG}(\widehat{\mathcal{A}}, x) \leftarrow \widehat{\mathcal{A}}_0\} \vdash e^l\{x \leftarrow v^{l_0}\}$. □

PREUVE

Par induction sur la structure de e^l.

- $\underline{e^l \widehat{=} c^l :}$ Pas de substitution.
- $\underline{e^l \widehat{=} x^l :}$ Par hypothèse, $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_0 \vdash v^{l_0}$. $x^l\{x \leftarrow v^{l_0}\} = v^{l_0}$ et $\widehat{\mathcal{A}}\{\mathbb{SG}(\widehat{\mathcal{A}}, x) \leftarrow \widehat{\mathcal{A}}_0\} = \widehat{\mathcal{A}}_0$. Donc
$$\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}\{\mathbb{SG}(\widehat{\mathcal{A}}, x) \leftarrow \widehat{\mathcal{A}}_0\} \vdash v^{l_0}$$
et
$$\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}\{\mathbb{SG}(\widehat{\mathcal{A}}, x) \leftarrow \widehat{\mathcal{A}}_0\} \vdash v^{l_0}$$
- $\underline{e^l \widehat{=} (\texttt{fun}\ y^{l_3}\ \texttt{=>}\ e_2^{l_2})^l :}$ Nous supposons que $y \neq x$, sinon il n'y a pas de substitution. Par définition de $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}[x \mapsto \widehat{\mathcal{C}}(l_1)], \widehat{\mathcal{A}} \vdash e^l$ nous avons
$$\widehat{\mathcal{C}}, \widehat{\mathcal{E}}[y \mapsto \widehat{\mathcal{C}}(l_3)][x \mapsto \widehat{\mathcal{C}}(l_1)], \widehat{\mathcal{A}}_2 \vdash e_2^{l_2} \tag{A.1}$$
et
$$l \in \widehat{\mathcal{C}}(l),\ [\![\mathrm{B}_l \xrightarrow{\varepsilon} \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}},\ \widehat{\mathcal{A}}_2 \sqsubseteq \widehat{\mathcal{A}} \tag{A.2}$$
D'après (A.1) et en utilisant les hypothèses d'induction, nous obtenons
$$\widehat{\mathcal{C}}, \widehat{\mathcal{E}}[y \mapsto \widehat{\mathcal{C}}(l_3)], \widehat{\mathcal{A}}_2\{\mathbb{SG}(\widehat{\mathcal{A}}_2, x) \leftarrow \widehat{\mathcal{A}}_1\} \vdash e_2^{l_2}\{x \leftarrow v^{l_1}\} \tag{A.3}$$
D'après (A.2) et (A.3) nous concluons que
$$\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}\{\mathbb{SG}(\widehat{\mathcal{A}}, x) \leftarrow \widehat{\mathcal{A}}_1\} \vdash (\texttt{fun}\ x^{l_3}\ \texttt{=>}\ (e_2^{l_2}\{x \leftarrow v^{l_1}\}))^l$$
et puisque $x \neq y$,
$$\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}\{\mathbb{SG}(\widehat{\mathcal{A}}, x) \leftarrow \widehat{\mathcal{A}}_1\} \vdash (\texttt{fun}\ x^{l_3}\ \texttt{=>}\ e_2^{l_2})^l\{x \leftarrow v^{l_1}\}$$
- $\underline{e^l \widehat{=} (\texttt{rec}\ f^{l_4}\ y^{l_3}\ \texttt{=>}\ e_2^{l_2})^l :}$ Similaire au cas précédent.
- $\underline{e^l \widehat{=} (e_2^{l_2}\ e_3^{l_3})^l :}$ Par induction
$$\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_2\{\mathbb{SG}(\widehat{\mathcal{A}}_2, x) \leftarrow \widehat{\mathcal{A}}_0\} \vdash e_2^{l_2}\{x \leftarrow v^{l_0}\} \tag{A.4}$$
$$\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_3\{\mathbb{SG}(\widehat{\mathcal{A}}_3, x) \leftarrow \widehat{\mathcal{A}}_0\} \vdash e_3^{l_3}\{x \leftarrow v^{l_0}\} \tag{A.5}$$

– si $e_2^{l_2} \widehat{=} x^{l_2}$ alors $e_2^{l_2}\{x \leftarrow v^{l_0}\} = v^{l_0}$. D'une part, nous avons

$$[\![\mathrm{B}_l \xrightarrow{\varepsilon}\!\circ\ \mathrm{B}_{l_0}]\!] \sqsubseteq \widehat{\mathcal{A}}\{\mathbb{SG}(\widehat{\mathcal{A}}_2, x) \leftarrow \widehat{\mathcal{A}}_0\}$$

et d'autre part

$$[\![\mathrm{E}_{l_0} \xrightarrow{\varepsilon}\!\circ\ \mathrm{B}_{l_3}]\!] \sqsubseteq \widehat{\mathcal{A}}\{\mathbb{SG}(\widehat{\mathcal{A}}_2, x) \leftarrow \widehat{\mathcal{A}}_0\}$$

De plus,

$$\widehat{\mathcal{C}}(l_0) \subseteq \widehat{\mathcal{C}}(l_1) = \widehat{\mathcal{E}}[x \mapsto \widehat{\mathcal{C}}(l_1)](x) \subseteq \widehat{\mathcal{C}}(l_2)$$

En conséquent les quantifications sont préservées.

– si $e_3^{l_3} \widehat{=} x^{l_3}$ alors

$$e_3^{l_3}\{x \leftarrow v^{l_0}\} = v^{l_0}$$

A nouveau,

$$[\![\mathrm{E}_{l_2} \xrightarrow{\varepsilon}\!\circ\ \mathrm{B}_{l_0}]\!] \sqsubseteq \widehat{\mathcal{A}}\{\mathbb{SG}(\widehat{\mathcal{A}}_2, x) \leftarrow \widehat{\mathcal{A}}_0\}$$

Soit $l_4 \in \widehat{\mathcal{C}}(l_3)$ tel que

$$(\texttt{fun}\ y^{l_6}\ \texttt{=>}\ e_5^{l_5})^{l_4} \in \mathrm{prg}$$

Nous avons

$$[\![\mathrm{E}_{l_0} \xrightarrow{\varepsilon}\!\circ\ \mathrm{B}_{l_5}]\!] \sqsubseteq \widehat{\mathcal{A}}\{\mathbb{SG}(\widehat{\mathcal{A}}, x) \leftarrow \widehat{\mathcal{A}}_0\}$$

et

$$\widehat{\mathcal{C}}(l_0) \subseteq \widehat{\mathcal{C}}(l_1) = \widehat{\mathcal{E}}[x \mapsto \widehat{\mathcal{C}}(l_1)](x) \subseteq \widehat{\mathcal{C}}(l_3)$$

CQFD

A.3 Preuve par réduction du sujet (expressions séquentielles)

Lemme 24 (Monotonie) *Si* $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}} \vdash e^l$ *et* $e^l \hookrightarrow e'^{l'}$ *alors* $\widehat{\mathcal{C}}(l') \subseteq \widehat{\mathcal{C}}(l)$. □

PREUVE

Voir la preuve de la proposition 18.

CQFD

Lemme 43 *Si* $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}} \vdash e^l$ *et* $e^l \hookrightarrow e'^{l'}$ *alors* $\mathrm{B}_l \xrightarrow{\varepsilon}\!\circ_*\ \mathrm{B}_{l'}$ *et* $\mathrm{E}_{l'} \xrightarrow{\varepsilon}\!\circ_*\ \mathrm{E}_l$ *dans* $\widehat{\mathcal{A}}$. □

PREUVE

Voir la preuve de la proposition 18.

CQFD

Proposition 26 (Réduction du sujet) *Si* $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}} \vdash e^l$ *et* $e^l \hookrightarrow e'^{l'}$ *alors* $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}' \vdash e'^{l'}$ *pour un certain* $\widehat{\mathcal{A}}'$ *tel que* $\widehat{\mathcal{A}}' \lessdot \widehat{\mathcal{A}}$. □

PREUVE

Nous prouvons que si $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}, \vdash e^l$ et $e^l \hookrightarrow e'^{l'}$ alors les propriétés suivantes sont vérifiées.

(i) $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}' \vdash e'^{l'}$ pour un certain $\widehat{\mathcal{A}}'$ tel que $\widehat{\mathcal{A}}' \lessdot \widehat{\mathcal{A}}$.

(ii) $\widehat{\mathcal{C}}(l') \subseteq \widehat{\mathcal{C}}(l)$.

(iii) $\mathrm{B}_l \overset{\varepsilon}{\multimap}_* \mathrm{B}_{l'}$ et $\mathrm{E}_{l'} \overset{\varepsilon}{\multimap}_* \mathrm{E}_l$ in $\widehat{\mathcal{A}}$.

– $e^l \widehat{=} (e_0^{l_0}\ e_1^{l_1})^l$ et

$$\frac{e_0^{l_0} \hookrightarrow e_2^{l_2}}{(e_0^{l_0}\ e_1^{l_1})^l \hookrightarrow (e_2^{l_2}\ e_1^{l_1})^l}$$

Par définition de $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}} \vdash e^l$, nous avons

$$\begin{array}{ll} \widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_0 \vdash e_0^{l_0} & \widehat{\mathcal{A}}_0 \sqsubseteq \widehat{\mathcal{A}} \\ \widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_1 \vdash e_1^{l_1} & \widehat{\mathcal{A}}_1 \sqsubseteq \widehat{\mathcal{A}} \\ [\![\mathrm{B}_l \overset{\varepsilon}{\multimap} \mathrm{B}_{l_0}]\!] \sqsubseteq \widehat{\mathcal{A}} & [\![\mathrm{E}_{l_0} \overset{\varepsilon}{\multimap} \mathrm{B}_{l_1}]\!] \sqsubseteq \widehat{\mathcal{A}} \end{array}$$

Par induction, $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_0 \vdash e_0^{l_0}$ et $e_0^{l_0} \hookrightarrow e_2^{l_2}$ implique

(i) $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_2 \vdash e_2^{l_2}$ for some $\widehat{\mathcal{A}}_2$ such that $\widehat{\mathcal{A}}_2 \lessdot \widehat{\mathcal{A}}_0$.

(ii) $\widehat{\mathcal{C}}(l_2) \subseteq \widehat{\mathcal{C}}(l_0)$.

(iii) $\mathrm{B}_{l_0} \overset{\varepsilon}{\multimap}_* \mathrm{B}_{l_2}$ et $\mathrm{E}_{l_2} \overset{\varepsilon}{\multimap}_* \mathrm{E}_{l_0}$ in $\widehat{\mathcal{A}}_0$.

soit $\widehat{\mathcal{A}}'$ tel que

$$\begin{array}{ll} \widehat{\mathcal{A}}_2 \sqsubseteq \widehat{\mathcal{A}}' & \widehat{\mathcal{A}}_1 \sqsubseteq \widehat{\mathcal{A}}' \\ [\![\mathrm{B}_l \overset{\varepsilon}{\multimap} \mathrm{B}_{l_2}]\!] \sqsubseteq \widehat{\mathcal{A}}' & [\![\mathrm{E}_{l_2} \overset{\varepsilon}{\multimap} \mathrm{B}_{l_1}]\!] \sqsubseteq \widehat{\mathcal{A}}' \end{array}$$

et tel que

$$\begin{array}{l} \forall l_5 \in \widehat{\mathcal{C}}(l_2)\ :\ (\mathtt{fun}\ x^{l_4} \Rightarrow e_3^{l_3})^{l_5} \in \mathrm{prg}, \\ \quad [\![\mathrm{E}_{l_1} \overset{\varepsilon}{\multimap} \mathrm{B}_{l_3}]\!] \sqsubseteq \widehat{\mathcal{A}}',\ [\![\mathrm{E}_{l_3} \overset{\varepsilon}{\multimap} \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}}' \end{array}$$

$$\begin{array}{l} \forall l_6 \in \widehat{\mathcal{C}}(l_2)\ :\ (\mathtt{rec}\ f^{l_4}\ x^{l_5} \Rightarrow e_3^{l_3})^{l_6} \in \mathrm{prg}, \\ \quad [\![\mathrm{E}_{l_1} \overset{\varepsilon}{\multimap} \mathrm{B}_{l_3}]\!] \sqsubseteq \widehat{\mathcal{A}}',\ [\![\mathrm{E}_{l_3} \overset{\varepsilon}{\multimap} \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}}',\ [\![\mathrm{E}_{l_1} \overset{\varepsilon}{\multimap} \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}}',\ [\![\mathrm{E}_{l_3} \overset{\varepsilon}{\multimap} \mathrm{B}_{l_3}]\!] \sqsubseteq \widehat{\mathcal{A}}' \end{array}$$

Puisque $\widehat{\mathcal{C}}(l_2) \subseteq \widehat{\mathcal{C}}(l_0)$, nous avons

$$\begin{array}{l}
\forall l_5 \in \widehat{\mathcal{C}}(l_2) \ : \ (\texttt{fun}\ x^{l_4}\ \texttt{=>}\ e_3^{l_3})^{l_5} \in \mathrm{prg}, \\
\left|\ \widehat{\mathcal{C}}(l_1) \subseteq \widehat{\mathcal{C}}(l_4),\ \widehat{\mathcal{C}}(l_3) \subseteq \widehat{\mathcal{C}}(l),\ [\![\mathrm{E}_{l_1} \overset{\varepsilon}{\multimap} \mathrm{B}_{l_3}]\!] \sqsubseteq \widehat{\mathcal{A}}',\ [\![\mathrm{E}_{l_3} \overset{\varepsilon}{\multimap} \mathrm{E}_{l}]\!] \sqsubseteq \widehat{\mathcal{A}}' \right. \\
\forall l_6 \in \widehat{\mathcal{C}}(l_2) \ : \ (\texttt{rec}\ f^{l_4}\ x^{l_5}\ \texttt{=>}\ e_3^{l_3})^{l_6} \in \mathrm{prg}, \\
\left|\ \begin{array}{l} \widehat{\mathcal{C}}(l_1) \subseteq \widehat{\mathcal{C}}(l_5),\ \widehat{\mathcal{C}}(l_3) \subseteq \widehat{\mathcal{C}}(l) \\ [\![\mathrm{E}_{l_1} \overset{\varepsilon}{\multimap} \mathrm{B}_{l_3}]\!] \sqsubseteq \widehat{\mathcal{A}}',\ [\![\mathrm{E}_{l_3} \overset{\varepsilon}{\multimap} \mathrm{E}_{l}]\!] \sqsubseteq \widehat{\mathcal{A}}',\ [\![\mathrm{E}_{l_1} \overset{\varepsilon}{\multimap} \mathrm{E}_{l}]\!] \sqsubseteq \widehat{\mathcal{A}}',\ [\![\mathrm{E}_{l_3} \overset{\varepsilon}{\multimap} \mathrm{B}_{l_3}]\!] \sqsubseteq \widehat{\mathcal{A}}' \end{array} \right.
\end{array}$$

et par conséquence $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}', \vdash (e_2^{l_2}\ e_1^{l_1})^l$. Nous devons montrer que $\widehat{\mathcal{A}}' \lessdot \widehat{\mathcal{A}}$. Nous avons

$$\widehat{\mathcal{A}}_2 \lessdot \widehat{\mathcal{A}}_0 \qquad [\![\mathrm{B}_l \overset{\varepsilon}{\multimap} \mathrm{B}_{l_2}]\!] \sqsubseteq \widehat{\mathcal{A}}' \qquad [\![\mathrm{E}_{l_2} \overset{\varepsilon}{\multimap} \mathrm{B}_{l_1}]\!] \sqsubseteq \widehat{\mathcal{A}}'$$

Dans $\widehat{\mathcal{A}}$, $\mathrm{B}_l \overset{\varepsilon}{\multimap}_* \mathrm{B}_{l_2}$ et $\mathrm{E}_{l_2} \overset{\varepsilon}{\multimap}_* \mathrm{B}_{l_1}$. Par conséquent nous obtenons $\widehat{\mathcal{A}}' \lessdot \widehat{\mathcal{A}}$.

– $e^l \widehat{=} (v^{l_0}\ e_1^{l_1})^l$ et

$$\frac{e_1^{l_1} \hookrightarrow e_2^{l_2}}{(v^{l_0}\ e_1^{l_1})^l \hookrightarrow (v^{l_0}\ e_2^{l_2})^l}$$

A nouveau, par définition de $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}} \vdash e^l$, nous avons

$$\begin{array}{ll}
\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_0 \vdash v^{l_0} & \widehat{\mathcal{A}}_0 \sqsubseteq \widehat{\mathcal{A}} \\
\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_1 \vdash e_1^{l_1} & \widehat{\mathcal{A}}_1 \sqsubseteq \widehat{\mathcal{A}} \\
[\![\mathrm{B}_l \overset{\varepsilon}{\multimap} \mathrm{B}_{l_0}]\!] \sqsubseteq \widehat{\mathcal{A}} & [\![\mathrm{E}_{l_0} \overset{\varepsilon}{\multimap} \mathrm{B}_{l_1}]\!] \sqsubseteq \widehat{\mathcal{A}}
\end{array}$$

Par induction, $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_1 \vdash e_1^{l_1}$ et $e_1^{l_1} \hookrightarrow e_2^{l_2}$ implique

(i) $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_2 \vdash e_2^{l_2}$ pour un certain $\widehat{\mathcal{A}}_2$ tel que $\widehat{\mathcal{A}}_2 \lessdot \widehat{\mathcal{A}}_1$.

(ii) $\widehat{\mathcal{C}}(l_2) \subseteq \widehat{\mathcal{C}}(l_1)$.

(iii) $\mathrm{B}_{l_1} \overset{\varepsilon}{\multimap}_* \mathrm{B}_{l_2}$ et $\mathrm{E}_{l_2} \overset{\varepsilon}{\multimap}_* \mathrm{E}_{l_1}$ in $\widehat{\mathcal{A}}_1$.

De plus, puisque $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}} \vdash (v^{l_0}\ e_1^{l_1})^l$ nous avons

$$\begin{array}{l}
\forall l_5 \in \widehat{\mathcal{C}}(l_0) \ : \ (\texttt{fun}\ x^{l_4}\ \texttt{=>}\ e_3^{l_3})^{l_5} \in \mathrm{prg}, \\
\left|\ \widehat{\mathcal{C}}(l_1) \subseteq \widehat{\mathcal{C}}(l_4),\ \widehat{\mathcal{C}}(l_3) \subseteq \widehat{\mathcal{C}}(l),\ [\![\mathrm{E}_{l_1} \overset{\varepsilon}{\multimap} \mathrm{B}_{l_3}]\!] \sqsubseteq \widehat{\mathcal{A}},\ [\![\mathrm{E}_{l_3} \overset{\varepsilon}{\multimap} \mathrm{E}_{l}]\!] \sqsubseteq \widehat{\mathcal{A}} \right. \\
\forall l_2 \in \widehat{\mathcal{C}}(l_0) \ : \ (\texttt{rec}\ f^{l_4}\ x^{l_5}\ \texttt{=>}\ e_3^{l_3})^{l_2} \in \mathrm{prg}, \\
\left|\ \begin{array}{l} \widehat{\mathcal{C}}(l_1) \subseteq \widehat{\mathcal{C}}(l_5),\ \widehat{\mathcal{C}}(l_3) \subseteq \widehat{\mathcal{C}}(l) \\ [\![\mathrm{E}_{l_1} \overset{\varepsilon}{\multimap} \mathrm{B}_{l_3}]\!] \sqsubseteq \widehat{\mathcal{A}},\ [\![\mathrm{E}_{l_3} \overset{\varepsilon}{\multimap} \mathrm{E}_{l}]\!] \sqsubseteq \widehat{\mathcal{A}},\ [\![\mathrm{E}_{l_1} \overset{\varepsilon}{\multimap} \mathrm{E}_{l}]\!] \sqsubseteq \widehat{\mathcal{A}},\ [\![\mathrm{E}_{l_3} \overset{\varepsilon}{\multimap} \mathrm{B}_{l_3}]\!] \sqsubseteq \widehat{\mathcal{A}} \end{array} \right.
\end{array}$$

soit $\widehat{\mathcal{A}}'$ tel que

$$\widehat{\mathcal{A}}_0 \sqsubseteq \widehat{\mathcal{A}}' \qquad \widehat{\mathcal{A}}_2 \sqsubseteq \widehat{\mathcal{A}}'$$
$$[\![\mathrm{B}_l \xrightarrow{\varepsilon}\!\circ\ \mathrm{B}_{l_0}]\!] \sqsubseteq \widehat{\mathcal{A}}' \qquad [\![\mathrm{E}_{l_0} \xrightarrow{\varepsilon}\!\circ\ \mathrm{B}_{l_2}]\!] \sqsubseteq \widehat{\mathcal{A}}'$$

et tel que pour toute fonction $(\texttt{fun}\ x^{l_4} \texttt{=>}\ e_3^{l_3})^{l_5} \in \mathrm{prg}$ avec $l_5 \in \widehat{\mathcal{C}}(l_0)$, $[\![\mathrm{E}_{l_2} \xrightarrow{\varepsilon}\!\circ\ \mathrm{B}_{l_3}]\!] \sqsubseteq \widehat{\mathcal{A}}'$. Puisque, pour toute fonction $(\texttt{fun}\ x^{l_4} \texttt{=>}\ e_3^{l_3})^{l_2} \in \mathrm{prg}$, $\widehat{\mathcal{C}}(l_2) \subseteq \widehat{\mathcal{C}}(l_1)$ implique $\widehat{\mathcal{C}}(l_2) \subseteq \widehat{\mathcal{C}}(l_4)$, nous avons

$$\begin{array}{l}
\forall l_5 \in \widehat{\mathcal{C}}(l_0)\ :\ (\texttt{fun}\ x^{l_4} \texttt{=>}\ e_3^{l_3})^{l_5} \in \mathrm{prg}, \\
\quad \widehat{\mathcal{C}}(l_2) \subseteq \widehat{\mathcal{C}}(l_4),\ \widehat{\mathcal{C}}(l_3) \subseteq \widehat{\mathcal{C}}(l),\ [\![\mathrm{E}_{l_2} \xrightarrow{\varepsilon}\!\circ\ \mathrm{B}_{l_3}]\!] \sqsubseteq \widehat{\mathcal{A}}',\ [\![\mathrm{E}_{l_3} \xrightarrow{\varepsilon}\!\circ\ \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}}' \\
\forall l_2 \in \widehat{\mathcal{C}}(l_0)\ :\ (\texttt{rec}\ f^{l_4}\ x^{l_5} \texttt{=>}\ e_3^{l_3})^{l_2} \in \mathrm{prg}, \\
\quad \widehat{\mathcal{C}}(l_2) \subseteq \widehat{\mathcal{C}}(l_5),\ \widehat{\mathcal{C}}(l_3) \subseteq \widehat{\mathcal{C}}(l) \\
\quad [\![\mathrm{E}_{l_2} \xrightarrow{\varepsilon}\!\circ\ \mathrm{B}_{l_3}]\!] \sqsubseteq \widehat{\mathcal{A}}',\ [\![\mathrm{E}_{l_3} \xrightarrow{\varepsilon}\!\circ\ \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}}',\ [\![\mathrm{E}_{l_2} \xrightarrow{\varepsilon}\!\circ\ \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}}',\ [\![\mathrm{E}_{l_3} \xrightarrow{\varepsilon}\!\circ\ \mathrm{B}_{l_3}]\!] \sqsubseteq \widehat{\mathcal{A}}'
\end{array}$$

Donc $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}' \vdash (e_2^{l_2}\ e_1^{l_1})^l$ et nous devons montrer que $\widehat{\mathcal{A}}' \lessdot \widehat{\mathcal{A}}$. $[\![\mathrm{E}_{l_0} \xrightarrow{\varepsilon}\!\circ\ \mathrm{B}_{l_2}]\!] \sqsubseteq \widehat{\mathcal{A}}'$ et

$$\forall l_5 \in \widehat{\mathcal{C}}(l_2)\ :\ (\texttt{fun}\ x^{l_4} \texttt{=>}\ e_3^{l_3})^{l_5} \in \mathrm{prg},\ [\![\mathrm{E}_{l_2} \xrightarrow{\varepsilon}\!\circ\ \mathrm{B}_{l_3}]\!] \sqsubseteq \widehat{\mathcal{A}}'$$

Puisque $\mathrm{B}_{l_1} \xrightarrow{\varepsilon}\!\circ_*\ \mathrm{B}_{l_2}$ et $\mathrm{E}_{l_2} \xrightarrow{\varepsilon}\!\circ_*\ \mathrm{E}_{l_1}$ in $\widehat{\mathcal{A}}_1$, nous avons $\widehat{\mathcal{A}}' \lessdot \widehat{\mathcal{A}}$.

– $e^l \mathrel{\widehat{=}} ((\texttt{fun}\ x^{l_4} \texttt{=>}\ e_3^{l_3})^{l_0}\ v^{l_1})^l$ et $e^l \hookrightarrow e_3^{l_3}\{x \leftarrow v^{l_1}\}$. $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}} \vdash e^l$ implique

$$\begin{array}{ll}
\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_0 \vdash (\texttt{fun}\ x^{l_4} \texttt{=>}\ e_3^{l_3})^{l_0} & \widehat{\mathcal{A}}_0 \sqsubseteq \widehat{\mathcal{A}} \\
\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_1 \vdash v^{l_1} &
\end{array}$$

$\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_0 \vdash (\texttt{fun}\ x^{l_4} \texttt{=>}\ e_3^{l_3})^{l_0}$ implique

$$\widehat{\mathcal{C}}, \widehat{\mathcal{E}}[x \mapsto \widehat{\mathcal{C}}(l_4)], \widehat{\mathcal{A}}_3 \vdash e_3^{l_3} \qquad l_0 \in \widehat{\mathcal{C}}(l_0)$$

Donc, comme cas particulier de $\forall l_5 \in \widehat{\mathcal{C}}(l_0)\ :\ (\texttt{fun}\ x^{l_4} \texttt{=>}\ e_3^{l_3})^{l_5} \in \mathrm{prg}, \ldots$ nous avons

$$\widehat{\mathcal{C}}(l_3) \subseteq \widehat{\mathcal{C}}(l) \qquad \widehat{\mathcal{C}}(l_1) \subseteq \widehat{\mathcal{C}}(l_4)$$

Puisque $\widehat{\mathcal{C}}(l_1) \subseteq \widehat{\mathcal{C}}(l_4)$ et $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}} \vdash v^{l_1}$, d'après le lemme 23 nous obtenons $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}' \vdash e_3^{l_3}\{x \leftarrow v^{l_1}\}$ avec

$$\widehat{\mathcal{A}}' \mathrel{\widehat{=}} \widehat{\mathcal{A}}_3\{\mathbb{SG}(\widehat{\mathcal{A}}_3, x) \leftarrow \widehat{\mathcal{A}}_1\}$$

Par examen de la façon dont $\widehat{\mathcal{A}}$ est construit, il est immédiat que $\mathrm{B}_l \xrightarrow{\varepsilon}\!\circ_*\ \mathrm{B}_{l_3}$ et $\mathrm{E}_{l_3} \xrightarrow{\varepsilon}\!\circ_*\ \mathrm{E}_l$ dans $\widehat{\mathcal{A}}$.

Nous devons montrer que $\widehat{\mathcal{A}}' \lessdot \widehat{\mathcal{A}}$. D'après $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}} \vdash e^l$, nous avons $\widehat{\mathcal{A}}_0 \sqsubseteq \widehat{\mathcal{A}}$ et

d'après $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}} \vdash e_0^{l_0}$ nous avons

$$\widehat{\mathcal{A}}_3 \sqsubseteq \widehat{\mathcal{A}}_0 \quad [\![\mathrm{B}_{l_0} \xrightarrow{\varepsilon}\!\circ\ \mathrm{E}_{l_0}]\!] \sqsubseteq \widehat{\mathcal{A}}_0 \quad [\![\mathrm{B}_{l_0} \xrightarrow{\mu}\!\circ\ \mathrm{B}_{l_3}]\!] \sqsubseteq \widehat{\mathcal{A}}_0 \quad [\![\mathrm{E}_{l_3} \xrightarrow{\mu}\!\circ\ \mathrm{E}_{l_0}]\!] \sqsubseteq \widehat{\mathcal{A}}_0$$

Donc nous avons $\widehat{\mathcal{A}}_3 \lessdot \widehat{\mathcal{A}}$ et les substitutions remplacent seulement des ε-transitions par d'autres. Donc, $\widehat{\mathcal{A}}' \lessdot \widehat{\mathcal{A}}$.

– $e^l \widehat{=} (\texttt{rec}\ f^{l_0}\ x^{l_1}\ \texttt{=>}\ e_2^{l_2})^l$ et

$$e^l \hookrightarrow (\texttt{fun}\ x^{l_1}\ \texttt{=>}\ e_2^{l_2}\{f \leftarrow (\texttt{rec}\ f^{l_0}\ x^{l_1}\ \texttt{=>}\ e_2^{l_2})^l\})^l$$

$\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}} \vdash e^l$ implique

$$\begin{array}{l} l \in \widehat{\mathcal{C}}(l),\ \widehat{\mathcal{C}}(l) \subseteq \widehat{\mathcal{C}}(l_0) \\ \widehat{\mathcal{C}}, \widehat{\mathcal{E}}[f \mapsto \widehat{\mathcal{C}}(l_0)][x \mapsto \widehat{\mathcal{C}}(l_1)], \widehat{\mathcal{A}}_2 \vdash e_2^{l_2} \end{array}$$

D'après le lemme de substitution, $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}[f \mapsto \widehat{\mathcal{C}}(l_0)][x \mapsto \widehat{\mathcal{C}}(l_1)], \widehat{\mathcal{A}}_2 \vdash e_2^{l_2}$, $\widehat{\mathcal{C}}(l) \subseteq \widehat{\mathcal{C}}(l_0)$ et $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}} \vdash e^l$ implique

$$\widehat{\mathcal{C}}, \widehat{\mathcal{E}}[x \mapsto \widehat{\mathcal{C}}(l_1)], \widehat{\mathcal{A}}_2\{\mathbb{SG}(\widehat{\mathcal{A}}_2, f) \leftarrow \widehat{\mathcal{A}}\} \vdash e_2^{l_2}\{f \leftarrow e^l\} \tag{A.6}$$

Soit $\widehat{\mathcal{A}}' \widehat{=} \widehat{\mathcal{A}}_2\{\mathbb{SG}(\widehat{\mathcal{A}}_2, f) \leftarrow \widehat{\mathcal{A}}\}$ et $\widehat{\mathcal{A}}''$ un automate tel que $\widehat{\mathcal{A}}' \sqsubseteq \widehat{\mathcal{A}}''$, $[\![\mathrm{B}_l \xrightarrow{\varepsilon}\!\circ\ \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}}''$, $[\![\mathrm{B}_l \xrightarrow{\mu}\!\circ\ \mathrm{Start}(\widehat{\mathcal{A}}')]\!] \sqsubseteq \widehat{\mathcal{A}}''$ et $[\![\mathrm{End}(\widehat{\mathcal{A}}') \xrightarrow{\mu}\!\circ\ \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}}''$. En utilisant les équations (A.6) et $l \in \widehat{\mathcal{C}}(l)$ nous obtenons

$$\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}'' \vdash (\texttt{fun}\ x^{l_1}\ \texttt{=>}\ e_2^{l_2}\{f \leftarrow (\texttt{rec}\ f^{l_0}\ x^{l_1}\ \texttt{=>}\ e_2^{l_2})^l\})^l$$

et $\widehat{\mathcal{A}}'' \lessdot \widehat{\mathcal{A}}$ puisque nous avons seulement ajouté des ε-transitions.

CQFD

A.4 Preuve par réduction du sujet (groupe de processus)

Lemme 44 (Contextes - 1) *Si $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}} \vdash E[e_0^{l_0}]$ alors $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_0 \vdash e_0^{l_0}$ et $\widehat{\mathcal{A}} = \widehat{\mathcal{A}}'[\widehat{\mathcal{A}}_0]$.* □

PREUVE

Simple examen de la structure de E[].

CQFD

Lemme 45 (Contextes - 2) *Si $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}[\widehat{\mathcal{A}}_0] \vdash E[e_0^{l_0}]$, $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_1 \vdash e_1^{l_1}$ et $\widehat{\mathcal{C}}(l_1) \subseteq \widehat{\mathcal{C}}(l_0)$ alors on a $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}[\widehat{\mathcal{A}}_1] \vdash E[e_1^{l_1}]$.* □

PREUVE

Simple examen de la structure de E[].

CQFD

Proposition 31 (Preuve par réduction) *Soit P un groupe de processus tel que $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_{\otimes}^{S} \models^{S} P$. Si $K, P \xrightarrow{[\ell]} K, P'$ alors nous avons $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_{\otimes}^{S\prime} \models^{S} P'$ pour un automate produit $\widehat{\mathcal{A}}_{\otimes}^{S\prime}$ tel que $\widehat{\mathcal{A}}_{\otimes}^{S\prime} \overset{\ell}{\preccurlyeq} \widehat{\mathcal{A}}_{\otimes}^{S}$.* □

PREUVE

–

$$\frac{e_0^{l_0} \hookrightarrow e_1^{l_1}}{K, P :: \langle p : E[e_0^{l_0}]\rangle \xrightarrow{[\varepsilon]} K, P :: \langle p : E[{e_1}^{l_1}]\rangle}$$

$$\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_{\otimes}^{S} \models^{S} P :: \langle p : E[e_0^{l_0}]\rangle$$

implique $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_p \vdash E[e_0^{l_0}]$ with $\widehat{\mathcal{A}}_p \mathrel{\widehat{=}} \widehat{\mathcal{A}}[\widehat{\mathcal{A}}_0]$ Ici, $\widehat{\mathcal{A}}_p$ est un élément de la collection $\widehat{\mathcal{G}} = (\widehat{\mathcal{A}}_p)_{p \in \mathrm{Dom}(P \cup \{p\})}$ d'automates telle que $\mathrm{A}_{\otimes}^{S}(\widehat{\mathcal{G}}) = \widehat{\mathcal{A}}_{\otimes}^{S}$. D'après le lemme 44 nous avons

$$\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_0 \vdash e_0^{l_0}$$

et $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_0 \vdash e_0^{l_0}$ et $e_0^{l_0} \hookrightarrow e_1^{l_1}$ implique

$$\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_1 \vdash e_1^{l_1} \qquad \widehat{\mathcal{A}}_1 \preccurlyeq \widehat{\mathcal{A}}_0 \qquad \widehat{\mathcal{C}}(l_1) \subseteq \widehat{\mathcal{C}}(l_0)$$

et $\mathrm{B}_{l_0} \overset{\varepsilon}{\multimap}_{*} \mathrm{B}_{l_1}$ et $\mathrm{E}_{l_1} \overset{\varepsilon}{\multimap}_{*} \mathrm{E}_{l_0}$ in $\widehat{\mathcal{A}}_0$. En utilisant le lemme 45, $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}'_p \vdash E[e_1^{l_1}]$ avec

$$\widehat{\mathcal{A}}'_p \mathrel{\widehat{=}} \widehat{\mathcal{A}}[\widehat{\mathcal{A}}_1]$$

Puisque $\widehat{\mathcal{A}}_1 \preccurlyeq \widehat{\mathcal{A}}_0$, nous avons $\widehat{\mathcal{A}}'_p \preccurlyeq \widehat{\mathcal{A}}_p$. Soit $\widehat{\mathcal{G}}'$ la collection obtenue en substituant $\widehat{\mathcal{A}}'_p$ à $\widehat{\mathcal{A}}_p$ dans $\widehat{\mathcal{G}}$. Nous avons

$$\widehat{\mathcal{A}}_{\otimes}^{S\prime} = \mathrm{A}_{\otimes}^{S}(\widehat{\mathcal{G}}') \preccurlyeq \mathrm{A}_{\otimes}^{S}(\widehat{\mathcal{G}})$$

et $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_{\otimes}^{S\prime} \models^{S} P :: \langle p : E[e_1^{l_1}]\rangle$.

– $K, P :: Q_1 \xrightarrow{[l_s, l_r]} K, P :: Q_2$ avec

$$Q_1 \mathrel{\widehat{=}} \langle p_s : E_s[(\mathtt{send}\ k^{l_0}\ v^{l_1})^{l_s}]\rangle :: \langle p_r : E_r[(\mathtt{receive}\ k^{l_2})^{l_r}]\rangle$$

$$Q_2 \mathrel{\widehat{=}} \langle p_s : E_s[v^{l_1}]\rangle :: \langle p_r : E_r[v^{l_1}]\rangle$$

La preuve se décompose en trois parties. Nous montrons que (*i*) $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}'_{p_s} \vdash E_s[v^{l_1}]$

pour un automate $\widehat{\mathcal{A}}'_{p_s}$ adéquat, (*ii*) $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}'_{p_r} \vdash E_r[v^{l_1}]$ pour un automate adéquat $\widehat{\mathcal{A}}'_{p_r}$, et finalement (*iii*) qu'en utilisant la nouvelle collection d'automates $\widehat{\mathcal{G}}'$ obtenue en substituant $\widehat{\mathcal{A}}'_{p_s}$ et $\widehat{\mathcal{A}}'_{p_r}$ à $\widehat{\mathcal{A}}_{p_s}$ et $\widehat{\mathcal{A}}_{p_r}$ dans $\widehat{\mathcal{G}}$ nous avons

$$\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}^{S\prime}_{\otimes} \models^S P :: Q_2$$

et

$$\widehat{\mathcal{A}}^{S\prime}_{\otimes} \widehat{=} \mathrm{A}^S_{\otimes}(\widehat{\mathcal{G}}') \overset{l_s, l_r}{\lessdot} \mathrm{A}^S_{\otimes}(\widehat{\mathcal{G}})$$

(i) $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}'_{p_s} \vdash E_s[v^{l_1}]$ pour un automate adéquat $\widehat{\mathcal{A}}'_{p_s}$. Par définition de $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}^S_{\otimes} \models^S P :: Q_1$ nous avons

$$\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_{p_s} \vdash E_s[(\mathtt{send}\ k^{l_0}\ v^{l_1})^{l_s}] \tag{A.7}$$

pour un certain $\widehat{\mathcal{A}}_p \in \widehat{\mathcal{G}}$ avec $\widehat{\mathcal{A}}_{p_s} \widehat{=} \widehat{\mathcal{A}}_s[\widehat{\mathcal{A}}'_s]$ et d'après le lemme 44

$$\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}'_s \vdash (\mathtt{send}\ k^{l_0}\ v^{l_1})^{l_s}$$

Aussi, $\widehat{\mathcal{C}}(l_1) \subseteq \widehat{\mathcal{C}}(l_s)$, $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_1 \vdash v^{l_1}$ et (A.7) impliquent en utilisant le lemme 45

$$\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}'_{p_s} \vdash E_s[v^{l_1}]$$

avec

$$\widehat{\mathcal{A}}'_{p_s} \widehat{=} \widehat{\mathcal{A}}_s[\widehat{\mathcal{A}}_1] \tag{A.8}$$

Remarquons que nous avons

$$\widehat{\mathcal{A}}'_{p_s} \overset{l_s}{\lessdot} \widehat{\mathcal{A}}_{p_s} \tag{A.9}$$

(ii) $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}'_{p_r} \vdash E_r[v^{l_1}]$ pour un automate adéquat $\widehat{\mathcal{A}}'_{p_r}$. A nouveau nous avons

$$\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}'_{p_r} \vdash E_r[(\mathtt{receive}\ k^{l_2})^{l_r}] \tag{A.10}$$

avec $\widehat{\mathcal{A}}'_{p_r} \widehat{=} \widehat{\mathcal{A}}_r[\widehat{\mathcal{A}}'_r]$. D'après le lemme 44

$$\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}'_r \vdash (\mathtt{receive}\ k^{l_2})^{l_r} \tag{A.11}$$

Nous avons $[\![\mathrm{E}_{l_1} \overset{l_s}{\multimap} \mathrm{E}_{l_s}]\!] \sqsubseteq \widehat{\mathcal{A}}_{p_s}$, $[\![\mathrm{E}_{l_2} \overset{l_r}{\multimap} \mathrm{E}_{l_r}]\!] \sqsubseteq \widehat{\mathcal{A}}_{p_r}$, $\mathrm{Start}(\widehat{\mathcal{A}}_{p_s}) \overset{\varepsilon}{\multimap}_* \mathrm{E}_{l_1}$ in $\widehat{\mathcal{A}}_{p_s}$ et $\mathrm{Start}(\widehat{\mathcal{A}}_{p_r}) \overset{\varepsilon}{\multimap}_* \mathrm{E}_{l_2}$ in $\widehat{\mathcal{A}}_{p_r}$. Donc par définition de $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}^S_{\otimes} \models^S P :: Q_1$, il existe un état $q \in Q^S$, $q = (\nu_1, \ldots, \mathrm{E}_{l_1}, \ldots, \mathrm{E}_{l_2}, \ldots, \nu_k)$ avec

$$[\![\omega \overset{l_s, l_r}{\multimap} \omega']\!] \sqsubseteq \mathrm{A}^S_{\otimes}(\widehat{\mathcal{G}})$$

et puisque $\widehat{\mathcal{C}}(l_0) \cap \widehat{\mathcal{C}}(l_2) \neq \varnothing$, nous obtenons $\widehat{\mathcal{C}}(l_1) \subseteq \widehat{\mathcal{C}}(l_r)$. D'après le lemme 45, $\widehat{\mathcal{C}}(l_1) \subseteq \widehat{\mathcal{C}}(l_r)$, $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_1 \vdash v^{l_1}$ et (A.10) impliquent

$$\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}'_{p_r} \vdash E_r[v^{l_1}]$$

avec

$$\widehat{\mathcal{A}}'_{p_r} \mathrel{\widehat{=}} \widehat{\mathcal{A}}[\widehat{\mathcal{A}}_1] \tag{A.12}$$

Notons que

$$\widehat{\mathcal{A}}'_{p_r} \overset{l_r}{\preccurlyeq} \widehat{\mathcal{A}}_{p_r} \tag{A.13}$$

(iii) $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}^{S\prime}_{\otimes} \models^S P :: Q_2$ pour un certain $\widehat{\mathcal{G}}'$ tel que

$$\widehat{\mathcal{A}}^{S\prime}_{\otimes} \overset{l_s,l_r}{\preccurlyeq} \widehat{\mathcal{A}}^{S}_{\otimes}$$

Soit $\widehat{\mathcal{G}}'$ la collection d'automates obtenue en substituant $\widehat{\mathcal{A}}'_{p_s}$ et $\widehat{\mathcal{A}}'_{p_r}$ tel qu'ils sont définis dans les équations (A.8) et (A.12) à $\widehat{\mathcal{A}}_{p_s}$ et $\widehat{\mathcal{A}}_{p_r}$ dans $\widehat{\mathcal{G}}$. Il existe $\mathrm{Q}^S \in Q^S$ et $\mathrm{Q}^{S\prime} \in Q^S$ avec

$$\mathrm{Q}^S \mathrel{\widehat{=}} (\nu_1, \ldots, \nu_{l_0}, \ldots, \nu_{l_2}, \ldots, \nu_n)$$

$$\mathrm{Q}^{S\prime} \mathrel{\widehat{=}} (\nu_1, \ldots, \nu_{l_s}, \ldots, \nu_{l_r}, \ldots, \nu_n)$$

En conséquence, $[\![\mathrm{Q}^S \overset{l_s,l_r}{\multimap} \mathrm{Q}^{S\prime}]\!] \sqsubseteq \widehat{\mathcal{A}}^S_{\otimes}$ et d'après les équations (A.9) et (A.13) nous obtenons

$$\widehat{\mathcal{A}}^{S\prime}_{\otimes} \mathrel{\widehat{=}} \mathrm{A}^S_{\otimes}(\widehat{\mathcal{G}}') \overset{l_s,l_r}{\preccurlyeq} \mathrm{A}^S_{\otimes}(\widehat{\mathcal{G}})$$

De plus,

$$\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}^{S\prime}_{\otimes} \models^S P :: Q_2$$

–

$$\frac{q \notin \mathrm{dom}(P)}{K, P :: \langle p : E[(\mathtt{fork}\, e_0^{l_0})^l] \rangle \xrightarrow{[l,l]} K, P :: \langle p : E[()^l] \rangle :: \langle q : e_0^{l_0} \rangle}$$

$\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}^S_{\otimes} \models^S P :: \langle p : E[(\mathtt{fork}\, e_0^{l_0})^l] \rangle$ implique $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_p \vdash E[(\mathtt{fork}\, e_0^{l_0})^l]$ avec

$$\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}} \vdash (\mathtt{fork}\, e_0^{l_0})^l \qquad \widehat{\mathcal{A}}_p = \widehat{\mathcal{A}}'_p[\widehat{\mathcal{A}}]$$
$$\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_0 \vdash e_0^{l_0} \qquad \widehat{\mathcal{A}}_0 \sqsubseteq \widehat{\mathcal{A}}$$
$$[\![\mathrm{B}_l \overset{l}{\multimap} \mathrm{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}} \qquad [\![\mathrm{B}_l \overset{l}{\multimap} \mathrm{B}_{l_0}]\!] \sqsubseteq \widehat{\mathcal{A}}$$

D'une part, par définition de $\widehat{\mathcal{A}}^S_\otimes$, puisque $\text{Start}(\widehat{\mathcal{A}}_p) = \text{B}_l$, il existe un état

$$q = (\nu_1, \ldots, \text{B}_l, \ldots, \nu_k) \in \widehat{\mathcal{A}}^S_\otimes$$

et $\widehat{\mathcal{A}}_p$ tel que $[\![\text{B}_l \overset{l}{\multimap} \text{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}}_p$, $[\![\text{B}_l \overset{l}{\multimap} \text{B}_{l_0}]\!] \sqsubseteq \widehat{\mathcal{A}}_p$. So,

$$[\![q \overset{l,l}{\multimap} (\nu_1, \ldots, \text{E}_l, \ldots, \nu_k, \text{B}_{l_0})]\!] \sqsubseteq \widehat{\mathcal{A}}^S_\otimes \tag{A.14}$$

et par examen de $\widehat{\mathcal{A}}_p$,

$$\text{Start}(\widehat{\mathcal{A}}_p) \overset{\varepsilon}{\multimap}_* q \tag{A.15}$$

D'autre part, d'après le lemme 45,

$$\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}'_p[\widehat{\mathcal{A}}'] \vdash E[(\,)^l] \tag{A.16}$$

avec $[\![\text{B}_l \overset{\varepsilon}{\multimap} \text{E}_l]\!] \sqsubseteq \widehat{\mathcal{A}}'$ et nous avons

$$\widehat{\mathcal{A}}' \overset{l}{\lessdot} \widehat{\mathcal{A}} \tag{A.17}$$

Par ailleurs,

$$\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_0 \vdash e^{l_0}_0 \tag{A.18}$$

et

$$\widehat{\mathcal{A}}_0 \overset{l}{\lessdot} \widehat{\mathcal{A}}_0 \tag{A.19}$$

Finalement, soit $\widehat{\mathcal{A}}^{S\prime}_\otimes \mathrel{\widehat{=}} \text{A}^S_\otimes(\widehat{\mathcal{G}}')$ où $\widehat{\mathcal{G}}'$ est la collection construite à partir de $\widehat{\mathcal{G}}$ et substituant $\widehat{\mathcal{A}}'_p[\widehat{\mathcal{A}}']$ à $\widehat{\mathcal{A}}_p$ et en ajoutant $\widehat{\mathcal{A}}_0$. D'après les équations (A.14), (A.15), (A.17), et (A.19), nous obtenons

$$\widehat{\mathcal{A}}^{S\prime}_\otimes \mathrel{\widehat{=}} \text{A}^S_\otimes(\widehat{\mathcal{G}}') \overset{l,l}{\lessdot} \widehat{\mathcal{A}}^S_\otimes$$

et

$$\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}^{S\prime}_\otimes \models^S P :: \langle p : E[()^l] \rangle :: \langle q : e^{l_0}_0 \rangle$$

CQFD

A.5 Relation entre les automates produits et réduits

Proposition 35 (Relation entre les automates produits et réduits) *Soit P un groupe de processus et $\widehat{\mathcal{G}} = (\widehat{\mathcal{A}}_p)_{p \in Dom(P)}$ une collection d'automates tels que pour tout $\langle p : e^l \rangle \in P$, $\widehat{\mathcal{C}}, \widehat{\mathcal{E}}, \widehat{\mathcal{A}}_p \vdash e^l$. La propriété*

$$\text{A}^S_\otimes(\widehat{\mathcal{G}}) \lessdot \text{A}^R_\otimes(\widehat{\mathcal{G}}) \tag{A.20}$$

est satisfaite. □

PREUVE

La preuve est par récurrence sur la longueur k des chemins dans les automates. Soient $\widehat{\mathcal{A}}^S = \mathrm{A}^S_{\otimes}(\widehat{\mathcal{G}})$ et $\widehat{\mathcal{A}}^R = \mathrm{A}^R_{\otimes}(\widehat{\mathcal{G}})$. Pour chaque chemin dans $\widehat{\mathcal{A}}^S$ nous mettons en évidence l'existence d'un chemin dans $\widehat{\mathcal{A}}^R$ avec les mêmes transitions étiquetées.

– Considérons les transitions $\mathrm{Start}(\widehat{\mathcal{A}}^S) \overset{\varepsilon}{\multimap}_* q_1^S \overset{l_s,l_r}{\multimap} q_2^S$ dans $\widehat{\mathcal{A}}^S$ avec

$$\begin{aligned} \mathrm{Start}(\widehat{\mathcal{A}}^S) &= (\nu_1, \ldots, \nu_s, \ldots, \nu_r, \ldots, \nu_n) \\ q_1^S &= (\nu'_1, \ldots, \nu'_s, \ldots, \nu'_r, \ldots, \nu'_n) \\ q_2^S &= (\nu''_1, \ldots, \nu''_s, \ldots, \nu''_r, \ldots, \nu''_n) \end{aligned}$$

Par comparaison de (i) dans les définitions 27 et 32, nous avons

$$\{\nu_s, \nu_r\} \subseteq L(q_0^R)$$

et puisque $\mathrm{Start}(\widehat{\mathcal{A}}^S) \overset{\varepsilon}{\multimap}_* q_1^S$, par l'équation (4.7) nous obtenons

$$\{\nu'_s, \nu'_r\} \subseteq L(q_0^R)$$

De plus, puisque $[\![q_1^S \overset{l_s,l_r}{\multimap} q_2^S]\!] \sqsubseteq \widehat{\mathcal{A}}^S$, il existe $\widehat{\mathcal{A}}_p \sqsubseteq \widehat{\mathcal{G}}$, $\widehat{\mathcal{A}}'_p \sqsubseteq \widehat{\mathcal{G}}$, s et r, $s \neq r$, $1 \leq s, r \leq n$ tels que

$$\begin{cases} [\![\nu'_s \overset{l_s}{\multimap} \nu''_s]\!] \sqsubseteq \widehat{\mathcal{A}}_p \\ [\![\nu'_r \overset{l_r}{\multimap} \nu''_r]\!] \sqsubseteq \widehat{\mathcal{A}}_{p'} \\ (\texttt{send}\ e_0^{l_0}\ e_1^{l_1})^{l_s} \in \mathrm{prg} \\ (\texttt{receive}\ e_2^{l_2})^{l_r} \in \mathrm{prg} \end{cases}$$

et il existe un état q_0^R tel que

$$\{\nu_s, \nu_r\} \subseteq L(q_0^R)$$

Aussi les conditions décrite dans l'équation (4.8) dans la Définition 32 sont satisfaites et nous obtenons

$$[\![q_0^R \overset{l_s,l_r}{\multimap} q_{l_s,l_r}]\!] \sqsubseteq \widehat{\mathcal{A}}^R$$

Par conséquent les deux automates possèdent un chemin partant de leurs états initiaux dont la première transition différente de ε est (l_s, l_r). Les `fork` sont traités de manière analogue.

– Supposons que la propriété est vérifiée pour les chemin de longueur $\leq k$ et que

$$q_k^S \overset{l_s,l_r}{\multimap} {q'}_k^S \overset{\varepsilon}{\multimap}_* q_{k+1}^S \overset{l'_s,l'_r}{\multimap} {q'}_{k+1}^S$$

avec $q_k^S \in \delta^{S*}(q_0^S, c_1 \dots c_{k-1})$ et

$$\begin{aligned} q_k^S &= (\nu_{k,1}, \dots, \nu_{k,s}, \dots, \nu_{k,r}, \dots, \nu_{k,n}) \\ q_{k+1}^S &= (\nu_{k+1,1}, \dots, \nu_{k+1,s'}, \dots, \nu_{k+1,r'}, \dots, \nu_{k+1,n}) \\ {q'}_k^S &= (\nu'_{k,1}, \dots, \nu'_{k,s}, \dots, \nu'_{k,r}, \dots, \nu'_{k,n}) \\ {q'}_{k+1}^S &= (\nu'_{k+1,1}, \dots, \nu'_{k+1,s'}, \dots, \nu'_{k+1,r'}, \dots, \nu'_{k+1,n}) \end{aligned}$$

Puisque $[\![q_{k+1}^S \overset{l'_s, l'_r}{\multimap} {q'}_{k+1}^S]\!] \sqsubseteq \widehat{\mathcal{A}}^S$,

$$\begin{aligned} &\exists \widehat{\mathcal{A}}_p \in \widehat{\mathcal{G}} \ : \ [\![\nu_{k+1,s'} \overset{l'_s}{\multimap} \nu'_{k+1,s'}]\!] \sqsubseteq \widehat{\mathcal{A}}_p \text{ et } (\texttt{send}\ e_0^{l_0}\ e_1^{l_1})^{l'_s} \in \text{prg} \\ &\exists \widehat{\mathcal{A}}_{p'} \in \widehat{\mathcal{G}} \ : \ [\![\nu_{k+1,r'} \overset{l'_r}{\multimap} \nu'_{k+1,r'}]\!] \sqsubseteq \widehat{\mathcal{A}}_{p'} \text{ et } (\texttt{receive}\ e_2^{l_2})^{l'_r} \in \text{prg} \end{aligned}$$

Afin de montrer que les conditions données dans l'équation (4.8) dans la définition 32 sont satisfaites, nous devons mettre en évidence l'existence d'un état $q \in Q^R$ tel que

$$\{\nu_{k+1,s'}, \nu_{k+1,r'}\} \subseteq L(q)$$

Nous distinguons deux cas.

– Si $s = s'$, c.à.d.. les deux dernières émissions impliquent le même processus. Puisque dans $\widehat{\mathcal{A}}^S$

$$q_k^S \overset{l_s, l_r}{\multimap} {q'}_k^S \overset{\varepsilon}{\multimap}_* {q'}_{k+1}$$

par définition de $\text{A}_{\otimes}^S(\widehat{\mathcal{G}})$, il existe $\widehat{\mathcal{A}}_p \in \widehat{\mathcal{G}}$ tel que $\widehat{\mathcal{A}}_p$ contient les transitions

$$\nu_{k,s} \overset{l_s}{\multimap} \nu'_{k,s} \overset{\varepsilon}{\multimap}_* \nu_{k+1,s}$$

A nouveau, nous distinguons deux cas, selon que $r = r'$ ou pas.

– Si $r = r'$, alors, par définition de $\text{A}_{\otimes}^S(\widehat{\mathcal{G}})$, il existe $\widehat{\mathcal{A}}'_p \in \widehat{\mathcal{G}}$ tel que $\widehat{\mathcal{A}}'_p$ contient les transitions

$$\nu_{k,r} \overset{l_r}{\multimap} \nu'_{k,r} \overset{\varepsilon}{\multimap}_* \nu_{k+1,r}$$

Par récurrence il y a deux états q et q_{l_s,l_r} dans Q^R tels que

$$[\![q \overset{l_s, l_r}{\multimap} q_{l_s,l_r}]\!] \sqsubseteq \widehat{\mathcal{A}}^R \qquad \{\nu_{k',s}, \nu_{k',r}\} \subseteq L(q_{l_s,l_r})$$

Aussi, d'après l'équation (4.7),

$$\{\nu_{k+1,s}, \nu_{k+1,r}\} \subseteq L(q_{l_s,l_r})$$

Donc par l'équation (4.8) nous avons

$$[\![q_{l_s,l_r} \overset{l'_s,l'_r}{\multimap} q_{l'_s,l'_r}]\!] \sqsubseteq \widehat{\mathcal{A}}^R$$

et par conséquent il y a les même chemins de longueur $k+1$ dans $\widehat{\mathcal{A}}^R$ que dans $\widehat{\mathcal{A}}^S$.

– Si $r \neq r'$, alors par hypothèse, il y a dans $\widehat{\mathcal{A}}^S$ le chemin

$$q_k^S \overset{l_s,l_r}{\multimap} {q'}_k^S \overset{\varepsilon}{\multimap}_* q_{k+1}^S \overset{l'_s,l'_r}{\multimap} {q'}_{k+1}^S \tag{A.21}$$

avec $q_k^S \in \delta^{S*}(q_0^S, c_1 \dots c_{k-1})$ et

$$\begin{aligned} q_k^S &= (\nu_{k,1}, \dots, \nu_{k,s}, \dots, \nu_{k,r}, \dots, \nu_{k,r'}, \dots \nu_{k,n}) \\ {q'}_k^S &= (\nu'_{k,1}, \dots, \nu'_{k,s}, \dots, \nu'_{k,r}, \dots, \nu_{k,r'}, \dots, \nu'_{k,n}) \\ q_{k+1}^S &= (\nu_{k+1,1}, \dots, \nu_{k+1,s'}, \dots, \nu_{k+1,r'}, \dots, \nu_{k+1,r'}, \dots, \nu_{k+1,n}) \\ {q'}_{k+1}^S &= (\nu'_{k+1,1}, \dots, \nu'_{k+1,s'}, \dots, \nu'_{k+1,r'}, \dots, \nu'_{k+1,r'}, \dots, \nu'_{k+1,n}) \end{aligned}$$

D'après l'équation (A.21), la définition 32, et les hypothèses de récurrence, il existe deux états q et q_{l_s,l_r} tels que

$$\{\nu_{k,s}, \nu_{k,r}, \nu_{k,r'}\} \subseteq L(q) \qquad \{\nu'_{k,s}, \nu'_{k,r}, \nu_{k,r'}\} \subseteq L(q)$$

et

$$[\![q \overset{l_s,l_r}{\multimap} q_{l_s,l_r}]\!] \sqsubseteq \widehat{\mathcal{A}}^R$$

De plus, d'après l'équation (A.21) il existe $\widehat{\mathcal{A}}_p \in \widehat{\mathcal{G}}$ qui contient les transitions $\nu_{k,r'} \overset{\varepsilon}{\multimap}_* \nu_{k+1,r'}$. Donc les conditions de l'équation (4.8) sont satisfaites, nous avons

$$\{\nu_{k+1,s}, \nu_{k+1,r'}\} \subseteq L(q_{l_s,l_r})$$

et par conséquent $[\![q_{l_s,l_r} \overset{l'_s,l'_r}{\multimap} q_{l'_s,l'_r}]\!] \sqsubseteq \widehat{\mathcal{A}}^R$. Il y a les mêmes chemins de longueur $k+1$ dans $\widehat{\mathcal{A}}^R$ que dans $\widehat{\mathcal{A}}^S$.

– $s \neq s'$: Ce cas est analogue au cas $r \neq r'$.

CQFD

Annexe B

Programmes relatifs au chapitre 5

Nous présentons ici les versions complètes des programmes décrits dans le chapitre 5 concernant les projections de Futamura dans le cadre de notre noyau non typé du langage Concurrent ML.

B.1 L'évaluateur partiel annoté

```
define Pev_a = rec Pev e ->
    if (isVar? @ e) then (getEnv @ e) else
    if (isLambda? @ e) then
      (_fun value -> (eval @ (substitute @ (fetchLambdaBody @ e)
                                           value
                                           (fetchLambdaId @ e)))) else
    if (isRec? @ e) then
      (_fun rvalue -> (Pev @ (substitute @ (substitute @ (fetchRecBody @ e)
                                                         rvalue
                                                         (fetchRecArg @ e)
                                            )
                                            e
                                            (fetchRecId @ e)))) else
    if (isApp? @ e) then
      ((eval @ (fetchAppFun @ e)) _@ (eval @ (fetchAppArg @ e))) else
    if (isCond? @ e) then
      _if (Pev @ (fetchCond @ e)) _then
        (Pev @ (fetchThen @ e))
      _else
        (Pev @ (fetchElse @ e)) else
    if (isConst? @ e) then lift e else
    if (isLift? @ e) then makeLift @ e else
    if (isChannel? @ e) then (createNewChan _@ ()) else
    if (isFork? @ e) then (_fork (eval @ (fetchChild @ e))) else
```

```
    if (isReceive? @ e) then _receive (eval @ (fetchRecChan @ e)) else
    if (isSend? @ e) then _send (eval @ (fetchSendChan @ e))
                                (eval @ (fetchSendVal @ e))

    if (isDLambda? @ e) then
      (_fun newVar ->
          (buildLambda _@ newVar (Pev @ (substitute @ (fetchLambdaBody @ e)
                                                      newVar
                                                      (fetchLambdaId @ e))))
      ) _@ (genSymb @ ()) else
    if (isDApp? @ e) then
      (buildApp _@ (Pev @ (fetchAppFun @ e)) (Pev @ (fetchAppArg @ e))) else
    if (isDRec? @ e) then
      (buildRec _@ (fetchDRecId @ e)
                   (fetchDRecArg @ e)
                   (Pev @ (fetchRecBody @ e))) else
    if (isDCond? @ e) then
      (buildCond _@ (Pev @ (fetchCond @ e))
                    (Pev @ (fetchThen @ e))
                    (Pev @ (fetchElse @ e))) else
    if (isDConst? @ e) then (buildConst _@ (fetchConst @ e)) else
    if (isDChannel? @ e) then dynChanCode else
    if (isDFork? @ e) then (buildFork _@ (Pev @ (fetchChild @ e))) else
    if (isDReceive? @ e) then
      (buildReceive _@ (Pev @ (fetchRecChan @ e))) else
    if (isDSend? @ e) then
      (buildSend _@ (Pev @ (fetchSendChan @ e))
                    (Pev @ (fetchSendVal @ e)))
    else (buildError @ e) ;
```

B.2 Le générateur de compilateurs

```
define Cogen = rec Cogen e ->
   if (isVar? @ e) then
     ((buildApp @ getEnv)
                @ e) else
   if (isLambda? @ e) then
     ((fun Symb2 ->
        ((buildLambda @ Symb2)
                      @ (Cogen @ (((substitute @ (fetchLambdaBody @ e))
                                                 @ Symb2)
                                                 @ (fetchLambdaId @ e)))
      )) @ (genSymb @ ())) else
   if (isRec? @ e) then
```

```
  ((fun Symb3 -> ((buildLambda
                   @ Symb3)
                   @ (Cogen @ (((substitute
                                 @ (((substitute @ (fetchRecBody @ e))
                                                 @ Symb3)
                                                 @ (fetchRecArg @ e)))
                                 @ e)
                                 @ (fetchRecId @ e)))
   )) @ (genSymb @ ())) else
if (isApp? @ e) then
  ((buildApp @ (Cogen @ (fetchAppFun @ e)))
             @ (Cogen @ (fetchAppArg @ e))) else
if (isCond? @ e) then
  (((buildCond @ ((buildApp @ getBase)
                          @ (Cogen @ (fetchCond @ e))))
               @ (Cogen @ (fetchThen @ e)))
               @ (Cogen @ (fetchElse @ e))) else
if (isConst? @ e) then e else
if (isLift? @ e) then makeLift @ e else
if (isChannel? @ e) then ((buildApp @ createNewChan)
                                    @ ()) else
if (isFork? @ e) then
  (buildFork @ (Cogen @ (fetchChild @ e))) else
if (isReceive? @ e) then
  (buildReceive @ (Cogen @ (fetchRecChan @ e))) else
if (isSend? @ e) then
  ((buildSend @ (Cogen @ (fetchSendChan @ e)))
              @ (Cogen @ (fetchSendVal @ e))) else
if (isDLambda? @ e) then
  ((buildApp @
     ((fun Symb4 ->
        ((buildLambda @ Symb4)
                       @ ((buildApp @ ((buildApp @ buildLambda)
                                                  @ Symb4))
                                    @ (Cogen @ (((substitute
                                                  @ (fetchLambdaBody @ e))
                                                  @ Symb4)
                                                  @ (fetchLambdaId @ e))))
                ) @ (genSymb @ ())))) @ ((buildApp @ genSymb)
                                                  @ ())) else
if (isDApp? @ e) then
  ((buildApp @ ((buildApp @ buildApp)
                          @ (Cogen @ (fetchAppFun @ e))))
             @ (Cogen @ (fetchAppArg @ e))) else
if (isDRec? @ e) then
```

```
  ((buildApp @ ((buildApp @ ((buildApp @ buildRec)
                                      @ (fetchRecId @ e)))
                           @ (fetchRecArg @ e)))
             @ (Cogen @ (fetchRecBody @ e))) else
if (isDCond? @ e) then
  ((buildApp @ ((buildApp @ ((buildApp @ buildCond)
                                      @ (Cogen @ (fetchCond @ e))))
                           @ (Cogen @ (fetchThen @ e))))
             @ (Cogen @ (fetchElse @ e))) else
if (isDConst? @ e) then
 ((buildApp @ buildConst)
            @ (fetchConst @ e)) else
if (isDChannel? @ e) then buildChanCode else
if (isDFork? @ e) then
  ((buildApp @ buildFork)
             @ (Cogen @ (fetchChild @ e))) else
if (isDReceive? @ e) then
  ((buildApp @ buildReceive)
             @ (Cogen @ (fetchRecChan@ e))) else
if (isDSend? @ e) then
  ((buildApp @ ((buildApp @ buildSend)
                           @ (Cogen @ (fetchSendChan @ e))))
             @ (Cogen @ (fetchSendVal @ e)))
else (buildError @ e) ;
```

Résumé

L'évaluation partielle est une technique qui permet d'exécuter un programme dont seule une partie des arguments est connue. Seules les opérations dont toutes les opérandes sont connues sont réalisées, alors que les autres opérations sont gelées. Nous obtenons ainsi un nouveau programme qui est composé des instructions n'ayant pas pu être exécutées et dans lequel les résultats des calculs effectués ont été insérés. L'évaluateur partiel utilise les résultats produits par une analyse des temps de liaison, afin de savoir quelles parties d'un programme dépendent uniquement d'arguments connus et peuvent être exécutées.

Dans cette thèse, nous nous intéressons à l'évaluation partielle et à l'analyse statique de systèmes de processus mobiles. Ces systèmes sont représentés par des programmes contenant des instructions de communication synchrones sur des canaux nommés. La mobilité est modélisée par le fait que des noms de canaux peuvent être créés dynamiquement et être communiqués. L'évaluation partielle consiste alors à exécuter les communications dépendant uniquement de données connues.

Tout d'abord, nous étudions les conséquences de l'absence de certaines communications dans les programmes évalués partiellement. Nous indiquons les conditions nécessaires pour que les programmes obtenus soient corrects, au sens où ils permettent les mêmes interactions avec le reste du système que ne le permettaient les programmes initiaux.

Ensuite, nous présentons une analyse statique qui calcule une approximation de la topologie des communications (dépendances entre les communications) réalisées par les programmes. Cette analyse est nécessaire pour déterminer précisément les communications qui peuvent être évaluées partiellement. C'est une analyse de flot de contrôle qui construit un automate produit réduit afin de décrire une approximation des interactions possibles entre les processus.

Enfin, nous présentons un évaluateur partiel et une analyse des temps de liaison (binding time analysis) pour un noyau de langage fonctionnel parallèle. L'évaluateur partiel est auto-applicable et permet, par les projections de Futamura, d'obtenir automatiquement un compilateur et un générateur de compilateurs. L'analyse des temps de liaison utilise les résultats de l'analyse de flot de contrôle décrite précédemment.

Les principaux résultats obtenus sont un évaluateur partiel auto-applicable pour un langage parallèle basé sur des communications. Les améliorations apportées aux analyses de temps de liaison et contrôle de flot concernent la prise en compte des dépendances entre les différentes communications.

Mots clés : Evaluation partielle, systèmes parallèles, analyse de flot de controle, analyse des temps de liaison, générateur de compilateurs.

Abstract

Partial evaluation is a technical tool that enables to execute a program for which only one part of the data is known. Only the instructions depending on the known data are executed, while the others are frozen. We obtain a new program, made of the pieces of code which could not be executed, and of the results of the achieved computations. A partial evaluator uses the results of a binding time analysis, in order to know which parts of a program solely depend on known data and can be executed.

In this thesis, we are interested in partial evaluation and static analysis of systems made of mobile processes. These systems are described by programs containing synchronous communication instructions over channels. Mobility is modelized by the ability to dynamically create and communicate channel names. In this context, partial evaluation consists of executing the communications depending on known data.

First, we study the consequences of the lack of some communications in the partially evaluated programs. We define the conditions required to ensure the correctness of the output programs, mainly in the sense that they realize the same interactions with the remainder of the system than the initial programs.

Second, we introduce a static analysis able to compute an approximation of the communication topology of programs (dependencies between the communications). This analysis is needed to determine which communications can be executed at partial evaluation time. It is a control flow analysis which builds a reduced product automaton, in order to compute an approximation of the possible interactions between the processes.

Finaly, we introduce a partial evaluator and a binding time analysis for a kernel of a concurrent functional language. The partial evaluator is self-applicable, and allows, by the Futamura's projections, to automatically compute a compiler and a compiler generator. The binding time analysis is based on the results of the control flow analysis previously described.

The main results we obtain are a self-applicable partial evaluator for a concurrent langage based on communications. The improvements of the binding time analyses and control flow analyses are due to the fact that we use the dependency informations between communications.

Keywords : Partial evaluation, concurrent systems, control flow analysis, binding time analysis, compiler generator.