Vérification symbolique de modèles à l'aide de systèmes de ré-écriture dédiés

Duy-Tùng Nguyên

UNIVERSITÉ D'ORLÉANS

ÉCOLE DOCTORALE
SCIENCES ET TECHNOLOGIES

LABORATOIRE : LIFO

THÈSE présentée par :

Duy-Tùng NGUYÊN

soutenue le : **21 octobre 2010**

pour obtenir le grade de : **Docteur de l'université d'Orléans**

Discipline/ Spécialité : **Informatique**

Vérification symbolique de modèles à l'aide de systèmes de ré-écritures dédiés

THÈSE dirigée par :

Jean-Michel COUVREUR	Professeur, Université d'Orléans

RAPPORTEURS :

Didier BUCHS	Professeur, Université de Genève
Fabrice KORDON	Professeur, Université de Paris VI

JURY :

Yohan BOICHUT	Maître de conférences, Université d'Orléans
Ahmed BOUAJJANI	Professeur, Université de Paris VII
Didier BUCHS	Professeur, Université de Genève
Jean-Michel COUVREUR	Professeur, Université d'Orléans
Serge HADDAD	Professeur, ENS Cachan
Fabrice KORDON	Professeur, Université de Paris VI

Remerciements

Je tiens en tout premier lieu à remercier M. Didier Buchs et M. Fabrice Kordon, pour m'avoir fait l'honneur de rapporter cette thèse, pour l'intérêt qu'ils ont porté à mon travail, et leurs précieux commentaires et remarques. Ma gratitude s'adresse aussi à M. Yohan Boichut, M. Ahmed Bouajjani, M. Serge Haddad et M. Grégoire Sutre pour avoir accepté de faire partie du jury de cette thèse. Malheureusement, M. Grégoire Sutre ne peux pas participer de la soutenance à cause du problème de transport.

Je remercie également M. Jean-Michel Couvreur, d'abord encadrant de stage de DEA au Laboratoire Bordelais de Recherche en Informatique (LaBRI), l'université de Bordeaux I, puis directeur de thèse à l'université d'Orléans, pour avoir suivi et supporté mon travail au cours de ces années.

Je tiens également à remercier M. Yohan Boichut. Son expérience dans le domaine du systèmes de ré-écriture a été les plus précieux.

Cette thèse n'aurait sans doute pas été une expérience aussi enrichissante et agréable sans la convivialité des membres du Laboratoire d'Informatique Fondamentale d'Orléans (LIFO), l'université d'Orléans. Ils sont trop nombreux pour que je les cite tous, mais je les remercie chaleureusement.

Merci à mes parents, à ma soeur ainsi et à ma copine pour leur appui moral. Et merci encore à tous mes amis que je n'ai pas cité ici, mais qui mériteraient de l'être.

Chapitre 1

Introduction générale

Avant-propos

Les techniques de **vérification** ont déjà plus de trente ans [Clarke 2000]. Elles ont été développées dans un premier temps par des équipes de chercheurs et sont de plus en plus utilisées dans le milieu industriel pour l'analyse d'une grande variété de systèmes (systèmes matériels, logiciels, systèmes réactifs, systèmes temps réel). Il est maintenant prouvé que ces techniques sont efficaces et sont fréquemment utilisées pour détecter des bogues dans des cas industriels. De nombreuses études sont en cours pour élargir leurs champs d'applications et améliorer leur efficacité. Ceci nous conduit à penser que les applications industrielles continuent à se multiplier de manière significative dans les prochaines années.

Dans les années 90, les industries des composants électroniques, dans leur recherche d'outils pour améliorer le niveau de confiance de leurs produits, ont adopté les **Diagrammes de Décisions Binaires** (BDD) [Bryant 1986, Bryant 1992], une structure de la **vérification symbolique**, pour traiter des composants de plus en plus complexes. Les BDD sont des structures codant des fonctions booléennes. Ils peuvent être vus comme des arbres où les états représentent des choix de valeurs de variables booléennes ; un ordre total sur les variables garantit l'unicité du codage d'une fonction. Les techniques de partage de structures, combinées à des méthodes de réductions, conduisent à des implémentations extrêmement efficaces en pratique [Minato 1990, Hulgaard 1999]. Ainsi, des vérifications exhaustives ont pu être réalisées sur des systèmes comprenant des milliards d'états [Minato 1990, Hulgaard 1999] (Voir cf. l'article *"Symbolic Model Checking :* 10^{20} *states and Beyond"* [Burch 1992]). Le pouvoir d'expression des BDD est suffisant pour manipuler une grande classe de systèmes finis [Burch 1992]. Comme le nombre de variables des systèmes étudiés est un facteur critique, de nombreuses structures *à la BDD* ont été proposées [Miner 1999, Ciardo 2000]. D'autres structures ont plutôt cherché à étendre le domaine d'application de ces techniques [Bahar 1997, Gupta 1993, Gupta 1994, Mauborgne 1999, Reffel 1999, Mauborgne 2000].

Dans le cadre de projets industriels, dans [Couvreur 2002], ils ont conçu une nouvelle structure *à la BDD*, les **Diagrammes de Décisions de Données** (DDD). L'objectif était de fournir un outil flexible qui peut être autant que possible adapté pour la vérification de tout type de modèles et qui offre des capacités de traitement similaires aux BDD. A la différence des BDD, les opérations sur ces structures ne sont pas pré-définies, mais une classe d'opérateurs, appelée homomorphismes, est introduite pour permettre à un utilisateur de concevoir ses propres opérations. Dans

ce modèle, les variables ne sont pas booléennes ; elles prennent leurs valeurs dans des domaines non nécessairement bornés. Une autre caractéristique intéressante est qu'aucun ordre sur les variables est présupposé dans la définition. De plus, une variable peut apparaître plusieurs fois dans un même chemin. Cette propriété est très utile quand on manipule des structures dynamiques comme les files. Grâce à la grande flexibilité de la structure, les DDD sont choisis (dans le cadre du projet MORSE, un projet RNTL [Bréant 2004]) comme les structures pour la vérification de systèmes décrits en LfP [Gilliers 2004], un langage de prototypage de haut niveau. Nous avons montré l'aptitude des DDD à traiter des programmes du LfV, un langage pour la vérification [Nguyen 2006, Nguyen 2007].

Les techniques de **Ré-écriture des termes** [Dershowitz 1990, Baader 1998, Ohlebusch 2002, Zantema 2003], avec une base mathématique solide et de nombreuses recherches sont applicables autour des problèmes inhérents à la ré-écriture *c.*à.*d.* terminaison, calcul des accessibles, etc. Parmi les techniques de vérification, nous retrouvons en ré-écriture l'équivalence du *model-checking* pour les systèmes finis. Pour les systèmes infinis, les techniques de ***Regular Model Checking*** ont été mise en place [Kesten 1997, Wolper 1998, Bouajjani 2000, Abdulla 2006, Genet 2001, Feuillade 2004, Boichut 2008]. Dans les deux, un calcul des accessibles ou d'une sur-approximation est le coeur de la technique.

Ceci nous conduit à construire une nouvelle structure de la vérification symbolique basée sur les systèmes de ré-écritures.

Objectifs

L'objectif de cette thèse est de construire un nouveau type de systèmes de ré-écriture pour la vérification symbolique de modèles. Nous appelons les *systèmes de ré-écriture fonctionnels*. Nous montrons que notre modèle a la puissance d'expression des systèmes de ré-écriture et qu'il est bien adapté à l'étude de propriétés de sûreté et de propriétés de logique temporelle de modèles.

Nous allons mettre en évidence une sous classe de systèmes fonctionnels, les *élémentaires* et les *élémentaires à droite*, préservant la puissance d'expression des systèmes fonctionnels et des techniques d'accélération des calculs aboutissant à un outil de vérification symbolique efficace.

Contributions

Nos contributions sont divisées en deux aspects : Aspect théorique et Aspect expérimental.

- **Aspect théorique.** Nous avons proposé un outil formel à base de systèmes de ré-écriture pour la vérification symbolique de modèles décidables. [Couvreur 2008, Boichut 2010].

  1. Conception des systèmes de ré-écriture fonctionnels et élémentaires.

Transformation entre des systèmes de ré-écriture classiques, des fonctionnels et des élémentaires.

2. Conception et optimisation des algorithmes d'évaluation des systèmes fonctionnels élémentaires.
3. Modélisation des réseaux de Petri ordinaires (P/T) et hiérarchiques par systèmes fonctionnels élémentaires.
4. Conception des systèmes de ré-écriture pour décrire et vérifier des propriétés de logiques temporelles telles que LTL et CTL ainsi que des propriétés de sûreté.

– **Aspect expérimental.** Nous avons implémenté un ensemble d'outils basés sur des systèmes fonctionnels élémentaires listé ci-dessous :

1. Implémentation d'un vérificateur de systèmes fonctionnels élémentaires.
2. Conception d'un outil de transformation automatique des modèles de réseaux de Petri ordinaires et hiérarchiques vers des systèmes fonctionnels élémentaires.
3. Réalisation d'un outil permettant de décrire et vérifier des propriétés des logiques temporelles telles que LTL et CTL.

Nous avons comparé notre vérificateur, d'une part avec des outils de ré-écriture tels que Timbuk, Maude et TOM sur des modèles arborescents, d'autre part avec des outils de vérification tels que SPIN, NuSMV, SMART et HSDD sur des modèles de réseaux de Petri afin de montrer nos performances pour des systèmes décidables.

Organisation de la thèse

Après le chapitre d'introduction, la thèse sera présentée en 4 grandes parties sous-divisées en 10 chapitres.

Dans la première partie (**Préliminaires**), nous rappelons les principes de la modélisation et de la vérification, en particulier les concepts à base de systèmes de ré-écriture. Cette partie contient également une étude de cas pour montrer le démarche de la modélisation et de la vérification à l'aide de systèmes de ré-écriture.

La deuxième partie (**Système de Ré-écriture Fonctionnel**) présente des principes de système ré-écriture fonctionnel et des algorithmes d'évaluation pour ces systèmes par une analyse d'accessibilité. Pour explorer tous les états possibles d'un système, on le parcourt en partant d'un état initial. Le problème de l'exploration exhaustive est qu'il y a beaucoup de possibilités, même pour un ordinateur. Plusieurs solutions sont disponibles, nous n'en présentons que quelques unes.

La troisième partie (**Application à la vérification**) montre la capacité d'expression des problèmes connus et la performance des systèmes fonctionnels élémentaires par des modèles de réseaux de Petri ordinaires et hiérarchiques ainsi que

des propriétés des logiques temporelles LTL, CTL, et des propriétés de sûreté. Les résultats statistiques nous disent la limite des outils sans utiliser les techniques d'accélération ainsi que la capacité de ce qui utilise ces techniques pour lutter contre l'explosion combinatoire dans l'analyse d'accessibilité.

Enfin, dans la dernière partie (**Perspectives et Conclusions**), nous proposons une extension de nos systèmes de ré-écriture : les systèmes ré-écritures fonctionnels paramétrés. Nous développons aussi un modèle de description de systèmes, le langage LfV, et montrons comment les modèles décrits dans ce langage peuvent analyser à l'aide de ces nouveaux systèmes de ré-écriture.

Table des matières

Première partie

Partie I : Préliminaires

Chapitre 2

Éléments sur la vérification de modèles

RÉSUMÉ DU CHAPITRE 2

Dans ce chapitre, nous réduisons le problème de modélisation aux principaux concepts tels que la structure de Kripke et le réseau de Petri pour modéliser des systèmes réactifs. A partir de la description du modèle, plusieurs techniques pour vérifier ce modèle peuvent être étudiées telles que les *logiques temporelles* [Pnueli 1977, Emerson 1980, Sistla 1982, Vardi 1986, Clarke 1986, Arnold 1988, Gerth 1995], la *vérification symbolique* [Bryant 1986, Bryant 1992, Clarke 2000], les *techniques de réduction basée sur des ordres partiels* [Holzmann 1991, Holzmann 1986, Clarke 2000] et les *techniques d'abstraction* [Clarke 2000]. Nous nous focalisons dans ce chapitre sur les *logiques temporelles* et la *vérification symbolique* qui seront étudiées dans la suite de ce mémoire.

Après avoir présenté la structure de Kripke et le réseau de Petri, nous disons comment nous pouvons construire des logiques temporelles à partir d'une telle structure. Ensuite, nous décrivons comment nous pouvons construire des structures symboliques à partir d'un tel modèle. Enfin, nous listons quelques outils de vérification connus basés sur ces deux techniques.

Sommaire

2.1 Modélisation des systèmes réactifs

Nous parlons d'abord des systèmes réactifs et leurs comportements. Un système réactif est capturé en ses *état*s. Il est également décrit par un franchissement d'état de l'un à l'autre. Un tel couple d'états est appelé une *transition*.

Un système réactif peut être modélisé par une structure de Kripke ou par un réseau de Petri. Nous pouvons utiliser les techniques des logiques temporelles et les techniques de la vérification symbolique pour vérifier ces modèles. Nous allons présenter ces techniques dans les sections 2.2 et 2.3.

2.1.1 Structure de Kripke

La structure de Kripke est un graphe de transitions-états capable de modéliser le comportement d'un système réactif. Formellement, une structure de Kripke est un quadruplet $M = (S, S_0, R, L)$ sur un ensemble fini de propositions atomiques AP tel que :

- S est un ensemble fini d'états.
- $S_0 \subseteq S$ est l'ensemble d'états initiaux.
- $R \subseteq S \times S$ est une relation de transitions. Celle-ci doit être totale ($\forall s \in S, \exists s' \mid (s, s') \in R$).
- $L : S \rightarrow 2^{AP}$ est une fonction associant à chaque état un ensemble de propositions atomiques vraies dans cet état.

Les comportements d'un système réactif sont définis par ses transitions. Un comportement est une séquence infinie d'états tel que chaque état est obtenu de l'état précédent par une transition. Un chemin dans la structure M à partir d'un état s_0 est une séquence infinie d'états $\pi = s_0 s_1 s_2 ...$ tel que $(s_i, s_{i+1}) \in R \ \forall i \geq 0$.

Nous allons illustrer cette structure par un exemple.

Exemple 1 *Soit une structure de Kripke $M = (S, S_0, R, L)$ comme la suivante :*

- $S = \{s_0, s_1\}$.
- $S_0 = \{s_0\}$.
- $R = \{(s_0, s_1), (s_1, s_0), (s_1, s_1)\}$.
- $L : s_0 \rightarrow \{a, b\}$ *et* $s_1 \rightarrow \{a, \neg b\}$.

Elle est décrite comme un graphe de transitions-états dans la figure 2.1. Ainsi, $\pi_0 =$

FIG. 2.1 – Une structure de Kripke

$s_0 s_0 s_1$ n'est pas un chemin dans M. Cependant, $\pi_1 = s_0 s_1 s_0 s_1 s_1 ... s_1$ est un chemin dans M.

A partir de cette structure de Kripke, nous pouvons utiliser les logiques temporelles et les techniques de la vérification symbolique pour vérifier ce modèle.

2.1.2 Réseaux de Petri

Un réseau de Petri est un modèle mathématique servant à représenter divers systèmes (informatiques, industriels, ...) travaillant sur des variables discrètes. Il est capable de modéliser le comportement d'un système réactif et sera vu comme un modèle de description concis de structures de Kripke.

Formellement, un réseau de Petri est un quintuple $R =< S, T, Pre, Post, m_0 >$ tel que :

- S est un ensemble fini de places.
- T est un ensemble fini de transitions.
- $Pre(t)$ et $Post(t)$ sont deux applications de T dans S qui à toute transition t de T associent les deux états $Pre(t)$ et $Post(t)$ qui sont respectivement l'*origine* et le *but* de la transition t.
- m_0 est un multi-ensemble de S appelé le marquage initial.

Un marquage m peut amener à un autre marquage m' (noté par $m[t > m'$) seulement s'il existe une transition t telle que $m > Pre(t)$ et $m' = m - Pre(t) + Post(t)$. La relation de franchissement peut être étendue inductivement à toute séquence de transitions par : $m_1[\varepsilon > m_2$ si $m_1 = m_2$, et $m_1[\sigma.t > m_2$ si $\exists m : m_1[\sigma > m$ et $m[t > m_2$. Un marquage m' est accessible à partir d'un marquage m si $\exists \sigma \in T^\star : m[\sigma > m'$. Nous notons $Reach(R, m_0)$ l'ensemble de marquages accessibles à partir de m_0.

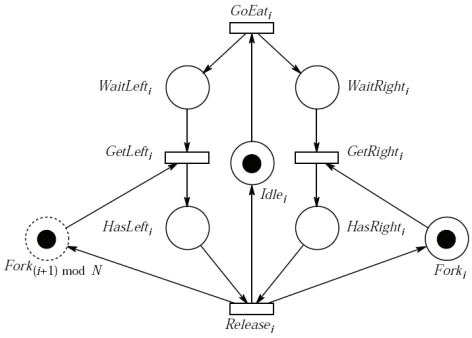
Le réseau de Petri est dit :

- **fini** si $\| S \| < +\infty$ et $\| T \| < +\infty$.
- **élémentaire** si $\forall p \in S, \forall t \in T$, $Pre(t)(p) \leq 1$ et $Post(t)(p) \leq 1$.
- **borné** si $\| Reach(R, m_0) \| < +\infty$.
- **sain** si $\forall m \in Reach(R, m_0)$, m est sain, *c.à.d.* $\forall p \in S, m(p) \leq 1$.

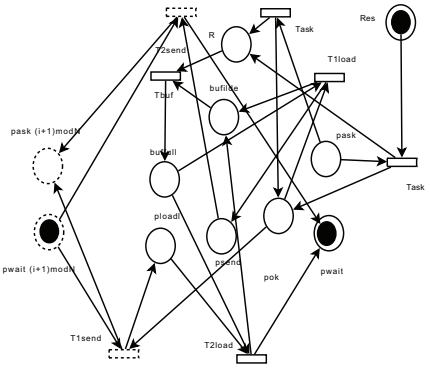
[Miner 1999] présente quelques exemples de réseaux de Petri. Nous allons les décrire ci-dessous.

Nous nous focalisons sur les réseaux de Petri paramétrés soit par un nombre de jetons N, soit par un nombre de sous réseaux identiques. Deux représentants des modèles paramétrés par un nombre de jetons N sont le système de Kanban et le système de FMS. Trois représentants des modèles paramétrés par un nombre de sous réseaux identiques sont le problème des Philosophes, le protocole de Slotted-Ring et le protocole de Round-Robin Mutex.

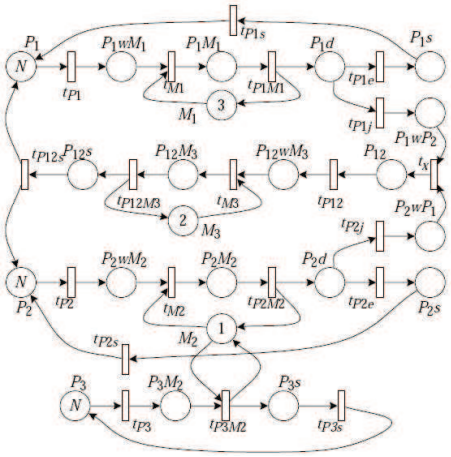
1. Le problème des Philosophes est un exemple explicatif d'un problème de concurrence. Au début, chaque philosophe est en état "idle" et décide de manger : cela le conduit à prendre les deux fourchettes de gauche et de droite. Puis, il les libère après avoir mangé. Le modèle des philosophes est composé par N sous réseaux identiques. Le sous réseau de Petri pour le i-ième philosophe est paru en figure 2.2 (a). Ce sous réseau représente un philosophe et la fourchette



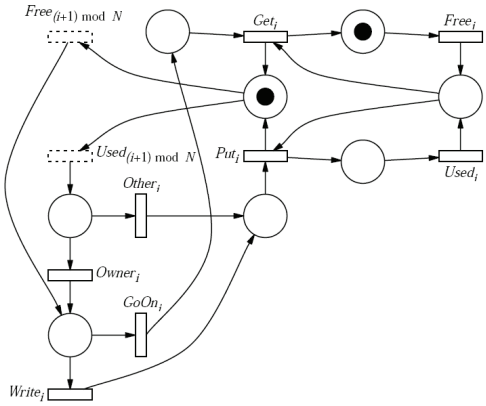
a) Problème des Philosophes



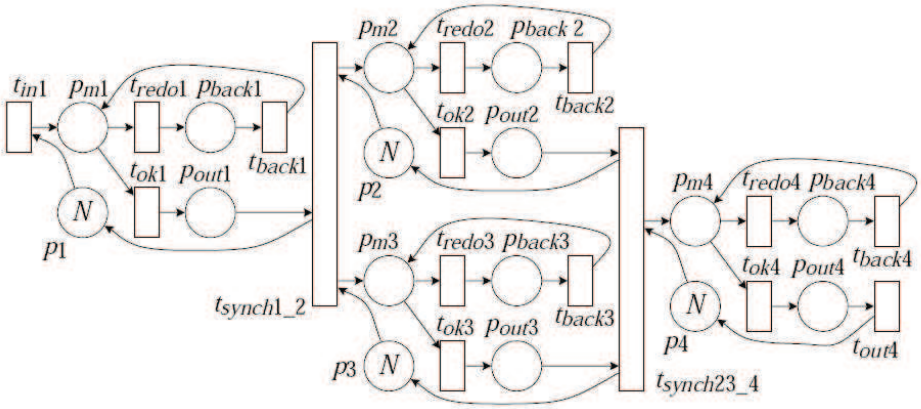
b) Protocole de Round-Robin Mutex



c) Système de FMS



d) Protocole de Slotted-Ring



e) Système de Kanban

FIG. 2.2 – Quelques réseaux de Petri

du côté droit du philosophe. La fourchette du côté gauche, représentée par la place pointillée $Fork(i+1)modN$, est une partie du sous réseau du philosophe suivant, cela illustre comment les sous réseaux communiquent entre eux. *Ce réseau est borné, élémentaire et sain.*

2. Modèle du Protocole de réseau Slotted Ring : Le réseau de Petri pour un seul noeud de ce protocole est représenté en figure 2.2 (d). Le modèle global est composé par N tel sous réseaux connectés en partageant les transitions ($Free(i+1)$ mod N et $Used(i+1)$ mod N). Comme le problème des philosophes, ce réseau est *borné, élémentaire et sain.*
3. Modèle du Protocole de Round-Robin Mutex (présenté par [Graf 1996]) : Voir la figure 2.2 (b). Ce protocole résout un type spécifique du problème d'exclusion mutuelle parmi N processus organisés dans une manière circulée, demandant d'accéder à une ressource partagée. Comme les deux réseaux ci-dessus, ce réseau *est borné, élémentaire et sain.*
4. Système de Kanban : La figure 2.2 (e) représente le réseau de Petri d'un système de Kanban. Ce modèle est paramétré par le nombre de jetons N au début en p_1, p_2, p_3, et p_4. Ce réseau est *borné, élémentaire* mais pas *sain* à cause du nombre de jetons N.
5. Système de FMS : Ce modèle représenté en figure 2.2(c) est paramétré par le nombre de jetons N dans P_1, P_2, and P_3. Comme le système de Kanban, ce réseau est *borné, élémentaire* mais pas *sain* à cause du nombre de jetons N.

Nous pouvons utiliser les logiques temporelles pour vérifier quelques propriétés intéressants d'un réseau de Petri.

2.2 Logiques temporelles

Les logiques temporelles sont capables de spécifier un système d'états / transitions ou des structures de Kripkes. Une formule des logiques temporelles décrit les propriétés des arbres de calcul. L'arbre de calcul montre toutes les exécutions possibles commençant à partir de l'état initial. Dans la suite, nous réduisons notre étude à deux parmi des logiques temporelles : La logique temporelle linéaire LTL et la logique temporelle arborescente CTL.

2.2.1 Logique temporelle linéaire LTL

La logique temporelle linéaire LTL [Pnueli 1977] permet de spécifier des propriétés le long des chemins d'exécution d'un système de transitions. Ces propriétés dynamiques font référence à un temps causal. Des opérateurs sont fournis ci-dessous pour décrire des événements qui suivent un seul chemin de calcul.

- Toujours ϕ_1 ($\mathbf{G}\phi_1$) : ϕ_1 est vraie dans tous les états de la séquence débutant en s_0
- Toujours ϕ_1 jusqu'à ϕ_2 ($\phi_1 \mathbf{U} \phi_2$) : ϕ_1 est vraie dans tous les états jusqu'à ce que ϕ_2 le devienne

- Au moins une fois ϕ_1 dans le futur : $\mathbf{F}\phi_1 = vrai\mathbf{U}\phi_1$
- ϕ_1 se produira à l'étape suivante : $\mathbf{X}\phi_1$ (ϕ_1 est vraie dans l'état suivant immédiatement s_0)

En pratique, les propriétés vérifiées font intervenir plusieurs opérateurs temporels (X, F, G ou U) composés entre eux :

- ϕ_1 se produira infiniment souvent : $\mathbf{GF}\phi_1$ (aussi loin que nous allions dans la séquence, nous pourrons toujours trouver un état pour lequel ϕ_1 est vraie)
- Dans le futur, ϕ_1 se produira toujours : $\mathbf{FG}\phi_1$ (à partir d'un certain rang, ϕ_1 sera toujours vraie)

La méthode la plus répandue de vérification de propriétés LTL est basée sur la construction du produit synchronisé du système à vérifier et d'un automate (appelé *automate de Büchi*) permettant de reconnaître des séquences infinies validant une propriété LTL [Vardi 1986].

Un *automate de Büchi* est un automate d'états fini, dont certains sont appelés les *états d'acceptation*. Une séquence d'un automate de Büchi est reconnue (ou acceptée) si elle comporte un cycle contenant au moins un état d'acceptation. [Gastin 2001, Couvreur 1999, Couvreur 2004] utilise un autre type d'automates : les automates à transitions. Dans ces automates, les conditions d'acceptation portent sur les transitions infiniment rencontrées plutôt que sur les états.

Nous montrons maintenant comment construire un automate de Büchi pour les modèles mentionnés ci-dessus. Considérons un exemple pour la structure de Kripke comme suit :

Exemple 2 *Soit une structure de Kripke décrit dans l'exemple 1. Nous considérons un comportement tel que : "Dans un moment dans le futur, est-ce qu'il existe a mais jamais b ?". Ainsi une formule de LTL est représentée comme la suivante :*

$$\phi = F(a \wedge G(\neg b))$$

Un chemin d'états qui vérifie cette formule termine par une séquence de s_1 comme le suivant : $s_0 s_1 ... s_0 s_1 ... s_1$. Plus précisément, nous obtenons :

$$(ab)(a\neg b)...(ab)(a\neg b)...(a\neg b)$$

Nous décrivons en figure 2.3 l'automate de Büchi de la formule LTL de cette formule.

Considérons un exemple pour le réseau de Petri. Nous revenons au problème des Philosophes.

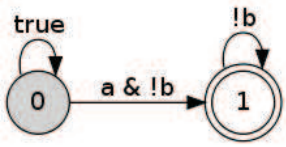


FIG. 2.3 – Automate de Büchi

Exemple 3 *Considérons le problème des Philosophes de la section 2.1. Le sous réseau de Petri pour le i-ième philosophe est paru en figure 2.2 (a).*

Nous sommes intéressés à un comportement tel que : "A un moment dans le futur, un philosophe prend la fourchette de gauche mais il n'aura jamais celle de droite" en utilisant la formule de LTL ci-dessous :

$$f_1 = \mathbf{F}(hasLeft_1 \wedge \mathbf{G}(\neg hasRight_1))$$

Voir l'automate de Büchi en figure 2.4 (a).

Nous sommes également intéressés à un comportement tel que : "A un moment dans le futur, au début, un philosophe prend la fourchette de droite, puis il la libère, et finalement, cette fourchette est prise par son voisin droite" en utilisant la formule de LTL suivante :

$$f_2 = (hasRight_1 \ \cup \ (Fork_1 \ \cup \ hasLeft_2))$$

Voir l'automate de Büchi en figure 2.4 (b).

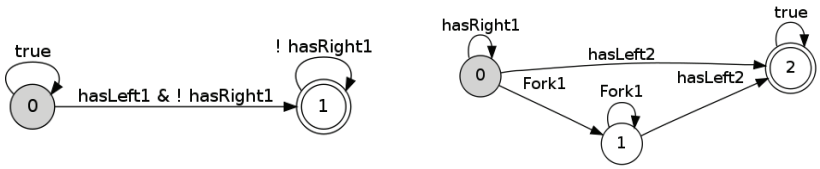


FIG. 2.4 – Automates de Büchi a) f_1 et b) f_2

Après avoir traduit d'une formule LTL en automate de Büchi [Wolper 2001], il nous reste la construction du produit synchronisé du système à vérifier et d'un automate de Büchi.

Vérification pour LTL

S'il est nécessaire de construire l'automate produit dans sa totalité pour prouver que $M \vDash \phi$, la réfutation de cette propriété peut être prouvée par l'existence d'un unique contre-exemple. La méthode de vérification dite *"à la volée"* consiste à

rechercher le cycle d'acceptation en même temps que nous construisons $A_M \otimes A_{\neg\phi}$ et à interrompre sa construction au premier cycle d'acceptation trouvé [Gerth 1995].

A partir de la négation de la formule LTL à vérifier, nous construisons l'automate de Büchi permettant de reconnaître un contre-exemple à la propriété. Ensuite nous procédons à la construction de l'automate de Büchi $A_{\neg\phi}$ permettant de reconnaître un contre-exemple de la propriété ϕ. Puis nous effectuons le produit synchronisé $A_M \otimes A_{\neg\phi}$ qui est aussi un automate de Büchi. Deux possibilités s'offrent alors à nous, suivant l'absence ou la présence de cycle d'acceptation (*c.à.d.* de contre-exemple) dans $A_M \otimes A_{\neg\phi}$.

2.2.2 Logique temporelle arborescente CTL

L'opérateur temporel qualifie les chemins possibles à partir d'un état donné [Clarke 1986]. Pour travailler sur l'arborescent, nous utilisons les préfixes modaux **A** (modalité universelle désignant tous les chemins possibles à partir de l'instant courant) et **E** (modalité existentielle désignant un chemin possible et existant à partir de l'instant courant). Ces préfixes sont toujours suivis des modalités (X, F, G ou U) de la logique temporelle linéaire. Par conséquent, la logique CTL est munie des opérateurs traditionnels de la logique booléenne, et des opérateurs temporels quantifiés : $EX, AX, EF, AF, EG, AG, EU, AU$.

Les formules CTL sont construites à partir des propositions atomiques AP d'une structure de Kripke. Leur syntaxe et leur sémantique sont les suivantes (ϕ_1 et ϕ_2 sont des formules CTL, ap est une proposition atomique) :

- $M, s \vDash ap \iff ap \in L(s)$
- $M, s \vDash \neg\phi_1 \iff M, s \nvDash \phi_1$
- $M, s \vDash \phi_1 \wedge \phi_2 \iff (M, s \vDash \phi_1) \wedge (M, s \vDash \phi_2)$
- $M, s \vDash \phi_1 \vee \phi_2 \iff (M, s \vDash \phi_1) \vee (M, s \vDash \phi_2)$
- $M, s \vDash EX\phi_1 \iff \exists\pi, s_0 = s \Rightarrow M, s_1 \vDash \phi_1$
- $M, s \vDash AX\phi_1 \iff \forall\pi, s_0 = s \Rightarrow M, s_1 \vDash \phi_1$
- $M, s \vDash EF\phi_1 \iff \exists\pi, \exists i \geq 0, s_0 = s \Rightarrow M, s_i \vDash \phi_1$
- $M, s \vDash AF\phi_1 \iff \forall\pi, \exists i \geq 0, s_0 = s \Rightarrow M, s_i \vDash \phi_1$
- $M, s \vDash EG\phi_1 \iff \exists\pi, \forall i \geq 0, s_0 = s \Rightarrow M, s_i \vDash \phi_1$
- $M, s \vDash AG\phi_1 \iff \forall\pi, \forall i \geq 0, s_0 = s \Rightarrow M, s_i \vDash \phi_1$
- $M, s \vDash E\phi_1 U\phi_2 \iff \exists\pi, \exists k \geq 0, s_0 = s \Rightarrow (M, s_k \vDash \phi_2) \wedge (\forall 0 \leq i < k, (M, s_i \vDash \phi_1))$
- $M, s \vDash A\phi_1 U\phi_2 \iff \forall\pi, \exists k \geq 0, s_0 = s \Rightarrow (M, s_k \vDash \phi_2) \wedge (\forall 0 \leq i < k, (M, s_i \vDash \phi_1))$.

Nous allons construire des formules de CTL pour l'exemple 1.

Exemple 4 *Soit la structure de Kripke décrit dans l'exemple 1. Ainsi, quelques formules de CTL sont calculées comme les suivantes :*

- $EX(\{s_0\}) = \{s_1\}$, *et* $EX(\{s_1\}) = \{s_0, s_1\}$.

- $AX(\{s_0\}) = \emptyset$, *et* $AX(\{s_1\}) = \{s_0\}$
- $E(\{s_0\})U(\{s_1\}) = \{s_0, s_1\}$ *et* $E(\{s_1\})U(\{s_0\}) = \{s_0, s_1\}$.
- $A(\{s_0\})U(\{s_1\}) = \{s_0, s_1\}$ *et* $A(\{s_1\})U(\{s_0\}) = \{s_0\}$.

Vérification pour CTL

La vérification symbolique de propriétés CTL utilise un type de données abstrait pour la représentation d'ensembles d'états. Il a été démontré [Emerson 1980] que ce type de données abstrait devait être muni des opérateurs ensemblistes traditionnels (union, intersection et différence) ainsi que de trois opérateurs suivants :

1. L'invariance *inv* dans l'ensemble $S_1 \subseteq S$ du sous-ensemble des états satisfaisant une condition booléenne φ :

$$S_2 = inv_\varphi(S_1) = s_2 \in S_1, (M, s_2 \vDash \varphi)$$

2. Le calcul *succ* de l'ensemble S_2 des états successeurs à partir d'un ensemble d'états $S_1 \subseteq S$ par la relation de transition R :

$$S_2 = succ(S_1) = \{s_2, \forall (s_1, s_2) \in S_1 \times S, (s_1, s_2) \in R\}$$

3. Le calcul en arrière *pred* de l'ensemble S_1 des états prédécesseurs à partir d'un ensemble d'états $S_2 \subseteq S$ par la relation de transition R :

$$S_1 = pred(S_2) = \{s_1, \forall (s_1, s_2) \in S \times S_2, (s_1, s_2) \in R\}$$

Ces trois opérateurs sont définis à partir de la structure de Kripke sur laquelle on souhaite vérifier des propriétés.

2.2.3 Comparaison des logiques temporelles

Pour comparer les deux logiques en terme de complexité, [Sistla 1982] a montré que la vérification de propriétés LTL est un problème PSPACE-complet avec une complexité temporelle en $2^{(\| \phi \|)} \times O(\| M \|)$, où $\| \phi \|$ désigne le nombre maximal d'opérateurs LTL imbriqués dans ϕ, et $\| M \|$ le nombre d'états accessibles du système. [Clarke 1986, Arnold 1988] ont montré que la vérification de propriétés CTL est un problème PSPACE-complet avec une complexité temporelle en $O(\| M \| \times \| \phi \|)$ ($\| M \|$ est le nombre d'états de M et $\| \phi \|$ le nombre d'opérateurs imbriqués dans ϕ).

2.3 Vérification symbolique

La complexité croissante des systèmes informatiques demande la mise en oeuvre de méthodes automatiques pour leur **Vérification formelle**. Les diagrammes de décisions binaires (BDD) [Bryant 1986, Bryant 1992] sont adoptés pour traiter des

composants électroniques de plus en plus complexes. Cependant, comme le nombre de variables des systèmes étudiés est un facteur critique, de nombreuses structures *à la BDD* ont été proposées. Quelques variantes de BDD peuvent être vues comme des arbres *à la BDD* où les états représentent des choix de valeurs de variables non booléennes.

2.3.1 Diagrammes de décisions binaires (BDD)

Les diagrammes de décisions binaires (BDD) [Bryant 1986, Bryant 1992] sont un succès pour la vérification de systèmes finis. Les BDD sont des structures codant des fonctions booléennes. Ils peuvent être vus comme des arbres où les états représentent des choix de valeurs de variables booléennes, un ordre total sur les variables garantit l'unicité du codage d'une fonction. Les techniques de partage de structures, combinées à des méthodes de réductions, conduisent à des implémentations extrêmement efficaces en pratique [Minato 1990, Hulgaard 1999]. Ainsi, des vérifications exhaustives ont pu être réalisées sur des systèmes comprenant des milliards d'états [Minato 1990, Hulgaard 1999] (Voir cf. l'article *"Symbolic Model Checking : 10^{20} states and Beyond"* [Burch 1992]). Le pouvoir d'expression des BDD est suffisant pour manipuler une grande classe de systèmes finis [Burch 1992].

Nous présentons maintenant comment nous pouvons construire des structures BDD à partir d'une structure de Kripke. Considérons désormais une structure de Kripke $M = (S, S_0, R, L)$ tel que :

- S est le codage des états, pour simplifier nous supposons que nous avons exactement 2^m états. Et $\Phi : \{0,1\} \rightarrow S$ est une fonction des vecteurs booléennes aux états.
- R utilise le même codage que S
- L est représenté séparément pour chaque proposition atomique $L_p = \{s, |p \in L(s)\}$.

Nous allons illustrer cette structure par un exemple.

Exemple 5 *Considérons la structure de Kripke $M = (S, S_0, R, L)$ décrit dans l'exemple 1. Dans ce cas-là, nous avons deux variables d'états s_0, s_1. De plus, nous introduisons deux variables additionnelles, s'_0, s'_1, pour coder des états de successeurs. Ainsi, nous allons présenter la transition de l'état s_0 à l'état s_1 par la conjonction :*

$$(a \wedge b \wedge a' \wedge \neg b')$$

La formule booléenne pour la relation entière est donnée ci-dessous :

$$(a \wedge b \wedge a' \wedge \neg b') \vee (a \wedge \neg b \wedge a' \wedge \neg b') \vee (a \wedge \neg b \wedge a' \wedge b')$$

Les points fort de la technique des BDDs sont listées ci-dessous :

– **Tableau d'unicité** : Nous n'acceptons qu'un représentant pour un objet de données. Cela permet les données de coder des modèles très compactes.
– **Calcul de cache** : Chaque calcul doit être mis en cache pour éviter de re-calculer un résultat déjà connu.

Ces techniques nous permettent de lutter contre l'explosion combinatoire surtout pour les modèles symétriques.

2.3.2 Variantes de BDD

Certains systèmes dynamiques peuvent être pris en compte avec ce type de techniques [Kolks 1993]. Comme le nombre de variables des systèmes étudiés est un facteur critique, de nombreuses structures *à la BDD* ont été proposées [Miner 1999, Ciardo 2000]. Les Diagrammes de Décision Multivalués (MDD) est une telle structure symbolique *à la BDD*. En conséquence, la compacité des données mène à un autre problème qui est l'effet des données intermédiaires. La stratégie de saturation [Ciardo 2003] proposée avec les MDD est une bonne solution pour ce problème. Cette stratégie de saturation n'est pas seulement applicable pour les MDD mais aussi pour les autres structures *à la BDD*.

Dans le cadre de projets industriels, nous avons conçu une nouvelle structure à la BDD, les Diagrammes de Décisions de Données (DDD) [Couvreur 2002]. L'objectif était de fournir un outil flexible qui peut être, autant que possible, adapté à la vérification de tout type de modèles et qui offre des capacités de traitement similaires aux BDD. A la différence des BDD, les opérations sur ces structures ne sont pas prédéfinies, mais une classe d'opérateurs, appelée homomorphismes, est introduite pour permettre à un utilisateur de concevoir ses propres opérations. Dans ce modèle, les variables ne sont pas booléennes, elles prennent leurs valeurs dans des domaines non nécessairement bornés. Une autre caractéristique intéressante est qu'aucun ordre sur les variables est présupposé dans la définition. De plus, une variable peut apparaître plusieurs fois dans un même chemin. Cette propriété est très utile quand nous manipulons des structures dynamiques comme les files. Les variantes (Les structures hiérarchiques) de DDD sont Diagrammes de Décision d'Ensembles (SDD) [Couvreur 2005], et Diagrammes de Décision d'Ensembles Hiérarchiques (HSDD) [Thierry-Mieg 2004, Thierry-Mieg 2008].

D'autres structures ont plutôt cherché à étendre le domaine d'application de ces techniques [Bahar 1997, Gupta 1993, Gupta 1994, Mauborgne 1999, Reffel 1999, Mauborgne 2000].

2.4 Quelques outils de vérification

Nous avons présenté dans les sections précédentes les logiques temporelles et la vérification symbolique. Dans cette section, nous présentons quelques vérificateurs connus dans ce domaine basés sur ces deux techniques.

SPIN [Holzmann 1991, Holzmann 1986] est un outil de vérification de propriétés LTL écrit en langage de modélisation PROMELA pour des systèmes distribués,

développé par une équipe dirigée par Gerard J. Holzmann (laboratoires Bell), qui a permis la vérification effective de protocoles de communication. Néanmoins, SPIN reste limité à l'analyse de systèmes représentables sur 10^6 à 10^7 états au maximum. Il est disponible à l'adresse *http ://spinroot.com/spin/whatispin.html*.

SMV le model-checkeur SMV (Symbolic Model Verifier) a été développé par l'équipe de l'Université de Carnegie Mellon dirigée par Edmund Clarke (Voir l'adresse *http ://www-2.cs.cmu.edu/ modelcheck/smv.html*). Il reprend les principes de l'article *"Symbolic Model Checking :* 10^{20} *states and Beyond"* [Burch 1992]. Cet article préconise l'emploi des BDDs pour une représentation compacte d'ensemble d'états. Depuis 1999, SMV est remplacé par NuSMV [Cimatti 2002] qui associe la vérification symbolique utilisant une représentation à base de BDD aux méthodes de résolution de problèmes SAT (Voir l'adresse *http ://nusmv.irst.itc.it/*), et à Cadence-SMV pour les raisonnements par abstraction et composition (Voir l'adresse *http ://www.kenmcmil.com/smv.html*).

SMART (pour *Stochastic Model checking Analyzer for Reliability and Timing* [Miner 1999]) est développé par l'équipe de l'université de Californie (Riverside) dirigée par Gianfranco Ciardo, est basé sur les MDDs et les techniques de saturation qui sont de bonnes solutions pour lutter contre l'effet des données intermédiaires des données à la BDD. Les performances de SMART pour l'analyse de réseaux de Petri ouvrent des perspectives de recherche prometteuses pour le model-checking symbolique. Il est disponible à l'adresse *http ://www.cs.ucr.edu/ ciardo/SMART/*.

DDD [Couvreur 2002] : Dans le projet MORSE [Bréant 2004], un projet RNTL, grâce à la grande flexibilité de la structure, nous avons montré l'aptitude des DDD à traiter des programmes [Nguyen 2006] comme des structures pour la vérification de systèmes d'écrits en LFP [Gilliers 2004], un langage de prototypage de haut niveau, traduit en LfV [Nguyen 2007], un langage pour la vérification. Le *Langage de spécification pivot* LfP permet de spécifier le comportement du système à l'aide de processus séquentiels communiquant via des médias. Le LfV est un langage formel plus simple et plus efficace pour la vérification des systèmes répartis. Il est disponible à l'adresse *http ://move.lip6.fr/software/DDD/*.

SDD [Couvreur 2005], **HSDD** [Thierry-Mieg 2008, Thierry-Mieg 2004] : Les variantes hiérarchiques de DDD. Il est également disponible à l'adresse *http ://sourceforge.net/projects/buddy/, http ://vlsi.colorado.edu/ fabio/CUDD/*.

Nous venons de présenter quelques outils de vérification. Dans le chapitre 8, nous allons comparer notre outil avec les outils de vérification listés ci-dessus. Nos benchmarks, étant les réseaux de Petri présentés en sous section 2.1.2, démontrent l'efficacité de nos nouveaux formalismes pour la vérification de modèles.

BILAN DU CHAPITRE 2

Dans ce chapitre, nous avons réduit le problème de modélisation aux principaux concepts de la structure de Kripke et du réseau de Petri pour des systèmes réactifs. A partir de ces modèles, nous pouvons utiliser les logiques temporelles et les techniques de la vérification symbolique pour vérifier quelques propriétés intéressantes.

En utilisant les mêmes principes, nos nouveaux formalismes, présentés en chapitre 4, héritent l'avantages des techniques de la vérification symbolique. Nous montrerons la capacité de ces formalismes pour modéliser les modèles dans le chapitre 6 et pour vérifier les propriétés de sûreté, l'inter-blocage et les propriétés de logiques temporelles dans le chapitre 7.

Dans le chapitre 8, nous allons comparer notre outil avec les outils de vérification listés dans ce chapitre. Nos benchmarks, étant les réseaux de Petri présentés en sous section 2.1.2, démontrent l'efficacité de nos nouveaux formalismes pour la vérification de modèles.

Chapitre 3

Principes de la vérification à base de TRSs

RÉSUMÉ DU CHAPITRE 3

Les recherches des systèmes de ré-écriture des termes (TRSs) et des automates d'arbres sont beaucoup étudiées en [Dershowitz 1990, Baader 1998, Comon 2002, Gilleron 1995, Bouajjani 2002, Abdulla 2006]. Dans ce chapitre, nous donnons des principaux éléments sur les systèmes de ré-écriture des termes (TRSs), ainsi que ceux des automates d'arbres de termes composés par des symboles fonctionnels binaires et la constante $\perp$.

Nous décrivons également les principes de la vérification à l'aide de TRSs et de transducteurs d'arbres en illustrant notre propos l'étude du protocole d'Arbitrage Arborescent. Ce protocole a été introduit par [Dill 1989] et est traité comme un problème de vérification asynchrone dans [Clarke 1989].

Sommaire

3.1 Principes

Nous allons introduire les préliminaires des TRSs, des automates d'arbre et des transducteurs d'arbres pour des termes composés par des symboles fonctionnels binaires et la constante $\perp$. Nous allons également montrer comment nous pouvons calculer la clôture réflexive et transitive en TRSs et en transducteurs d'arbres. Nous pouvons résoudre les problèmes de vérification tels que la propriété de sûreté, l'inter-blocage, les logiques CTL et LTL grâce à cette clôture réflexive et transitive.

3.1.1 Vérification à base de systèmes de ré-écriture

Dans cette section, nous introduisons les préliminaires des TRSs pour des termes composés par des symboles fonctionnels binaires et la constante $\perp$. Ensuite, nous montrons comment nous pouvons calculer la clôture réflexive transitive en utilisant des TRSs.

3.1.1.1 Systèmes de Ré-écriture (TRSs)

Comme déjà mentionné ci-dessus, nous réduisons notre étude aux termes composés par des symboles fonctionnels binaires et la constante $\perp$. Nous notons par $\mathcal{F}_{bin}$ cet ensemble de symboles. Soit un ensemble dénombrable de variables $\mathcal{X}$. $\mathcal{T}(\mathcal{F}_{bin}, \mathcal{X})$ dénote l'ensemble de termes construits sur ces symboles et dotés des variables, et $\mathcal{T}(\mathcal{F}_{bin})$ signifie l'ensemble de *termes clos* (termes sans des variables). Une substitution est une fonction σ de $\mathcal{X}$ à $\mathcal{T}(\mathcal{F}_{bin}, \mathcal{X})$, qui peut être étendue uniquement à un endomorphisme de $\mathcal{T}(\mathcal{F}_{bin}, \mathcal{X})$.

Exemple 6 *Soit $\mathcal{F}_{bin} = \{t, i, \perp\}$ et $\mathcal{X} = \{x, y\}$. Le terme $t_1 = t(i(\perp, \perp), i(\perp, \perp))$ est un terme clos, cependant le terme $t_2 = t(x, i(\perp, y))$ n'est pas ce cas-là.*
Considérons une substitution $\sigma = \{x \leftarrow i(\perp, \perp), y \leftarrow \perp\}$ (voir la figure 3.1), alors :

$$t_2\sigma = t_2\{x \leftarrow i(\perp, \perp), y \leftarrow \perp\} = t_1$$

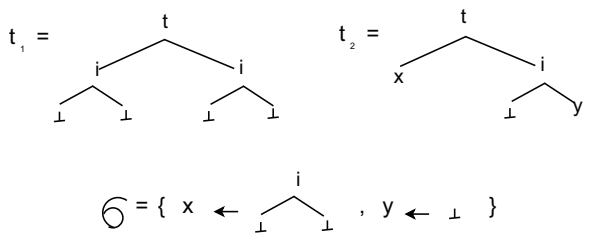


FIG. 3.1 – Terme et substitution

Une position p pour un terme t est un mot sur $\mathbb{N}$. La séquence vide ε dénote la plus haute position. L'ensemble $\mathcal{P}os(t)$ de positions d'un terme t est inductivement défini par $\mathcal{P}os(f(t_1,t_2)) = \{\varepsilon\} \cup \{0.p \mid \text{ et } p \in \mathcal{P}os(t_1)\} \cup \{1.p \mid \text{ et } p \in \mathcal{P}os(t_2)\}$, et $\mathcal{P}os(t) = \{\varepsilon\}$ quand $t \in \mathcal{X}$ ou $t = \perp$. Soit un ordre partiel $\preceq$ sur des positions. Soit p_1 et p_2 deux mots de $\mathbb{N}^\star$, nous disons que $p_1 \preceq p_2$ s'il existe $\omega \in \mathbb{N}^\star$ tel que $p_1 = p_2.\omega$.

Si $p \in \mathcal{P}os(t)$, alors $t|_p$ signifie les sous-termes de t à la position p et $t[s]_p$ dénote le terme obtenu par le remplacement du sous-terme $t|_p$ à la position p par le terme s. Nous dénotons également par $t(p)$ le symbole apparaissant dans t à la position p. Etant donné un terme $t \in \mathcal{T}(\mathcal{F}_{bin}, \mathcal{X})$ et A un ensemble de symboles, fait $\mathcal{P}os_A(t) = \{p \in \mathcal{P}os(t) \mid t(p) \in A\}$. Par conséquent, $\mathcal{P}os_{\mathcal{F}}(t)$ est l'ensemble de positions de t, à chacune dont un symbole fonctionnel apparaît. L'ensemble de positions *frontier* d'un terme t, dénoté par $\mathcal{FP}\text{os}(t)$, est défini ci-dessous : $\mathcal{FP}\text{os}(t) = \{p \in \mathcal{P}os(t) \mid p.0 \notin \mathcal{P}os(t)\}$. L'ensemble de variables apparaissant un terme $t \in \mathcal{T}(\mathcal{F}_{bin}, \mathcal{X})$ est dénoté par $\mathcal{V}ar(t)$. Plus formellement, $\mathcal{V}ar(t) = \{t|_p \mid p \in \mathcal{P}os_{\mathcal{X}}(t)\}$.

Exemple 7 *Etant donnés $\mathcal{F}_{bin} = \{t, i, \perp\}$ et $\mathcal{X} = \{x, y\}$ comme dans l'exemple 6. Considérons le terme $t_2 = t(x, i(\perp, y))$ avec la racine de t_2 est t. Alors, l'ensemble de de positions frontier $\mathcal{FP}\text{os}(t_2)$ est $\{10\}$ et puisque $\mathcal{V}ar(t_2) = \{x, y\}$, l'ensemble de variables $\mathcal{P}os_{\mathcal{X}}(t_2)$ est $\{0, 11\}$.*

Un TRS $\mathcal{R}$ est un ensemble de *règles de ré-écriture* $(l, r) \in \mathcal{T}(\mathcal{F}_{bin}, \mathcal{X}) \times \mathcal{T}(\mathcal{F}_{bin}, \mathcal{X})$, également dénoté par $l \to r$ où $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$ et $l \notin \mathcal{X}$. Le TRS $\mathcal{R}$ induit une relation de ré-écriture $\to_{\mathcal{R}}$ sur des termes dont la clôture réflexive et transitive est écrite $\to^\star_{\mathcal{R}}$. Plus précisément, nous disons que $t \to_{\mathcal{R}} t'$ s'il existe une position p de $\mathcal{P}os(t)$, une règle $l \to r \in \mathcal{R}$ et une substitution $\sigma : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F}_{bin})$ telle que $t|_p = l\sigma$ et $t' = t[r\sigma]_p$.

Exemple 8 *Soit $\mathcal{F}_{bin} = \{t, i, r, \perp\}$, $\mathcal{X} = \{x, y\}$ et un terme clos $t_1 = t(i(\perp, \perp), i(\perp, \perp))$. Considérons une règle $r_1 = i(\perp, \perp) \to r(\perp, \perp)$, alors nous avons 2 possibilités de réduction comme les suivantes :*

$$t_1 \to_{r_1} t(r(\perp, \perp), i(\perp, \perp))$$

$$t_1 \to_{r_1} t(i(\perp, \perp), r(\perp, \perp))$$

Considérons une règle $r_2 = t(x, i(\perp, y) \to t(y, r(\perp, x))$, alors il existe une seule possibilité de réduction comme la suivante : $t_1 \to_{r_2} t(\perp, r(\perp, i(\perp, \perp)))$.

3.1.1.2 Vérification par l'analyse d'accessibilité

La technique de la vérification par l'analyse d'accessibilité est basée sur l'ensemble de $\mathcal{R}$-descendants d'un ensemble de termes.

L'ensemble de $\mathcal{R}$-descendants (appelé également la clôture réflexive et transitive ou plus simple l'ensemble des états accessibles) d'un ensemble de termes $E \subseteq \mathcal{T}(\mathcal{F}_{bin})$ est $\mathcal{R}^\star(E) = \{s \in \mathcal{T}(\mathcal{F}_{bin}) \mid t \rightarrow^\star_\mathcal{R} s \wedge t \in E\}$.

Sur un ensemble de termes de $\mathcal{F}_{bin}$, nous avons les propriétés linéaires suivantes :

- $a(s, s_1) + a(s, s_2) = a(s, s_1 + s_2)$
- $a(s_1, s) + a(s_2, s) = a(s_1 + s_2, s)$

avec a un symbole de $\mathcal{F}_{bin}$.

Soient t un terme dans $\mathcal{T}(\mathcal{F}_{bin})$ et s un ensemble de termes clos. Nous dirons que $t \in s$ ssi, en considérant les propriétés linéaires précédentes, s peut être écrit par cette manière : $s = t_1 + \ldots + t_n$ avec $t_i \in T(F)$ et il existe j tel que $t_j = t$. En pratique, une telle transformation simple est appelée la canonisation. Nous dénotons $\emptyset$ un ensemble vide.

Nous pouvons obtenir l'ensemble de termes accessibles à partir d'un terme initial *init* en calculant $\mathcal{R}^\star(\{init\})$. De plus, nous pouvons également vérifier si un terme *att* est atteignable à partir du terme *init* puis que $init \rightarrow^\star_\mathcal{R} att$.

Exemple 9 *Revenons à l'exemple 8, nous obtenons l'ensemble de termes accessibles à partir du terme* t_1 *:*

$$\mathcal{R}^\star(\{t_1\}) = \left\{ \begin{array}{l} t(i(\bot,\bot), i(\bot,\bot)), \\ t(r(\bot,\bot), i(\bot,\bot)), \\ t(i(\bot,\bot), r(\bot,\bot)), \\ t(r(\bot,\bot), r(\bot,\bot)) \end{array} \right\}.$$

Nous pouvons écrire également :

$$\mathcal{R}^\star(\{t_1\}) =$$

$$t(i(\bot,\bot), i(\bot,\bot)) + t(r(\bot,\bot), i(\bot,\bot)) + t(i(\bot,\bot), r(\bot,\bot)) + t(r(\bot,\bot), r(\bot,\bot))$$

$$= t(r(\bot,\bot), i(\bot,\bot) + r(\bot,\bot)) + t(i(\bot,\bot), i(\bot,\bot) + r(\bot,\bot))$$

$$= t(t(i(\bot,\bot) + r(\bot,\bot), i(\bot,\bot) + r(\bot,\bot))$$

En fin, nous avons prouvé que les termes $t(r(\bot,\bot), i(\bot,\bot)), t(i(\bot,\bot), r(\bot,\bot))$ *et* $t(r(\bot,\bot), r(\bot,\bot))$ *sont atteignables par ré-écriture à partir de* t_1*.*

Pour résumer, nous retrouvons en ré-écriture l'équivalence du *model-checking* pour les systèmes finis. Dans la section 3.2.2, nous donnons une illustration de cette technique pour la vérification du Protocole d'Arbitrage Arborescent.

3.1.2 Vérification à l'aide de automates d'arbres

Dans cette section, nous introduisons les préliminaires des automates d'arbre et des transducteurs d'arbres pour des termes finis composés par des symboles fonctionnels binaires et la constante $\perp$. Ensuite, nous montrons comment nous pouvons calculer la clôture réflexive transitive en utilisant des transducteurs d'arbres.

3.1.2.1 Automate d'arbres et Transducteur d'arbres

- **Automate d'arbres**

  Un automate d'arbres fini sur $\mathcal{F}_{bin}$ est un quadruple $\mathcal{A} = (Q, \mathcal{F}_{bin}, Q_f, \triangle)$ où Q est un ensemble d'états (unaires), $Q_f \subseteq Q$ est un ensemble d'états finaux, et $\triangle$ est un ensemble de règles de transition de type suivante :

  $$f(q_1(x_1), q_2(x_2)) \to q(f(x_1, x_2)),$$

  où $f \in \mathcal{F}_{bin}, q, q_1, q_2 \in Q, x_1, x_2 \in \mathcal{X}$.

  L'automate d'arbres sur $\mathcal{F}_{bin}$ fonctionne sur des termes clos de $\mathcal{F}_{bin}$. Un automate d'arbres commence aux feuilles et monte pas à pas de bas en haut, associant durant le trajet inductivement un état avec chaque sous-terme. Il faut noter qu'il n'y a pas d'état initial dans un automate d'arbres fini, mais, pour un constant symbole $\perp$, une règle de transition est de la forme $\perp \to q(\perp)$. Par conséquent, les règles de transition pour ce symbole constant peuvent être considérées comme les *règles initiales*. Si les sous-termes directs u_1, u_2 de $t = f(u_1, u_2)$ sont étiquetés avec les états q_1, q_2, alors le terme t sera étiqueté par quelques états q avec $f(q_1(x_1), q_2(x_2)) \to q(f(x_1, x_2)) \in \triangle$. Nous définissons formellement la relation de déplacement par un automate d'arbres fini.

  Soit $\mathcal{A} = (Q, \mathcal{F}_{bin}, Q_f, \triangle)$ un automate d'arbres fini sur $\mathcal{F}_{bin}$. La relation de déplacement $\to_{\mathcal{A}}$ est définie par : Soit $t, t' \in \mathcal{T}(\mathcal{F}_{bin} \cup Q)$,

  $$t \to_{\mathcal{A}} t' \iff \begin{cases} \exists C \in C(\mathcal{F}_{bin} \cup Q), \exists u_1, u_2 \in \mathcal{T}(\mathcal{F}_{bin})), \\ \exists f(q_1(x_1), q_2(x_2)) \to q(f(x_1, x_2)) \in \triangle, \\ t = C[f(q_1(u_1), q_2(u_2))], \\ t' = C[q(f(u_1, u_2))]. \end{cases}.$$

  $\to^{\star}_{\mathcal{A}}$ est la clôture transitive et réflexive de $\to_{\mathcal{A}}$.

  Un terme clos t dans $\mathcal{T}(\mathcal{F}_{bin})$ est accepté par un automate d'arbres fini $\mathcal{A} = (Q, \mathcal{F}_{bin}, Q_f, \triangle)$ si

  $$t \to^{\star}_{\mathcal{A}} q(t)$$

  pour quelques états q dans Q_f . Notez que cette définition correspond à la notion d'automate d'arbres fini non-déterministe car ce modèle d'automate d'arbres fini permet zéro, une ou plus de règles de transition avec le même côté gauche. En conséquence, c'est possible d'avoir plus d'une réduction à partir du même terme clos. Et, un terme clos t est accepté s'il y a une réduction

(parmi toutes les réductions possibles) à partir de ce terme clos et menant à une configuration de la forme $q(t)$ où q est un état final.

Le langage d'arbre $L(\mathcal{A})$ reconnu par $\mathcal{A}$ est l'ensemble de tous les termes clos acceptés par $\mathcal{A}$. Un ensemble L de termes clos est reconnaissable si $L = L(\mathcal{A})$ pour quelques automates d'arbres finis $\mathcal{A}$. Nous pouvons dire que cet ensemble de termes est évalué par l'automate d'arbres $\mathcal{A}$.

Exemple 10 *Soit* $\mathcal{F}_{bin} = \{t, i, \perp\}$. *Considérons l'automate d'arbres* $\mathcal{A} = (Q, \mathcal{F}_{bin}, Q_f, \triangle)$ *défini par :* $Q = \{q_t, q_i, q_\perp\}$, $Q_f = \{q_t\}$, *et* $\triangle$ *l'ensemble de règles de transition est donné ci-dessous :*

$$\{\perp \to q_\perp(\perp), \quad i(q_\perp(x), q_\perp(y)) \to q_i(i(x,y)), \quad t(q_i(x), q_i(y)) \to q_t(t(x,y))\}$$

Cet ensemble peut être représenté plus simple comme le suivante :

$$\{\xrightarrow{\perp} q_\perp, \quad (q_\perp, q_\perp) \xrightarrow{i} q_i, \quad (q_i, q_i) \xrightarrow{t} q_t\}$$

Nous donnons l'exemple de réduction avec la relation de déplacement $\to_{\mathcal{A}_1}$ *:* $t_1 = t(i(\perp,\perp), i(\perp,\perp)) \to_{\mathcal{A}_1} t(i(q_\perp(\perp), q_\perp(\perp)), i(q_\perp(\perp), q_\perp(\perp))) \to_{\mathcal{A}_1} t(q_i(i(\perp,\perp)), q_i(i(\perp,\perp))) \to_{\mathcal{A}_1} q_t(t(i(\perp,\perp), i(\perp,\perp)))$. *Enfin, le terme* t_1 *est accepté par* $\mathcal{A}_1$, *c.à.d.*$t_1 \in L(\mathcal{A}_1)$. *Puisque* t_1 *est l'unique terme accepté par* $\mathcal{A}_1$, *ainsi* $L(\mathcal{A}_1) = \{t_1\}$

- **Transducteur d'arbres**

  Un transducteur d'arbres ascendant est un quintuple $\mathcal{T} = (Q, \mathcal{F}_{bin}, \mathcal{F}'_{bin}, Q_f, \delta)$ où Q est un ensemble fini des états, $\mathcal{F}_{bin}$ et $\mathcal{F}'_{bin}$ (les ensembles finis des lettres entrées et des lettres sorties) sont des alphabets, $Q_f \subseteq Q$ est des ensembles finaux, et δ est un ensemble de règles de traduction de type suivante :

$$f(q_1(x_1), q_2(x_2)) \to_\delta q(u), u \in \mathcal{T}_{\mathcal{F}'_{bin}}[\{x_1, x_2\}] \tag{3.1}$$

$$q(x) \to_\delta q'(u), u \in \mathcal{T}_{\mathcal{F}'_{bin}}[x] \tag{3.2}$$

$$\perp \to_\delta q(u), u \in \mathcal{T}_{\mathcal{F}'_{bin}} \tag{3.3}$$

  où $\perp \in \mathcal{F}_{bin}, f \in \mathcal{F}_{bin}, x, x_1, x_2 \in \mathcal{X}$, et $q_1, q_2, q, q' \in Q$.

  Etant donné t un terme entré, $\mathcal{T}$ poursuit le précédent : il commence en remplaçant quelques feuilles selon les règles 3.3. Pour l'instance, si une feuille est étiquetée $\perp$ et la règle $\perp \to_\delta q(u)$ est dans δ, puis $\perp$ est remplacé par $q(u)$. La substitution poursuit jusqu'à la racine. Si la règle $f(q_1(x_1), q_2(x_2)) \to_\delta q(u)$ est dans δ, alors $\mathcal{T}$ remplace une occurrence d'un sous-arbre $f(q_1(t_1), q_2(t_2))$

par le terme $q(u[x_1 \leftarrow t_1, x_2 \leftarrow t_2])$, où chaque occurrence de la variable x_i dans t est remplacée par t_i. Le calcul continue jusqu'à ce que la racine de t soit atteinte.

Le transducteur $\mathcal{T}$ défini la relation régulière suivante entre des arbres $\mathcal{R}_\mathcal{T} = \{(t, t') \in \mathcal{T}_{\mathcal{F}_{bin}} \times \mathcal{T}_{\mathcal{F}'_{bin}} | t \rightarrow^\star_\delta q(t'), \text{ pour quelques } q \in Q_f\}$.

Soit $L \subseteq \mathcal{T}_{\mathcal{F}_{bin}}$ un langage d'arbre, nous définissons donc l'ensemble $\mathcal{R}_\mathcal{T}(L) = \{t' \in \mathcal{F}'_{bin} | \exists t \in L, (t, t') \in \mathcal{R}_\mathcal{T}\}$.

3.1.2.2 *Regular Tree Model Checking*

Nous pouvons calculer la clôture réflexive transitive en utilisant des transducteurs d'arbres finis ou infinis. Nous retrouvons en transducteurs d'arbres finis l'équivalence du *model-checking* en ré-écriture pour les systèmes finis.

[Bouajjani 2002] dénote par $\mathcal{R}^n_\mathcal{T}$ la composition de $\mathcal{R}_\mathcal{T}$, n fois. Ainsi, $\mathcal{R}^\star_\mathcal{T} = \bigcup_{n \geq 0} \mathcal{R}^n_\mathcal{T}$ dénote la clôture transitive et réflexive de $\mathcal{R}_\mathcal{T}$. [Abdulla 2002] nous présentons également un moyen de calculer la clôture transitive des relations régulières d'arbres en utilisant un transducteur d'arbres d'histoire. Plus précisément, étant donné un transducteur d'arbres $\mathcal{T}$, nous générons un nouvel transducteur d'arbres infini H, appelé le transducteur d'arbres d'histoire tel que $\mathcal{R}_H = \mathcal{R}^\star_\mathcal{T}$.

Exemple 11 *Soit $\mathcal{A}_1$ l'automate d'arbre de l'exemple 10. Soit un transducteur $\mathcal{T} = (Q, \mathcal{F}_{bin}, \mathcal{F}'_{bin}, Q_f, \delta)$ défini par : $Q = \{q_t, q_i, q_r, q_\perp\}$, $\mathcal{F}_{bin} = \{t, i, \perp\}$, $\mathcal{F}'_{bin} = \{t, i, r, \perp\}$, $Q_f = \{q_t\}$ et δ l'ensemble de transitions comme les suivantes :*

$$\xrightarrow{(\perp,\perp)} q_\perp \qquad (q_\perp, q_\perp) \xrightarrow{(i,i)} q_i \qquad (q_\perp, q_\perp) \xrightarrow{(i,r)} q_r$$
$$(q_i, q_i) \xrightarrow{(t,t)} q_t \qquad (q_i, q_r) \xrightarrow{(t,t)} q_t \qquad (q_r, q_i) \xrightarrow{(t,t)} q_t$$
$$(q_r, q_r) \xrightarrow{(t,t)} q_t$$

Nous avons donc $\mathcal{R}^\star_\mathcal{T}(L(\mathcal{A}_1)) = \mathcal{R}^\star_\mathcal{T}(\{t_1\}) = \mathcal{R}^\star_\mathcal{T}(\{t(i(\perp,\perp), i(\perp,\perp))\}) = \{t(i(\perp,\perp), i(\perp,\perp)),\ t(r(\perp,\perp), i(\perp,\perp)),\ t(i(\perp,\perp), r(\perp,\perp)),\ t(r(\perp,\perp), r(\perp,\perp))\}$.

Regular Model Checking est proposé comme une technique pour la vérification des systèmes infinis d'états [Kesten 1997, Wolper 1998, Bouajjani 2000, Abdulla 2006]. Dans cette technique, les systèmes sont modélisés et analysés en utilisant des représentations symboliques basées sur des automates : configurations des systèmes sont codées par des mots ou des arbres. Cela suggère l'utilisation de l'automate de mot/arbre fini régulier afin de to représenter des ensembles de configurations, et l'utilisation du transducteur de mot/arbre fini régulier afin de modéliser les transitions entre des configurations. Ensuite, le problème de vérification basé sur l'analyse d'accessibilité est réduit au calcul des clôtures des langages réguliers sur des transducteurs de mot/arbre fini régulier *c.à.d.*, soit une relation régulière R et un langage

régulier L, Calculons $R^\star(L)$, où $R^\star$ est la clôture réflexive-transitive de R. Un problème le plus général est de construire une représentation de la relation $R^\star$ comme un transducteur fini.

Dans les systèmes finis et infinis, un calcul des accessibles ou d'une sur-approximation [Feuillade 2004] est le coeur de la technique. On peut démontrer qu'un terme *natt* n'est pas accessible par le calcul d'une sur-approximation de l'ensemble des termes réellement accessibles et par la non-appartenance de *natt* à cette sur-approximation.

Dans la section 3.2.3, Nous donnons une illustration de cette technique pour la vérification du Protocole d'Arbitrage Arborescent.

3.2 Protocole d'Arbitrage Arborescent (TAP)

Cette section montre comment le protocole d'Arbitrage Arborescent (*Tree Arbiter Protocol*, TAP), un protocole résolvant le problème d'exclusion mutuelle pour accéder à une ressource partagée, peut être formalisé en termes, en TRSs et en automate d'arbres.

3.2.1 Description du protocole TAP

Comme mentionné ci-dessus, le TAP est un circuit asynchrone qui résout le problème d'exclusion mutuelle en construisant un arbre des cellules d'arbitre. Ce circuit était introduit par [Dill 1989]. Il était traité comme un problème de vérification asynchrone en [Clarke 1989].

Une cellule d'arbitre a trois chaînes de communication dénotées par C_0, C_1 et C_p. Chaque chaîne se compose de deux signaux, r et g, représentant un **r***equest* et un *acknowled***g***ement* (ack) (Voir cette cellule au niveau de port à la figure 3.2 (a) et au niveau de transistor à la figure 3.2 (b). Une demande **r***equest* sur une des chaînes C_0, C_1 est transférée au serveur sur la chaîne C_p. Après avoir reçu un *ack* sur C_p, cette cellule va le passer au processeur. A ce moment-là, ce processeur prend le droit d'accès de la ressource partagée. Quand il finit avec la ressource, un autre cycle de **r***equest* / *ack* est lancé. En combinant plusieurs cellules dans un arbre binaire, nous pouvons former un arbitre pour n'importe quel nombre de processeurs. Par exemple dans la figure 3.2 (c) avec le nombre de processeurs $N = 4$, nous avons $N - 1 = 3$ cellules d'arbitre.

Le circuit fonctionne en exécutant des tours d'élimination : une cellule d'arbitres arbitre entre ces deux enfants. Les feuilles de l'arbre sont des processeurs, qui peut-être veulent accéder en asynchronie à une ressource partagée. Les N processeurs au niveau le plus bas sont arbitrés par N/2 cellules. Les gagnants de ce niveau-là sont arbitrés par le niveau suivant, et ainsi de suite.

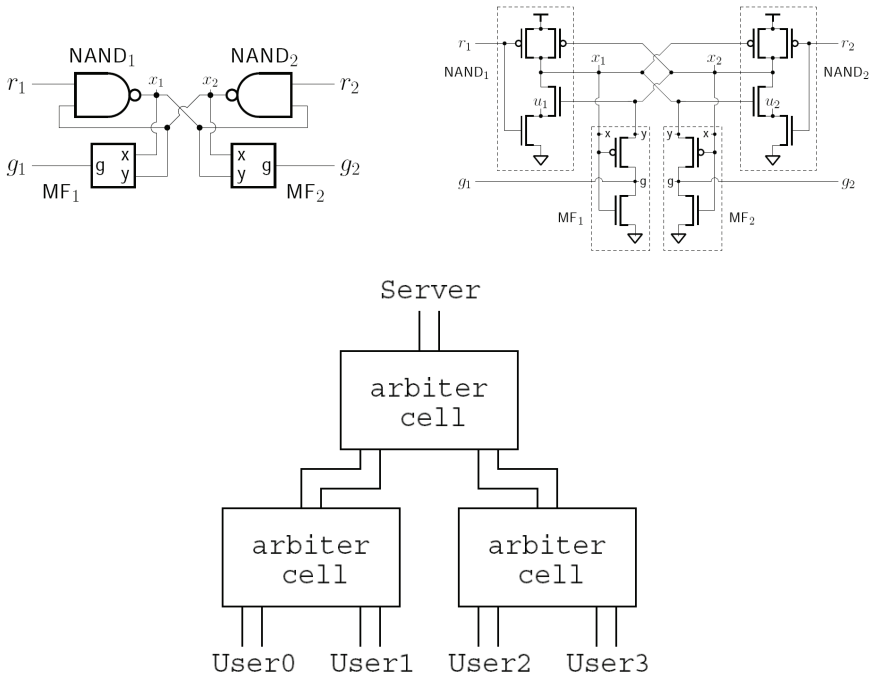


FIG. 3.2 – Circuit d'arbitrage a) Cellule au niveau de port b) Cellule au niveau de transistor et c) Modèle de 3 cellules

3.2.2 Vérification du TAP à base de TRSs

Nous utilisons les symboles binaires i, r, t et b respectivement pour la spécification des états *idle, requesting, token* et *token below* où :

- état *idle* signifie que tous les enfants d'une cellule (ou un processeur) ne font (fait) rien.
- état *requesting* signifie que ce processeur (ou un des enfants de cette cellule) veulent accéder à la ressource partagée.
- état *token* signifie que ce processeur (ou cette cellule) a la ressource partagée.
- état *token below* signifie que la ressource partagée (ou le jeton) est quelque parts dans un des sous-arbres au-dessous de ce noeud.

Etant donné un état initial *total idle* pour un modèle de $N = 4$ processeurs $u = t(i(i(\bot,\bot),i(\bot,\bot)),i(i(\bot,\bot),i(\bot,\bot)))$ où tous les processeurs sont *idle* et la cellule de racine tient le jeton (Voir la figure 3.3 (a). Nous remarquons que ce terme est exactement la représentation du terme du modèle en figure 3.2 (c).

Des requêtes sont propagées de bas en haut jusqu'à la racine qui tient le jeton :

$$i(x, r(y, z)) \rightarrow r(x, r(y, z)) \tag{3.4}$$
$$i(r(x, y), z) \rightarrow r(r(x, y), z) \tag{3.5}$$
$$i(\bot, \bot) \rightarrow r(\bot, \bot) \tag{3.6}$$

En utilisant seulement ces règles de requêtes (appelées $\mathcal{R}^{req}$), le système peut arriver à un état *total requesting* $v = t(r(r(\bot,\bot),r(\bot,\bot)),r(r(\bot,\bot),r(\bot,\bot)))$ (Voir la figure 3.3 (b).

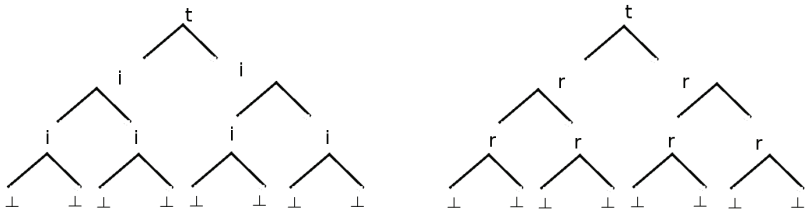


FIG. 3.3 – a) état global *total idle* u et b) état global *total requesting* v

Le terme *total requesting* v est atteignable à partir du terme *total idle* u puis que $u \to^{\star}_{\mathcal{R}^{req}} v$. Nous listons ci-dessus la réduction détaillée :

$$\begin{aligned}
u \;=\; & t(i(i(\bot,\bot),i(\bot,\bot)),i(i(\bot,\bot),i(\bot,\bot))) \to_{r_{3.6}} \\
& t(i(\mathbf{r}(\bot,\bot),i(\bot,\bot)),i(i(\bot,\bot),i(\bot,\bot))) \to_{r_{3.6}} \\
& t(i(\mathbf{r}(\bot,\bot),\mathbf{r}(\bot,\bot)),i(i(\bot,\bot),i(\bot,\bot))) \to_{r_{3.6}} \\
& t(i(\mathbf{r}(\bot,\bot),\mathbf{r}(\bot,\bot)),i(\mathbf{r}(\bot,\bot),i(\bot,\bot))) \to_{r_{3.6}} \\
& t(i(\mathbf{r}(\bot,\bot),\mathbf{r}(\bot,\bot)),i(\mathbf{r}(\bot,\bot),\mathbf{r}(\bot,\bot))) \to_{r_{3.5}} \\
& t(\mathbf{r}(\mathbf{r}(\bot,\bot),\mathbf{r}(\bot,\bot)),i(\mathbf{r}(\bot,\bot),\mathbf{r}(\bot,\bot))) \to_{r_{3.4}} \\
& t(\mathbf{r}(\mathbf{r}(\bot,\bot),\mathbf{r}(\bot,\bot)),\mathbf{r}(\mathbf{r}(\bot,\bot),\mathbf{r}(\bot,\bot))) \;=\; v
\end{aligned}$$

Nous remarquons que ce chemin n'est pas unique, *c.*à.*d.* il existe au moins un autre chemin de u à v.

Nous continuons à présenter les autres règles de TRS. La cellule de racine accorde la ressource a au plus un enfant, et le privilège se propage de haut en bas jusqu'à un des processeurs. Si les deux enfants d'une cellule demandent la ressource, la cellule fait donc un choix non-déterministe :

$$t(x, r(y,z)) \to b(x, t(y,z)) \tag{3.7}$$
$$t(r(x,y), z) \to b(t(x,y), z) \tag{3.8}$$

Par conséquent, de cette façon, seul un processeur ayant une demande de la ressource l'obtiendra. Quand le processeur a fini avec la ressource, il envoie une requête de libération qui est propagée de bas en haut.

$$b(x, t(\bot,\bot)) \to t(x, i(\bot,\bot)) \tag{3.9}$$
$$b(t(\bot,\bot), z) \to t(i(\bot,\bot), z) \tag{3.10}$$

Pour suivre le *retour à la racine* du jeton, trois comportements sont possibles pour une cellule :

- Si tous les enfants d'une telle cellule sont en état *idle*, cette cellule donne le jeton à la cellule voisine plus haute suivante et va également en état *idle*.

$$b(x, t(i(y,z), i(u,v))) \to t(x, i(i(y,z), i(u,v))) \tag{3.11}$$
$$b(t(i(x,y), i(u,v)), z) \to t(i(i(x,y), i(u,v)), z) \tag{3.12}$$

- Si un des enfants d'une telle cellule demande la ressource, alors :
  - soit l'état suivant de cette cellule sera *requesting* et le jeton va à la cellule voisine plus haute suivante en utilisant des règles suivantes

$$b(x, t(r(y,z), i(u,v))) \to t(x, r(r(y,z), i(u,v))) \tag{3.13}$$
$$b(x, t(i(y,z), r(u,v))) \to t(x, r(i(y,z), r(u,v))) \tag{3.14}$$
$$b(t(r(x,y), i(u,v)), z) \to t(r(r(x,y), i(u,v)), z) \tag{3.15}$$
$$b(t(i(x,y), r(u,v)), z) \to t(r(i(x,y), r(u,v)), z) \tag{3.16}$$

  - soit l'état suivant de cette cellule sera *token below* car la cellule a choisi de donner le jeton à un des enfants *requesting* en utilisant des règles (3.7) ou (3.8).

Nous notons par $\mathcal{R}^{TAP}$ le TRS complet défini en cette section. Nous pouvons obtenir l'ensemble de termes accessibles en calculant $(\mathcal{R}^{TAP})^\star(\{u\})$. De plus, nous pouvons également vérifier que le terme *total requesting* v est atteignable à partir du terme *total idle* u puis que $u \to^\star_{\mathcal{R}^{TAP}} v$.

3.2.3 Vérification du TAP à l'aide de automates d'arbres

Nous utilisons le même vocabulaire $\mathcal{F}_{bin}$ avec la section précédente *c.à.d* les symboles binaires i, r, t et b et la constante $\perp$.

Nous avons donc un automate d'arbre fini U acceptant u comme suit :

$$\xrightarrow{\perp} q_\perp$$
$$(q_\perp, q_\perp) \xrightarrow{i} q_i \qquad (q_i, q_i) \xrightarrow{i} q_i \qquad (q_i, q_i) \xrightarrow{t} q_t$$

Nous avons également un automate d'arbre fini V acceptant v comme suit :

$$\xrightarrow{\perp} q_\perp$$
$$(q_\perp, q_\perp) \xrightarrow{r} q_r \qquad (q_r, q_r) \xrightarrow{r} q_r \qquad (q_r, q_r) \xrightarrow{t} q_t$$

où q_t est un état accepté.

Remarques 1 *Il faut noter que dans l'exemple précédent : l'automate d'arbre U n'accepte pas seulement le terme u mais également un état total idle avec n'importe quelle taille. Similairement, l'automate d'arbre V n'accepte pas seulement le terme v mais également un état total requesting avec n'importe quelle taille. Des automates d'arbres ainsi que des transducteurs d'arbres qui n'acceptent que u et v sont plus complexes.*

[Abdulla 2006] nous présente un transducteur d'arbres pour résoudre le problème du TAP. Ainsi, les états sont appelés $q_i, q_r, q_t, q_{req}, q_{rel}, q_{grant}, q_m, q_{rt}$. Intuitivement, ces états signifient comme les suivants :

- q_i : Tous les noeuds au-dessous du noeud courant ainsi que ce noeud sont *idle*.
- q_r : Tous les noeuds au-dessous du noeud courant sont soit *idle* soit *requesting*, avec au moins un *requesting*. Il y a aucune propagation au-dessous de ce noeud.
- q_t : Le jeton est soit au-dessous de ce noeud, soit le jeton ne bougeait pas.
- q_{req} : Le noeud courant est en train de demander le jeton pour lui même ou pour un noeud au-dessous de ce noeud. Cette requête était propagée.
- q_{rel} : Le jeton est déplacé de bas en haut à partir du noeud courant.
- q_{grant} : Le jeton est déplacé de haut en bas jusqu'au noeud courant.
- q_m : Le jeton est dans ou au-dessous ce noeud, le jeton bougeait (C'est un état accepté).
- q_{rt} : Le jeton est soit dans soit au-dessous de ce noeud, *c.à.d*, il n'y a pas de changement ci-dessus du noeud courant (C'est un état accepté).

En utilisant ces états, les transitions du transducteur sont les suivantes :

$$\xrightarrow{(\perp,\perp)} q_\perp$$

$$
\begin{array}{lll}
(q_\perp, q_\perp) \xrightarrow{(i,i)} q_i & (q_\perp, q_\perp) \xrightarrow{(i,r)} q_{req} & (q_\perp, q_\perp) \xrightarrow{(r,r)} q_r \\
(q_\perp, q_\perp) \xrightarrow{(r,t)} q_{grant} & (q_\perp, q_\perp) \xrightarrow{(t,t)} q_t & (q_\perp, q_\perp) \xrightarrow{(t,i)} q_{rel} \\
(q_i, q_i) \xrightarrow{(i,i)} q_i & (q_r, q_i) \xrightarrow{(i,i)} q_i & (q_i, q_r) \xrightarrow{(i,i)} q_i \\
(q_r, q_r) \xrightarrow{(i,i)} q_r & (q_{req}, q_i) \xrightarrow{(i,i)} q_{req} & (q_i, q_{req}) \xrightarrow{(i,i)} q_{req} \\
(q_{req}, q_r) \xrightarrow{(i,i)} q_{req} & (q_r, q_{req}) \xrightarrow{(i,i)} q_{req} & (q_r, q_r) \xrightarrow{(i,r)} q_{req} \\
(q_r, q_i) \xrightarrow{(i,r)} q_{req} & (q_i, q_r) \xrightarrow{(i,r)} q_{req} & (q_r, q_r) \xrightarrow{(r,r)} q_r \\
(q_r, q_i) \xrightarrow{(r,r)} q_r & (q_i, q_r) \xrightarrow{(r,r)} q_r & (q_{req}, q_r) \xrightarrow{(r,r)} q_{req}
\end{array}
$$

$$
\begin{array}{lll}
(q_r, q_{req}) \xrightarrow{(r,r)} q_{req} & (q_r, q_i) \xrightarrow{(r,t)} q_{grant} & (q_i, q_r) \xrightarrow{(r,t)} q_{grant} \\
(q_r, q_r) \xrightarrow{(r,t)} q_{grant} & (q_i, q_i) \xrightarrow{(t,t)} q_t & (q_i, q_r) \xrightarrow{(t,t)} q_t \\
(q_r, q_i) \xrightarrow{(t,t)} q_t & (q_r, q_r) \xrightarrow{(t,t)} q_t & (q_i, q_{req}) \xrightarrow{(t,t)} q_{rt} \\
(q_{req}, q_i) \xrightarrow{(t,t)} q_{rt} & (q_r, q_{req}) \xrightarrow{(t,t)} q_{rt} & (q_{req}, q_r) \xrightarrow{(t,t)} q_{rt} \\
(q_i, q_{grant}) \xrightarrow{(t,b)} q_m & (q_{grant}, q_i) \xrightarrow{(t,b)} q_m & (q_{grant}, q_r)(t,b) \rightarrow q_m \\
(q_r, q_{grant}) \xrightarrow{(t,b)} q_m & (q_t, q_i) \xrightarrow{(b,b)} q_t & (q_t, q_r) \xrightarrow{(b,b)} q_t \\
(q_r, q_t) \xrightarrow{(b,b)} q_t & (q_i, q_t) \xrightarrow{(b,b)} q_t & (q_m, q_r) \xrightarrow{(b,b)} q_m \\
(q_m, q_i) \xrightarrow{(b,b)} q_m & (q_r, q_m) \xrightarrow{(b,b)} q_m & (q_i, q_m) \xrightarrow{(b,b)} q_m \\
(q_t, q_{req}) \xrightarrow{(b,b)} q_{rt} & (q_{req}, q_t) \xrightarrow{(b,b)} q_{rt} & (q_r, q_{rt}) \xrightarrow{(b,b)} q_{rt} \\
(q_{rt}, q_r) \xrightarrow{(b,b)} q_{rt} & (q_i, q_{rt}) \xrightarrow{(b,b)} q_{rt} & (q_{rt}, q_i) \xrightarrow{(b,b)} q_{rt} \\
(q_i, q_{rel}) \xrightarrow{(b,t)} q_m & (q_{rel}, q_i) \xrightarrow{(b,t)} q_m & (q_r, q_{rel}) \xrightarrow{(b,t)} q_m \\
(q_{rel}, q_r) \xrightarrow{(b,t)} q_m & (q_i, q_i) \xrightarrow{(t,i)} q_{rel} & (q_i, q_r) \xrightarrow{(t,r)} q_{rel} \\
(q_r, q_i) \xrightarrow{(t,r)} q_{rel} & (q_r, q_r) \xrightarrow{(t,r)} q_{rel} &
\end{array}
$$

3.3 Vérificateurs basés sur des TRSs et des automates d'arbres

Dans cette section, nous présentons quelques vérificateurs connus dans ce domaine basés sur des TRSs et des automates d'arbres.

Timbuk [Genet 2000, Genet 2001, Feuillade 2004, Genet 2003, Boichut 2007] est un ensemble d'outils pour l'analyse d'accessibilité sur TRSs. Cet outil manipule des automates d'arbres pour représenter des ensembles infinis possible de termes.

Maude [Denker 1998, Clavel 2001, Meseguer 2003] est un moteur de ré-écriture utilisé dans un contexte de vérification. Comme Timbuk, Maude peuvent être utilisé pour la vérification de systèmes infinis.

TOM [Balland 2007] Un langage dédié à la transformation de structures arborescentes permet d'embarquer des constructions de filtrage et de ré-écriture dans des langages généralistes comme Java ou C.

Dans le chapitre 8 , nous allons comparer notre outil avec les outils de vérification listés dans cette section. Nos benchmarks, étant le protocole TAP et les autres modèles arborescents présentés dans la section 6.1, démontrent l'efficacité de nos nouveaux formalismes pour la vérification de modèles.

BILAN DU CHAPITRE 3

Nous avons introduit les préliminaires des TRSs, des automates d'arbre et des transducteurs d'arbres pour des termes composés par des symboles fonctionnels binaires et la constante $\perp$. Nous avons également montré comment nous pouvons calculer la clôture transitive en TRSs et en transducteurs d'arbres. A partir de cette clôture transitive, on va plus loin pour résoudre les problèmes de vérification tels que la propriété de sûreté, l'inter-blocage, les logiques CTL et LTL, etc.

D'autre part, nous trouvons que le contrôle du processus de ré-écriture est encore illisible, cela empêche l'efficacité des techniques telles que le *cache à la BDD* (BDD-like cache). En outre, les techniques d'accélération habituellement utilisé dans l'algorithme de saturation sur les structures de données *à la BDD* sont inutilisables.

Ainsi, les motivations d'un nouvel outil que nous allons proposer dans la partie suivante seront d'une part de faciliter le contrôle du processus de ré-écriture, de rendre complètement indépendant de l'implémentation du moteur de ré-écriture, et d'autre part d'offrir un formalisme bien adapté aux techniques d'accélération.

Deuxième partie

Partie II : Système de Ré-écriture Fonctionnel (FTRS)

Chapitre 4

Systèmes de Ré-écriture Fonctionnels

RÉSUMÉ DU CHAPITRE 4

Dans ce chapitre, nous proposons les systèmes de ré-écriture fonctionnels (FTRSs). Nous montrons que notre modèle a la puissance d'expression des systèmes de ré-écriture (TRSs) malgré la simplicité des règles et le mode de fonctionnement légèrement différent par rapport aux systèmes de ré-écriture classiques.

Nous allons mettre en évidence une sous classe de systèmes fonctionnels, les élémentaires (EFTRSs), préservant la puissance d'expression des systèmes fonctionnels et des techniques d'accélération des calculs aboutissant à un outil de vérification symbolique efficace.

En fait, les EFTRSs sont des FTRSs ayant le côté gauche de la règle simplifié. De plus, nous allons continuer à simplifier le côté droit des règles d'EFTRSs pour que les systèmes de ré-écriture soient vraiment élémentaires. Nous les appelons les élémentaires à droite (REFTRSs). D'une part, ils préservent également la puissance d'expression des systèmes fonctionnels et d'autre part ils sont capables de mieux appliquer des techniques d'accélération des calculs.

La relation entre les systèmes de ré-écriture listés ci-dessus est présentée en figure 4.1. La flèche grasse signifie une transformation directe quand la flèche pointillée est une conversion indirecte en considérant les systèmes fonctionnels comme un type spécial des systèmes classiques.

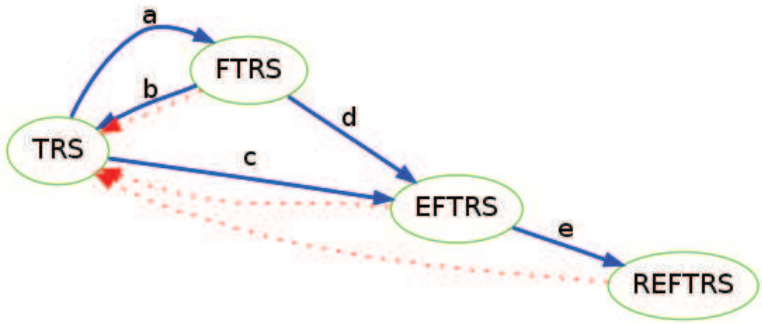


FIG. 4.1 – Relation entre des systèmes de ré-écriture

Dans la figure 4.1, les transformations entre TRSs et FTRSs sont représentées par les flèches (a) et (b). La transformation de TRSs vers EFTRSs est représentée par la flèche (c). Similairement, la flèche (d) signifie la transformation de FTRSs vers EFTRSs, et la flèche (e) signifie la transformation de EFTRSs vers REFTRSs. Enfin, les flèches pointillées signifient les conversions indirectes des systèmes fonctionnels vers le système classique.

Les transformations sont plus simples dans notre étude de cas (TAP) mais en général, les transformations sont techniques pour traiter tous les cas possibles des règles données.

Sommaire

4.1 Systèmes de Ré-écriture Fonctionnels (FTRSs)

Dans cette section, nous proposons le *système de ré-écriture fonctionnel (Functional Term Rewriting Systems, FTRS)* capable de simuler les TRSs classiques. Les objectifs de ce nouvel outil sont d'une part de faciliter le contrôle du processus de ré-écriture, de rendre complètement indépendant de l'implémentation du moteur de ré-écriture, et d'autre part d'offrir un formalisme bien adapté aux techniques d'accélération habituellement utilisé dans l'algorithme de saturation sur les structures de données *à la BDD*.

Cette section contient également les transformations entre les systèmes de ré-écriture fonctionnels et les systèmes de ré-écriture classiques.

4.1.1 Principes

Tout d'abord, nous introduisons de nouveaux symboles fonctionnels appelés non-terminaux. Nous dénotons par $\mathcal{F}_{NT}$ l'ensemble de symboles non-terminaux tel que $\mathcal{F}_{NT} \cap \mathcal{F}_{bin} = \emptyset$ et pour chaque $H \in \mathcal{F}_{NT}$, l'arité de H est une. Pour faire la distinction entre des symboles fonctionnels plus facile, désormais, seuls les symboles fonctionnels de $\mathcal{F}_{NT}$ seront écrits en lettres majuscules. Un TRS fonctionnel $\mathcal{R}_\lambda$ est un ensemble de règles de la forme $H(t) \to \alpha$ où $H \in \mathcal{F}_{NT}$, $t \in \mathcal{T}(\mathcal{F}_{bin}, \mathcal{X})$ et $\alpha \in \mathcal{T}(\mathcal{F}_{bin} \cup \mathcal{F}_{NT}, \mathcal{X})$. Un FTRS $\mathcal{R}_\lambda$ induit une relation de ré-écriture fonctionnelle $\to_{\mathcal{R}_\lambda}$ sur des termes dans $\mathcal{T}(\mathcal{F}_{bin} \cup \mathcal{F}_{NT})$: $u \to_{\mathcal{R}_\lambda} v$ ss'il existe une position p de u et une règle $H(t) \to \alpha$ dans $\mathcal{R}_\lambda$ telles qu'il existe une substitution $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}_{bin})$ avec $u|_p = H(t)\sigma$ et $v = u[\alpha\sigma]_p$.

Nous dénotons également par $\to^\star_{\mathcal{R}_\lambda}$ la clôture transitive de la relation de ré-écriture fonctionnelle $\to_{\mathcal{R}_\lambda}$. La définition suivante décrit l'ensemble de termes de $\mathcal{T}(\mathcal{F}_{bin})$ accessibles par la ré-écriture fonctionnelle d'un terme de $\mathcal{T}(\mathcal{F}_{bin} \cup \mathcal{F}_{NT})$.

Définition 1 ($\mathcal{R}^\star_\lambda$) *Soient $\mathcal{R}_\lambda$ un FTRS et $E \subseteq \mathcal{T}(\mathcal{F}_{bin} \cup \mathcal{F}_{NT})$ tels que $E \cap \mathcal{T}(\mathcal{F}_{bin}) = \emptyset$. L'ensemble de termes accessibles de $\mathcal{T}(\mathcal{F}_{bin})$ à partir de E est dénoté par $\mathcal{R}^\star_\lambda(E)$ et est défini ci-dessous :*

$$\mathcal{R}^\star_\lambda(E) = \{\beta \in \mathcal{T}(\mathcal{F}_{bin}) | \alpha \to^\star_{\mathcal{R}_\lambda} \beta \wedge \alpha \in E\}.$$

Remarques 2 *Un système de ré-écriture fonctionnel a trois différences par rapport aux systèmes de ré-écriture classiques :*

1. *Simplicité des règles (avec des non-terminaux de $\mathcal{F}_{NT}$).*
2. *Mode de fonctionnement : La règle de FTRS $H(t) \to \alpha$ est applicable sur un terme de la forme $H(t)$ seulement si le terme t ne contient aucun non-terminal.*
3. *Calcul des termes accessibles (avec le contrôle du processus de ré-écriture par des non-terminaux).*

Les propriétés linéaires des termes sont déjà parlé à la section 3.1. Maintenant, sur un ensemble de termes de $\mathcal{F}_{bin} \cup \mathcal{F}_{NT}$, nous avons la propriété linéaire suivante :

- $H(s_1) + H(s_2) = H(s_1 + s_2)$

avec H un symbole de $\mathcal{F}_{NT}$. En pratique, une telle transformation simple est appelée la canonisation.

Nous allons illustrer cette structure par un exemple.

Exemple 12 *Fonctionnement de FTRS.*

Soit $\mathcal{R}_\lambda$ le FTRS suivant :

$$\mathcal{R}_\lambda = \left\{ \begin{array}{l} F(x) \to H(G(x)) \\ H(x) \to a(x,x) \\ G(x) \to 0(\bot,\bot) \\ G(x) \to 1(\bot,\bot) \end{array} \right\}.$$

L'ensemble de termes accessibles $\mathcal{R}_\lambda^\star(\{\bot\})$ est calculé :

- *en mode fonctionnel :*

$$\begin{aligned} & F(\bot) \to H(G(\bot)) \\ & \to H(0(\bot,\bot)) \ + \ H(1(\bot,\bot)) \\ & \to a(0(\bot,\bot),0(\bot,\bot)) \ + \ a(1(\bot,\bot),1(\bot,\bot)) \end{aligned}$$

- *en mode non-fonctionnel (comme le mode de fonctionnement des TRSs classiques) :*

$$\begin{aligned} & F(\bot) \to H(G(\bot) \\ & \to H(0(\bot,\bot)) \ + \ H(1(\bot,\bot)) \ + \ a(G(\bot),G(\bot)) \\ & \to a(0(\bot,\bot),0(\bot,\bot)) \ + \ a(1(\bot,\bot),1(\bot,\bot)) \\ & + a(0(\bot,\bot),1(\bot,\bot)) \ + \ a(1(\bot,\bot),0(\bot,\bot)) \end{aligned}$$

La différence est dans le mode non-fonctionnel,$H(x) \to a(x,x)$ est applicable même si $t = G(\bot)$ contient un non-terminal G.

Remarques 3 *Si un FTRS est linéaire gauche et droite, nous pouvons montrer que l'ensemble des états accessibles sans non-terminaux de ce FTRS en utilisant un mode de fonctionnement des TRSs classiques.*

Spécification du protocole d'Arbitrage Arborescent en FTRS

Dans la section 3.2, nous avons montré comment nous pouvons traiter le TAP, un protocole résolvant le problème d'exclusion mutuelle pour accéder à une ressource partagée, en TRSs.

Cette section montre comment le TAP peut être formalisé en FTRSs sous forme d'un problème de calcul d'accessibilité.

Soit $\mathcal{F}_{NT}$ un ensemble de symboles non-terminaux tels que $\mathcal{F}_{NT} = \{H, Arbiter\}$. L'ensemble $\mathcal{F}_{bin}$ est celui défini dans la section 3.2, *c.à.d.*, $\{i, r, t, b, \bot\}$.

– **Règles générées.** Chaque règle normale de TRS $\alpha \to \beta$ sera être convertie en un FTRS règle de la manière suivante : $H(\alpha) \to \beta$ avec $\alpha, \beta \in \mathcal{T}(\mathcal{F}_{bin}, \mathcal{X})$. Par exemple, $\mathcal{R}_\lambda^{req}$ générées à partir des règles de ré-écriture (3.4), (3.5) et (3.6) données dans la section 3.2.

$$H(i(x, r(y, z))) \to r(x, r(y, z)) \tag{4.1}$$
$$H(i(r(x, y), z)) \to r(r(x, y), z) \tag{4.2}$$
$$H(i(\bot, \bot)) \to r(\bot, \bot) \tag{4.3}$$

– **Règles de circulation $\mathcal{R}_\lambda^{cir}$.** Pour que les règles générées soient applicables, nous avons besoin d'un ensemble de règles de FTRS comme indiqué ci-dessous :

$$H(a(x, y)) \to a(H(x), y),$$

$$H(a(x, y)) \to a(x, H(y)) | a \in \mathcal{F}_{bin}, H \in \mathcal{F}_{NT}$$

Ces règles nous permettent de circuler le non-terminal H à tous les endroits du terme donné.

– **Règles de Point Fixe $\mathcal{R}_\lambda^{FP}$.** Le calcul de point fixe est simulé par les règles suivantes :

$$Arbiter(x) \to x$$

$$Arbiter(x) \to Arbiter(H(x)) | Arbiter, H \in \mathcal{F}_{NT}$$

Ces règles nous permettent de générer le non-terminal H autant qu'on a besoin.

Nous pouvons obtenir l'ensemble de termes accessibles en calculant $(\mathcal{R}_\lambda^{req})^\star(\{Arbiter(u)\})$. De plus, nous pouvons également vérifier que le terme *total requesting* v est atteignable à partir du terme *total idle* u puis que $u \to^\star_{\mathcal{R}_\lambda^{req} \cup \mathcal{R}_\lambda^{cir} \mathcal{R}_\lambda^{FP}} v$.

Le terme *total requesting* v est atteignable à partir du terme *total idle* u puis que $u \to^\star v$. Nous listons ci-dessus la réduction détaillée :

$$\mathbf{Arbiter}(u) \;=\; \mathbf{Arbiter}(t(i(i(\bot, \bot), i(\bot, \bot)), i(i(\bot, \bot), i(\bot, \bot)))) \to_{\mathcal{R}_\lambda^{FP}}$$
$$\mathbf{Arbiter}(\mathbf{H}(t(i(i(\bot, \bot), i(\bot, \bot)), i(i(\bot, \bot), i(\bot, \bot))))) \to^\star_{\mathcal{R}_\lambda^{cir}}$$
$$\mathbf{Arbiter}(t(i(\mathbf{H}(i(\bot, \bot)), i(\bot, \bot)), i(i(\bot, \bot), i(\bot, \bot)))) \to_{r_{4.3}}$$
$$\mathbf{Arbiter}(t(i(i(\mathbf{r}(\bot, \bot)), i(\bot, \bot)), i(i(\bot, \bot), i(\bot, \bot)))) \to_{\mathcal{R}_\lambda^{FP}}$$
$$\mathbf{Arbiter}(\mathbf{H}(t(i(i(\mathbf{r}(\bot, \bot), i(\bot, \bot)), i(i(\bot, \bot), i(\bot, \bot))))) \to^\star$$
$$\ldots$$
$$\mathbf{Arbiter}(t(\mathbf{r}(\mathbf{r}(\bot, \bot), \mathbf{r}(\bot, \bot)), \mathbf{r}(\mathbf{r}(\bot, \bot), \mathbf{r}(\bot, \bot)))) \to_{\mathcal{R}_\lambda^{FP}}$$
$$t(\mathbf{r}(\mathbf{r}(\bot, \bot), \mathbf{r}(\bot, \bot)), \mathbf{r}(\mathbf{r}(\bot, \bot), \mathbf{r}(\bot, \bot))) \;=\; v$$

Nous notons par $\mathcal{R}_\lambda^{TAP}$ le FTRS complet défini en cette section.

4.1.2 Transformation de TRSs vers FTRSs

Nous discutons maintenant l'expressivité de FTRSs et de TRSs. La proposition 1 montre qu'un pas de ré-écriture utilisant $\mathcal{R}$ peut être simulé par un FTRS $\mathcal{R}_\lambda$ avec un seul non-terminal $F_{l\to r}$.

Proposition 1 *Soit $\mathcal{R}$ un TRS sur $\mathcal{T}(\mathcal{F}_{bin},\mathcal{X})\times\mathcal{T}(\mathcal{F}_{bin},\mathcal{X})$, $\alpha\in\mathcal{T}(\mathcal{F}_{bin})$, $\beta\in\mathcal{T}(\mathcal{F}_{bin})$ et $l\to r\in\mathcal{R}$. Soit $\mathcal{R}_\lambda$ le FTRS tel que*

$$\mathcal{R}_\lambda=\{F_{l\to r}(l)\to r\}\cup\left\{\begin{array}{l}F_{l\to r}(a(x,y))\to a(F_{l\to r}(x),y),\\ F_{l\to r}(a(x,y))\to a(x,F_{l\to r}(y))|\\ a\in\mathcal{F}_{bin}\end{array}\right\}.$$

Ainsi,

$$\alpha\to_{\{l\to r\}}\beta\Leftrightarrow F_{l\to r}(\alpha)\to^\star_{\mathcal{R}_\lambda}\beta.$$

Remarques 4 *Il faut noter que dans $F_{l\to r}(\alpha)\to^\star_{\mathcal{R}_\lambda}\beta$ est la clôture transitive de la relation de ré-écriture fonctionnelle pour que $\alpha\to_{\{l\to r\}}\beta$ est établie exactement une fois.*

Preuve 1 (Preuve succincte (Voir cf. dans l'annexe A)) .

- $\alpha\to_{\{l\to r\}}\beta\Rightarrow F_{l\to r}(\alpha)\to^\star_{\mathcal{R}_\lambda}\beta$:
  Selon la définition, il existe une position p de α et une substitution $\sigma:\mathcal{X}\to\mathcal{T}(\mathcal{F}_{bin})$ telles que $\alpha|_p=l\sigma$ et $\beta=\alpha[r\sigma]_p$. Soit α' un terme construit comme le suivant : $\alpha'=\alpha[F_{l\to r}(\alpha|_p)]_p$.
  Clairement, en utilisant seulement les règles de circulation, c.à.d. les règles de la forme $F_{l\to r}(a(x,y))\to a(F_{l\to r}(x),y)$ et $F_{l\to r}(a(x,y))\to a(x,F_{l\to r}(y))$, nous avons

$$F_{l\to r}(\alpha)\to^\star_{\mathcal{R}_\lambda}\alpha'. \tag{4.4}$$

  Selon la construction de α', il existe une règle générée de $\mathcal{R}_\lambda$, c.à.d $F_{l\to r}(l)\to r$, une substitution $\sigma:\mathcal{X}\to\mathcal{T}(\mathcal{F}_{bin})$ et une position p telles que $\alpha'|_p=F_{l\to r}(l)\sigma$. En conséquence, nous pouvons construire le terme $\alpha'[r\sigma]$ qui est égal à β par la construction de α'. Nous avons donc :

$$\alpha'\to_{\mathcal{R}_\lambda}\beta. \tag{4.5}$$

  Ainsi, en utilisant (4.4) et (4.5), nous pouvons déduire que $F_{l\to r}(\alpha)\to^\star_{\mathcal{R}_\lambda}\beta$. Par conséquent, $\alpha\to_{\{l\to r\}}\beta\Rightarrow F_{l\to r}(\alpha)\to^\star_{\mathcal{R}_\lambda}\beta$.
- $\alpha\to_{\{l\to r\}}\beta\Leftarrow F_{l\to r}(\alpha)\to^\star_{\mathcal{R}_\lambda}\beta$: *Effectivement, poursuivons en induction sur le chemin de ré-écriture qui nous conduit à la revendication.*

Ainsi, la transformation (présentée par la flèche (a) de la figure 4.1), étant le résultat principal de cette section, est donné par le théorème 1 *c.à.d.* que pour tous TRS $\mathcal{R}$, nous pouvons construire un FTRS $\mathcal{R}_\lambda$ simulant $\mathcal{R}$ en utilisant des non-terminaux $F_{l\to r}$ pour chaque $l\to r\in\mathcal{R}$ et un non-terminal G pour le point fixe.

Théorème 1 *Soit $\mathcal{R}$ un TRS sur $\mathcal{T}(\mathcal{F}_{bin},\mathcal{X})\times\mathcal{T}(\mathcal{F}_{bin},\mathcal{X})$, $E\subseteq\mathcal{T}(\mathcal{F}_{bin})$. Soit $l\to r$ une règle de $\mathcal{R}$ et $\mathcal{R}_{l\to r}$ le FTRS tels que*

$$\mathcal{R}_{l\to r}=\{F_{l\to r}(l)\to r\}\cup\left\{\begin{array}{l}F_{l\to r}(a(x,y))\to a(F_{l\to r}(x),y),\\ F_{l\to r}(a(x,y))\to a(x,F_{l\to r}(y))|\\ a\in\mathcal{F}_{bin}\end{array}\right\}.$$

Soit $\mathcal{R}_{FP}$ le FTRS tel que

$$\mathcal{R}_{FP}=\{G(x)\to G(F_{l\to r}(x))\mid l\to r\in\mathcal{R}\}\cup\{G(x)\to x\}.$$

Soit $\mathcal{R}_\lambda$ le FTRS tel que

$$\mathcal{R}_\lambda=\bigcup_{l\to r\in\mathcal{R}}\mathcal{R}_{l\to r}\cup\mathcal{R}_{FP}.$$

Ainsi,

$$\mathcal{R}^\star(E)=\mathcal{R}_\lambda^\star(E')$$

avec $E'=\{G(t)|t\in E\}$.

Remarques 5 *Il faut noter que la clôture réflexive et transitive de TRS est simulée par $\mathcal{R}_{FP}=\{G(x)\to G(F_{l\to r}(x))\mid l\to r\in\mathcal{R}\}\cup\{G(x)\to x\}$.*

Preuve 2 (Preuve succincte (Voir cf. dans l'annexe A)) .

- *$\mathcal{R}^\star(E)\subseteq\mathcal{R}_\lambda^\star(E')$: Soit $\alpha\in E$ et $\beta\in\mathcal{R}^\star(\{\alpha\})$. Selon la définition de $\mathcal{R}^\star$, il existe $l_1\to r_1,\dots l_n\to r_n\in\mathcal{R}$ et $t_0,\dots,t_n$ telles que $\alpha\to_{\{l_0\to r_0\}}t_0\to_{\{l_1\to r_1\}}\dots\to_{\{l_{n-1}\to r_{n-1}\}}t_{n-1}\to_{\{l_n\to r_n\}}t_n=\beta$. Le FTRS $\mathcal{R}_{l_0\to r_0}$ remplit les conditions attendues spécifiées à la proposition 1.*
  En généralisant ce processus, nous obtenons le chemin de ré-écriture ci-dessous : $G(t_0)\to^\star_{\mathcal{R}_\lambda}G(t_1)\dots\to^\star_{\mathcal{R}_\lambda}G(t_n)\to_{G(x)\to x}t_n=\beta$. Par conséquent, $\beta\in\mathcal{R}_\lambda^\star(E')$.
  Ainsi, nous pouvons déduire que

$$\mathcal{R}^\star(\{\alpha\})\subseteq\mathcal{R}_\lambda^\star(\{G(\alpha)\}).\tag{4.6}$$

- *$\mathcal{R}_\lambda^\star(E')\subseteq\mathcal{R}^\star(E)$: Étudions $\mathcal{R}_\lambda^\star(\{G(\alpha)\})$.*
  - *Supposons qu'une règle de la forme $G(x)\to G(F_{l\to r}(x))$ est appliquée. Ainsi, $G(\alpha)\to_{G(x)\to G(F_{l\to r}(x))}G(F_{l\to r}(\alpha)))$. De plus, d'après la proposition 1, nous pouvons déduire que*

$$\mathcal{R}^\star(\{\alpha\})\supseteq\mathcal{R}_\lambda^\star(\{G(\alpha)\}).\tag{4.7}$$

  - *Supposons qu'aucune règle de la forme $G(x)\to G(F_{l\to r}(x))$ est appliquée. Par conséquent, aucune règle de $\bigcup_{l\to r\in\mathcal{R}}\mathcal{R}_{l\to r}$ peut être appliquée .*
    L'unique terme atteint de $\mathcal{T}(\mathcal{F}_{bin})$ est donc obtenu en appliquant la règle $G(x)\to x$. Ainsi, $\alpha\in\mathcal{R}_\lambda^\star(\{G(\alpha)\})$. D'après la définition, $\alpha\in\mathcal{R}^\star(\{\alpha\})$.

  Pour conclusion, pour tous $\alpha\in E$, de (4.6) et (4.7), nous obtenons que $\mathcal{R}^\star(E)=\mathcal{R}_\lambda^\star(E')$.

4.1.3 Transformation de FTRSs vers TRSs

Dans la section précédente, nous avons prouvé que soit un TRS, il existe un FTRS correspondant. La réciproque est posée dans cette section.

La transformation (présentée par la flèche (b) de la figure 4.1) est donnée par le lemme 1 *c.à.d* pour un FTRS $\mathcal{R}_\lambda$ satisfaisant une telle condition, nous pouvons construire un TRS $\mathcal{R}$ correspondant .

Lemme 1 *Soit un FTRS $\mathcal{R}_\lambda$ sur $\mathcal{T}(\mathcal{F}_{bin} \cup \mathcal{F}_{NT}, \mathcal{X}) \times \mathcal{T}(\mathcal{F}_{bin} \cup \mathcal{F}_{NT}, \mathcal{X})$. Soit le TRS sur $\mathcal{T}(\mathcal{F}_{bin}, \mathcal{X}) \times \mathcal{T}(\mathcal{F}_{bin}, \mathcal{X})$ tels que pour chaque règle $l \to r \in \mathcal{R}$, il existe un sous ensemble FTRS $\mathcal{R}_{l\to r} \subset \mathcal{R}_\lambda$ tel que*

$$\mathcal{R}_{l\to r} = \{F_{l\to r}(l) \to r\} \cup \left\{ \begin{array}{l} F_{l\to r}(a(x,y)) \to a(F_{l\to r}(x), y), \\ F_{l\to r}(a(x,y)) \to a(x, F_{l\to r}(y))| \\ a \in \mathcal{F}_{bin} \end{array} \right\}.$$

et il existe également une règle $r_{FP} \in \mathcal{R}_{FP}$ tel que

$$r_{FP} \;=\; G(x) \to G(F_{l\to r}(x))$$

et $G(x) \to x \;\in \mathcal{R}_{FP}$ tels que

$$\bigcup_{l\to r\in\mathcal{R}} \mathcal{R}_{l\to r} \cup \mathcal{R}_{FP} = \mathcal{R}_\lambda.$$

Ainsi,

$$\mathcal{R}^\star_\lambda(E') = \mathcal{R}^\star(E)$$

avec $E \subseteq \mathcal{T}(\mathcal{F}_{bin})$ et $E' = \{G(t)|t \in E\}$.

Preuve 3 (Preuve succincte) *Supposons un tel TRS construit comme dans le lemme 1. Selon le théorème 1, nous obtenons $\mathcal{R}^\star_\lambda(E') = \mathcal{R}^\star(E)$.*

Etant donné un TRS, il existe toujours un FTRS correspondant. Mais la réciproque n'est pas souvent possible. Regardons l'exemple donné ci-dessous.

Exemple 13 *Considérons un FTRS qui travaille comme un filtrage OnlyOneRequest des états du TAP ayant une seule demande de ressource qui est déjà propagée jusqu'à la racine. Ainsi,*

$$\mathcal{R}_\lambda = \left\{ \begin{array}{l} OnlyOne(t(r(x,y), i(z,t))) \to t(OnlyOne(r(x,y)), i(z,t)) \\ OnlyOne(t(i(x,y), r(z,t))) \to t(i(x,y), OnlyOne(r(z,t))) \\ OnlyOne(r(r(x,y), i(z,t))) \to r(OnlyOne(r(x,y)), i(z,t)) \\ OnlyOne(r(i(x,y), r(z,t))) \to r(i(x,y), OnlyOne(r(z,t))) \\ OnlyOne(r(\bot,\bot)) \to r(\bot,\bot) \end{array} \right\}.$$

Il n'existe pas de TRS qui fonctionne de la même manière. Mais nous pouvons définir une propriété *OnlyOneRequest* sur $\alpha \in \mathcal{T}(\mathcal{F}_{bin})$ telle que pour tout $p, p' \in \mathcal{P}os(\alpha)$, si $\alpha(p) = r, p \neq p'$, alors $p \preceq p'$ ou $p' \preceq p$. Dans la suite, nous allons revenir à ce sujet à la section *Analyse d'accessibilité pour les invariants* du chapitre 7.

Dans la section 4.4, nous allons chercher une conversion indirecte en considérant les systèmes fonctionnels comme un type spécial des systèmes classiques.

4.2 FTRSs Elémentaires (EFTRSs)

4.2.1 Principes

Dans cette section, étant donné un TRS $\mathcal{R}$ sur $\mathcal{T}(\mathcal{F}_{bin}, \mathcal{X}) \times \mathcal{T}(\mathcal{F}_{bin}, \mathcal{X})$ et un terme $t \in \mathcal{T}(\mathcal{F}_{bin})$, nous montrons que c'est possible de calculer exactement le même ensemble de termes accessibles en utilisant un FTRS de la forme particulière *c.à.d.* FTRS élémentaire.

Définition 2 *Soit $\mathcal{R}_{\lambda^e}$ un FTRS. $\mathcal{R}_{\lambda^e}$ est dit d'être élémentaire (également appelé EFTRS) seulement si chaque règle de $\mathcal{R}_{\lambda^e}$ est une des formes ci-dessous :*

1. $H(a(x, y)) \rightarrow \alpha$ *avec* $a \in \mathcal{F}_{bin}$, $H \in \mathcal{F}_{NT}$, $x, y \in \mathcal{X}$, $x \neq y$, $\alpha \in \mathcal{T}(\mathcal{F}_{bin} \cup \mathcal{F}_{NT}, \mathcal{X})$ *et* $\mathcal{V}ar(\alpha) = \{x, y\}$
2. $H(\perp) \rightarrow \alpha$ *avec* $\alpha \in \mathcal{T}(\mathcal{F}_{bin} \cup \mathcal{F}_{NT})$
3. $H(a(x, \perp)) \rightarrow x$ *avec* $x \in \mathcal{X}$, $H \in \mathcal{F}_{NT}$ *et* $a \in \mathcal{F}_{bin}$.

En fait, les EFTRSs de la définition 2, sont suffisamment expressives pour la spécification du mécanisme de ré-écriture pour une règle $l \rightarrow r \in \mathcal{R}$:

1. de trouver une position pour la ré-écriture,
2. de vérifier si l concorde avec le sous-terme courant, puis
3. de calculer la substitution résultant σ, et
4. de replacer le sous-terme par $r\sigma$.

Spécification du protocole d'Arbitrage Arborescent en EFTRS

Dans la section 3.2, nous avons montré comment nous pouvons traiter le TAP en TRSs.

Cette section montre comment le TAP peut être formalisé en EFTRSs sous forme d'un problème de calcul d'accessibilité.

Soit $\mathcal{F}_{NT}$ un ensemble de symboles non-terminaux tel que $\mathcal{F}_{NT} = \{H, R, RT, TI, I, Arbiter\}$. L'ensemble $\mathcal{F}_{bin}$ est celui défini dans la section 3.2, *c.à.d*, $\{i, r, t, b, \perp\}$.

Les règles de ré-écriture (3.4), (3.5) et (3.6) données dans la section 3.2 sont transformées comme les règles d'EFTRS $\mathcal{R}_{\lambda^e}^{req}$ comme indiqué ci-dessous.

$$H(i(x,y)) \to r(x,R(y)) \tag{4.8}$$
$$H(i(x,y)) \to r(R(x),y) \tag{4.9}$$
$$R(r(x,y)) \to r(x,y) \tag{4.10}$$
$$R(\bot) \to \bot \tag{4.11}$$

Nous illustrons la ré-écriture de la configuration suivante $t_0 = H(t(i(\bot,\bot),i(\bot,\bot)))$. Aucune règle mentionnée ci-dessus ne peut être appliquée. Ainsi, aucun terme de $\mathcal{T}(\mathcal{F}_{bin})$ n'est accessible de $H(t(i(\bot,\bot),i(\bot,\bot)))$.

Nous avons donc besoin de règles pour spécifier la circulation du symbole H avant d'appliquer les règles (4.8), ..., (4.11). Ces règles $\mathcal{R}_{\lambda^e}^{cir}$ sont données ci-dessous et sont définies pour chaque symbole de $\mathcal{F}_{bin}$ sauf $\bot$.

$$H(i(x,y)) \to i(H(x),y) \tag{4.12}$$
$$H(i(x,y)) \to i(x,H(y)) \tag{4.13}$$
$$H(r(x,y)) \to r(H(x),y) \tag{4.14}$$
$$H(r(x,y)) \to r(x,H(y)) \tag{4.15}$$
$$H(b(x,y)) \to b(H(x),y) \tag{4.16}$$
$$H(b(x,y)) \to b(x,H(y)) \tag{4.17}$$
$$H(t(x,y)) \to t(H(x),y) \tag{4.18}$$
$$H(t(x,y)) \to t(x,H(y)) \tag{4.19}$$

Maintenant nous illustrons l'utilisation des règles (4.12),..., (4.19). En appliquant la règle (4.12) (resp. (4.13)) sur t_0, le terme $t_1' = t(H(i(\bot,\bot)),i(\bot,\bot))$ (resp. $t_1'' = t(i(\bot,\bot),H(i(\bot,\bot))))$ est obtenu. En appliquant la règle (4.8) et la règle (4.11) sur t_1' (resp. t_1''), nous obtenons le terme $t_2' = t(r(\bot,\bot),i(\bot,\bot))$ (resp. $t_2'' = t(i(\bot,\bot),r(\bot,\bot))$).

Notez que t_2' et t_2'' ne contiennent aucun symbole de $\mathcal{F}_{NT}$. L'EFTRS ne peut donc plus s'appliquer. Ainsi, nous introduisons le symbole *Arbiter* dont les sémantiques sont décrites par les règles (4.30) à (4.33). Autrement dit, ce symbole nous permet de ré-écrire autant que nous avons besoin. Les règles indiquées ci-dessous remplissent la spécification du TAP par un EFTRS.

$$H(t(x,y)) \to b(x, RT(y)) \tag{4.20}$$
$$H(t(x,y)) \to b(RT(x), y) \tag{4.21}$$
$$RT(r(x,y)) \to t(x,y) \tag{4.22}$$
$$H(b(x,y)) \to t(x, TI(y)) \tag{4.23}$$
$$H(b(x,y)) \to t(TI(x), y) \tag{4.24}$$
$$TI(t(x,y)) \to i(I(x), I(y)) \tag{4.25}$$
$$TI(t(x,y)) \to r(R(x), I(y)) \tag{4.26}$$
$$TI(t(x,y)) \to r(I(x), R(y)) \tag{4.27}$$
$$I(i(x,y)) \to i(x,y) \tag{4.28}$$
$$I(\bot) \to \bot \tag{4.29}$$
$$Arbiter(t(x,y)) \to t(x,y) \tag{4.30}$$
$$Arbiter(t(x,y)) \to Arbiter(H(t(x,y))) \tag{4.31}$$
$$Arbiter(b(x,y)) \to b(x,y) \tag{4.32}$$
$$Arbiter(b(x,y)) \to Arbiter(H(b(x,y))) \tag{4.33}$$

Maintenant considérons le terme du début $t_0 = Arbiter(t(i(\bot,\bot), i(\bot,\bot)))$. De t_0 et en appliquant les règles (4.31), (4.12), (4.8) et (4.11), le terme $t_1 = Arbiter(t(i(\bot,\bot), r(\bot,\bot)))$ est calculé. En appliquant les règles (4.20) et (4.22) de t_1, le terme t_2 est calculé avec $t_2 = Arbiter(b(i(\bot,\bot), t(\bot,\bot)))$. Notez que de t_1 (resp. t_2), nous pouvons également appliquer la règle (4.30) (resp. 4.32) et puis obtient le terme $t(i(\bot,\bot), r(\bot,\bot))$ (resp. $b(i(\bot,\bot), t(\bot,\bot))$). Les deux termes mentionnés ci-dessus représentent deux configurations vraiment accessibles.

Nous pouvons obtenir l'ensemble de termes accessibles à partir du terme *total idle* en calculant $(\mathcal{R}^{req}_{\lambda^e})^\star(\{Arbiter(u)\})$. De plus, nous pouvons également vérifier que le terme *total requesting* v est atteignable à partir du terme *total idle* u puis que $u \to^\star_{\mathcal{R}^{req}_{\lambda^e}} v$.

Enfin, nous notons par $\mathcal{R}^{TAP}_{\lambda^e}$ l'EFTRS complet défini en cette section. Nous remarquons que $\mathcal{R}^{TAP}_{\lambda^e}$ est un des benchmarks utilisés dans le chapitre 8.

Remarques 6 *C'est possible de simuler la règle $H(x) \to \alpha(x)$ par les élémentaires. Nous traitons deux cas de variable x de cette règle :*

- $H(\bot) \to \alpha(\bot)$.
- $H(a(x,y)) \to \alpha(a(x,y))$. *avec* $x \in \mathcal{X}$, $H \in \mathcal{F}_{NT}$, $a \in \mathcal{F}_{bin}$, $\alpha(x), \alpha(\bot)$ *et* $\alpha(a(x,y)) \in \mathcal{T}(\mathcal{F}_{bin} \cup \mathcal{F}_{NT}, \mathcal{X})$.

Pour simplifier, nous considérons désormais que la règle $H(x) \to \alpha(x)$ est élémentaire. Par conséquent, les règles d'EFTRSs de point fixe 4.30 à 4.33 peuvent être écrites comme les suivant :

$$Arbiter(x) \to x$$
$$Arbiter(x) \to Arbiter(H(x))$$

La transformation est évidente dans notre étude de cas (TAP) mais en général, la transformation de TRSs vers EFTRSs est technique pour traiter tous les cas possibles des règles données.

4.2.2 Transformation de TRSs vers EFTRSs

Nous présentons 4 ensembles de règles d'EFTRSs de concordance $\mathcal{R}^{l\to r}_{check}$, de construction d'arbre de variables $\mathcal{R}^{l\to r}_{\sigma}$, de test de substitution $\mathcal{R}^{l\to r}_{GS}$ et de substitution $\mathcal{R}^{l\to r}_{\sigma-apply}$ de la sous section 4.2.2.1 à la sous section 4.2.2.4, et montrer comment les utiliser pour simuler la ré-écriture des TRSs dans la sous section 4.2.2.5.

Les principes de chaque ensemble de règles sont listés ci-dessus :

1. Règles d'EFTRSs de concordance $\mathcal{R}^{l\to r}_{check}$ est un EFTRS nous permettant de vérifier si un terme de $\mathcal{T}(\mathcal{F}_{bin})$ *concorde* avec l. Quand la concordance entre t et l n'est pas possible, $(\mathcal{R}^{l\to r}_{check})^*(\{F^{\varepsilon}_{l\to r}(t)\}) = \emptyset$.
2. Règles d'EFTRSs de construction d'arbre de variables $\mathcal{R}^{l\to r}_{\sigma}$ est un EFTRS nous permettant de construire une liste ordonnée de termes indexés par des variables de l. Cette liste est représentée par un terme et sémantiquement au courant, nous pouvons considérer cette liste comme la substitution résultant du pas *concordance*. Soit t' le terme résultant de $(\mathcal{R}^{l\to r}_{check})^*(\{F^{\varepsilon}_{l\to r}(t)\})$. $(\mathcal{R}^{l\to r}_{\sigma})^*(F^{l\to r}_{copy}(t'))$ mène à l'unique terme t'' représentant la substitution résultant de la *concordance* entre t et l.
3. Règles d'EFTRSs de test de substitution $\mathcal{R}^{l\to r}_{GS}$ est un EFTRS spécifiant la vérification d'une substitution bien-formée. Effectivement, si l n'est pas linéaire alors une variable x apparaît deux fois au moins. Si toutes les occurrences de x partagent le même terme comme valeur alors la substitution est bien-formée. $(\mathcal{R}^{l\to r}_{check})^*(F^{l\to r}_{check}(t'')) = \{t''\}$ est la substitution bien-formée, $\emptyset$ sinon.
4. Règles d'EFTRSs de substitution $\mathcal{R}^{l\to r}_{\sigma-apply}$ est un EFTRS spécifiant l'application de la substitution bien-formée résultant de t et l sur le terme r. Ainsi, $(\mathcal{R}^{l\to r}_{\sigma-apply})^*(F^{l\to r}_{rewrite}(t'')) = \{r\sigma\}$.

En utilisant les 4 ensembles de règles d'EFTRSs, nous sommes capables de simuler la ré-écriture d'un TRS.

4.2.2.1 Règles d'EFTRS de concordance : $\mathcal{R}^{l\to r}_{check}$

$\mathcal{R}^{l\to r}_{check}$ est un EFTRS nous permettant de vérifier si un terme de $\mathcal{T}(\mathcal{F}_{bin})$ *concorde* avec l. L'application de cet EFTRS est renvoyé par la présence du symbole non-terminal $F^{\varepsilon}_{l\to r}$. Etant donné un terme $t \in \mathcal{T}(\mathcal{F}_{bin})$ concordant avec l, $(\mathcal{R}^{l\to r}_{check})^*(\{F^{\varepsilon}_{l\to r}(t)\})$ est $\{t'\}$ où

- pour tout $p \in \mathcal{P}os_{\mathcal{F}}(l)$, $t(p) = t'(p)$;
- pour tout $p \in \mathcal{P}os_{\mathcal{X}}(l)$, $t'(p) = \oplus_{x,p} \in \mathcal{F}_{bin}$;
- pour tout $p \in \mathcal{FP}\mathrm{os}(l)\backslash\mathcal{P}os_{\mathcal{X}}(l)$, et pour tout $p' \in \mathcal{P}os(t|_p)$, $t(p.p') = t'(p.1.p')$.

Le symbole $\oplus_{x,p}$ est un marqueur signifiant que le terme sous ce marqueur est la valeur que la variable x à la position p dans l obtient quand la concordance apparaît. Quand la concordance entre t et l n'est pas possible, $(\mathcal{R}^{l\to r}_{check})^*(\{F^{\varepsilon}_{l\to r}(t)\}) = \emptyset$. Notez que, à ce point, nous ne vérifions pas le cas où deux variables identiques doivent partager la même valeur. Donc $(\mathcal{R}^{l\to r}_{check})^*(\{F^{\varepsilon}_{l\to r}(t)\}) = \emptyset$ seulement si t et l sont structurellement différents.

Définition 3 ($\mathcal{R}^{l\to r}_{check}$) *Soit $l \to r$ une règle de $\mathcal{T}(\mathcal{F}_{bin}, \mathcal{X}) \times \mathcal{T}(\mathcal{F}_{bin}, \mathcal{X})$. Nous dénotons par $\mathcal{R}^{l\to r}_{check}$ le EFTRS construit à partir de l comme suit :*

1. $F^{p}_{l\to r}(x) \to \oplus_{l|_p}(x) \in \mathcal{R}^{l\to r}_{check}$ *si* $p \in \mathcal{P}os_{\mathcal{X}}(l)$ *et où* $\oplus_{l|_p}$ *est un symbole spécial que nous considérons d'être dans* $\mathcal{F}_{bin}$
2. $F^{p}_{l\to r}(a(x,y)) \to a(F^{p.0}_{l\to r}(x), F^{p.1}_{l\to r}(y)) \in \mathcal{R}^{l\to r}_{check}$ *si* $p \in \mathcal{P}os(l) \setminus \mathcal{FP}\text{os}(l)$ *et* $a = l(p)$
3. $F^{p}_{l\to r}(\bot) \to \bot \in \mathcal{R}^{l\to r}_{check}$ *si* $p \in \mathcal{FP}\text{os} \setminus \mathcal{P}os_{\mathcal{X}}(l)$

Exemple 14 *Considérons la règle $a(b(x,\bot), c(d(\bot,y),z)) \to r$ comme une règle de ré-écriture, $x, y, z \in \mathcal{X}$ et $\mathcal{V}ar(r) \subseteq \{x,y,z\}$. Pour des raisons de lisibilité, nous dénotons ce terme par l. Ainsi,*

$$\mathcal{R}^{l\to r}_{check} = \left\{ \begin{array}{l} F^{\varepsilon}_{l\to r}(a(x,y)) \to a(F^{1}_{l\to r}(x), F^{2}_{l\to r}(y)) \\ F^{1}_{l\to r}(b(x,y)) \to b(F^{11}_{l\to r}(x), F^{12}_{l\to r}(y)) \\ F^{11}_{l\to r}(x) \to \oplus_x(x) \\ F^{12}_{l\to r}(\bot) \to \bot \\ F^{2}_{l\to r}(c(x,y)) \to c(F^{21}_{l\to r}(x), F^{22}_{l\to r}(y)) \\ F^{21}_{l\to r}(d(x,y)) \to d(F^{211}_{l\to r}(x), F^{212}_{l\to r}(y)) \\ F^{211}_{l\to r}(\bot) \to \bot \\ F^{212}_{l\to r}(x) \to \oplus_y(x) \\ F^{22}_{l\to r}(x) \to \oplus_z(x) \end{array} \right\}.$$

Le but de l'ensemble de règles de ré-écriture $\mathcal{R}^{l\to r}_{check}$ présenté ci-dessus est de vérifier qu'un terme t donné est de la forme souhaitée *c.à.d* l. En appliquant l'EFTRS de l'exemple 14 sur le terme $F^{\varepsilon}_{l\to r}(t)$ dèsque nous obtenons un point fixe de l'ensemble de termes successeurs dans $\mathcal{T}(\mathcal{F}_{bin})$, si cet ensemble est vide, alors t ne concorde pas avec l. Sinon, l'ensemble de termes successeurs est vraiment un singleton doté du terme t marqué à la position des variables dans l avec un symbole spécial $\oplus_x$ où x est un variable de l.

4.2.2.2 Règles d'EFTRS de construction d'arbre de variables : $\mathcal{R}^{l\to r}_{\sigma}$

$\mathcal{R}^{l\to r}_{\sigma}$ est un EFTRS nous permettant de construire une liste ordonnée de termes indexés par des variables de l. Cette liste est représentée par un terme et sémantiquement au courant, nous pouvons considérer cette liste comme la substitution résultant du pas *concordance*. Soit t' le terme résultant de $(\mathcal{R}^{l\to r}_{check})^*(\{F^{\varepsilon}_{l\to r}(t)\})$. $(\mathcal{R}^{l\to r}_{\sigma})^*(F^{l\to r}_{copy}(t'))$ mène à l'unique terme t'' représentant la substitution résultant de la *concordance* entre t et l.

Nous introduisons la notion d'*arbre de variables* construite à partir d'un terme. Pour simplifier, les variables de l sont renommées dans des variables x_i avec i étant un entier.

Définition 4 *Soit t un terme de $\mathcal{T}(\mathcal{F}_{bin}, \mathcal{X})$. Soit $x_0, \ldots, x_n$ les variables apparaissant dans t de gauche à droite. L'arbre de variables vide (arbre de variable en bref) de t, dénoté par $\top_{\mathcal{X}}(t)$, est défini par le terme suivant : $\top(\oplus_{x_0}(\bot), \top(\oplus_{x_1}(\bot), \ldots, \top(\oplus_{x_n}(\bot), \bot_{\top}) \ldots))$.*

Exemple 15 *Soit t un terme de $\mathcal{T}(\mathcal{F}_{bin}, \mathcal{X})$ tel que $t = a(b(x_0, \bot), c(d(\bot, x_1), x_2))$. Ainsi,*

$$\top_{\mathcal{X}}(t) = \top(\oplus_{x_0}(\bot), \top(\oplus_{x_1}(\bot), \top(\oplus_{x_2}(\bot), \bot_{\top}))).$$

Considérer le terme t, soit t' est le terme $a(b(a(\bot, \bot), \bot), c(d(\bot, \bot), a(\bot, \bot)))$. Clairement, t filtre t'.

Pour obtenir un EFTRS nous permettant de construire l'arbre de variables instancié de t, nous avons l'intention de définir un FTRS tel que, l'unique terme accessible de t' est $\top(a(\bot, \bot), \top(\bot, \top(a(\bot, \bot), \bot_{\top})))$.

La difficulté ici est que nous voulons seulement des EFTRSs. Le point clé est les symboles $\oplus_x$. Effectivement, t'' contient de tels symboles. Pour chaque sous-terme s de t'' tel que $s = \oplus_{x,p}(s')$, nous pouvons l'interpréter de la manière suivante : pour la substitution σ telle que $t = t'\sigma$, $\sigma(x) = s'$. Nous pouvons donc considérer les symboles $\oplus_x$ comme des marqueurs pour initialiser la copie de $\sigma(x)$.

Pour résumer la situation courante, pour deux termes $t \in \mathcal{T}(\mathcal{F}_{bin})$ et $t' \in \mathcal{T}(\mathcal{F}_{bin}, \mathcal{X})$, pour lesquels il existe une substitution $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F})$ telle que $t'\sigma = t$,

1. nous pouvons construire un terme t'' doté des marqueurs ayant le lien avec σ ;
2. Ces marqueurs sont également dans l'arbre des variables $\top_{\mathcal{X}}(t')$ et leur position dans $\top_{\mathcal{X}}(t')$ est connue puisque t' est donné.

Le but de l'EFTRS donné dans la définition 5 est

1. de vérifier qu'un terme t' est de la forme donnée (t)
2. de marquer dans t' la position de variables de t et
3. de copier symbole par symbole chaque valeur sur des marqueurs dans l'arbre des variables t.

Définition 5 ($\mathcal{R}_{\sigma}^{l \to r}$) *Soit $l \to r$ une règle de $\mathcal{T}(\mathcal{F}_{bin}, \mathcal{X}) \times \mathcal{T}(\mathcal{F}_{bin}, \mathcal{X})$. Soit $x_0, \ldots, x_n$ l'ensemble de variables apparaissant dans l et qui peut être lu de gauche à droite dans l. Ainsi, l'EFTRS $\mathcal{R}_{\sigma-compute}^{l \to r}$ est défini comme suit :*

1. $F_{copy}^{l \to r}(x) \to D_{copy}(RW_{x_0}(RW_{x_1}(\ldots RW_{x_n}(\top(F_t^{\varepsilon}(x), \top_{\mathcal{X}}(t))) \ldots)))$
2. $RW_{x_i}(x) \to RW_{x_i}(RW_{x_i}(x))$
3. $RW_{x_i}(x) \to NMA_{x_i}(x)$
4. $\{NMA_{x_i}(a(x, y)) \to a(NMA_{x_i}(x), NMA_{x_i}(y)) \mid a \in \mathcal{F}_{bin}\}$
5. $\{NMA_{x_i}(\oplus_{x_j}(x)) \to \oplus_{x_j,}(x) \mid x_i \neq x_j\}$

6. $NMA_{x_i}(\bot) \rightarrow \bot$
7. $NMA_{x_i}(\top(x,y)) \rightarrow \top(NMA_{x_i}(x), NMA_{x_i}(y))$
8. $NMA_{x_i}(\bot_\top) \rightarrow \bot_\top$
9. $RW_{x_i}(x) \rightarrow RW_{x_i,a}(x)$ *avec* $a \in \mathcal{F}_{bin}$
10. $RW_{x_i,a}(\top(x,y)) \rightarrow \top(R_{x_i,a}(x), SW^{x_i}_{x_i,a}(y))$ *avec* $a \in \mathcal{F}_{bin}$
11. $\{SW^x_{x_0,a}(\top(x,y)) \rightarrow \top(W_{x,a}(x), y)\} \cup \{SW^x_{x_i,a}(\top(x,y)) \rightarrow \top(x, SW^x_{x_{i-1},a}(y)) \mid i > 0 \wedge i \leq n \wedge x \in \mathcal{V}ar(t)\}$
12. $R_{x,a}(g(x,y)) \rightarrow g(R_{x,a}(x), y)$ *avec* $g \in \mathcal{F}_{bin}$
13. $R_{x,a}(g(x,y)) \rightarrow g(?_x(x), R_{x,a}(y))$ *avec* $g \in \mathcal{F}_{bin}$
14. $R_{x,a}(\oplus_x(x)) \rightarrow R_{\oplus_{x,a}}(x)$ *avec* $a \in \mathcal{F}_{bin}$
15. $R_{\oplus_{x,a}}(a(x,y)) \rightarrow a(\oplus_x(x), \oplus_x(y))$
16. $R_{\oplus_{x,\bot}}(\bot) \rightarrow \bot$
17. $W_{x,a}(g(x,y)) \rightarrow g(W_{x,a}(x), y)$ *avec* $g \in \mathcal{F}_{bin}$
18. $W_{x,a}(g(x,y)) \rightarrow g(?_{\oplus_x}(x), W_{x,a}(y))$ *avec* $g \in \mathcal{F}_{bin}$
19. $W_{x,a}(\oplus_x(\bot)) \rightarrow a(\oplus_x(\bot), \oplus_x(\bot))$
20. $W_{x,\bot}(\oplus_x(\bot)) \rightarrow \bot$
21. $D_{copy}(\top(x,y)) \rightarrow D'_{copy}(\top(y, D(x))$
22. $D'_{copy}(\top(x,\bot)) \rightarrow x$
23. $D(x) \rightarrow D(D(x))$
24. $D(\bot) \rightarrow \bot$
25. $D(x) \rightarrow D'(x)$
26. $D'(g(x,y)) \rightarrow g(D'(x), D'(y))$
27. $D'(\bot) \rightarrow \bot$
28. $D'(g(x,\bot)) \rightarrow x$
29. $?_{\oplus_x}(g(x,y)) \rightarrow g(?_{\oplus_x}(x), ?_{\oplus_x}(y))$ *avec* $g \in \mathcal{F}_{bin}$ *et* $x \in \mathcal{V}ar(t)$
30. $?_{\oplus_x}(\bot) \rightarrow \bot$

Nous décrivons le rôle des règles de ré-écriture présentées dans la définition 5.

La règle 1. est une règle principale ou une composition des opérations. Cette règle lance la copie la valeur obtenue par chaque variable après avoir marquée que le terme stocké dans la variable x. Une copie du terme étiqueté est établie à la racine de l'arbre de variables.

Exemple 16 *Soit t' le terme $a(b(a(\bot,\bot),\bot), c(d(\bot, b(\bot,\bot)), c(\bot,\bot)))$ et soit t le terme $a(b(x_0,\bot), c(d(\bot,x_1), x_2))$.*

La première règle est donc

$$F_{copy\_t}(x) \rightarrow$$

$$D_{copy}(RW_{x_0}(RW_{x_1}(RW_{x_2}(\top(F^{\varepsilon}_t(x), \top(\oplus_{x_0}(\bot), \top(\oplus_{x_1}(\bot), \top(\oplus_{x_2}(\bot)))))))))).$$

La ré-écriture peut être réalisée sur le terme $F_{copy}(t')$ et puis le terme $D_{copy}(RW_{x_0}(RW_{x_1}(\ RW_{x_2}(\top(F^{\varepsilon}_t(t'), \top(\oplus_{x_0}(\bot), \top(\oplus_{x_1}(\bot), \top(\oplus_{x_2}(\bot)))))))))$.

En utilisant les règles de $\mathcal{R}^t_{check}$ données dans l'exemple 14, nous obtenons le terme suivant :

$D_{copy}(RW_{x_0}(RW_{x_1}(RW_{x_2}(\ \top(t'', \top(\oplus_{x_0}(\bot), \top(\oplus_{x_1}(\bot), \top(\oplus_{x_2}(\bot), \bot_\top)))))))))$
où t'' est le terme $a(b(\oplus_{x_0}(a(\bot,\bot)),\bot), c(d(\bot, \oplus_{x_1}(b(\bot,\bot))), \oplus_{x_2}(c(\bot,\bot))))$.

La règle 2. spécifie qu'un nouveau pas de la copie de la valeur stockée dans la variable x_i fonctionne. Effectivement, dans le meilleur cas, une application de RW_{x_i} correspond à la copie d'un symbole, ainsi à copier entièrement la valeur stockée dans x_i, nous avons besoin d'autant moins du nombre d'applications que le nombre de positions fonctionnelles dans la valeur.

Exemple 17 *Étudions le terme t'' de l'exemple précédent. sur le symbole fonctionnel $\oplus_{x_2}$, il y a trois positions fonctionnelles correspondant aux symboles c, $\bot$ et $\bot$. Ainsi la règle $RW_{x_2}(x) \to RW_{x_2}(RW_{x_2}(x))$ devra être appliquée au moins deux fois pour une copie complète de la valeur stockée sur $\oplus_{x_2}$.*

Puis, nous pouvons obtenir le terme t''' = $D_{copy}(RW_{x_0}(RW_{x_1}(RW_{x_2}(RW_{x_2}(RW_{x_2}(\top(t'', \top(\oplus_{x_0}(\bot), \top(\oplus_{x_1}(\bot), \top(\oplus_{x_2}(\bot$ $), \bot_\top))))))))))$.

La règle 3. exprime que la copie de la variable x_i est terminé dès qu'il n'y a aucun marqueur correspondant à x_i dans l'arbre de variables. Les règles 3, 4, 5 et 6. lancent une recherche dans l'arbre de variables du marqueur correspondant à chaque variable x_i. En fait, s'il existe un tel marqueur, puisque ce cas est pas traité par la règle 4, alors le symbole fonctionnel NMA_{x_i} ne peut pas être effacé et puis aucun terme successeur en $\mathcal{T}(\mathcal{F}_{bin})$ de tel terme existe. Par conséquent, un nouveau pas de copie devrait être lancé.

Les règles 7 à 18 décrivent d'une part, comment un symbole est lu dans le premier élément de l'arbre de variables et d'autre part, comment ce symbole est simultanément écrit dans l'arbre de variables finale elle-même. Plus précisément, la règle 7. spécifie que la lecture et l'écriture simultanées sont ordonnées pour un symbole a de $\mathcal{F}_{bin}$. En fait, il y a une telle règle par symbole dans $\mathcal{F}_{bin}$ ($\bot$ compris).

Les règles 10 à 14 expliquent comment nous vérifions que le symbole suivant à lire dans le terme stocké dans la variable x est vraiment celle supposée intégrée dans le symbole fonctionnel $R_{x,a}$. Plus précisément, $R_{x,a}$ est récursivement appelé dans le top du terme de l'arbre de variable. Notez que la règle 11. spécifie que nous cherchons dans la partie droite d'un symbole seulement s'il n'y a aucun symbole $\oplus_x$ dans sa partie gauche. Encore une fois, supposons que cette règle est appliquée quand il y a encore le symbole $\oplus_x$ dans la partie gauche. ainsi, suivons les règles 29 et 30 nous voyons que le symbole $?_{\oplus_x}$ apparaîtra encore dans tous les termes successeurs et puis aucun terme de $\mathcal{T}(\mathcal{F}_{bin})$ sera possible de dériver. La lecture du symbole fonctionnel suivant est spécifiée par les règles 12 et 13. Effectivement, les sémantiques de $R_{x,a}$ sont que le symbole suivant à lire doit être a. Dès que le marqueur de variable x est mis dans la règle 12, le pas suivant est de vérifier que le symbole suivant est effectivement a , et si c'est ce cas, les marqueurs suivants sont établis directement sur ce symbole. Le processus d'écriture dans l'arbre de variables est proche de celui

décrit ci-dessus, sauf que : Dès que le marqueur est trouvé (spécifié par la règle 17), un nouveau terme est créé *c.à.d* $a(\bot, \bot)$.

La figure 4.2 illustre la copie de la valeur stockée dans la variable x_2. Le processus est le même pour les variables x_1 et x_0.

Et finalement, dès que la copie des valeurs stockées dans des variables est terminée, le terme au top de l'arbre de variables est obsolète. Alors, les règles 19 à 26 de la définition 5 décrivent comment détruire le terme obsolète et comment retourner à l'arbre de variables final instancié.

4.2.2.3 Règles d'EFTRS de test de substitution : $\mathcal{R}_{GS}^{l\to r}$

$\mathcal{R}_{GS}^{l\to r}$ est un EFTRS spécifiant la vérification d'une substitution bien-formée. Effectivement, si l n'est pas linéaire alors une variable x apparaît deux fois au moins. Si toutes les occurrences de x partagent le même terme comme valeur alors la substitution est bien-formée. $(\mathcal{R}_{check}^{l\to r})^*(F_{check}^{l\to r}(t'')) = \{t''\}$ est la substitution bien-formée, $\emptyset$ sinon. $\mathcal{R}_{GS}^{l\to r}$ traite chaque variable apparaissant plus d'une fois comme mentionné ci-dessus.

Définition 6 ($\mathcal{R}_{GS}^{l\to r}$) *Soit* $l \to r$ *une règle de* $\mathcal{T}(\mathcal{F}_{bin}, \mathcal{X}) \times \mathcal{T}(\mathcal{F}_{bin}, \mathcal{X})$. *Soit* $x_0, \ldots, x_n$ *les variables de* l *de gauche à droite. Soit Indexes l'ensemble de coupes d'indexes de variable correspondant à la même variable initiale. Soit* $\mathcal{R}_{GS}^{l\to r}$ *l' EFTRS réservé à vérifier que deux variables ont le même terme sur l'arbre de variables instantané. Ainsi,* $\mathcal{R}_{GS}^{l\to r}$ *est composé par des règles suivantes :*

1. $F_{check}^{l\to r}(x) \to CFor_{i_n,j_n}(CFor_{i_{n-1},j_{n-1}}(\ldots CFor_{i_1,j_1}(x)\ldots))$ *avec* $Indexes = \{(i_1,j_1),\ldots,(i_n,j_n)\}$
2. $PutMarks_i(\top(x,y)) \to \top(x, PutMarks_{i-1}(y))$ *pour* $i > 1$
3. $PutMarks_0(\top(x,y)) \to \top(\oplus_=(x), y)$
4. $CFor_{i,j}(x) \to Test_{=_{i,j}}(PutMarks_i(PutMarks_j(x)))$
5. $Test_{=_{i,j}}(x) \to Test_{=_{i,j}}(RR_{i,j}(x))$
6. $Test_{=_{i,j}}(x) \to ?_{x,p}(x)$
7. $\{RR_{i,j}(x) \to RR_{i,j,g}(x)\}$ *avec* $g \in \mathcal{F}_{bin}$
8. $\{RR_{i,j,g}(\top(x,y)) \to \top(x, RR_{i-1,j-1,g}(y))\}$ *avec* $g \in \mathcal{F}_{bin}$ *et* $i > 0$
9. $\{RR_{0,j,g}(\top(x,y)) \to \top(R_g(x), R_{j-1,g}(y))\}$ *avec* $g \in \mathcal{F}_{bin}$ *et* $i > 0$
10. $\{RR_{i,g}(\top(x,y)) \to \top(x, R_{i-1,g}(y))\}$ *avec* $g \in \mathcal{F}_{bin}$ *et* $i > 0$
11. $\{RR_{0,g}(\top(x,y)) \to \top(R_g(x), y)\}$ *avec* $g \in \mathcal{F}_{bin}$ *et* $i > 0$
12. $\{R_g(g(x,y)) \to g(R_g(x), y), R_g(g(x,y)) \to g(?_{\oplus_=}(x), R_g(y))\}$ *avec* $g \in \mathcal{F}_{bin}$
13. $R_g(\oplus_=(x)) \to R_{g,\oplus_=}(x)$
14. $R_{g,\oplus_=}(g(x,y)) \to g(\oplus_=(x), \oplus_=(y))$
15. $R_{\bot,\oplus_=}(\bot) \to \bot$

4.2.2.4 Règles d'EFTRS de substitution : $\mathcal{R}_{\sigma-apply}^{l\to r}$

$\mathcal{R}_{\sigma-apply}^{l\to r}$ est un EFTRS spécifiant l'application de la substitution bien-formée résultant de t et l sur le terme r. Ainsi, $(\mathcal{R}_{\sigma-apply}^{l\to r})^*(F_{rewrite}^{l\to r}(t'')) = \{r\sigma\}$.

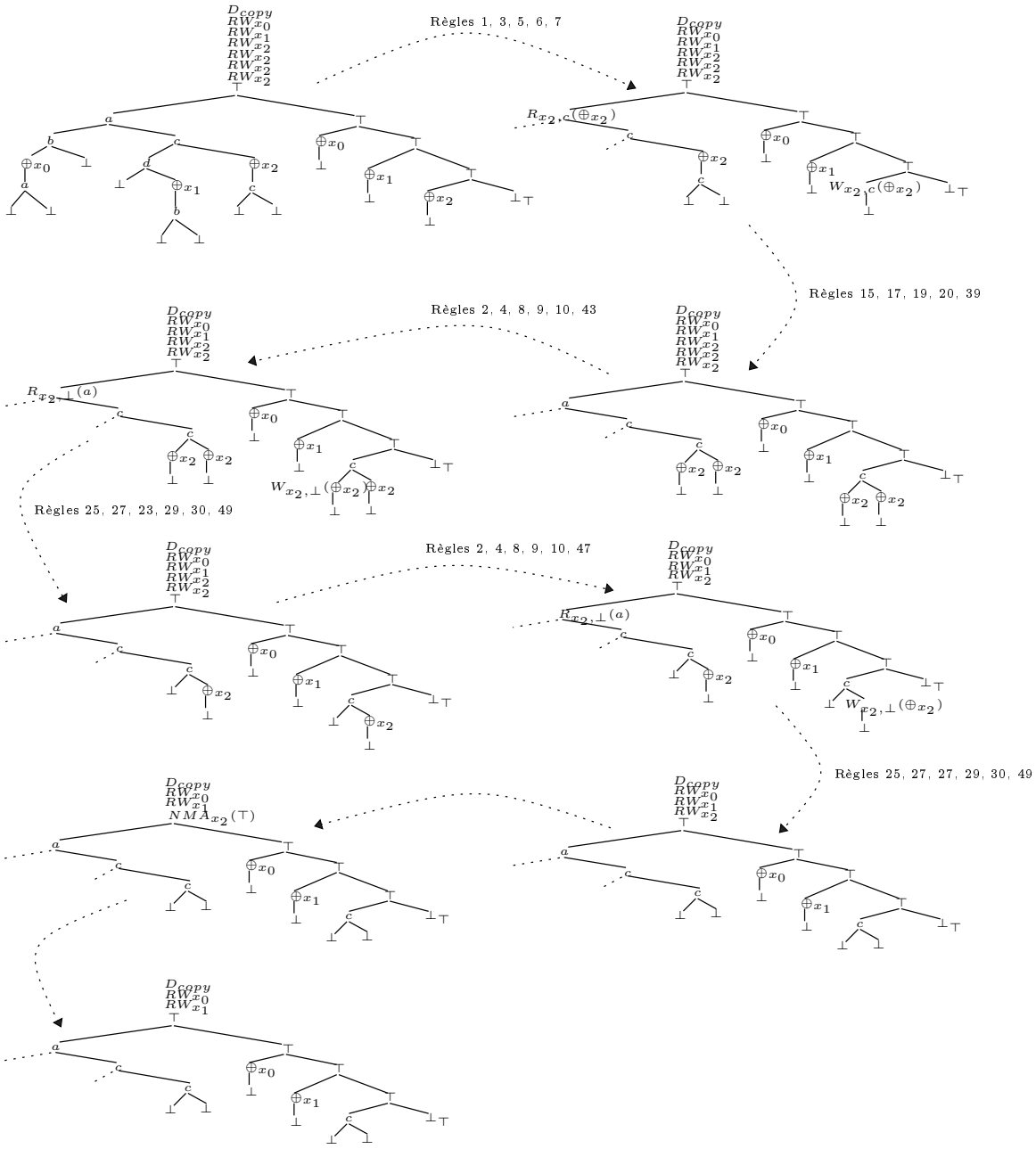


FIG. 4.2 – Copie de la valeur stockée dans x_2

Définition 7 ($\mathcal{R}^{l\to r}_{\sigma-apply}$) *Soit $l \to r$ une règle de $\mathcal{T}(\mathcal{F}_{bin}, \mathcal{X}) \times \mathcal{T}(\mathcal{F}_{bin}, \mathcal{X})$. Ainsi, $\mathcal{R}^{l\to r}_{\sigma-apply}$ contient les règles suivantes :*

1. $F^{l\to r}_{rewrite}(x) \to D_\top(RW^{rew}_{x_0}(RW^{rew}_{x_1}(\dots RW^{rew}_{x_n}(\top(r\sigma_{l\to r}, x))\dots)))$
2. $RW^{rew}_{x_i}(x) \to RW^{rew}_{x_i}(RW^{rew}_{x_i}(x))$
3. $RW^{rew}_{x_i}(x) \to NMA_{x_i}(x)$
4. $RW^{rew}_{x_i}(x) \to RW^{rew}_{x_i,a}(x)$ *avec* $a \in \mathcal{F}_{bin}$
5. $RW^{rew}_{x_i,a}(\top(x,y)) \to \top(W_{x_i,a}(x), SR_{x_i,a}(MS^{x_i}_{x_i}(?_{\oplus_{x_i}} y)))$ *avec* $a \in \mathcal{F}_{bin}$
6. $\{MS^z_{x_0}(\top(x,y)) \to \top(\oplus_z(x), y)\} \cup \{MS^z_{x_i}(\top(x,y)) \to \top(x, MS^z_{x_{i-1}}(y)) \mid i > 0 \wedge i \leq n \wedge z \in \mathcal{V}ar(r)\}$
7. $\{SR_{x_0,a}(\top(x,y)) \to \top(R_{x_0,a}(x), y)\} \cup \{SR_{x_i,a}(\top(x,y)) \to \top(x, SR_{x_{i-1},a}(y)) \mid i > 0 \wedge i \leq n\}$
8. $D_\top(\top(x,y)) \to D_\top(\top(x, D(y))$
9. $D_\top(\top(x,\bot)) \to x$
10. $D'(\top(x,y)) \to \top(D'(x), D'(y))$
11. $D'(\top(x,\bot_\top)) \to x$
12. $D'(\top(x,\bot)) \to x$

Nous allons expliquer l'ensemble de règles construites dans la définition 7. La règle 1. spécifie que toutes les variables apparaissant dans r doivent être remplacées par leurs valeurs stockées dans l'arbre de variable. Notez que les variables de r sont remplacées par des marqueurs et ce terme est établi au position topmost de l'arbre de variables. Les règles 2 et 3 décrivent que la copie d'une variable donnée doit être faite jusqu'à ce qu'aucun marqueur de cette variable apparaîte dans l'arbre total. Pour une variable donnée, la règle 4 lance la copie pour chaque symbole de $\mathcal{F}_{bin}$. Au contraire de la définition 5, r peut contenir quelques fois la même variable, et puis, la copie de chaque variable non linéaire doit être traitée prudemment. En particulier, si nous avons l'intention de copier variable après variable, nous devons alors nous assurer que une nouvelle copie ne peut pas commencer avant la fin de la précédente. En fait, c'est facile de le traiter pour une variable donnée x. Effectivement, en vérifiant la présence du symbole $\oplus_x$ dans l'arbre de variable réel (l'arbre de variable sans le premier élément), on peut dire si la lecture est en route. Dès qu'une copie d'une variable donnée est terminée, s'il y a une nouvelle copie à faire pour la même variable, alors un marqueur est établi dans l'arbre de variables en utilisant les règles 5 et 6. Le pas suivant est de recommencer la lecture pour la copie. Ce processus est implémenté par la règle 7. Dès que chaque variable de r est substituée par leurs valeurs correspondantes, l'arbre de variables est détruit, en utilisant des règles 8, 9, 10, 11, 12 comme des règles 23, 24, 25, 26, 27, 28 de la définition 5. Finalement, l'instance de r est obtenue.

4.2.2.5 Transformation de TRSs vers EFTRSs

En conséquence, en utilisant les 4 ensembles de règles définis ci-dessus avec un ensemble de règles d'EFTRS de circulation $\mathcal{R}^{l\to r}_{visit}$ tel que

$\mathcal{R}^{l\to r}_{visit} = \{F_{l\to r}(a(x,y)) \to a(F_{l\to r}(x), y),\ F_{l\to r}(a(x,y)) \to a(x, F_{l\to r}(y))\}$

et un ensemble de règles d'EFTRS de relance $\mathcal{R}^{l\to r}_{TV}$ tel que

$\mathcal{R}^{l\to r}_{TV} = \{F_{l\to r}(x) \to F^{l\to r}_{rewrite}(F^{l\to r}_{check}(F^{l\to r}_{copy}(\ F^{\varepsilon}_{l\to r}(x))))\}$,

nous sommes capables de simuler la ré-écriture d'un TRS.

Nous remarquons que les règles d'EFTRS de circulation $\mathcal{R}_{visit}^{l\to r}$ nous permettent de circuler le non-terminal $F_{l\to r}$ à tous les endroits du terme donné quand les règles d'EFTRS de relance $\mathcal{R}_{TV}^{l\to r}$ est une composition de 4 non-terminaux $F_{rewrite}^{l\to r}, F_{check}^{l\to r}, F_{copy}^{l\to r}$ et $F_{l\to r}^{\varepsilon}$ (coresspondant avec 4 ensembles de règles définis ci-dessus).

En conséquence, nous avons le résultat pour un pas de ré-écriture comme suit :

Proposition 2 *Soit $\mathcal{R}$ un TRS sur $\mathcal{T}(\mathcal{F}_{bin},\mathcal{X})\times\mathcal{T}(\mathcal{F}_{bin},\mathcal{X})$, $\alpha\in\mathcal{T}(\mathcal{F}_{bin})$, $\beta\in\mathcal{T}(\mathcal{F}_{bin})$ and $l\to r\in\mathcal{R}$. Il existe un EFTRS $\mathcal{R}_{\lambda^e}$ tel que*

$$\alpha\to_{\{l\to r\}}\beta\Leftrightarrow F_{l\to r}(\alpha)\to_{\mathcal{R}_{\lambda^e}}^{*}\beta.$$

Preuve 4 (Preuve succincte (Voir cf. dans l'annexe A)) *Le point clé est de construire un EFTRS $\mathcal{R}_{\lambda^e}$ qui implémente le processus de ré-écriture.*

Le EFTRS $\mathcal{R}_{\lambda^e}$ est construit comme le suivant :

$\mathcal{R}_{\lambda^e} = \mathcal{R}_{visit}^{l\to r}\cup\mathcal{R}_{TV}^{l\to r}\cup\mathcal{R}_{check}^{l\to r}\cup\mathcal{R}_{\sigma}^{l\to r}\cup\mathcal{R}_{GS}^{l\to r}\cup\mathcal{R}_{\sigma-apply}^{l\to r}$

- *$\alpha\to_{\{l\to r\}}\beta\Rightarrow F_{l\to r}(\alpha)\to_{\mathcal{R}_{\lambda^e}}^{*}\beta$: Effectivement, $F_{rewrite}^{l\to r}(t)\to_{\mathcal{R}_{\sigma-apply}^{l\to r}}^{*} r\sigma\in\mathcal{T}(\mathcal{F}_{bin})$. En conséquence, $\alpha[F_{rewrite}^{l\to r}(t)]_p\to_{\mathcal{R}_{\sigma-apply}^{l\to r}}^{*}\alpha[r\sigma]_p\in\mathcal{T}(\mathcal{F}_{bin})$ et puisque $\alpha[r\sigma]_p=\beta$, $\beta\in\mathcal{R}_{\lambda^e}^{*}(\{F_{l\to r}(\alpha)\})$.*

- *$\alpha\to_{l\to r}\beta\Leftarrow F_{l\to r}(\alpha)\to_{\mathcal{R}_{\lambda^e}}^{*}\beta$: La preuve pour ce cas est proche au cas similaire traité dans la preuve de la proposition 1 dans le sens que les pas de ré-écriture sont ordonnés et dépendants du terme initial c.à.d $F_{l\to r}(\alpha)$.*
  Ainsi, pour résumé, il existe une position de α et une substitution $\sigma:\mathcal{X}\to\mathcal{T}(\mathcal{F}_{bin})$ telles que $\alpha|_p=l\sigma$, $\beta=\alpha[r\sigma]_p$ et $F_{l\to r}(\alpha)\to_{\mathcal{R}_{\lambda^e}}^{*}\beta$.
  Nous pouvons déduire que $\alpha\to_{l\to r}\beta\Rightarrow F_{l\to r}(\alpha)\to_{\mathcal{R}_{\lambda^e}}^{*}\beta$ qui conclut la preuve.

En généralisant le résultat ci-dessus, nous obtenons le calcul de l'ensemble des états accessibles, étant également le résultat de la transformation (c) de la figure 4.1.

Théorème 2 *Soit $\mathcal{R}$ un TRS sur $\mathcal{T}(\mathcal{F}_{bin},\mathcal{X})\times\mathcal{T}(\mathcal{F}_{bin},\mathcal{X})$ et $E\subseteq\mathcal{T}(\mathcal{F}_{bin})$. Ainsi, il existe un EFTRS $\mathcal{R}_{\lambda^e}$ et un symbole $G\in\mathcal{F}_{NT}$ tels que :*

$$\mathcal{R}^{*}(E)=\mathcal{R}_{\lambda^e}^{\star}(E'),$$

où $E'=\{G(\alpha)|\alpha\in E\}$.

Preuve 5 *Étant donné l'EFTRS $\mathcal{R}_{\lambda^e}=\bigcup_{l\to r\in\mathcal{R}}(\mathcal{R}_{visit}^{l\to r}\cup\mathcal{R}_{TV}^{l\to r}\cup\mathcal{R}_{check}^{l\to r}\cup\mathcal{R}_{\sigma}^{l\to r}\cup\mathcal{R}_{GS}^{l\to r}\cup\mathcal{R}_{\sigma-apply}^{l\to r})\cup\mathcal{R}_{FP}$ avec $\mathcal{R}_{FP}=\{G(a(x,y))\to G(F_{l\to r}(a(x,y)))\mid l\to r\in\mathcal{R}\wedge a\in\mathcal{F}_{bin}\}\cup\{G(a(x,y))\to a(x,y)\mid a\in\mathcal{F}_{bin}\}$, la preuve est très proche à celle du théorème 1.*

Dans cette section, nous avons prouvé que soit un TRS, il existe un EFTRS correspondant. Ensuite, à partir des EFTRSs, nous pouvons les convertir vers des TRSs comme dans la section 4.1.

4.2.3 Transformation de FTRSs vers EFTRSs

Enfait, il existe un chemin de FTRSs vers EFTRSs en passant par les flèches (b) et (d) de la figure 4.1. Autrement dit, Cependant, la transformation directe de la figure 4.1 (b) de FTRS vers TRS n'est pas toujours faisable, nous devons alors procéder de manière indirecte.

Considérons un exemple :

Exemple 18 *Revenons à l'exemple 13 avec le FTRS OnlyOneRequest L'EFTRS correspondant est le suivant :*

$$\mathcal{R}_{\lambda^e} = \left\{ \begin{array}{l} OnlyOne(t(x,y)) \rightarrow t(OnlyOne(R(x)), I(y)) \\ OnlyOne(t(x,y)) \rightarrow t(I(x), OnlyOne(R(y))) \\ OnlyOne(r(x,y)) \rightarrow r(OnlyOne(R(x)), I(y)) \\ OnlyOne(r(x,y)) \rightarrow r(I(x), OnlyOne(R(y))) \\ OnlyOne(r(\bot,\bot)) \rightarrow r(\bot,\bot) \\ R(r(\bot,\bot)) \rightarrow r(\bot,\bot) \\ I(i(\bot,\bot)) \rightarrow i(\bot,\bot) \end{array} \right\}.$$

Nous proposons dans cette section une transformation directe du FTRS vers un ensemble de règles de EFTRSs qui sont suffisamment expressives pour la spécification du mécanisme de ré-écriture. Tout d'abord, nous montrons qu'un pas de ré-écriture utilisant $\mathcal{R}_\lambda$ (*c.à.d* une règle FTRS de la forme $H\ (t) \rightarrow \alpha$) peut être simulé par un EFTRS $\mathcal{R}_{\lambda^e}$.

1. de trouver une position pour la ré-écriture,
2. de vérifier si $H\ (t)$ concorde avec le sous-terme courant en considérant H comme un symbole spécial en $\mathcal{F}_{bin}$, puis
3. de calculer la substitution résultant σ, et
4. de remplacer le sous-terme par $\alpha\sigma$.

La transformation (d) de la figure 4.1 est donnée par la proposition suivante.

Proposition 3 *Soit $\mathcal{R}_\lambda$ un FTRS $\alpha \in \mathcal{T}(\mathcal{F}_{bin})$. Il existe un EFTRS $\mathcal{R}_{\lambda^e}$ tel que*

$$\{\alpha\} \cup \bigcup_{H \in \mathcal{F}_{NT}, \beta \in Reachables} (\mathcal{R}_\lambda^\star(\{H(\beta)\})) = \{\alpha\} \cup \bigcup_{H \in \mathcal{F}_{NT}, \beta \in Reachables} (\mathcal{R}_{\lambda^e}{}^\star(\{H(\beta)\})).$$

Preuve 6 *Étant donné l'EFTRS $\mathcal{R}_{\lambda^e}$, la preuve est très proche à celle de la proposition 2.*

Le point clé est de construire un EFTRS $\mathcal{R}_{\lambda^e}$ qui implémente le processus de ré-écriture en considérant H comme un symbole spécial en $\mathcal{F}_{bin}$ et en ignorant les règles de circulation.

Le EFTRS $\mathcal{R}_{\lambda^e}$ est construit comme le suivant :

$$\mathcal{R}_{\lambda^e} = \mathcal{R}_{TV}^{H(t)\rightarrow\alpha} \cup \mathcal{R}_{check}^{H(t)\rightarrow\alpha} \cup \mathcal{R}_{\sigma}^{H(t)\rightarrow\alpha} \cup \mathcal{R}_{GS}^{H(t)\rightarrow\alpha} \cup \mathcal{R}_{\sigma-apply}^{H(t)\rightarrow\alpha}$$

4.3 EFTRSs à Droite (REFTRSs)

Nous allons présenter dans cette section une sous classe de systèmes fonctionnels élémentaires, les élémentaires à droite (REFTRSs). D'une part, ils préservent également la puissance d'expression des systèmes fonctionnels et d'autre part ils sont capables de mieux appliquer des techniques d'accélération des calculs. Il est possible de transformer d'un EFTRS vers des REFTRSs.

4.3.1 Principes

Cette section présente une sous classe de systèmes fonctionnels élémentaires, les élémentaires à droite (REFTRSs), préservant la puissance d'expression des systèmes fonctionnels. D'une part, l'implémentation d'un moteur de ré-écriture pour les REFTRSs est plus simple que les autres FTRSs, notamment la procédure de concordance des termes. D'autre part, l'utilisation flexible des non-terminaux en REFTRSs peut augmenter l'efficacité de la technique de cache *à la BDD* et l'algorithme de saturation.

Etant donné un EFTRS $\mathcal{R}_{\lambda^e}$, nous allons le traiter par l'analyse de cas.

1. $H(\perp) \rightarrow \beta$ avec $\beta \in \mathcal{T}(\mathcal{F}_{bin} \cup \mathcal{F}_{NT})$: Ce type est considérée comme un *REFTRS type A* : $\mathcal{R}_{\lambda^e \rightarrow A}$.
2. $H(a(x, \perp)) \rightarrow x$ avec $x \in \mathcal{X}$, $H \in \mathcal{F}_{NT}$ and $a \in \mathcal{F}_{bin}$: Ce type est considérée comme un *REFTRS type B* : $\mathcal{R}_{\lambda^e \rightarrow B}$.
3. $H(a(x, y)) \rightarrow \alpha$ avec $a \in \mathcal{F}_{bin}$, $H \in \mathcal{F}_{NT}$, $x, y \in \mathcal{X}$, $\alpha \in \mathcal{T}(\mathcal{F}_{bin} \cup \mathcal{F}_{NT}, \mathcal{X})$ et $\mathcal{V}ar(\alpha) = \{x, y\}$: Ce type n'est pas encore *élémentaire* au côté droit comme les autres. Par conséquent, nous traitons des types de $H(a(x, y)) \rightarrow \alpha$ pour simplifier le côté droit de ce type de règle.

Nous allons distinguer trois types du côté droit de la règle $H(a(x, y)) \rightarrow \alpha$ avec $a \in \mathcal{F}_{bin}$, $H \in \mathcal{F}_{NT}$, $x, y \in \mathcal{X}$, $\alpha \in \mathcal{T}(\mathcal{F}_{bin} \cup \mathcal{F}_{NT}, \mathcal{X})$ et $\mathcal{V}ar(\alpha) = \{x, y\}$. Les deux premiers REFTRS type I et type II qui conservent le symbole a après avoir appliqué le non-terminal H sont appelés *Conservé la forme*. Malgré le type très simple qui remplit bien les conditions pour les techniques d'accélération basées sur l'algorithme de saturation, le REFTRS type I apparaît très souvent, notamment dans la circulation. Le REFTRS type II est également importante car il nous permet de composer une suite d'autant d'opérations que nous voulons. Le troisième REFTRS type III qui nous permettent de changer le lettre courante a, de permuter deux sous-termes (concordant avec x et y), et d'ajouter une lettre à l'extérieur du terme courant est appelé *non conservé la forme* .

Tout d'abord, nous commençons par le REFTRS type I (appelé *Conservé la forme 1*) le type le plus simple des règles de type $H(a(x, y)) \rightarrow \alpha$. Malgré le type très simple qui remplit bien les conditions pour les techniques d'accélération basées sur l'algorithme de saturation, elles apparaissent très souvent, notamment dans la circulation.

Définition 8 *REFTRS type I .*
$\mathcal{R}_{\lambda^e}$ est dit REFTRS type I si toutes les règles ayant côté gauche $H\ (a(x,y))$ sont sous forme
- $H\ (a(x,y)) \rightarrow a\ (x,y)$
- $H\ (a(x,y)) \rightarrow a(G\ (x),y)$
- $H\ (a(x,y)) \rightarrow a(x,D\ (y))$

avec $H,G,D \in \mathcal{F}_{NT}, a \in \mathcal{F}_{bin}$.
Nous dénotons par $\mathcal{R}_{\lambda^e \rightarrow I}$.

Deuxièmement, nous définissons le REFTRS type II, qui est encore appelé *Conservé la forme*, et qui nous permet d'exécuter une composition de deux autres opérations. C'est une règle très importante car elle nous permet de composer une suite d'autant d'opérations que nous voulons.

Définition 9 *REFTRS type II.*
$\mathcal{R}_{\lambda^e}$ est dit REFTRS type II si toutes les règles ayant côté gauche $H\ (a(x,y))$ sont sous forme
- $H\ (a(x,y)) \rightarrow G\ (D\ (a(x,y)))$

avec $H,G,D \in \mathcal{F}_{NT}, a \in \mathcal{F}_{bin}$.
Nous dénotons par $\mathcal{R}_{\lambda^e \rightarrow II}$.

En utilisant seulement des règles de type I et II, nous pouvons simuler la règle $H\ (a(x,y)) \rightarrow a(G\ (x), D\ (y))$ comme les suivantes :
- $H\ (a(x,y)) \rightarrow G'\ (D'\ (a(x,y)))$
- $G'\ (a(x,y)) \rightarrow a(G\ (x),y)$
- $D'\ (a(x,y)) \rightarrow a(x,D\ (y))$.

Nous allons définir ci-dessous les règles primitives REFTRS type III qui nous permettent de changer la lettre courante, de permuter deux sous-termes, et d'ajouter une lettre à l'extérieur du terme courant.

Définition 10 *REFTRS type III.*
$\mathcal{R}_{\lambda^e}$ est dit REFTRS type III s'il existe une règle telle que
- $H\ (a(x,y)) \rightarrow b(x,y)$
- $H\ (a(x,y)) \rightarrow a(y,x)$
- $H\ (a(x,y)) \rightarrow a(\perp, a(x,y))$

avec $H \in \mathcal{F}_{NT}$ et $a,b \in \mathcal{F}_{bin}$.
Nous dénotons par $\mathcal{R}_{\lambda^e \rightarrow III}$.

Spécification du TAP en REFTRS

Cette section montre comment le TAP peut être formalisé en REFTRSs sous forme d'un problème de calcul d'accessibilité.

Les règles de ré-écriture(4.10) et (4.11) données en section 4.2 TAP en EFTRS sont déjà élémentaires à droite.

Dans tous les autres cas non élémentaires, nous créons d'abord une liste des non-terminaux $\mathcal{F}_{NT}$ nécessaires. Puis chaque $H \in \mathcal{F}_{NT}$ va effectuer une opération élémentaire. Par exemple, les règles de ré-écriture (4.8) et (4.9) données dans la section 4.2 sont transformées comme les règles de REFTRS $\mathcal{R}^{req}_{\lambda^e}{}_{\to\cup}$ indiquées ci-dessous.

$$H(i(x,y)) \to H_{4.8}(H'_{4.8}(i(x,y))) \tag{4.34}$$
$$H_{4.8}(i(x,y)) \to r(x,y) \tag{4.35}$$
$$H'_{4.8}(i(x,y)) \to i(x,R(y)) \tag{4.36}$$
$$H(i(x,y)) \to H_{4.9}(H'_{4.9}(i(x,y))) \tag{4.37}$$
$$H_{4.9}(i(x,y)) \to r(x,y) \tag{4.38}$$
$$H'_{4.9}(i(x,y)) \to i(R(x),y) \tag{4.39}$$

Similairement, nous avons les règles de circulation $\mathcal{R}^{cir}_{\lambda^e}{}_{\to\cup} = \mathcal{R}^{cir}_{\lambda^e}$. Les règles indiquées ci-dessous remplissent la spécification du TAP par un REFTRS.

$$H(t(x,y)) \to H_{4.20}(H'_{4.20}(t(x,y))) \tag{4.40}$$
$$H_{4.20}(t(x,y)) \to b(x,y) \tag{4.41}$$
$$H'_{4.20}(t(x,y)) \to t(x,RT(y)) \tag{4.42}$$
$$H(t(x,y)) \to H_{4.21}(H'_{4.21}(t(x,y))) \tag{4.43}$$
$$H_{4.21}(t(x,y)) \to b(x,y) \tag{4.44}$$
$$H'_{4.21}(t(x,y)) \to t(RT(x),y) \tag{4.45}$$
$$RT(r(x,y)) \to t(x,y) \tag{4.46}$$
$$H(b(x,y)) \to H_{4.23}(H'_{4.23}(b(x,y))) \tag{4.47}$$
$$H_{4.23}(b(x,y)) \to t(x,y) \tag{4.48}$$
$$H'_{4.23}(b(x,y)) \to b(x,TI(y)) \tag{4.49}$$
$$\tag{4.50}$$

$$H(b(x,y)) \to H_{4.24}(H'_{4.24}(b(x,y))) \tag{4.51}$$
$$H_{4.24}(b(x,y)) \to t(x,y) \tag{4.52}$$
$$H'_{4.24}(b(x,y)) \to b(TI(x),y) \tag{4.53}$$
$$H(t(x,y)) \to H_{4.25}(H'_{4.25}(t(x,y))) \tag{4.54}$$
$$H(t(x,y)) \to H'''_{4.25}(H''_{4.25}(t(x,y))) \tag{4.55}$$
$$H'''_{4.25}(t(x,y)) \to i(x,y) \tag{4.56}$$
$$H'_{4.25}(t(x,y)) \to t(I(x),y) \tag{4.57}$$
$$H''_{4.25}(t(x,y)) \to t(x,I(y)) \tag{4.58}$$
$$H(t(x,y)) \to H_{4.26}(H'_{4.26}(t(x,y))) \tag{4.59}$$
$$H(t(x,y)) \to H'''_{4.26}(H''_{4.26}(t(x,y))) \tag{4.60}$$
$$H'''_{4.26}(t(x,y)) \to r(x,y) \tag{4.61}$$
$$H'_{4.26}(t(x,y)) \to t(R(x),y) \tag{4.62}$$
$$H''_{4.26}(t(x,y)) \to t(x,I(y)) \tag{4.63}$$
$$\tag{4.64}$$

$$H(t(x,y)) \to H_{4.27}(H'_{4.27}(t(x,y))) \tag{4.65}$$
$$H(t(x,y)) \to H'''_{4.27}(H''_{4.27}(t(x,y))) \tag{4.66}$$
$$H'''_{4.27}(t(x,y)) \to r(x,y) \tag{4.67}$$
$$H'_{4.27}(t(x,y)) \to t(I(x),y) \tag{4.68}$$
$$H''_{4.27}(t(x,y)) \to t(x,R(y)) \tag{4.69}$$
$$I(i(x,y)) \to i(x,y) \tag{4.70}$$
$$I(\bot) \to \bot \tag{4.71}$$
$$Arbiter(t(x,y)) \to t(x,y) \tag{4.72}$$
$$Arbiter(t(x,y)) \to Arbiter(H(t(x,y))) \tag{4.73}$$
$$Arbiter(b(x,y)) \to b(x,y) \tag{4.74}$$
$$Arbiter(b(x,y)) \to Arbiter(H(b(x,y))) \tag{4.75}$$

Nous notons par ${\mathcal{R}_{\lambda^e}^{TAP}}_{\to\cup}$ le REFTRS complet défini en cette section. Nous pouvons obtenir l'ensemble de termes accessibles à partir du terme *total idle* en calculant $({\mathcal{R}_{\lambda^e}^{req}}_{\to\cup})^\star(\{Arbiter(u)\})$. De plus, nous pouvons également vérifier que le terme *total requesting* v est atteignable à partir du terme *total idle* u puis que $u \to^\star_{{\mathcal{R}_{\lambda^e}^{req}}_{\to\cup}} v$.

4.3.2 Transformation de EFTRSs vers REFTRSs

Actuellement, des REFTRSs, présentés par les définitions 8, 9 et 10 sont suffisamment expressives pour la spécification du mécanisme de ré-écriture pour une règle EFTRS ayant côté gauche $H\ (a(x,y)) \to \alpha\ \in \mathcal{R}_{\lambda^e}$:

1. de trouver une position pour la ré-écriture,
2. de vérifier si $H\ (a(x,y))$ concorde avec le sous-terme courant, puis
3. de calculer la substitution résultant σ, et
4. de remplacer le sous-terme par $\alpha\sigma$ avec α (Des substitutions des variables peuvent être multipliées).

En conséquence, nous avons le résultat listé ci-dessous. Nous remarquons que les systèmes de ré-écriture primitives listés ci-dessus sont linéaires droites et sont capables de simuler n'importe quel système EFTRS linéaire droite ou même non linéaire droite.

La transformation (e) de la figure 4.1 est donnée par la proposition suivante.

Proposition 4 *Soit $\mathcal{R}_{\lambda^e}$ un EFTRS et $\alpha \in \mathcal{T}(\mathcal{F}_{bin})$. Il existe des REFTRSs $\mathcal{R}_{\lambda^e \to I}$, $\mathcal{R}_{\lambda^e \to II}$, $\mathcal{R}_{\lambda^e \to III}$, $\mathcal{R}_{\lambda^e \to A}$ et $\mathcal{R}_{\lambda^e \to B}$ (en bref $\mathcal{R}_{\lambda^e \to \cup}$) tels que*

$$\{\alpha\} \cup \bigcup_{H \in \mathcal{F}_{NT}, \beta \in Reachables} (\mathcal{R}^\star_{\lambda^e}(\{H(\beta)\})) = \{\alpha\} \cup \bigcup_{H \in \mathcal{F}_{NT}, \beta \in Reachables} (\mathcal{R}_{\lambda^e}{}^\star_{\to\cup}(\{H(\beta)\})).$$

Preuve 7 (Preuve succincte (Voir cf. dans l'annexe A)) *Étant donné les REFTRSs $\mathcal{R}_{\lambda^e\to I}$, $\mathcal{R}_{\lambda^e\to II}$, $\mathcal{R}_{\lambda^e\to III}$, $\mathcal{R}_{\lambda^e\to A}$ et $\mathcal{R}_{\lambda^e\to B}$, la preuve est très proche à celle de la proposition 2.*

Le point clé est de construire un REFTRS qui implémente le processus de réécriture.

Le REFTRS est construit comme le suivant :

$$\mathcal{R}_{\lambda^e\to\cup} = \mathcal{R}_{TV}^{H\ (a(x,y))\to\alpha} \cup \mathcal{R}_{check}^{H\ (a(x,y))\to\alpha} \cup \mathcal{R}_{\sigma}^{H\ (a(x,y))\to\alpha} \cup \mathcal{R}_{\sigma-apply}^{H\ (a(x,y))\to\alpha}$$

Nous traitons un exemple pour éclairer la proposition précédente.

Exemple 19 *Etant donnée une règle EFTRS de la forme $H(a(x,y)) \longrightarrow b(a(x,\bot),a(\bot,y))$. Elle est simulée par les règles REFTRSs comme suit :*

- $H\ (a(x,y)) \to H"(H'(a(x,y)))$
- $H'\ (a(x,y)) \to b(x,y)$
- $H"\ (b(x,y)) \to b(G\ (x), D\ (y))$

Pour $D(x) \to a(\bot,x)$, nous avons un ensemble de règles suivantes :

- $D\ (\bot) \to a(\bot,\bot)$
- $D(c(x,y)) \to A(D'(c(x,y)))\ \forall c \in \mathcal{F}_{bin}$
- $A\ (c(x,y)) \to a(x,y)$
- $D'\ (c(x,y)) \to c(\bot,c(x,y))$
- $D'\ (\bot) \to a(\bot,\bot)$

Pour $G(x) \to a(x,\bot)$, nous faisons un appel de D :

- $G\ (\bot) \to a(\bot,\bot)$
- $G(c(x,y)) \to M(D(c(x,y)))\ \forall c \in \mathcal{F}_{bin}$
- $M\ (a(x,y)) \to a(y,x)$

Intuitivement, nous obtenons le résultat plus général, étant la combinaison des transformation (présentées par les flèches (c) et (e) de la figure 4.1), présenté cidessous : Pour tout EFTRS, nous pouvons construire des REFTRSs équivalents pour l'accessibilité sur $\mathcal{T}(\mathcal{F}_{bin})$. En résumé, tous TRS peut être simulé par les REFTRSs.

Théorème 3 *Soit $\mathcal{R}$ un TRS sur $\mathcal{T}(\mathcal{F}_{bin},\mathcal{X}) \times \mathcal{T}(\mathcal{F}_{bin},\mathcal{X})$ et $\alpha \in \mathcal{T}(\mathcal{F}_{bin})$. Ainsi, il existe un REFTRS $\mathcal{R}_{\lambda^e\to\cup}$ tel que, considérons $Reachables = \{\alpha\} \cup \bigcup_{H\in\mathcal{F}_{NT},\beta\in Reachables}(\mathcal{R}_{\lambda^e\to\cup}^{\star}(\{H(\beta)\}))$, $\mathcal{R}^{\star}(\alpha) = Reachables$.*

Preuve 8 *Selon les propositions 4 et la théorème 2.*

4.4 Conversion des systèmes fonctionnels vers TRSs

Dans cette section, notre ambition est de chercher une conversion indirecte en considérant les systèmes fonctionnels comme un type spécial des systèmes classiques. Elle est représentée par les flèches pointillées de la figure 4.1.

Soit $\mathcal{R}_\lambda$ un FTRS sur $\mathcal{T}(\mathcal{F}_{bin} \cup \mathcal{F}_{NT})$. Nous construisons un nouveau TRS $\mathcal{R}_{conv}$. Tout d'abord, l'ensemble de symboles $\mathcal{T}(\mathcal{F}_{bin})$ à partir de $\mathcal{F}_{NT}$ et $\mathcal{F}_{bin}$ est dénoté par $\mathcal{F}^{conv}$ est défini ci-dessous :

- Pour chaque symbole binaire $a \in \mathcal{F}_{bin}$ (y compris $\perp$), ajouter $a \in \mathcal{F}^{conv}$
- Pour chaque symbole unaire $H \in \mathcal{F}_{bin}$, ajouter $h \in \mathcal{F}^{conv}$ tel que h est binaire et $h \notin \mathcal{F}_{bin}$

Ensuite, $\mathcal{R}_{conv}$ est construit par des règles de $\mathcal{R}_\lambda$ en remplaçant tous les symboles unaires $H(...)$ par des symboles binaires $h(..., \perp)$ où $\perp$ va remplir le sous-terme droit de h non utilisé.

Exemple 20 *Considérons le FTRS de l'exemple 13. Puisque OnlyOne* $\in \mathcal{F}_{NT}$ *est traduit à onlyone* $\in \mathcal{F}^{conv}$*, ainsi le TRS* $\mathcal{R}_{conv}$ *obtenu est comme le suivant :*

$$\mathcal{R}_{conv} = \left\{ \begin{array}{l} onlyone(t(r(x,y),i(z,t)),\perp) \to t(onlyone(r(x,y),\perp),i(z,t)) \\ onlyone(t(i(x,y),r(z,t)),\perp) \to t(i(x,y),onlyone(r(z,t),\perp)) \\ onlyone(r(r(x,y),i(z,t)),\perp) \to r(onlyone(r(x,y),\perp),i(z,t)) \\ onlyone(r(i(x,y),r(z,t)),\perp) \to r(i(x,y),onlyone(r(z,t),\perp)) \\ onlyone(r(\perp,\perp),\perp) \to r(\perp,\perp) \end{array} \right\}.$$

Malheureusement, le problème désormais vient du mode de fonctionnement. Dans le cas de l'exemple 20, le mode de fonctionnement de ce TRS est similaire au mode du FTRS. Cependant, dans le cas de l'exemple 12, nous avons obtenu les résultats différents.

En résumé, nous avons prouvé que pour tout TRS, il existe un FTRS correspondant. Pour la réciproque, si un FTRS satisfait la condition de la section , il existe un TRS correspondant. D'autre part, si ce FTRS contient que des règles respectent le mode de fonctionnement de TRS, nous pouvons transfomer vers un TRS par une manière indirecte.

BILAN DU CHAPITRE 4

Dans ce chapitre, nous avons proposé les systèmes de ré-écriture fonctionnels. Nous montrons que notre modèle a la puissance d'expression des systèmes de ré-écriture et qu'il est bien adapté à l'étude de propriétés de sûreté de et de propriétés de logique temporelle de systèmes présentée en détail à la partie III.

Nous avons mis en évidence une sous classe de systèmes fonctionnels, les élémentaires, préservant la puissance d'expression des systèmes fonctionnels et des techniques d'accélération des calculs aboutissant à un outil de vérification symbolique efficace. Nous expliquons comment ils sont évalués au chapitre 5.

De plus, nous avons mis en évidence une sous classe de systèmes fonctionnels élémentaires, les élémentaires à droite, préservant la puissance d'expression des systèmes fonctionnels et des techniques d'accélération des calculs.

Dans la partie expérimentale (en chapitre 8), nous allons comparer notre outil basé sur notre formalisme soit avec des outils de ré-écriture tels que Timbuk et Maude, soit avec des outils de vérification tels que SPIN, NuSMV, SMART, HSDD, etc afin de montrer nos performances compétitives.

Chapitre 5

Évaluation des systèmes élémentaires

RÉSUMÉ DU CHAPITRE 5

Dans le chapitre précédent, nous avons proposé les systèmes de ré-écriture fonctionnels (FTRSs), les élémentaires (EFTRSs) et les élémentaires à droite (REFTRSs) préservant la puissance d'expression des systèmes fonctionnels.

Dans ce chapitre, nous présentons les algorithmes d'évaluation classiques pour les systèmes de ré-écriture élémentaires (EFTRSs ou REFTRSs). La relation entre les systèmes de ré-écriture listés ci-dessus et les algorithmes d'évaluation est présentée par deux flèches de *EFTRS* à *Évaluation* et de *REFTRS* à *Évaluation* dans la figure 5.1. Les autres (les classiques et les FTRSs) doivent passer par une transformation simple (proposée au chapitre précédent) avant l'évaluation par nos algorithmes.

Ce chapitre va étudier la capacité d'optimisation de quelques types de règles élémentaires à droite. La relation entre les REFTRSs et les algorithmes d'optimisation est présentée par la flèche de *REFTRS* à *Saturation* dans la figure 5.1. Évidemment, l'efficacité de cette optimisation dépend du nombre de règles de ré-écritures applicables, de l'importance de ces règles ainsi que du niveau de symétrie du modèle donné.

Le mécanisme d'évaluation de nos algorithmes (classiques et optimisés), qui respecte bien le mode fonctionnel, est illustré par l'étude de cas (TAP) sous forme d'un problème de calcul d'accessibilité.

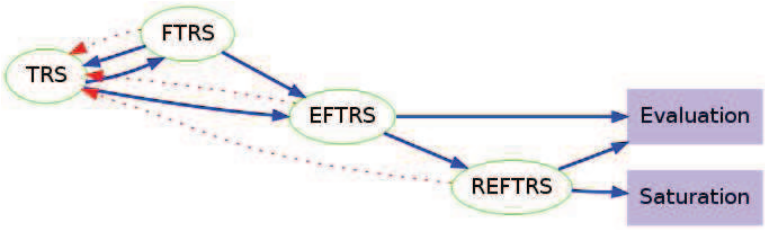


FIG. 5.1 – Relation entre des systèmes de ré-écriture et les algorithmes d'évaluation

Sommaire

5.1 Algorithme d'évaluation classique

Dans cette section, nous présentons les algorithmes d'évaluation classiques pour les systèmes de ré-écriture élémentaires (EFTRSs ou REFTRSs). Les autres (les classiques et les FTRSs) doivent passer par une transformation simple (proposée au chapitre précédent) avant l'évaluation par nos algorithmes.

Soit un EFTRS $\mathcal{R}_{\lambda^e}$ (y compris un REFTRS) qui induit une relation de ré-écriture $\rightarrow_{\mathcal{R}_{\lambda^e}}$ sur des termes $E \subseteq \mathcal{T}(\mathcal{F}_{bin})$, l'objectif est de calculer la clôture transitive écrite $\mathcal{R}^{\star}_{\lambda^e}(E)$. Donc cet EFTRS $\mathcal{R}_{\lambda^e}$ ne contient que des règles de la forme

1. $H(a(x,y)) \rightarrow \alpha$ avec $a \in \mathcal{F}_{bin}$, $H \in \mathcal{F}_{NT}$, $x,y \in \mathcal{X}$, $x \neq y$, $\alpha \in \mathcal{T}(\mathcal{F}_{bin} \cup \mathcal{F}_{NT}, \mathcal{X})$ et $\mathcal{V}ar(\alpha) = \{x,y\}$
2. $H(\perp) \rightarrow \beta$ avec $\beta \in \mathcal{T}(\mathcal{F}_{bin} \cup \mathcal{F}_{NT})$
3. $H(a(x,\perp)) \rightarrow x$ avec $x \in \mathcal{X}$, $H \in \mathcal{F}_{NT}$ et $a \in \mathcal{F}_{bin}$.

Maintenant nous donnons quelques algorithmes qui expliquent

1. comment appliquer l'ensemble de règles $\mathcal{R}_{\lambda^e}$ sur le terme $H(s)$ et
2. comment calculer le point fixe.

Nous présentons l'algorithme 1 qui calcule un pas de ré-écriture en traitant chaque cas de forme de règle élémentaire. Si la règle est bien de la forme $H(a(x,y)) \rightarrow \alpha$, l'algorithme 1 va faire appel à l'algorithme 2 pour traiter n'importe quel $\alpha \in \mathcal{T}(\mathcal{F}_{bin} \cup \mathcal{F}_{NT}, \mathcal{X})$. Le dernier traite le cas où il existe les règles de point fixe $H(a(x,y)) \rightarrow a(x,y), H(a(x,y)) \rightarrow H(H'(a(x,y))) \in \mathcal{R}_{\lambda^e}$. Si c'est bien ce cas, l'algorithme 3 sera appelé.

Algorithme 1 Calcul de $H_{\mathcal{R}_{\lambda^e}}(s)$

Précondition : $s \subseteq \mathcal{T}(\mathcal{F}_{bin})$, $H \in \mathcal{F}_{NT}$ et $\mathcal{R}_{\lambda^e}$ un ensemble de règles d'EFTRSs

1: $res \leftarrow \emptyset$
2: **si** $\perp \in s$ et $H(\perp) \rightarrow \beta \in \mathcal{R}_{\lambda^e}$ **alors**
3: res $+\leftarrow \beta$
4: **fin si**
5: **pour** $a(u,v) \in s$ et $H(a(x,y)) \rightarrow \alpha \in \mathcal{R}_{\lambda^e}$ **faire**
6: res $+\leftarrow \alpha_{\mathcal{R}_{\lambda^e}}(u,v)$
7: **fin pour**
8: **pour** $a(u,\perp) \in s$ et $H(a(x,\perp)) \rightarrow x \in \mathcal{R}_{\lambda^e}$ **faire**
9: res $+\leftarrow u$
10: **fin pour**
11: **pour** $a(u,v) \in s$ et $H(a(x,y)) \rightarrow a(x,y), H(a(x,y)) \rightarrow H(H'(a(x,y))) \in \mathcal{R}_{\lambda^e}$ **faire**
12: res $+\leftarrow FP^{H'}_{\mathcal{R}_{\lambda^e}}(\{a(u,v)\})$
13: **fin pour**
14: **retourner** res

Algorithme 2 Calcul de $\alpha_{\mathcal{R}_{\lambda^e}}(u, v)$

Précondition : $u, v \in \mathcal{T}(\mathcal{F}_{bin})$, $\alpha \in \mathcal{T}(\mathcal{F}_{bin} \cup \mathcal{F}_{NT}, \mathcal{X})$ et $\mathcal{R}_{\lambda^e}$ un ensemble de règles d'EFTRSs

1: **si** $(\alpha = \bot)$ **alors**
2: **retourner** $\bot$
3: **sinon si** $(\alpha = x)$ **alors**
4: **retourner** u
5: **sinon si** $(\alpha = y)$ **alors**
6: **retourner** v
7: **sinon si** $(\alpha = H(\alpha')$ avec $H \in \mathcal{F}_{NT})$ **alors**
8: **retourner** $H_{\mathcal{R}_{\lambda^e}}(\alpha'_{\mathcal{R}_{\lambda^e}}(u, v))$
9: **sinon si** $(\alpha = a(\alpha_1(x, y), \alpha_2(x, y))$ avec $a \in \mathcal{F}_{bin})$ **alors**
10: **retourner** $a(\alpha_{1\mathcal{R}_{\lambda^e}}(u, v), \alpha_{2\mathcal{R}_{\lambda^e}}(u, v))$
11: **fin si**

Dans l'algorithme 1, la notation $\alpha_{\mathcal{R}_{\lambda^e}}(u, v)$ visite le côté droit de $H(a(x, y)) \to \alpha \in \mathcal{R}_{\lambda^e}$ et substitue respectivement les variables x, y par u et v dans α. De plus, si un symbole non-terminal de $\mathcal{F}_{NT}$ est rencontré pendant l'exploration, $H_{\mathcal{R}_{\lambda^e}}$ est appelé. Cela garantit le mécanisme d'évaluation en mode fonctionnel. Ce processus est décrit dans l'algorithme 2. Le dernier traite le cas où $\mathcal{R}_{\lambda^e}$ n'est pas linéaire droite.

Nous allons illustrer cet algorithme par notre étude de cas TAP.

Exemple 21 *Soit l'ensemble de règles de ré-écriture $\mathcal{R}_{\lambda^e}^{req}$ (4.8), (4.9), (4.10) et (4.11) ainsi que les règles de circulation $\mathcal{R}_{\lambda^e}^{cir}$: (4.12), ... (4.19) de notre étude de cas TAP dans la section 4.2. Au début, nous voulons calculer un pas de ré-écriture fonctionnel en utilisant l'algorithme 1 :*

$$H_{\mathcal{R}_{\lambda^e}^{req} \cup \mathcal{R}_{\lambda^e}^{cir}{}_{\to \cup}}(u) = \quad \mathbf{H}(t(i(i(\bot, \bot), i(\bot, \bot)), i(i(\bot, \bot), i(\bot, \bot))))$$

En appliquant les règles de circulation $\mathcal{R}_{\lambda^e}^{cir}$, nous trouvons des termes ci-dessous :

$$t(\mathbf{H}(i(i(\bot, \bot), i(\bot, \bot))), i(i(\bot, \bot), i(\bot, \bot))) \quad + \quad t(i(i(\bot, \bot), i(\bot, \bot)), \mathbf{H}(i(i(\bot, \bot), i(\bot, \bot))))$$

Et puis les sous termes ayant H à la racine continuent à être simplifiés en utilisant le $\mathcal{R}_{\lambda^e}^{cir}$:

$$\begin{array}{lll} t((i(\mathbf{H}(i(\bot, \bot)), i(\bot, \bot)), i(i(\bot, \bot), i(\bot, \bot))) & + & t((i(i(\bot, \bot), \mathbf{H}(i(\bot, \bot))), i(i(\bot, \bot), i(\bot, \bot))) + \\ t(i(i(i(\bot, \bot), i(\bot, \bot)), i(\mathbf{H}(i(\bot, \bot)), i(\bot, \bot))) & + & t(i(i(i(\bot, \bot), i(\bot, \bot)), i(i(\bot, \bot), \mathbf{H}(i(\bot, \bot))))). \end{array}$$

A ce moment, les règles $\mathcal{R}_{\lambda^e}^{cir}$ deviennent applicables, nous alors atteignons un ensemble de termes clos ci-dessous :

$$\begin{array}{lll} t(i(r(\bot, \bot), i(\bot, \bot)), i(i(\bot, \bot), i(\bot, \bot))) & + & t(i(i(i(\bot, \bot), r(\bot, \bot)), i(i(\bot, \bot), i(\bot, \bot))) + \\ t(i(i(i(\bot, \bot), i(\bot, \bot)), i(r(\bot, \bot), i(\bot, \bot))) & + & t(i(i(i(\bot, \bot), i(\bot, \bot)), i(i(\bot, \bot), r(\bot, \bot)))) \end{array}$$

Après la canonisation, $H_{\mathcal{R}_{\lambda^e}^{req} \cup \mathcal{R}_{\lambda^e}^{cir}}(u) =$

$$\begin{array}{l} t(\mathbf{i}(\mathbf{r}(\bot, \bot), \mathbf{i}(\bot, \bot)) + \mathbf{i}(\mathbf{i}(\bot, \bot), \mathbf{r}(\bot, \bot)), i(i(\bot, \bot), i(\bot, \bot))) + \\ t(i(i(\bot, \bot), i(\bot, \bot)), \mathbf{i}(\mathbf{r}(\bot, \bot), \mathbf{i}(\bot, \bot)) + \mathbf{i}(\mathbf{i}(\bot, \bot), \mathbf{r}(\bot, \bot))) \end{array}$$

Algorithme 3 Calcul de $FP^{H}_{\mathcal{R}_{\lambda^e}}(s)$

Précondition : $s \subseteq \mathcal{T}(\mathcal{F}_{bin})$, $H \in \mathcal{F}_{NT}$ et $\mathcal{R}_{\lambda^e}$ un ensemble de règles d'EFTRSs

1: $res_1 \leftarrow \emptyset$;
2: $res_2 \leftarrow s$;
3: **tantque** $res_1 \neq res_2$ **faire**
4: $\quad res_1 \leftarrow res_2$;
5: $\quad res_2 + \leftarrow H_{\mathcal{R}_{\lambda^e}}(res_2)$
6: **fin tanque**
7: **retourner** res_2

Sémantiquement, dans cet exemple, nous pouvons considérer le symbole H comme un marqueur indiquant à quel endroit notre système de ré-écriture est en train de traiter. Autrement dit, chaque terme u de l'ensemble de termes donné s spécifiant un EFTRS, $H(u)$ spécifie que le terme u doit être mis à jour. Par conséquent, nous appliquons $\mathcal{R}_{\lambda^e}$ sur $H(u)$ dans le but d'implémenter la mise-à-jour autant que possible.

Autrement dit, les algorithmes 1 et 2 appliquent le symbole H sur chaque terme de l'ensemble de termes clos et appliquent le système de ré-écriture autant que possible.

Il faut savoir également que la sémantique du symbole fonctionnel H est, dans un certain sens, définie grâce à l'EFTRS $\mathcal{R}_{\lambda^e}$, pour un ensemble de termes. Il est intéressant de calculer toutes les configurations possibles en appliquant H sur $s \subseteq \mathcal{T}(\mathcal{F}_{bin})$ un nombre non-borné de fois et appliquer $\mathcal{R}_{\lambda^e}$ autant de fois que possible, *c.à.d.*. $FP_{\mathcal{R}_{\lambda^e}}(s)$. Si un point fixe est obtenu, l'ensemble de termes dans $\mathcal{T}(\mathcal{F}_{bin})$ accessibles est calculé.

L'algorithme 1 calcule l'ensemble de configurations accessibles de l'ensemble de termes de $\mathcal{T}(\mathcal{F}_{bin})$ après avoir appliqué H à la position la plus haute. Autrement dit, il montre la robustesse de cet algorithme.

Finalement, pour calculer l'ensemble des accessibles (comme mentionné dans le chapitre 4), nous avons besoin de définir le calcul du point fixe. C'est le rôle de l'algorithme 3.

En fait, l'algorithme 1, l'algorithme 2 et l'algorithme 3 proposés ci-dessus ont profité les techniques d'accélération de la vérification symbolique *à la BDD* [Bryant 1986, Bryant 1992] telles que :

- **Tableau d'unicité** *à la BDD* de l'ensemble de termes de $\mathcal{T}(\mathcal{F}_{bin})$: Techniquement dit, nous n'acceptons qu'un unique représentant pour un terme de $\mathcal{T}(\mathcal{F}_{bin})$. Cela permet les données codées des modèles très compactes.
- **Canonisation** de l'ensemble de termes de $\mathcal{T}(\mathcal{F}_{bin})$: Les algorithmes profitent bien les propriétés linéaires des termes, déjà parlé à la section 3.1 et à la section 4.1.1, pour que le codage des modèles soient plus compactes.

– **Cache** *à la BDD* : Chaque calcul de l'algorithme 1 et l'algorithme 2 doit être mis en cache pour éviter de re-faire un résultat déjà calculé.

L'exemple suivant va illustrer comment le point fixe est calculé pour notre étude de cas TAP.

Exemple 22 *Etant donnés des ensembles de règles* ${\mathcal{R}_{\lambda^e}}^{cir} \cup {\mathcal{R}_{\lambda^e}}^{req}$. *de l'exemple 21, nous avons calculé* $u_1 = H_{\mathcal{R}^{req}_{\lambda^e} \cup \mathcal{R}^{cir}_{\lambda^e}}(u) = \{u_{11}, u_{12}, u_{13}, u_{14}\}$ *avec*

$$u_{11} = t(i(\boldsymbol{r}(\bot,\bot), i(\bot,\bot)), i(i(\bot,\bot), i(\bot,\bot))), \quad u_{12} = t(i(i(\bot,\bot), \boldsymbol{r}(\bot,\bot)), i(i(\bot,\bot), i(\bot,\bot))),$$
$$u_{13} = t(i(i(\bot,\bot), i(\bot,\bot)), i(\boldsymbol{r}(\bot,\bot), i(\bot,\bot))), \quad u_{14} = t(i(i(\bot,\bot), i(\bot,\bot)), i(i(\bot,\bot), \boldsymbol{r}(\bot,\bot))).$$

Le deuxième pas de ré-écriture est de re-faire exactement l'exemple précédent avec l'ensemble de termes de u_1, *c.à.d nous calculons*

$$u_2 = H_{\mathcal{R}^{req}_{\lambda^e} \cup \mathcal{R}^{cir}_{\lambda^e}}(u_1) = H_{\mathcal{R}^{req}_{\lambda^e} \cup \mathcal{R}^{cir}_{\lambda^e}}(u_{11} + u_{12} + u_{13} + u_{14}) =$$

$$H(t(i(\boldsymbol{r}(\bot,\bot), i(\bot,\bot)), i(i(\bot,\bot), i(\bot,\bot)))) \quad + \quad H(t(i(i(\bot,\bot), \boldsymbol{r}(\bot,\bot)), i(i(\bot,\bot), i(\bot,\bot)))) +$$
$$H(t(i(i(\bot,\bot), i(\bot,\bot)), i(\boldsymbol{r}(\bot,\bot), i(\bot,\bot)))) \quad + \quad H(t(i(i(\bot,\bot), i(\bot,\bot)), i(i(\bot,\bot), \boldsymbol{r}(\bot,\bot)))).$$

Après l'élimination des termes redondants et la canonisation, nous calculons comme suit

$$t(i(\boldsymbol{r}(\bot,\bot), \boldsymbol{r}(\bot,\bot)), i(i(\bot,\bot), i(\bot,\bot))) \quad + \quad t(i(i(\bot,\bot), i(\bot,\bot)), i(\boldsymbol{r}(\bot,\bot), \boldsymbol{r}(\bot,\bot))) +$$
$$t(i(\boldsymbol{r}(\bot,\bot), i(\bot,\bot)), i(\boldsymbol{r}(\bot,\bot), i(\bot,\bot))) \quad + \quad t(i(\boldsymbol{r}(\bot,\bot), i(\bot,\bot)), i(i(\bot,\bot), \boldsymbol{r}(\bot,\bot))) +$$
$$t(i(i(\bot,\bot), \boldsymbol{r}(\bot,\bot)), i(\boldsymbol{r}(\bot,\bot), i(\bot,\bot))) \quad + \quad t(i(i(\bot,\bot), \boldsymbol{r}(\bot,\bot)), i(i(\bot,\bot), \boldsymbol{r}(\bot,\bot))) +$$
$$t(\boldsymbol{r}(\boldsymbol{r}(\bot,\bot), i(\bot,\bot)), i(i(\bot,\bot), i(\bot,\bot))) \quad + \quad t(\boldsymbol{r}(i(\bot,\bot), \boldsymbol{r}(\bot,\bot)), i(i(\bot,\bot), i(\bot,\bot))) +$$
$$t(i(i(\bot,\bot), i(\bot,\bot)), \boldsymbol{r}(\boldsymbol{r}(\bot,\bot), i(\bot,\bot))) \quad + \quad t(i(i(\bot,\bot), i(\bot,\bot)), \boldsymbol{r}(i(\bot,\bot), \boldsymbol{r}(\bot,\bot)))$$

Après la canonisation, nous pouvons obtenir une représentation plus compact ci-dessous :

$$t(i(i(\bot,\bot), i(\bot,\bot)), \boldsymbol{r}(\boldsymbol{r}(\bot,\bot), i(\bot,\bot)) \; + \; \boldsymbol{r}(i(\bot,\bot), \boldsymbol{r}(\bot,\bot)) \; + \; i(\boldsymbol{r}(\bot,\bot), \boldsymbol{r}(\bot,\bot))) +$$
$$t(i(\boldsymbol{r}(\bot,\bot), \boldsymbol{r}(\bot,\bot)) \; + \; \boldsymbol{r}(\boldsymbol{r}(\bot,\bot), i(\bot,\bot)) \; + \; \boldsymbol{r}(i(\bot,\bot), \boldsymbol{r}(\bot,\bot)), i(i(\bot,\bot), i(\bot,\bot))) +$$
$$t(i(\boldsymbol{r}(\bot,\bot), i(\bot,\bot)), i(\boldsymbol{r}(\bot,\bot), i(\bot,\bot)) \; + \; i(i(\bot,\bot), \boldsymbol{r}(\bot,\bot))) +$$
$$t(i(i(\bot,\bot), \boldsymbol{r}(\bot,\bot)), i(\boldsymbol{r}(\bot,\bot), i(\bot,\bot)) \; + \; i(i(\bot,\bot), \boldsymbol{r}(\bot,\bot)))$$

Les pas de ré-écriture suivants fonctionneront de la même manière. Il faut noter que dans ce cas-là $u, v \notin H_{\mathcal{R}^{req}_{\lambda^e} \cup \mathcal{R}^{cir}_{\lambda^e}}(u)$ *mais* $u, v \in FP_{\mathcal{R}^{req}_{\lambda^e} \cup \mathcal{R}^{cir}_{\lambda^e}}(u)$.

Intuitivement, nous voyons que le terme final est quasi-similaire au terme initial sauf le changement des symboles des sous-arbres de la racine étiquetée par le symbole t. D'autre part, les opérations de circulation pour des sous-arbres de la racine font le même comportement.

Une question évidente est *Est-il possible d'accélérer le calcul en évitant la répétition des opérations pour la circulation en descendant une seule fois à l'endroit intéressé et saturer le noeud courant autant que possible.*

Ces techniques nous permettent de lutter contre l'explosion combinatoire pour les modèles symétriques. Malheureusement, la compacité des données nous conduit à un autre problème qui est l'effet des données intermédiaires. Les points fixes locaux (LFP) (ou la stratégie de saturation [Ciardo 2003]) présentées dans la section suivante sont une bonne solution pour ce problème.

Algorithme 4 Calcul de $FP^{H}_{\mathcal{R}_{\lambda^e}}(s)$ avec des règles de point fixe

Précondition : $s \in \mathcal{T}(\mathcal{F}_{bin})$, $H, H^\star \in \mathcal{F}_{NT}$ et $\mathcal{R}_{\lambda^e}$ un ensemble de règles d'EFTRSs

1: $res_1 \leftarrow \emptyset$;
2: $res_2 \leftarrow s$;
3: **tantque** $res_1 \neq res_2$ **faire**
4: $\quad res_1 \leftarrow res_2$;
5: $\quad res_2+ \leftarrow H^\star_{\mathcal{R}_{\lambda^e}}(res_2)$
6: $\quad res_2+ \leftarrow H_{\mathcal{R}_{\lambda^e}}(res_2)$
7: **fin tanque**
8: **retourner** res_2

5.2 Optimisation

Cette section traite l'optimisation des algorithmes d'évaluation en utilisant des techniques d'accélération des calculs aboutissant à un outil de vérification symbolique efficace.

Pour quelques types de règles élémentaires à droite, nous allons montrer la capacité d'accélération des calculs. Évidemment, l'efficacité de cette optimisation dépend du nombre de règles de ré-écriture applicables, de l'importance de ces règles ainsi que du niveau de symétrie du modèle donné.

5.2.1 Évaluation avec la stratégie de saturation

Notre but est d'accélérer le calcul en évitant la répétition des opérations pour la circulation en descendant une seule fois à l'endroit intéressé et saturer le noeud courant tant que possible. Les points fixes locaux (LFP) (ou la stratégie de saturation) sont une bonne solution.

La stratégie de saturation (appelée également Les *Points Fixes Locaux, LFP*) est proposée dans [Ciardo 2003]. Elle lutte contre l'effet de la taille des données intermédiaires. Effectivement, cet algorithme est très efficace pour des structures très compactes à la BDD tels que MDD [Ciardo 2003], DDD [Couvreur 2002], SDD [Couvreur 2005], HSDD [Thierry-Mieg 2008], [Thierry-Mieg 2004].

Tout d'abord, nous introduisons un nouveau symbole non-terminal $H^\star$ pour chaque symbole non-terminal H. Ces nouveaux symboles peuvent être utilisés dans des règles ré-écriture comme un symbole non-terminal normal. Ils sont interprétés comme H^n pour des entiers non négatifs n.

Définition 11 *Transformation des REFTRSs de type I.*
Soit $\mathcal{R}_{\lambda^e \to I}$ un ensemble de règles ayant un côté gauche qui est de la forme $H\ (a(x, y))$ où $H \in \mathcal{F}_{NT}$ et $a \in \mathcal{F}_{bin}$. Selon des cas de règles du côté droit, nous avons les nouvelles règles suivantes :

$\mathcal{R}_{\lambda^e \to I}$	$\mathcal{R}'_{\lambda^e \to I}$
$H\ (a(x,y)) \to a(x,y)$	$H^\star\ (a(x,y)) \to a(x,y)$
$H\ (a(x,y)) \to a(G\ (x),y)$	$H^\star\ (a(x,y)) \to a(G^\star\ (x),y)$
$H\ (a(x,y)) \to a(x,D\ (y))$	$H^\star\ (a(x,y)) \to a(x,D^\star\ (y))$
$H\ (a(x,y)) \to a(G\ (x),y)$ $H\ (a(x,y)) \to a(x,D\ (y))$	$H^\star\ (a(x,y)) \to a(G^\star\ (x),D^\star\ (y))$
$H\ (a(x,y)) \to a(x,y)$ $H\ (a(x,y)) \to a(x,D\ (y))$	$H^\star\ (a(x,y)) \to a(x,D^\star\ (y))$
$H\ (a(x,y)) \to a(x,y)$ $H\ (a(x,y)) \to a(G\ (x),y)$	$H^\star\ (a(x,y)) \to a(G^\star\ (x),y)$
$H\ (a(x,y)) \to a(x,y)$ $H\ (a(x,y)) \to a(G\ (x),y)$ $H\ (a(x,y)) \to a(x,D\ (y))$	$H^\star\ (a(x,y)) \to a(G^\star\ (x),D^\star\ (y))$

avec $H,\ G,\ D \in \mathcal{F}_{NT}$, $H^\star,\ G^\star,\ D^\star \in \mathcal{F}_{NT}$ *et* $a \in \mathcal{F}_{bin}$

Il faut noter que la transformation des REFTRS de type I peut être appliquée séparément pour l'ensemble de règles $H\ (a(x,y)) \to \alpha$ et l'ensemble de règles $H\ (b(x,y)) \to \beta$ où $\alpha, \beta \in \mathcal{T}(\mathcal{F}_{bin} \cup \mathcal{F}_{NT}, \mathcal{X})$ s'il n'existe pas de règle de type III telle que $H\ (a(x,y)) \to b(x,y)$ ou $H\ (b(x,y)) \to a(x,y)$.

Définition 12 *Transformation des REFTRSs de type II.*
Soit $\mathcal{R}_{\lambda^e \to II}$ *un ensemble de règles ayant un côté gauche qui est de la forme* $H\ (a(x,y))$ *où* $H \in \mathcal{F}_{NT}$ *et* $a \in \mathcal{F}_{bin}$. *Selon les cas de règles du côté droit, nous avons de nouvelles règles comme les suivantes :*

$\mathcal{R}_{\lambda^e \to II}$	$\mathcal{R}'_{\lambda^e \to II}$
$H\ (a(x,y)) \to G(D(a(x,y)))$ $G\ (a(x,y)) \to a(x,y)$	$H^\star\ (a(x,y)) \to D^\star\ (a(x,y))$
$H\ (a(x,y)) \to G(D(a(x,y)))$ $D\ (a(x,y)) \to a(x,y)$	$H^\star\ (a(x,y)) \to G^\star\ (a(x,y))$
$H\ (a(x,y)) \to G(D(a(x,y)))$ $G\ (b(x,y)) \to b(x,y)$ $G\ (a(x,y)) \to c(x,y)$ $G\ (c(x,y)) \to \alpha$ $D\ (a(x,y)) \to b(x,y)$ $D\ (c(x,y)) \to c(x,y)$ $D\ (b(x,y)) \to \beta$	$H^\star\ (a(x,y)) \to G^\star\ (c(x,y))$ $H^\star\ (a(x,y)) \to D^\star\ (b(x,y))$

avec $H,\ G,\ D \in \mathcal{F}_{NT}$, $H^\star,\ G^\star,\ D^\star \in \mathcal{F}_{NT}$ *et* $a \in \mathcal{F}_{bin}$, $\alpha, \beta \in \mathcal{T}(\mathcal{F}_{bin} \cup \mathcal{F}_{NT}, \mathcal{X})$, $\mathcal{V}ar(\alpha) = \{x,y\}$ *et* $\mathcal{V}ar(\beta) = \{x,y\}$.

Sémantiquement dit, nous pouvons considérer le symbole $H^\star$ comme un marqueur indiquant à quel endroit notre système de ré-écriture est en train de lancer un point fixe. Par conséquent, les règles transformation peuvent nous aider à lancer

Algorithme 5 Calcul de $\alpha_{\mathcal{R}_{\lambda^e}}(u, v)$ avec des règles de point fixe

Précondition : $u, v \in \mathcal{T}(\mathcal{F}_{bin})$, $\alpha \in \mathcal{T}(\mathcal{F}_{bin} \cup \mathcal{F}_{NT}, \mathcal{X})$ et $\mathcal{R}_{\lambda^e}$ un ensemble de règles d'EFTRSs

1: **si** $(\alpha = \bot)$ **alors**
2: **retourner** $\bot$
3: **sinon si** $(\alpha = x)$ **alors**
4: **retourner** u
5: **sinon si** $(\alpha = y)$ **alors**
6: **retourner** v
7: **sinon si** $(\alpha = H(\alpha')$ avec $H \in \mathcal{F}_{NT})$ **alors**
8: **retourner** $H_{\mathcal{R}_{\lambda^e}}(\alpha'_{\mathcal{R}_{\lambda^e}}(u, v))$
9: **sinon si** $(\alpha = H^\star(\alpha')$ avec $H^\star \in \mathcal{F}_{NT})$ **alors**
10: **retourner** $FP^H_{\mathcal{R}_{\lambda^e}}(\alpha'_{\mathcal{R}_{\lambda^e}}(u, v))$
11: **sinon si** $(\alpha = a(\alpha_1(x, y), \alpha_2(x, y))$ avec $a \in \mathcal{F}_{bin})$ **alors**
12: **retourner** $a(\alpha_{1\mathcal{R}_{\lambda^e}}(u, v), \alpha_{2\mathcal{R}_{\lambda^e}}(u, v))$
13: **fin si**

les calculs de LFP dans les endroits intéressants pour optimiser le temps de calcul. Désormais, nous essayons d'intégrer ces transformations dans nos algorithmes d'évaluation classiques.

L'algorithme 4 est la nouvelle version de l'algorithme 3 (l'algorithme du point fixe) qui compte les nouveaux symboles non-terminaux $H^\star$. Dans la boucle de ce nouvel algorithme, les règles de l'opérateur de point fixe sont appliquées en appelant la fonction $H^\star_{\mathcal{R}_{\lambda^e}}(s)$ (l'algorithme 1 avec des symboles non-terminaux $H^\star$).

L'algorithme 2 est également substitué par l'algorithme 5. En regardant le nouveau symbole non-terminal $H^\star$, cet algorithme appelle l'algorithme 4 (l'algorithme du point fixe).

Intuitivement, selon la proposition 5 et la construction des algorithmes listée au dessous :

- Lignes 5 de l'algorithme 4
- Lignes 9 et 10 de l'algorithme 5

on peut conclure que l'évaluation par l'algorithme 4 et par l'algorithme 3 donne le même résultat.

En fait, en lançant le calcul $H^\star_{\mathcal{R}_{\lambda^e}}(res_2)$(Ligne 5 de l'algorithme 4) avant le calcul $H_{\mathcal{R}_{\lambda^e}}(res_2)$(Ligne 6 de l'algorithme 4) nous pouvons atteindre le point fixe plus vite que l'algorithme 3.

Proposition 5 *Soit $\mathcal{R}_{\lambda^e}$ un EFTRS et $s \in \mathcal{T}(\mathcal{F}_{bin})$. S'il existe des $\mathcal{R}_{\lambda^e \to I} \subseteq \mathcal{R}_{\lambda^e}$ ou $\mathcal{R}_{\lambda^e \to II} \subseteq \mathcal{R}_{\lambda^e}$ remplissant des conditions de la définition 11 ou de la définition 12 ainsi*

$$(\mathcal{R}_{\lambda^e} \cup (\mathcal{R}'_{\lambda^e \to I} \cup \mathcal{R}'_{\lambda^e \to II}))^\star(E') = \mathcal{R}^\star_{\lambda^e}(E').$$

avec $E \subseteq \mathcal{T}(\mathcal{F}_{bin})$ *et* $E' = \{H(t)|t \in E\}$.

Preuve 9 – $\mathcal{R}^{\star}_{\lambda^e}(E') \subseteq (\mathcal{R}_{\lambda^e} \cup (\mathcal{R}'_{\lambda^e \to I} \cup \mathcal{R}'_{\lambda^e \to II}))^{\star}(E')$, *c'est évident selon* $\mathcal{R}_{\lambda^e} \subseteq (\mathcal{R}_{\lambda^e} \cup (\mathcal{R}'_{\lambda^e \to I} \cup \mathcal{R}'_{\lambda^e \to II}))$

– $(\mathcal{R}_{\lambda^e} \cup (\mathcal{R}'_{\lambda^e \to I} \cup \mathcal{R}'_{\lambda^e \to II}))^{\star}(E') \subseteq \mathcal{R}^{\star}_{\lambda^e}(E')$. *Poursuivons en traitant l'analyse de cas :*

  – $(\mathcal{R}_{\lambda^e} \cup \mathcal{R}'_{\lambda^e \to I})^{\star}(E') \subseteq \mathcal{R}^{\star}_{\lambda^e}(E')$:

    – $\mathcal{R}'_{\lambda^e}{}^{\star}_{H^{\star}\ (a(x,y)) \to a(x,y)}(E') \subseteq \mathcal{R}_{\lambda^e}{}^{\star}_{H\ (a(x,y)) \to a(x,y)}(E')$

    – $\mathcal{R}'_{\lambda^e}{}^{\star}_{H^{\star}\ (a(x,y)) \to a(G^{\star}\ (x),y)}(E') \subseteq \mathcal{R}_{\lambda^e}{}^{\star}_{H\ (a(x,y)) \to a(G\ (x),y)}(E')$ *car* $H^n\ (a(x,y)) \to^{\star}_{H\ (a(x,y)) \to a(G\ (x),y)} a(G^n\ (x), y)$ *avec* n *suffisamment grand.*

    – *Similairement pour le reste.*

  – $(\mathcal{R}_{\lambda^e} \cup \mathcal{R}'_{\lambda^e \to II})^{\star}(E') \subseteq \mathcal{R}^{\star}_{\lambda^e}(E')$:

    – *Similairement pour les autres cas.*

Nous allons illustrer ces nouveaux algorithmes par notre étude de cas TAP.

Exemple 23 *Soit* $\mathcal{R}^{cir}_{\lambda^e}$ *défini dans l'exemple précédent. Les règles de circulation (4.12), ... (4.19) de* $\mathcal{R}^{cir}_{\lambda^e}$ *étant des élémentaires à droite type I (*$\mathcal{R}^{cir}_{\lambda^e \to I}$*) peuvent être optimisées pour le Calcul de point fixe comme la façon suivante :*

$$H^{\star}(i(x,y)) \to i(H^{\star}(x), H^{\star}(y)) \tag{5.1}$$
$$H^{\star}(r(x,y)) \to r(H^{\star}(x), H^{\star}(y)) \tag{5.2}$$
$$H^{\star}(b(x,y)) \to b(H^{\star}(x), H^{\star}(y)) \tag{5.3}$$
$$H^{\star}(t(x,y)) \to t(H^{\star}(x), H^{\star}(y)) \tag{5.4}$$

En appliquant l'algorithme de point fixe à la circulation pour les règles de circulation optimisées $\mathcal{R}'^{cir}_{\lambda^e \to I}$ *(5.1), (5.2), (5.3), (5.4)* $FP^{H}_{\mathcal{R}^{cir}_{\lambda^e} \cup \mathcal{R}^{req}_{\lambda^e} \cup \mathcal{R}'^{cir}_{\lambda^e \to I}}(u)$ *donne après quelques calculs. La symétrie des sous-termes nous permet de re-prendre les résultats des points fixes locaux déjà calculés en cache. Finalement, nous atteignons le point fixe global* $FP^{H}_{\mathcal{R}^{cir}_{\lambda^e} \cup \mathcal{R}^{req}_{\lambda^e} \cup \mathcal{R}'^{cir}_{\lambda^e \to I}}(u) = t(a(a(\bot,\bot), a(\bot,\bot)), a(a(\bot,\bot), a(\bot,\bot)))$ *où* $a \in \{i, r\}$.

Il faut noter que les arbres sont des structures très compactes grâce au tableau d'unicité et à la canonisation.

Nous remarquons aussi que quelques calculs pour des règles dans $\mathcal{R}_{\lambda^e \to I}$ et $\mathcal{R}_{\lambda^e \to II}$ sont inutiles car ils sont déjà calculés. Poursuivons les nouveaux algorithmes pour réduire ces inconvénients dans la section suivante.

Algorithme 6 Calcul de $FP^{H}_{\mathcal{R}_{\lambda^e}}(s)$ avec de nouvelles règles de point fixe

Précondition : $s \in \mathcal{T}(\mathcal{F}_{bin})$, $H, H^{\star} \in \mathcal{F}_{NT}$ et $\mathcal{R}_{\lambda^e}$ un ensemble de règles d'EFTRSs

1: $res_1 \leftarrow \emptyset$;
2: $res_2 \leftarrow s$;
3: **tantque** $res_1 \neq res_2$ **faire**
4: $\quad res_1 \leftarrow res_2$;
5: $\quad res_2+ \leftarrow H^{\star}_{\mathcal{R}'_{\lambda^e \rightarrow I} \cup \mathcal{R}'_{\lambda^e \rightarrow II}}(res_2)$
6: $\quad res_2+ \leftarrow H_{\mathcal{R}_{\lambda^e}, \mathcal{R}'_{\lambda^e \rightarrow I}, \mathcal{R}'_{\lambda^e \rightarrow II}, \mathcal{R}_{\lambda^e \rightarrow I}, \mathcal{R}_{\lambda^e \rightarrow II}}(res_2)$
7: **fin tanque**
8: **retourner** res_2

5.2.2 Algorithme d'évaluation optimisée

Comme déjà mentionné, nous proposons de nouveaux algorithmes pour éviter les re-calculs inutiles pour des règles dans $\mathcal{R}_{\lambda^e \rightarrow I} \cup \mathcal{R}_{\lambda^e \rightarrow II}$.

L'algorithme 6 est la nouvelle version de l'algorithme 3 (l'algorithme du point fixe) qui compte l' introduction automatique du nouveau TRS. Dans la boucle de l'algorithme, des règles avec l'opérateur de point fixe sont appliquées en appelant une fonction $H^{\star}_{\mathcal{R}'_{\lambda^e \rightarrow I} \cup \mathcal{R}'_{\lambda^e \rightarrow II}}(s)$ et en appelant l'algorithme 1 $H_{\mathcal{R}_{\lambda^e}}(s)$ limité pour la règle étant dans $\mathcal{R}_{\lambda^e \rightarrow I}$ ou $\mathcal{R}_{\lambda^e \rightarrow II}$ L'algorithme 7 donne une façon d'appliquer $H_{\mathcal{R}_{\lambda^e}}(s)$ sans utiliser aucune règle dans $\mathcal{R}_{\lambda^e \rightarrow I}$ et $\mathcal{R}_{\lambda^e \rightarrow II}$ pour la première étape de la transformation.

L'application des nouveaux algorithmes donne exactement le même calcul comme le calcul de systèmes utilisant les règles optimisées de la section précédente pour le même problème dans un temps de calcul quasi-identique.

Intuitivement, selon la proposition 6 et la construction des algorithmes listée ci-dessous :

- Lignes 5 et 6 de l'algorithme 6
- Lignes 5 et 6 de l'algorithme 7

nous pouvons conclure que l'évaluation par l'algorithme 6 et par l'algorithme 4 donne le même résultat de l'algorithme 3.

Nous avons donc la proposition 6, un résultat plus fort que la proposition 5.

Proposition 6 *Soit $\mathcal{R}_{\lambda^e}$ un EFTRS et $s \in \mathcal{T}(\mathcal{F}_{bin})$. S'il existe des $\mathcal{R}_{\lambda^e \rightarrow I} \subseteq \mathcal{R}_{\lambda^e}$ ou $\mathcal{R}_{\lambda^e \rightarrow II} \subseteq \mathcal{R}_{\lambda^e}$ remplissant des conditions de la définition 11 ou de la définition 12 ainsi*

$$(\mathcal{R}_{\lambda^e} \cup (\mathcal{R}'_{\lambda^e \rightarrow I} \cup \mathcal{R}'_{\lambda^e \rightarrow II}) \setminus (\mathcal{R}_{\lambda^e \rightarrow I} \cup \mathcal{R}_{\lambda^e \rightarrow II}))^{\star}(E') = \mathcal{R}^{\star}_{\lambda^e}(E').$$

avec $E \subseteq \mathcal{T}(\mathcal{F}_{bin})$ et $E' = \{H(t) | t \in E\}$.

Algorithme 7 Calcul de $H_{\mathcal{R}_{\lambda^e},\mathcal{R}'_{\lambda^e\to I},\mathcal{R}'_{\lambda^e\to II},\mathcal{R}_{\lambda^e\to I},\mathcal{R}_{\lambda^e\to II}}(s)$ avec de nouvelles règles de point fixe

Précondition :
$s \in \mathcal{T}(\mathcal{F}_{bin})$, $H \in \mathcal{F}_{NT}$
$\mathcal{R}_{\lambda^e}, \mathcal{R}'_{\lambda^e\to I}, \mathcal{R}'_{\lambda^e\to II}, \mathcal{R}_{\lambda^e\to I}, \mathcal{R}_{\lambda^e\to II}$ sont des ensembles de règles EFTRSs
1: $res \leftarrow \emptyset$
2: **si** $\bot \in s$ et $H(\bot) \to \beta \in \mathcal{R}_{\lambda^e} \setminus(\mathcal{R}_{\lambda^e\to I} \cup \mathcal{R}_{\lambda^e\to II}) \cup(\mathcal{R}'_{\lambda^e\to I} \cup \mathcal{R}'_{\lambda^e\to II})$ **alors**
3: res $+ \leftarrow \beta$
4: **fin si**
5: **pour** $a(u,v) \in s$ et $H(a(x,y)) \to \alpha \in \mathcal{R}_{\lambda^e} \setminus(\mathcal{R}_{\lambda^e\to I} \cup \mathcal{R}_{\lambda^e\to II}) \cup(\mathcal{R}'_{\lambda^e\to I} \cup \mathcal{R}'_{\lambda^e\to II})$ **faire**
6: res $+ \leftarrow \alpha_{\mathcal{R}_{\lambda^e}\cup(\mathcal{R}'_I\cup\mathcal{R}'_{II})}(u,v)$
7: **fin pour**
8: **pour** $a(u,\bot) \in s$ et $H(a(x,\bot)) \to x \in \mathcal{R}_{\lambda^e} \setminus(\mathcal{R}_{\lambda^e\to I} \cup \mathcal{R}_{\lambda^e\to II}) \cup(\mathcal{R}'_{\lambda^e\to I} \cup \mathcal{R}'_{\lambda^e\to II})$ **faire**
9: res $+ \leftarrow u$
10: **fin pour**
11: **pour** $a(u,v) \in s$ et $H(a(x,y)) \to a(x,y), H(a(x,y)) \to H(H'(a(x,y))) \in \mathcal{R}_{\lambda^e}$ **faire**
12: res $+ \leftarrow FP^{H'}_{\mathcal{R}_{\lambda^e}}(\{a(u,v)\})$
13: **fin pour**
14: **retourner** *res*

Preuve 10 *[Preuve succincte de la proposition 6]*

$$(\mathcal{R}_{\lambda^e} \cup (\mathcal{R}'_{\lambda^e\to I} \cup \mathcal{R}'_{\lambda^e\to II}) \setminus (\mathcal{R}_{\lambda^e\to I} \cup \mathcal{R}_{\lambda^e\to II}))^\star(E') = \mathcal{R}^\star_{\lambda^e}(E').$$

avec $E \subseteq \mathcal{T}(\mathcal{F}_{bin})$ *et* $E' = \{H(t)|t \in E\}$.

D'après la proposition 5 nous obtenons

$$(\mathcal{R}_{\lambda^e} \cup (\mathcal{R}'_{\lambda^e\to I} \cup \mathcal{R}'_{\lambda^e\to II}))^\star(E') = \mathcal{R}^\star_{\lambda^e}(E').$$

avec $E \subseteq \mathcal{T}(\mathcal{F}_{bin})$ *et* $E' = \{H(t)|t \in E\}$.

D'autre part, au moment donné, nous pouvons prouver qu'une règle en $\mathcal{R}'_{\lambda^e\to I}$ *est applicable seulement si la règle correspondante en* $\mathcal{R}_{\lambda^e\to I}$ *est applicable. De plus le résultat d'un pas de ré-écriture de la règle en* $\mathcal{R}'_{\lambda^e\to I}$ *est identique à celui de la règle correspondante en* $\mathcal{R}_{\lambda^e\to I}$. *Similaire pour* $\mathcal{R}'_{\lambda^e\to II}$ *avec* $\mathcal{R}_{\lambda^e\to II}$. *Cela nous permet d'obtenir le résultat d'un pas de ré-écriture sans utiliser la règle en* $(\mathcal{R}_{\lambda^e\to I}\cup\mathcal{R}_{\lambda^e\to II})$.

Cela montre la revendication.

Résultats expérimentaux 1 *Dans la partie expérimentale, nous avons implémenté sur des modèles arborescents, et sur des modèles de réseaux de Petri afin de montrer l'efficacité de la technique de LFP.*

Problème	*Taille maximale du modèle*		*Temps de calcul*	
	NoLFP	*LFP*	*NoLFP*	*LFP*
Arborescents	2^5	$2^{400} \rightarrow 2^{500}$	*$\tilde{}$10 secondes*	*< 1 seconde*
Réseaux de Petri	$2^2 \rightarrow 2^4$	$2^4 \rightarrow 2^{35}$	*> 3 heures*	$2 \rightarrow 20$ *minutes*

où les modèles arborescents sont le protocole d'Arbitrage Arborescent (TAP), le protocole de Percolate (PP) et le protocole d'Election Arborescent (LEP), et les modèles de réseaux de Petri sont le problème des Philosophes, le protocole de Slotted-Ring et le protocole de Round-Robin Mutex.

Remarques 7 *Nous revenons aux $\mathcal{R}_{\lambda^e \rightarrow III}$ les règles REFTRS de type III de la forme*

- $H\ (a(x,y)) \rightarrow b(x,y)$
- $H\ (a(x,y)) \rightarrow a(y,x)$
- $H\ (a(x,y)) \rightarrow a(\perp, a(x,y))$

avec $H \in \mathcal{F}_{NT}$ et $a, b \in \mathcal{F}_{bin}$.

Elles ne sont pas encore traitées car la saturation deviendra très complexe à exécuter avec de telles formes. Dans les cas plus simples, par exemple $H(a(x,y)) \rightarrow b(x,y)$ quand des sous-termes de a, b sont $\perp$, nous pouvons effectuer une transformation simple. De plus, pour quelques types de REFTRS, les calculs de points fixes globaux sont terminés.

BILAN DU CHAPITRE 5

Dans ce chapitre, nous avons présenté les algorithmes d'évaluation classiques pour les systèmes de ré-écriture élémentaires (EFTRSs ou REFTRSs). Les autres (les classiques et les FTRSs) doivent passer par une transformation simple (proposée au chapitre précédent) avant l'évaluation par nos algorithmes.

Ce chapitre contient également une partie d'optimisation des algorithmes d'évaluation en utilisant des techniques d'accélération des calculs aboutissant à un outil de vérification symbolique efficace. Évidemment, l'efficacité dépend du nombre de règles de ré-écriture applicables, de l'importance de ces règles ainsi que du niveau de symétrie du modèle donné.

Dans la partie expérimentale au chapitre 8, nous allons comparer des algorithmes listés ci-dessus afin de montrer l'efficacité remarquable de l'algorithme de saturation pour les modèles symétriques tels que le TAP ainsi que les autres protocoles de taille du modèle énorme.

Troisième partie

Partie III : Application à la vérification

Chapitre 6

Modélisation par EFTRS

RÉSUMÉ DU CHAPITRE 6

Dans le chapitre 4, nous avons présenté EFTRS préservant la puissance d'expression des systèmes fonctionnels et des techniques d'accélération des calculs aboutissant à un outil de vérification symbolique efficace. Dans le chapitre 5, nous avons montré comment nous pouvons évaluer des EFTRSs. Les formalismes et les algorithmes d'évaluation de l'EFTRS sont illustrés par le protocole TAP.

Dans ce chapitre, nous allons montrer la puissance de l'EFTRS dans la modélisation. Après le TAP, nous continuons à expliquer comment nous pouvons simuler quelques autres protocoles tels que le protocole de Percolate (PP), le protocole d'Election Arborescent (LEP). Nos ambitions sont également d'éclaircir la relation entre TRS et FTRS, et entre FTRS et EFTRS sur ces protocoles pour l'objectif de réaliser le premier pas de la transformation automatique.

D'autre part, la capacité de la simulation d'un réseau de Petri soit de type ordinaire (appelé également P/T) soit de type hiérarchique avec des EFTRS est également discutée dans ce chapitre.

Sommaire

6.1 Modèles arborescents

Dans la section 4.2.1, nous avons présenté notre système EFTRS sur usage avec le protocole TAP. Dans cette section, nous continuons à expliquer comment nous pouvons simuler quelques autres protocoles tels que le protocole de Percolate, le protocole d'Election Arborescent. Nos ambitions sont d'éclaircir la relation entre TRS et FTRS, et entre FTRS et EFTRS avec l'objectif de réaliser le premier pas de la transformation automatique.

Pour chaque problème, nous allons construire pas à pas des systèmes ré-écriture de la manière suivante :

1. **Construction d'un TRS classique**.
2. **Transformation du TRS vers un FTRS** (Section 4.1.1) :
   - Tout d'abord, chaque règle normale de TRS $\alpha \to \beta$ sera convertie vers une règle FTRS de type : $H(\alpha) \to \beta$ avec $\alpha, \beta \in \mathcal{T}(\mathcal{F}_{bin}, \mathcal{X})$.
   - Puis nous avons besoin d'un ensemble de règles de circulation pour que des règles $H(\alpha) \to \beta$ soient applicables :

$$H(a(x, y)) \to a(H(x), y),$$

$$H(a(x, y)) \to a(x, H(y))$$

   - Un ensemble de règles de point fixe est construit ci-dessous :

$$F(x) \to x,$$

$$F(x) \to F(H(x))$$

3. **Transformation du FTRS vers un EFTRS** : Pour chaque problème, après la transformation en général (Voir cf. la section 4.2.3), nous allons présenter dans cette section une solution simplifiée mais plus évidente en EFTRS.
4. **Optimisation** (Section 5.2) : Nous observons que les règles de circulation sont élémentaires à droite de type I, ainsi elles sont transformées grâce à la règle suivante :

$$H^\star(a(x, y)) \to a(H^\star(x), H^\star(y))$$

Il faut noter que les quatre systèmes de ré-écriture sont listés ci-dessus (Construction d'un TRS classique, Transformation du TRS vers un FTRS, Transformation du FTRS vers un EFTRS, et Optimisation) comme objectif de calculer l'ensemble des états accessibles du modèle. Nous discuterons comment nous pouvons extraire des propriétés intéressantes à partir de l'ensemble des états accessibles dans le chapitre suivant.

6.1.1 Protocole de Percolate

Un protocole de Percolate (présenté par [Kesten 1997]) contient un arbre de processus, chacun d'eux possède une étiquette locale ayant une valeur possible dans un

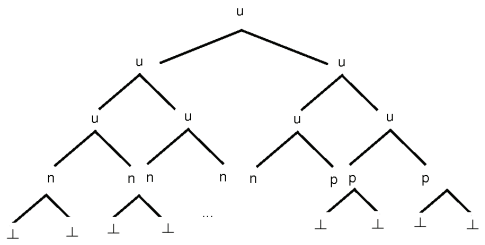


FIG. 6.1 – Ensemble initial E_0

ensemble d'étiquettes $\{p, n, u\}$. Les valeurs p (ou n) peuvent être interprétées comme *"positive"* (ou *"négative"*), la valeur u peut être interprétée comme *"indéfini"*, qui implique qu'à la fin, elle changera soit en p soit en n.

Au début, toutes les feuilles dans l'arbre sont présentées comme $p(\perp, \perp)$ ou $n(\perp, \perp)$, et les autres ont la valeur u. Le comportement de ce protocole est de passer à la racine de l'arbre la valeur p si au moins une des feuilles a la valeur p, ou n, si toutes les feuilles sont étiquetées par la valeur n. Si un processus n'a pas encore une valeur définie mais toutes les étiquettes de ses enfants sont définies, alors ce processus établit sa valeur à la disjonction des valeurs de ses enfants.

Par conséquent, nous pouvons représenter une configuration du protocole de Percolate comme un arbre sur $\mathcal{F}_{bin} = \{p, n, u, \perp\}$. Un ensemble initial $E_0 \subseteq \mathcal{T}(\mathcal{F}_{bin})$ est décrit dans la figure 6.1.

1. **Construction d'un TRS classique** : **TRS** $\mathcal{R}^{PP}$:

$$u(n(x,y), n(z,t)) \to n(n(x,y), n(z,t))$$
$$u(p(x,y), z) \to p(p(x,y), z)$$
$$u(x, p(y,z)) \to p(x, p(y,z))$$

2. **Transformation du TRS vers un FTRS** : **FTRS** correspondant $\mathcal{R}^{PP}_{\lambda}$:
   - *Règles générées (Règles ayant le côté gauche $H(u(...))$)*

$$H(u(p(x,y), z)) \to p(p(x,y), z) \qquad H(u(x, p(y,z))) \to p(x, p(y,z))$$
$$H(u(n(x,y), n(z,t))) \to n(n(x,y), n(z,t))$$

   - *Règles de circulation (Règles uniquement ayant le côté gauche $H(u(x,y))$. Les autres c.à.d $H(p(x,y))$ et $H(n(x,y))$ ne sont pas nécessaires à générer ici)*

$$H(u(x,y)) \to u(H(x), y)$$
$$H(u(x,y)) \to u(x, H(y))$$

   - *Règles de point fixe*

$$Percolate(x) \to x$$
$$Percolate(x) \to Percolate(H(x))$$

3. **Transformation du FTRS vers un EFTRS** : **EFTRS** correspondant $\mathcal{R}^{PP}_{\lambda^e}$:

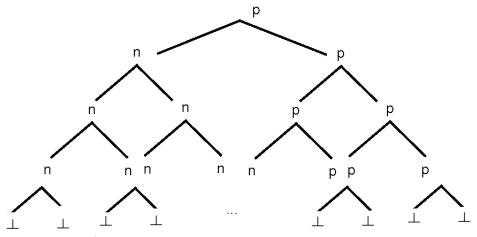


FIG. 6.2 – Terme accessible contient seulement des étiquettes n, p et $\perp$

– *Règles générées*

$$\begin{array}{ll} H(u(x,y)) \to n(Neg(x), Neg(y)) & Neg(n(x,y)) \to n(x,y) \\ H(u(x,y)) \to p(Pos(x), y) & Pos(p(x,y)) \to p(x,y) \\ H(u(x,y)) \to p(x, Pos(y)) & \end{array}$$

– *Règles de circulation (Elles sont identiques à celles de FTRS)*

$$\begin{array}{l} H(u(x,y)) \to u(H(x), y) \\ H(u(x,y)) \to u(x, H(y)) \end{array}$$

– *Règles de point fixe (Nous traduisons de celle de FTRS en traitant des cas possibles des symboles au-dessous de la racine)*

$$\begin{array}{ll} Percolate(u(x,y)) \to u(x,y) & Percolate(u(x,y)) \to Percolate(H(u(x,y))) \\ Percolate(p(x,y)) \to p(x,y) & Percolate(n(x,y)) \to n(x,y) \end{array}$$

4. **Optimisation** : Les règles de circulation sont optimisées comme la suite : $H^\star(a(x,y)) \to a(H^\star(x), H^\star(y))$.

TRS $\mathcal{R}^{PP}$, son FTRS $\mathcal{R}^{PP}_\lambda$ et son EFTRS $\mathcal{R}^{PP}_{\lambda^e}$ correspondant fournissent le même résultat $E_1 \subseteq \mathcal{T}(\mathcal{F}_{bin})$ à partir de l'ensemble initial E_0 :

$$E_1 = \mathcal{R}^{PP^\star}(E_0) = \mathcal{R}^{PP^\star}_\lambda(Percolate(E_0)) = \mathcal{R}^{PP^\star}_{\lambda^e}(Percolate(E_0)).$$

Résultats expérimentaux 2 *En utilisant notre outil (détaillé au chapitre 8), nous sommes capables de traiter le protocole PP ayant* 2^{500} *processus en* $0,396$ *seconde.*

Nous sommes intéressés uniquement à un ensemble de propriétés *définies*, appelé E'_1. En fait étant un sous ensemble de E_1, l'ensemble E'_1 contient seulement des étiquettes n, p et $\perp$ (Voir un tel terme dans la figure 6.2). Voir cf. des règles d'invariance en EFTRS dans le chapitre suivant.

6.1.2 Protocole d'Election Arborescent

Le protocole d'Election Arborescent ([Abdulla 2006]) est un protocole d'élection d'un leader. Il contient un ensemble de processus comme le protocole de Percolate, dénoté par les feuilles. Chacun d'eux décide d'abord s'il est candidat ou non. Le processus d'élection se déroule en deux étapes.

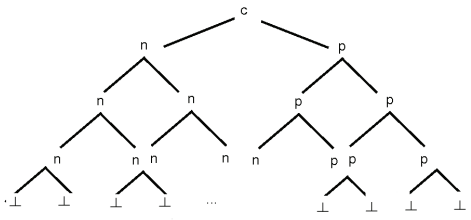


FIG. 6.3 – Ensemble initial $E"_1$ de la deuxième étape

La première étape se compose des noeuds intérieurs faisant une requête à leur noeuds d'enfant pour savoir si au moins un d'eux est candidaté. Dans un tel cas, le noeud intérieur devient un candidat et sera étiqueté p, sinon il sera étiqueté n.

Similairement au le protocole Percolate, au début, toutes les feuilles dans l'arbre sont présentées comme $p(\bot, \bot)$ ou $n(\bot, \bot)$, les autres ont la valeur u. L'objectif de la première étape est de propager à la racine de l'arbre la valeur p si au moins une des feuilles est étiquetée par p, et n, si toutes les feuilles sont étiquetées par n. Si un processus n'a pas encore une valeur définie que toutes les étiquettes de ses enfants sont définies, alors ce processus établit sa valeur à la disjonction des valeurs de ses enfants.

La deuxième étape est la procédure d'élection réelle. Dans le cas où la racine a la valeur n, nous n'avons pas de candidat et donc impossible de continuer l'élection. Dans le cas contraire, il y a au moins un candidat.

A partir de l'ensemble de termes *définis* trouvé après la première étape (Voir la figure 6.2), nous préparons pour la deuxième étape en marquant la racine par une étiquette c (Voir $E_1^"$ dans la figure 6.3).

Ainsi, la racine choisit un candidat en faisant un choix non-déterministe parmi ses enfants étiquetés par p. Un noeud intérieur qui était sélectionné (noté par une étiquette c), choisit un de ses enfants qui a déclaré lui-même un candidat.

1. **Construction d'un TRS classique** : **TRS** $\mathcal{R}^{LEP}$:

$$\begin{array}{ll} c(p(x,y),p(z,t)) \to c(c(x,y),p(z,t)) & c(p(x,y),p(z,t)) \to c(p(x,y),c(z,t)) \\ c(p(x,y),n(z,t)) \to c(c(x,y),n(z,t)) & c(n(x,y),p(z,t)) \to c(n(x,y),c(z,t)) \end{array}$$

2. **Transformation du TRS vers un FTRS** : **FTRS** correspondant $\mathcal{R}_\lambda^{LEP}$:
   - *Règles générées*

$$\begin{array}{ll} H(c(p(x,y),p(z,t))) \to c(c(x,y),p(z,t)) & H(c(p(x,y),p(z,t))) \to c(p(x,y),c(z,t)) \\ H(c(p(x,y),n(z,t))) \to c(c(x,y),n(z,t)) & H(c(n(x,y),p(z,t))) \to c(n(x,y),c(z,t)) \end{array}$$

   - *Règles de circulation (Règles uniquement ayant le côté gauche $H(c(x,y))$. Les autres c.à.d $H(p(x,y))$ et $H(n(x,y))$ ne sont pas nécessaires à générer ici)*

$$\begin{array}{l} H(c(x,y)) \to c(H(x),y) \\ H(c(x,y)) \to c(x,H(y)) \end{array}$$

   - *Règles de point fixe*

$$\begin{array}{l} Election(x) \to x \\ Election(x) \to Election(H(x)) \end{array}$$

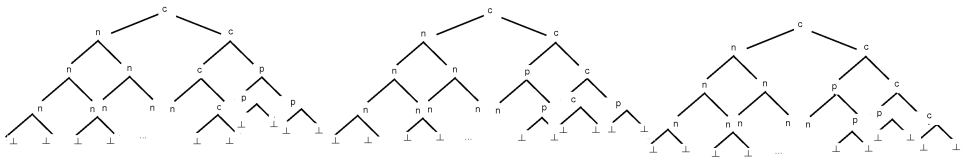


FIG. 6.4 – Termes ayant un chemin de c

3. **Transformation du FTRS vers un EFTRS** : **EFTRS** correspondant $\mathcal{R}_{\lambda^e}^{LEP}$:

   – *Règles générées*

$$\begin{array}{ll} H(c(x,y)) \to c(Elec(x), Neg(y)) & H(c(x,y)) \to c(Elec(x), Pos(y)) \\ H(c(x,y)) \to c(Pos(x), Elec(y)) & H(c(x,y)) \to c(Neg(x), Elec(y)) \\ Elec(p(x,y)) \to c(x,y) & Neg(n(x,y)) \to n(x,y) \\ Pos(p(x,y)) \to p(x,y) & Elec(\bot) \to \bot \\ Pos(\bot) \to \bot & Neg(\bot) \to \bot \end{array}$$

   – *Règles de circulation (Elles sont identiques à celles du FTRS)*

$$\begin{array}{l} H(c(x,y)) \to c(H(x), y) \\ H(c(x,y)) \to c(x, H(y)) \end{array}$$

   – *Règles de point fixe (Nous traduisons de celle du FTRS en traitant des cas possibles des symboles au-dessous de la racine)*

$$\begin{array}{ll} Election(c(x,y)) \to c(x,y) & Election(c(x,y)) \to Election(H(c(x,y))) \\ Election(b(x,y)) \to b(x,y) & Election(b(x,y)) \to Election(H(b(x,y))) \end{array}$$

4. **Optimisation** : Les règles de circulation sont optimisées comme la suite : $H^\star(a(x,y)) \to a(H^\star(x), H^\star(y))$.

Le TRS $\mathcal{R}^{LEP}$, son FTRS $\mathcal{R}_{\lambda}^{LEP}$ et son EFTRS $\mathcal{R}_{\lambda^e}^{LEP}$ correspondant fournissent le même résultat E_2 à partir de l'ensemble initial $E_1^"$:

$$E_2 = \mathcal{R}^{LEP^\star}(E_1^") = \mathcal{R}_{\lambda}^{LEP^\star}(Election(E_1^")) = \mathcal{R}_{\lambda^e}^{LEP^\star}(Election(E_1^")).$$

Résultats expérimentaux 3 *En utilisant notre outil (détaillé au chapitre 8), nous sommes capables de traiter le protocole LEP ayant* 2^{400} *processus en* $138,699$ *secondes.*

A partir du résultat de l'ensemble E_2, nous sommes intéressés par l'ensemble de propriétés ayant un chemin contenant uniquement c de la racine à la feuille (Voir la figure 6.4). Voir cf. des règles d'invariance en EFTRS dans le chapitre suivant.

6.2 Réseaux de Petri P/T (Place/Transition)

Dans cette section, nous présentons quelques codages (soit de type classique soit de type hiérarchique) pour simuler un réseau de Petri borné élémentaire. Nous montrons également comment nous pouvons calculer des successeurs d'un marquage donné ainsi que l'ensemble de marquages accessibles à partir d'un marquage initial donné par nos systèmes.

Tout d'abord, nous devons évoquer le codage d'un entier pour simuler un nombre de jeton dans une place d'un réseau de Petri.

Remarques 8 *En fait, dans les TRSs, on utilise les termes tels que* 0 *pour zéro,* $s(\perp, 0)$ *pour* 1, $s(\perp, s(\perp, 0))$ *pour* 2, *et ainsi de suite. De cette manière, on peut simuler un ensemble infini d'entiers. Cependant, nous préférons utiliser des termes dans* $\mathcal{F}_{bin}$ *qui contient un seul symbole de constante* $\perp$ *avec des symboles binaires tels que* $zero, one, two, three....$

En détail, les entiers $0, 1, 2...$ sont représentés par $zero(\perp, \perp), one(\perp, \perp)$, etc.

Pour plus de simplicité, nous supposons que tous les codages en EFTRS peuvent désormais avoir plusieurs constantes. Donc une règle $TestChange(zero(x, y)) \rightarrow une(x, y)$ peut être représentée par $TestChange(0) \rightarrow 1$.

Ainsi, d'une part, pour modifier le nombre de jetons dans une place, nous utilisons deux règles primitives $Minus, Plus$ listées ci-dessous :

- Pour des réseaux de Petri **sains** :

$$Plus(0) \rightarrow 1$$
$$Minus(1) \rightarrow 0$$

- Pour des réseaux de Petri **non sains** :

$$Plus(1) \rightarrow 2...$$
$$Minus(2) \rightarrow 1...$$

D'autre part, pour seulement tester un entier, nous utilisons souvent deux règles de $Test$ listées ci-dessous :

- Pour des réseaux de Petri **sains** :

$$Test_0(0) \rightarrow 0$$
$$Test_1(1) \rightarrow 1$$

- Pour des réseaux de Petri **non sains** :

$$Test_2(2) \rightarrow 2$$
$$Test_3(3) \rightarrow 3$$

Nous allons présenter trois codages d'EFTRS pour les réseaux de Petri bornés élémentaires $R =< S, T, Pre, Post, m_0 >$. L'idée principale est

1. de coder un marquage en un terme dans $\mathcal{T}(\mathcal{F}_{bin})$. Au début, le marquage initial m_0 est représenté par un terme initial t_{m_0}
2. ensuite de construire un ensemble de règles $\mathcal{R}^t_{\lambda^e}$ pour chaque transition $t \in T$. Ainsi, un franchi des transitions du marquage m à un autre m' est modélisé par la ré-écriture du t_m au $t_{m'}$ en utilisant des règles $\mathcal{R}^t_{\lambda^e}$.

Algorithme 8 Calcul de marquages accessibles $Reach_{\mathcal{R}^T_{\lambda^e}}(t_{m_0})$

Précondition : Un terme initial $t_{m_0} \in \mathcal{T}(\mathcal{F}_{bin})$,
$\mathcal{R}^T_{\lambda^e}$ un ensemble de règles d'EFTRSs tel que $\mathcal{R}^T_{\lambda^e} = \bigcup_{t_i \in T} \mathcal{R}^{t_i}_{\lambda^e}$

1: $res_1 \leftarrow \emptyset$;
2: $res_2 \leftarrow t_{m_0}$;
3: **tantque** $res_1 \neq res_2$ **faire**
4: $res_1 \leftarrow res_2$;
5: **pour** $\alpha \in res_2$ **faire**
6: $res_2 + \leftarrow (\mathcal{R}^T_{\lambda^e})^\star(\{Trans_{t_i}(\alpha) | t_i \in T\})$
7: **fin pour**
8: **fin tanque**
9: **retourner** res_2

Enfin, nous calculons l'ensemble de marquages accessibles $Reach[R, m_0]$ par EFTRS comme dans l'algorithme 8.

Nous commençons par un codage naïf (Codage 1) et ensuite deux autres (Codages 2 et 3) en partageant des données intermédiaires et en profitant le calcul basé sur le cache de mieux en mieux.

Codage 1

Considérons S, l'ensemble de N places d'un réseau de Petri. Pour plus de lisibilité et sans perte de généralité, nous supposons qu'elles sont nommées $p_0, p_1, ..., p_{N-1}$.

Définition 13 *Codage 1.*

Soit un réseau de Petri borné élémentaire $< S, T, Pre, Post, m_0 >$.

Construisons un terme t_{m_0} *dans* $\mathcal{T}(\mathcal{F}_{bin})$ *comme une liste des places et leurs marquages en conservant le nom de chaque place*

$$t_{m_0} = p_0(v_0, p_1(v_1, ...p_{N-1}(v_{N-1}, \perp)))$$

tel que pour tout $p_i \in S, i = 0..N-1$ *:*

- $v_i = 1$ *si* $p_i \in m_0$
- $v_i = 0$ *sinon.*

Construisons un ensemble de règles de ré-écriture $\mathcal{R}^t_{\lambda^e}$ *pour chaque transition* $t \in T$. *Nous considérons quatre cas de* p *:*

- $p \notin Pre(t)$ *et* $p \notin Post(t)$ *:*

$$Trans_t(p(x, y)) \rightarrow p(x, Trans_t(y)) \tag{6.1}$$

- $p \notin Pre(t)$ *et* $p \in Post(t)$ *:*

$$Trans_t(p(x, y)) \rightarrow p(Plus(x), Trans_t(y)) \tag{6.2}$$

- $p \in Pre(t)$ *et* $p \notin Post(t)$ *:*

$$Trans_t(p(x, y)) \rightarrow p(Minus(x), Trans_t(y)) \tag{6.3}$$

– $p \in Pre(t)$ *et* $p \in Post(t)$ *: Nous vérifions que le marquage est supérieure à zéro.*

$$Trans_t(p(x,y)) \rightarrow p(Test_1(x), Trans_t(y)) \tag{6.4}$$
$$Trans_t(p(x,y)) \rightarrow p(Test_2(x), Trans_t(y)) \tag{6.5}$$
$$... \tag{6.6}$$

Nous avons également besoin une règle pour la terminaison de la liste de places : $Trans_t(\perp) \rightarrow \perp$*. Ainsi, le changement d'état de* m_0 *à* m_1 *en utilisant la transition* t *est modélisé par*

$$t_{m_1} \in (\mathcal{R}^t_{\lambda^e})^\star(\{Trans_t(t_{m_0})\})$$

Nous allons illustrer ce codage par un exemple.

Exemple 24 *Codage 1. Soit un réseau de Petri borné du Problème à 2 philosophes. Soit* S *un ensemble de 12 places :*

$$S = \{Idle_0, WaitLeft_0, HasLeft_0, WaitRight_0, HasRight_0, Fork_0,$$
$$Idle_1, WaitLeft_1, HasLeft_1, WaitRight_1, HasRight_1, Fork_1\}$$

Soit $m_0 = \{Idle_0, Fork_0, Idle_1, Fork_1\}$

Un terme t_{m_0} *est donc représenté comme suit (Voir la figure 6.5.a) :*

$$\mathbf{idle_0}(\mathbf{1}, waitleft_0(0, hasleft_0(0, waitright_0(0, hasright_0(0, \mathbf{fork_0}(\mathbf{1},$$
$$\mathbf{idle_1}(\mathbf{1}, waitleft_1(0, hasleft_1(0, waitright_1(0, hasright_1(0, \mathbf{fork_1}(\mathbf{1}, \perp))))))))))$$

Soit une transition $GoEat_0$ *telle que* $Pre(GoEat_0) = \{Idle_0\}$ *et* $Post(GoEat_0) = \{WaitLeft_0, WaitRight_0\}$*. Et une autre* $GoLeft_0$ *telle que* $Pre(GoLeft_0) = \{Fork_1, WaitLeft_0\}$ *et* $Post(GoLeft_0) = \{HasLeft_0\}$*.*

Elles vont être produites par les règles de ré-écriture 6.1 à 6.5 ci-dessus. Alors $m_1 = m_0 - Pre(GoEat_0) + Post(GoEat_0)$ *sera calculé par un pas de ré-écriture :*

$$t_{m_1} = idle_0(0, \mathbf{waitleft_0}(\mathbf{1}, hasleft_0(0, \mathbf{waitright_0}(\mathbf{1}, hasright_0(0, \mathbf{fork_0}(\mathbf{1},$$
$$\mathbf{idle_1}(\mathbf{1}, waitleft_1(0, hasleft_1(0, waitright_1(0, hasright_1(0, \mathbf{fork_1}(\mathbf{1}, \perp))))))))))$$

Alors $m_2 = m_1 - Pre(GoLeft_0) + Post(GoLeft_0)$ *sera calculé par un pas de ré-écriture :*

$$t_{m_2} = idle_0(0, waitleft_0(0, \mathbf{hasleft_0}(\mathbf{1}, \mathbf{waitright_0}(\mathbf{1}, hasright_0(0, \mathbf{fork_0}(\mathbf{1},$$
$$\mathbf{idle_1}(\mathbf{1}, waitleft_1(0, hasleft_1(0, waitright_1(0, hasright_1(0, fork_1(0, \perp))))))))))$$

Résultats expérimentaux 4 *En utilisant notre outil (détaillé au chapitre 8), ce codage est capable de traiter le problème ayant 8 philosophes en* $2.666,321$ *secondes.*

Points Forts du codage de réseau de Petri 1 *Simple, facile à concevoir.*

Points Faibles du codage de réseau de Petri 1 *Nous observons que cette structure de données n'est pas compacte, l'avantage des EFTRSs n'est donc pas utilisé.*

C'est la raison pour laquelle, nous présentons le codage suivant, codage 2.

Codage 2

Au début, nous supposons que toutes les places dans S sont nommées de p_0 à p_{N-1} comme dans le Codage 1. Tout d'abord, nous discutons des règles primitives du codage 2 $Post_p$ et Pre_p pour une place p. $Post_p$ nous permet d'ajouter un jeton dans la place p quand Pre_p permet de retirer un jetons de la place p. Les deux peuvent également déplacer à la place suivante si la place courante n'est pas p.

$$Post_p(p(x,y)) \rightarrow p(Plus(x),y)$$
$$Pre_p(p(x,y)) \rightarrow p(Minus(x),y)$$

Définition 14 *Codage 2.*

Soit un réseau de Petri borné élémentaire $< S, T, Pre, Post, m_0 >$.

Construisons un terme t_{m_0} *dans* $\mathcal{T}(\mathcal{F}_{bin})$ *comme une liste des places et leurs marquages en conservant le nom de chaque place*

$$t_{m_0} = p_0(v_0, p_1(v_1, ...p_{N-1}(v_{N-1}, \perp)))$$

tel que pour tout $p_i \in S, i = 0..N-1$ *:*

- $v_i = 1$ *si* $p_i \in m_0$
- $v_i = 0$ *sinon.*

Construisons un ensemble de règles de ré-écriture pour chaque transition $t \in T$ *en commençant par une règle de composition principale de la forme*

$$Trans_t(x) \rightarrow Post_0(Pre_0(x))$$

avec Pre_0 *une règle de composition de source* $Pre(t)$ *et* $Post_0$ *celle de destination* $Post(t)$.

- *Chaque* $p \in Pre(t)$ *est interprétée par*
  $Pre_i(x) \rightarrow Pre_p(Pre_{i+1}(x)), 0 \leq i <\| Pre(t) \|$ *et* $Pre_{\|Pre(t)\|}(x) \rightarrow x$.
- *Chaque* $p \in Post(t)$ *est interprétée par*
  $Post_i(x) \rightarrow Post_p(Post_{i+1}(x)), 0 \leq i <\| Post(t) \|$ *et* $Post_{\|Post(t)\|}(x) \rightarrow x$

Des règles primitives $Post_p$ *et* Pre_p *pour une place p.* $Post_p$ *nous permet d'ajouter un jeton dans la place p quand* Pre_p *permet de retirer un jeton de la place p. Les deux peuvent également sauter à la place suivante si la place actuelle n'est pas p.*

- *Chaque place p :* $Pre_p(p(x,y)) \rightarrow p(Minus(x),y)$.
- *Pour tous les* $p' \neq p$ *:* $Pre_p(p'(x,y)) \rightarrow p'(x, Pre_p(y))$.
- *Chaque place p :* $Post_p(p(x,y)) \rightarrow p(Plus(x),y)$.

– *Pour tous les* $p' \neq p$: $Post_p(p'(x,y)) \rightarrow p'(x, Post_p(y))$.

Ainsi, le changement d'état de m_0 *à* m_1 *utilisant la transition* t *est modélisé par*

$$t_{m_1} \in (\mathcal{R}^t_{\lambda^e})^\star(\{Trans_t(t_{m_0})\})$$

Il faut noter que tout est généré et appliqué à partir de la racine. Nous allons illustrer ce codage par un exemple.

Exemple 25 *Codage 2. Soit le réseau de Petri de l'exemple 24. Soit* S *un ensemble de 12 places :*

$$S = \{Idle_0, WaitLeft_0, HasLeft_0, WaitRight_0, HasRight_0, Fork_0,$$
$$Idle_1, WaitLeft_1, HasLeft_1, WaitRight_1, HasRight_1, Fork_1\}$$

Soit $m_0 = \{Idle_0, Fork_0, Idle_1, Fork_1\}$ *Un terme* t_{m_0} *est donc représenté comme le suivant (Voir la figure 6.5.a) :*

$$\mathbf{idle_0}(\mathbf{1}, waitleft_0(0, hasleft_0(0, waitright_0(0, hasright_0(0, \mathbf{fork_0}(\mathbf{1},$$
$$\mathbf{idle_1}(\mathbf{1}, waitleft_1(0, hasleft_1(0, waitright_1(0, hasright_1(0, \mathbf{fork_1}(\mathbf{1}, \bot)))))))))))$$

Soit une transition $GoEat_0$ *telle que* $Pre(GoEat_0) = \{Idle_0\}$ *et* $Post(GoEat_0) = \{WaitLeft_0, WaitRight_0\}$. *Alors* $m_1 = m_0 - Pre(GoEat_0) + Post(GoEat_0)$ *sera calculé par un pas de ré-écriture avec une règle de composition de source* $Pre(GoEat_0)$, *c.à.d. Elle va être produite par les règles de ré-écriture indiquées ci-dessous.*

$$\mathcal{R}^{Pre(GoEat_0)}_{\lambda^e} = \left\{ \begin{array}{l} GoEat_0 Pre_0(x) \rightarrow Pre_{Idle_0}(x) \\ Pre_{Idle_0}(p(x,y)) \rightarrow p(x, Pre_{Idle_0}(y)) \forall p \in S, p \neq idle_0 \\ Pre_{Idle_0}(idle_0(x,y)) \rightarrow idle_0(Minus(x), y) \end{array} \right\}.$$

et celle de destination $Post(GoEat_0)$, *c.à.d.*

$$\mathcal{R}^{Post(GoEat_0)}_{\lambda^e} = \left\{ \begin{array}{l} GoEat_0 Post_0(x) \rightarrow Post_{WaitLeft_0}(Post_{WaitRight_0}(x)) \\ Post_{WaitLeft_0}(p(x,y)) \rightarrow p(x, Post_{WaitLeft_0}(y)) \ \forall p \in S, p \neq waitleft_0 \\ Post_{WaitLeft_0}(waitleft_0(x,y)) \rightarrow waitleft_0(Plus(x), y) \\ Post_{WaitRight_0}(p(x,y)) \rightarrow p(x, Post_{WaitRight_0}(y)) \ \forall p \neq waitright_0 \\ Post_{WaitRight_0}(waitright_0(x,y)) \rightarrow waitright_0(Plus(x), y) \end{array} \right\}.$$

Ainsi, nous obtenons

$$t_{m_1} = idle_0(0, \mathbf{waitleft_0}(\mathbf{1}, hasleft_0(0, \mathbf{waitright_0}(\mathbf{1}, hasright_0(0, \mathbf{fork_0}(\mathbf{1},$$
$$\mathbf{idle_1}(\mathbf{1}, waitleft_1(0, hasleft_1(0, waitright_1(0, hasright_1(0, \mathbf{fork_1}(\mathbf{1}, \bot)))))))))))$$

Et une autre $GoLeft_0$ *telle que* $Pre(GoLeft_0) = \{Fork_1, WaitLeft_0\}$ *et* $Post(GoLeft_0) = \{HasLeft_0\}$. *Alors* $m_2 = m_1 - Pre(GoLeft_0) + Post(GoLeft_0)$ *sera calculé par un pas de ré-écriture avec une règle de composition de source* $Pre(GoLeft_0)$, *c.à.d. Elle va être produite par les règles ré-écriture indiquées ci-dessous.*

$$\mathcal{R}^{Pre(GoLeft_0)}_{\lambda^e} = \left\{ \begin{array}{l} GoLeft_0 Pre_0(x) \rightarrow Pre_{Fork_1}(Pre_{WaitLeft_0}(x)) \\ \textit{et ainsi de suite.} \end{array} \right\}.$$

et celle de destination $Post(GoLeft_0)$, *c.à.d.*

$$\mathcal{R}^{Post(GoLeft_0)}_{\lambda^e} = \left\{ \begin{array}{l} GoLeft_0 Post_0(x) \rightarrow Post_{HasLeft_0}(x) \\ \textit{et ainsi de suite.} \end{array} \right\}.$$

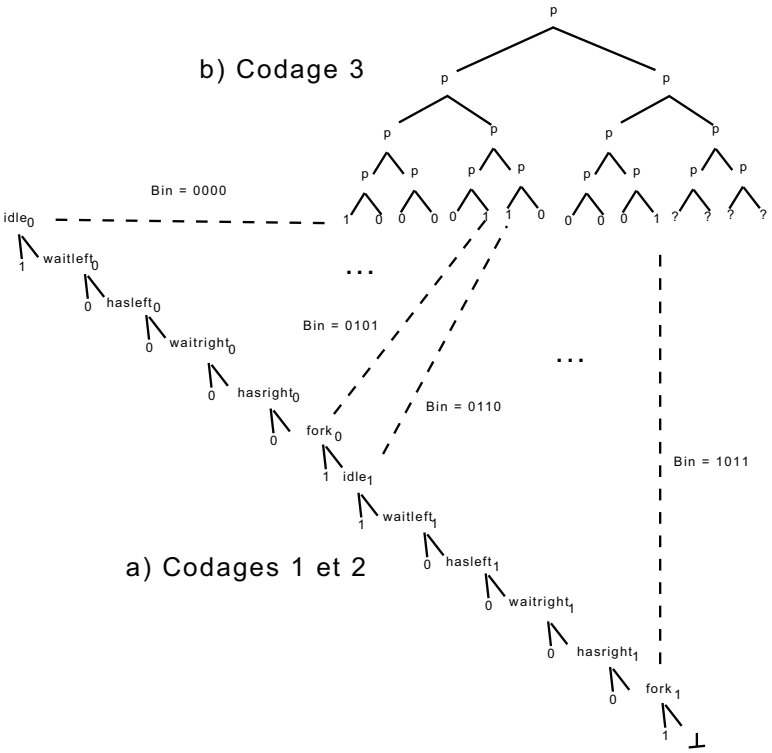


FIG. 6.5 – Codage d'un réseau de Petri à 2 philosophes en $\mathcal{T}(\mathcal{F}_{bin})$

Finalement, nous obtenons

$$t_{m_2} = idle_0(0, waitleft_0(0, \mathbf{hasleft_0}(\mathbf{1}, \mathbf{waitright_0}(\mathbf{1}, hasright_0(0, \mathbf{fork_0}(\mathbf{1},$$

$$\mathbf{idle_1}(\mathbf{1}, waitleft_1(0, hasleft_1(0, waitright_1(0, hasright_1(0, fork_1(0, \bot)))))))))))$$

Résultats expérimentaux 5 *En utilisant notre outil (détaillé au chapitre 8), ce codage est capable de traiter le problème ayant 4 philosophes en* $63,632$ *secondes.*

Points Forts du codage de réseau de Petri 2 *Nous observons que le partage des règles pour les opérations primitives des places est bien organisé dans ce codage. Grâce à la mise en cache des règles Pre et Post, nous profitons bien de l'avantage des EFTRSs.*

Points Faibles du codage de réseau de Petri 2 *Nous observons également que le partage des règles pour les transitions n'existe pas dans ce codage.*

Nous présentons le codage suivant, le codage 3. Dans ce codage, le partage des règles pour les transitions est organisé mieux que le codage précédent.

Codage 3

Au début, supposons que toutes les places dans S vont de p_0 à p_{N-1} comme dans le codage 1 et 2. L'indexe de chaque place est utilisée pour déterminer une unique position (une feuille) dans l'arbre. Nous définissons $Bin(p)|k$ comme le chiffre à la position k de $Bin(p)$. Par exemple, avec $Bin(p) = 01..0$, nous avons $Bin(p)|0 = 0$, $Bin(p)|1 = 1$ et $Bin(p)|(h-1) = 0$. Par conséquent, la feuille 0 est réservée pour la place p_0 ($Bin(0) = 00..0$),... quand la feuille $N-1$ (avec $Bin(N-1) = 11..1$) est réservée pour la place p_{N-1}. Nous n'avons donc plus besoin de conserver le nom de chaque place, ce qui réduit la taille du codage.

Définition 15 *Codage 3.*

Soit un réseau de Petri borné élémentaire $< S, T, Pre, Post, m_0 >$. *Construisons un terme* t *dans* $\mathcal{T}(\mathcal{F}_{bin})$ *de hauteur* $h = log_2(N)$ *tel que pour chaque* i *: si* $marquage = p_i$, *alors* $i \in \mathcal{P}os_{\mathcal{F}}(marquage)$, *avec* $marquage \in \{0, 1\}$ *et* $0 \leq i < N$.

Chaque transition $t \in T$ *doit être produite par les règles de ré-écriture commençant par une règle principale comme dans le codage 2 :*

$$Trans_t(x) \rightarrow Post_0(Pre_0(x))$$

telle que Pre_0 *pour l'ensemble de source* $Pre(t)$ *et* $Post_0$ *pour l'ensemble de destination* $Post(t)$.

- *Chaque* $p \in Pre(t)$ *est interprétée par*
  $Pre_i(x) \rightarrow Pre^h_{Bin(p)}(Pre_{i+1}(x)), 0 \leq i <\| Pre(t) \|$.
- *Chaque* $p \in Post(t)$ *est interprétée par*
  $Post_i(x) \rightarrow Post^h_{Bin(p)}(Post_{i+1}(x)), 0 \leq i <\| Post(t) \|$.

Des règles de circulation sont générées et appliquées à partir de la racine :

- *Chaque* Pre^h, *pour* $0 < k \leq h$
  - $Pre^k_{Bin(p)}(p(x, y) \rightarrow p(Pre^{k-1}_{Bin(p)}(x), y)$ *si* $Bin(p)|k = 0$
  - $Pre^k_{Bin(p)}(p(x, y) \rightarrow p(x, Pre^{k-1}_{Bin(p)}(y))$ *sinon.*

  Enfin, quand nous arrivons à la position de la place indiquée :
  $Pre_0(p(x, y)) \rightarrow p(Minus(x), y)$
  $Pre_1(p(x, y)) \rightarrow p(x, Minus(y))$
- *Chaque* $Post^h$, *pour* $0 < k \leq h$
  - $Post^k_{Bin(p)}(p(x, y) \rightarrow p(Post^{k-1}_{Bin(p)}(x), y)$ *si* $Bin(p)|k = 0$
  - $Post^k_{Bin(p)}(p(x, y) \rightarrow p(x, Post^{k-1}_{Bin(p)}(y))$ *sinon.*

  Enfin, les opérations pour ajouter ou retirer un jeton d'une place quand nous arrivons à la position indiquée sont :
  $Post_0(p(x, y)) \rightarrow p(Plus(x), y)$
  $Post_1(p(x, y)) \rightarrow p(x, Plus(y))$

Ainsi, le changement d'état de m_0 *à* m_1 *utilisant la transition* t *est modélisé par*

$$t_{m_1} \in (\mathcal{R}^t_{\lambda^e})^\star(\{Trans_t(t_{m_0})\})$$

Des règles de circulation exigent seulement de descendre jusqu'à la feuille correspondante de la place indiquée et modifier son marquage. De cette manière, nous pouvons partager des règles primitives.

Nous allons illustrer ce codage par un exemple.

Exemple 26 *Codage 3. Soit le réseau de Petri de l'exemple 24. Soit S un ensemble de 12 places :*

$$S = \{Idle_0, WaitLeft_0, HasLeft_0, WaitRight_0, HasRight_0, Fork_0,$$

$$Idle_1, WaitLeft_1, HasLeft_1, WaitRight_1, HasRight_1, Fork_1\}$$

Nous avons besoin d'un arbre de hauteur $h = 4$ pour coder l'ensemble S, c.à.d. nous utilisons 12 premières feuilles de cet arbre qui sont réservées pour les 12 places de S, donc les 4 dernières ne sont pas utilisés. Soit $m_0 = \{Idle_0, Fork_0, Idle_1, Fork_1\}$. Un terme t_{m_0} est donc représenté comme en figure 6.5.b.

Soit une transition $GoEat_0$ telle que $Pre(GoEat_0) = \{Idle_0\}$ et $Post(GoEat_0) = \{WaitLeft_0, WaitRight_0\}$. Alors $m_1 = m_0 - Pre(GoEat_0) + Post(GoEat_0)$ sera calculé par un pas de ré-écriture avec une règle de composition de source $Pre(GoEat_0)$, c.à.d. Elle va être produite par les règles de ré-écriture indiquées ci-dessous :

$$\mathcal{R}_{\lambda^e}{}^{Pre(GoEat_0)} = \left\{ \begin{array}{l} GoEat_0 Pre_0(x) \rightarrow Pre_{0000}(x) \\ Pre_{0000}(p(x,y)) \rightarrow p(Pre_{000}(x), y) \\ Pre_{000}(p(x,y)) \rightarrow p(Pre_{00}(x), y) \\ Pre_{00}(p(x,y)) \rightarrow p(Pre_0(x), y) \\ Pre_0(p(x,y)) \rightarrow p(Minus(x), y) \end{array} \right\}.$$

et celles de destination $Post(GoEat_0)$, c.à.d.

$$\mathcal{R}_{\lambda^e}{}^{Post(GoEat_0)} = \left\{ \begin{array}{l} GoEat_0 Post_0(x) \rightarrow Post_{0001}(Post_{0011}(x)) \\ \\ Post_{0001}(p(x,y)) \rightarrow p(Post_{001}(x), y) \\ Post_{001}(p(x,y)) \rightarrow p(Post_{01}(x), y) \\ Post_{01}(p(x,y)) \rightarrow p(Post_1(x), y) \\ \\ Post_{0011}(p(x,y)) \rightarrow p(Post_{011}(x), y) \\ Post_{011}(p(x,y)) \rightarrow p(Post_{11}(x), y) \\ Post_{11}(p(x,y)) \rightarrow p(x, Post_1(y)) \\ \\ \end{array} \right\}.$$

La règle $Post_1(p(x,y)) \rightarrow p(x, Plus(y))$ avec les règles primitives sont partagées.

Résultats expérimentaux 6 *En utilisant notre outil (détaillé au chapitre 8), ce codage est capable de traiter le problème ayant 8 philosophes en $454,978$ secondes.*

Points Forts du codage de réseau de Petri 3 *Dans ce codage, nous ne distinguons pas le nom des places, nous n'avons besoin que d'un symbole p à la position du nom de toutes les places. Cela réduit le nombre de règles de ré-écriture pour la traiter. Autrement dit, nous réduisons la taille du codage. Ainsi, nous avons besoin d'un arbre ayant une hauteur $log_2(N)$ pour coder la liste des places numérotées.*

Points Faibles du codage de réseau de Petri 3 *Comme mentionné ci-dessus, pour coder un ensemble de places S, nous avons besoin d'un arbre de hauteur $h = log_2(\parallel S \parallel)$. Quand $N = \parallel S \parallel$ n'est pas un entier de la forme 2^n, nous risquons d'avoir un grand nombre de feuilles non utilisées, qui réduisent la performance du codage. Une solution raisonnable est de décomposer N en somme de puissances de 2. Ensuite, nous construisons un codage de réseau de Petri 3 pour chacun. Enfin, nous agrégeons ces codages.*

D'autre part, soit un codage constitué de composants (appelés les sous-réseaux) identiques. Effectivement, les sous-réseaux identiques sont souvent codés dans les endroits totalement différents du terme. Ce codage ne profite donc pas de l'avantage des EFTRSs dans ce cas-là.

Proposition 7 *Soit un réseau de Petri borné élémentaire $R =< S, T, Pre, Post, m_0 >$. Il existe des codages de réseaux de Petri 1, 2 et 3 tels que les marquages accessibles de ce réseau sont calculés par la clôture transitive $\mathcal{R}^{\star}_{\lambda^e}$ sur t_{m_0} de ces codages.*

Preuve 11 (Preuve succincte)
Au moment donné, nous pouvons prouver qu'une règle en $\mathcal{R}_{\lambda^e}$ est applicable sur t_m ssi la transition correspondante en T est franchissable sur m. Effectivement, nous pouvons prouver qu'il existe une fonction f de la clôture transitive $\mathcal{R}^{\star}_{\lambda^e}$ sur t_{m_0} à l'ensemble des marquages accessibles de ce réseau Petri.

Nous allons étudier des sous-réseaux identiques pour améliorer la capacité de modéliser des réseaux de Petri dans la section suivante.

6.3 Réseaux de Petri hiérarchiques

Un réseau de Petri hiérarchique [Miner 1999] peut être défini par une composition des réseaux de Petri bornés élémentaires ordinaires (appelé sous réseaux dans cette section).

Définition 16 *Réseaux de Petri hiérarchiques. Un réseau de Petri hiérarchique est défini comme un quadruplet $< 2^R, T_{sync}, Pre_{sync}, Post_{sync} >$ où*

- *2^R un ensemble de sous réseaux de Petri $R =< S_{local}, T_{local}, Pre_{local}, Post_{local}, m_{0local} >$.*
- *T_{sync} est un ensemble fini de transitions.*
- *Pre_{sync} et $Post_{sync}$ sont deux applications de T_{sync} dans 2^R qui à toute transition t_{sync} de T_{sync} associent les deux états $Pre(t_{sync})$ et $Post(t_{sync})$ qui sont respectivement l'origine et le but de la transition t_{sync}.*

Il faut noter que les sous réseaux peuvent être identiques. Par conséquent, le comportement des sous réseaux sont naturellement similaires. Nous nous intéressons aux codages ayant uniquement des sous réseaux identiques tels que le problème des Philosophes ou le protocole de Slotted-Ring. D'autre part, nous étudions également

le protocole de Round-Robin Mutex qui contient un ensemble de processus identiques et une ressource partagée. Cette ressource peut être considérée comme un sous réseau différent des autres.

Construction des termes

Dans la première étape, construisons un terme pour coder le niveau de sous-réseau identique : $local(u, v)$ avec u et v sont les codages du codage identique.

Dans la deuxième étape, construisons un codage identique niveau 1 en composant deux sous-réseaux identiques tels que

$$sys(local(u, v), local(u, v))$$

A partir des codage identique de niveau 1, nous pouvons construire le codage identique de niveau 2 de la même manière, et ainsi de suite.

Quand il existe un sous réseau $local'(u', v')$ différent aux autres (c.à.d.$local(u, v)$), nous organisons comme le suivant :

$$globalsys(sys'(u', v'), system)$$

avec le côté droit *system* est le codage identique de niveau n.

Remarques 9 *Grâce à la symétrie du modèle, chaque terme peut être construit en dépliant des sous-termes identiques de niveau plus bas.*

Maintenant il nous reste à construire des règles pour qu'elles puissent être appliquées sur tout le système construit par le pliage récursif.

Construction des règles de ré-écriture

Maintenant considérons une transition effectuée seulement pour les places locales, nous créons un ensemble de règles de circulation qui descend de la racine jusqu'au niveau de chaque sous-réseau. A ce niveau là, nous ré-écrivons :

$$Local(local(x, y)) \rightarrow \gamma \tag{6.7}$$

où γ est une composition de quelques règles primitives $Post_p$ and Pre_p pour quelques places.

D'autre part, il existe des transitions plus complexes (les transitions synchrones) c.à.d une transition effectuée pour des places dans au moins deux sous-réseaux différents, par exemple deux voisins. Nous créons des **règles de synchronisation entre deux voisins** (y compris deux sous réseaux I et $I + 1$ ainsi que deux sous réseaux 0 et $N - 1$) en descendant de la racine jusqu'au niveau de chaque sous-réseau :

$$H(sys(x, y)) \rightarrow Sync_{II+1}(sys(x, y)) \tag{6.8}$$

$$H(sys(x, y)) \rightarrow Sync_{0N-1}(sys(x, y)) \tag{6.9}$$

Nous avons besoin de règle de circulation pour descendre du niveau k au niveau k-1 du codage :

$$Sync_{II+1}(sys(x,y)) \rightarrow sys(Sync_{II+1}(x),y) \tag{6.10}$$
$$Sync_{II+1}(sys(x,y)) \rightarrow sys(x,Sync_{II+1}(y)) \tag{6.11}$$

Puis nous lançons les règles de synchronisation à chaque niveau du codage :

$$Sync_{II+1}(sys(x,y)) \rightarrow sys(Sync_I(x),Sync_{I+1}(y)) \tag{6.12}$$
$$Sync_{0N-1}(sys(x,y)) \rightarrow sys(Sync_0(x),Sync_{N-1}(y)) \tag{6.13}$$
$$Sync_I(sys(x,y)) \rightarrow sys(x,Sync_I(y)) \tag{6.14}$$
$$Sync_{I+1}(sys(x,y)) \rightarrow sys(Sync_{I+1}(x),y) \tag{6.15}$$
$$Sync_0(sys(x,y)) \rightarrow sys(Sync_0(x),y) \tag{6.16}$$
$$Sync_{N-1}(sys(x,y)) \rightarrow sys(x,Sync_{N-1}(y)) \tag{6.17}$$

Finalement, au niveau d'un sous-réseau du codage :

$$Sync_0(local(x,y)) \rightarrow \alpha_0 \tag{6.18}$$
$$Sync_{N-1}(local(x,y)) \rightarrow \alpha_{N-1} \tag{6.19}$$
$$Sync_I(local(x,y)) \rightarrow \beta_I \tag{6.20}$$
$$Sync_{I+1}(local(x,y)) \rightarrow \beta_{I+1} \tag{6.21}$$

où α, β sont des compositions de quelques règles primitives $Post_p$ and Pre_p pour quelques places.

Exemple 27 *Problème des Philosophes. En fait, un sous réseau ayant 6 places* $\{Idle, WaitLeft, HasLeft, WaitRight, HasRight, Fork\}$ *avec seulement 2 jetons dans les places* $Idle$ *et* $Fork$ *est codé ci-dessous :*

$$t_{local} = local(\mathbf{idle}(\mathbf{1}, waitleft(0, hasleft(0, waitright(0, hasright(0, \mathbf{fork}(\mathbf{1}, \perp)))))))$$

Ainsi, un codage de niveau 1 pour 2 philosophes est une composition de deux sous-réseaux identiques est codé comme suit :

$$t_{sys_1} = sys(t_{local}, t_{local})$$

Ainsi, un codage niveau 2 pour 4 philosophes est composé de la même manière :

$$t_{sys_2} = sys(t_{sys_1}, t_{sys_1})$$

Évidemment, après le problème des Philosophes, le codage hiérarchique en composant des sous-réseaux identiques convient bien encore avec le protocole de Slotted Ring et le protocole de Round Robin Mutex. Ce dernier protocole a un comportement un peu différent des autres car il doit synchroniser tous les sous réseaux avec une place partagée.

Notre implémentation sur ces codages de réseaux de Petri montre que la transformation Petri-EFTRS automatique fournit le même nombre de configurations par rapport aux autres approches.

Remarques 10 *Grâce au dépliage récursif, l'application des règles profite bien l'avantage de la technique cache à la BDD.*

Optimisation : Nous observons que les règles de circulation 6.10 et 6.11 sont élémentaires à droite de type I, ainsi elles sont remplacées par la règle suivante :

$$Sync^{\star}_{II+1}(sys(x,y)) \rightarrow sys(Sync^{\star}_{II+1}(x), Sync^{\star}_{II+1}(y))$$

Par conséquent, les algorithmes d'optimisation de la section 5.2 sont utilisables dans le codage hiérarchique.

Remarques 11 *Grâce au dépliage récursif, l'application des règles profite bien l'avantage de la technique LFP (ou l'algorithme de saturation).*

Résultats expérimentaux 7 *En utilisant notre outil (détaillé au chapitre 8), il est possible de traiter le problème ayant environ 30 milliards philosophes en* $135,407$ *secondes.*

Remarques 12 *Ce codage est efficace quand le nombre de sous réseaux identiques N est une puissance de 2. Sinon comme le codage de réseaux de Petri 3, il risque d'y avoir de nombreuses de feuilles non utilisées qui réduisent la performance du codage. Une solution raisonnable est également de décomposer N en somme de puissances de 2. Ensuite, nous construisons un codage hiérarchique pour chacun. Enfin, nous pouvons synchroniser facilement ces codages.*

Remarques 13 *Il faut noter qu'il existe encore un autre type de réseau hiérarchique tel que le paramètre N est le nombre de jeton de chaque place, comme deux exemples suivants : le système de Kanban et le système de FMS. Ici la notion de "sous-réseau" devient inutile, donc actuellement il n'existe pas encore de solution efficace pour optimiser ces codages.*

BILAN DU CHAPITRE 6

Dans ce chapitre, nous avons montré la puissance de l'EFTRS dans la modélisation de quelques protocoles pour montrer la relation entre TRS et FTRS, et entre FTRS et EFTRS dans objectif de réaliser le premier pas de la transformation automatique.

Nous avons également présenté trois codages pour simuler un réseau de Petri borné ordinaire, et montré comment nous pouvons calculer l'état des successeurs dans ces codages. A partir de cet opérateur, nous pouvons calculer les états accessibles, l'état précédent (le calcul en arrière), le calcul du chemin, etc. Ce sont des calculs primitifs pour la logique CTL. D'autre part, le calcul de l'état des successeurs est utilisé également pour la logique LTL.

Nous avons également présenté une structure adaptée étant un type de réseau hiérarchique avec des sous réseaux identiques tel que le problème des Philosophes, le protocole de Slotted Ring et le protocole de Round Robin Mutex. La transformation automatique d'un réseau de Petri hiérarchique vers un EFTRS nous conduit à un autre problème qui est de détecter automatiquement des sous-réseaux identiques à partir d'un réseau de Petri. De plus, nous sommes à la recherche d'une solution efficace pour simuler le type de réseaux hiérarchiques tels que le système de Kanban et le système de FMS. Les tels réseaux sont paramétrés par un nombre de jetons de chaque place. Cela ne nous donne pas des opérations primitives comme le type de réseau hiérarchique avec le pliage récursif des sous réseaux identiques. Le pliage récursif n'est pas applicable ici, les techniques de LFPs ne sont donc pas efficaces.

Chapitre 7

Vérification par EFTRS

RÉSUMÉ DU CHAPITRE 7

Dans le chapitre 4, nous avons présenté l'EFTRS préservant la puissance d'expression des systèmes fonctionnels et des techniques d'accélération des calculs aboutissant à un outil de vérification symbolique efficace. Dans le chapitre 5, nous avons expliqué comment nous pouvons évaluer les EFTRSs.

Dans le chapitre 6, nous avons montré la puissance des EFTRSs dans la modélisation par quelques protocoles. Dans ce chapitre, nous continuons à montrer leur capacité à vérifier certaines propriétés connues pour ces modèles.

Tout d'abord, nous discutons un problème d'analyse d'accessibilité. A partir de ce problème, nous pouvons résoudre d'autres problèmes tels que le calcul des propriétés de sûreté ou l'inter-blocage par EFTRS.

Ce chapitre contient également des démonstrations de la spécification des formules de la logique CTL ou de la logique LTL pour les traduire en EFTRS.

Sommaire

7.1 Analyse d'accessibilité pour l'inter-blocage

Nous discuterons d'abord du calcul des accessibles par les EFTRSs. A partir de ce calcul, nous pouvons résoudre des autres problèmes tels que le calcul des propriétés de sûreté ou l'inter-blocage par EFTRS. L'ensemble des états accessibles est utilisé également pour des calculs de propriété pour la logique CTL ou LTL.

Etant donné un état initial *init*, nous pouvons obtenir tous les états accessibles en calculant l'ensemble de termes clos descendant de l'état initial *init* $\mathcal{R}^{\star}_{\lambda^e}(F(init))$.

Après le calcul des états accessibles $\mathcal{R}^{\star}_{\lambda^e}$ discuté ci-dessus. Nous avons besoin d'un autre calcul primitif : le calcul arrière $\mathcal{R}^{-1}_{\lambda^e}$. Actuellement, nous ne traitons pas le cas général. Cependant, pour les modèles de réseaux de Petri, le calcul arrière est calculable en changeant la direction des transitions des réseaux de Petri donnés (Post devient Pre et Pre devient Post).

Définition 17 *Calcul arrière*
Soient u un terme dans $\mathcal{F}_{bin}$ et $\mathcal{R}^{T}_{\lambda^e}$ un ensemble de règles d'EFTRSs tel que $\mathcal{R}^{T}_{\lambda^e} = \bigcup_{t_i \in T} \mathcal{R}^{t_i}_{\lambda^e}$. S'il existe v tel que $v = \mathcal{R}^{t}_{\lambda^e}(Trans_t(u))$ avec $t \in T$, ainsi l'état précédent du terme v est défini comme suivant :

$$\mathcal{R}^{-1}_{\lambda^e}(v) = u$$

Maintenant nous pouvons utiliser deux calculs primitifs pour calculer des états bloquants comme dans la proposition suivante :

Proposition 8 *Soient init un terme initial dans $\mathcal{F}_{bin}$ et $\mathcal{R}_{\lambda^e}$ un ensemble de règles d'EFTRS. L'inter-blocage d'un ensemble de termes de $\mathcal{F}_{bin}$ (appelé deadlock) est calculé sur $\mathcal{R}_{\lambda^e}$ à partir de init comme suivant :*

$$deadlock = \mathcal{R}^{\star}_{\lambda^e}(init) \setminus \mathcal{R}^{-1}_{\lambda^e}(\mathcal{R}^{\star}_{\lambda^e}(init))$$

Preuve 12 (Preuve succincte)

$$deadlock = \mathcal{R}^{\star}_{\lambda^e}(init) \backslash \mathcal{R}^{-1}_{\lambda^e}(\mathcal{R}^{\star}_{\lambda^e}(init))$$

- *$deadlock \subseteq \mathcal{R}^{\star}_{\lambda^e}(init) \backslash \mathcal{R}^{-1}_{\lambda^e}(\mathcal{R}^{\star}_{\lambda^e}(init))$: Intuitivement selon $deadlock \subseteq \mathcal{R}^{\star}_{\lambda^e}(init)$ et $deadlock \cap \mathcal{R}^{-1}_{\lambda^e}(\mathcal{R}^{\star}_{\lambda^e}(init)) = \emptyset$*
- *$\mathcal{R}^{\star}_{\lambda^e}(init) \backslash \mathcal{R}^{-1}_{\lambda^e}(\mathcal{R}^{\star}_{\lambda^e}(init)) \subseteq deadlock$: Effectivement parce que s'il existe t accessible et il n'est pas dans $\mathcal{R}^{-1}_{\lambda^e}(\mathcal{R}^{\star}_{\lambda^e}(init))$, alors il doit être un état bloquant.*

Considérons le problème des Philosophes discuté dans le chapitre précédent.

Exemple 28 *Problème à deux philosophes.*
Le codage identique niveau 1 pour 2 philosophes est une composition de deux sous-réseaux identiques comme le suivant :

$$t_{sys_1} = sys(t_{local}, t_{local})$$

où un sous réseau ayant 6 places parmi lesquelles les places Idle et Fork contiennent un jeton est codé ci-dessous :

$$t_{local} = local(\mathbf{idle}(\mathbf{1}, waitleft(0, hasleft(0, waitright(0, hasright(0, \mathbf{fork}(\mathbf{1}, \bot))))))$$

Les philosophes, s'ils agissent tous de façons naïves et identiques, risquent fort de se retrouver en situation d'inter-blocage. En effet, il suffit que chacun saisisse sa fourchette de gauche et, qu'ensuite, chacun attende que sa fourchette de droite se libère pour qu'aucun d'entre eux ne puisse manger.

Nous calculons les états accessibles ensuite nous calculons l'état précédent de ces états. Finalement, nous pouvons détecter deux inter-blocages connus ci-dessous :

$$t^0_{local} = local(idle(0, \mathbf{waitleft}(\mathbf{1}, hasleft(0, waitright(0, \mathbf{hasright}(\mathbf{1}, fork(0, \bot))))))$$

$$t^1_{local} = local(idle(0, \mathbf{waitleft}(\mathbf{1}, hasleft(0, waitright(0, \mathbf{hasright}(\mathbf{1}, fork(0, \bot))))))$$

Et l'état symétrique :

$$t^0_{local} = local(idle(0, waitleft(0, \mathbf{hasleft}(\mathbf{1}, \mathbf{waitright}(\mathbf{1}, hasright(0, fork(0, \bot))))))$$

$$t^1_{local} = local(idle(0, waitleft(0, \mathbf{hasleft}(\mathbf{1}, \mathbf{waitright}(\mathbf{1}, hasright(0, fork(0, \bot))))))$$

Nous montrons que le calcul de l'inter-blocage reste faisable avec des modèles de grande taille dans la partie expérimentale (Voir cf. le chapitre 8).

Résultats expérimentaux 8 *En utilisant notre outil (détaillé au chapitre 8), il est possible de détecter deux états bloquants du problème de* 32.000 *philosophes en moins d'une heure.*

7.2 Analyse d'accessibilité pour les invariants

Les invariants Φ sont les propriétés de sûreté les plus communes et les plus utiles. Nous définirons ci-dessous des invariances primitives pour un terme dans $\mathcal{F}_{bin}$. Nous utilisons ces invariances primitives pour vérifier des propriétés intéressantes pour nos études de cas.

Définition 18 *Soit t un terme de $\mathcal{T}(\mathcal{F}_{bin})$. Nous dirons que ce terme t satisfait Φ si Φ est une des formes ci-dessous :*

$$\Phi ::= TRUE \mid \bot \mid a(\Phi_1, \Phi_2) \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid \alpha$$

avec $a \in \mathcal{F}_{bin}$ et $\alpha \in \mathcal{T}(\mathcal{F}_{bin}, \mathcal{X})$.

Il existe des TRSs classiques pour vérifier une telle propriété.

Conjecture 1 *Soit t un terme de $\mathcal{T}(\mathcal{F}_{bin})$. Nous dirons que ce terme t satisfait la propriété ϕ définie dans la définition 18 s'il existe un TRS modélisant une invariance tel que $(\mathcal{R}^{Inv})^\star(\{t\})$ n'est pas vide dans une des formes ci-dessous :*

1. *Inv_{all} avec n'importe quel terme t*

$$\mathcal{R}^{Inv_{all}} = \{\ x \to x\ \}.$$

2. *$Inv_\perp$ avec $t = \perp$*

$$\mathcal{R}^{Inv_\perp} = \{\ \perp \to \perp\ \}.$$

3. *$Inv^a_{Inv_l, Inv_r}$ avec $t = a(t_l, t_r)$, $a \in \mathcal{F}_{bin}$, $Inv_l(t_l)$ et $Inv_r(t_r)$ sont vraies*

$$a(x, y) \to a(x, y)$$

avec $\mathcal{P}os(t_l) = 00, \mathcal{P}os(t_r) = 01$ et

$$(\mathcal{R}^{Inv_l})^\star(t_l) \cap (\mathcal{R}^{Inv_r})^\star(t_r) \neq \emptyset$$

4. *$(Inv_1 \vee Inv_2)(t)$ avec $Inv_1(t)$ ou $Inv_2(t)$ est vraies :*

$$(\mathcal{R}^{Inv_1 \vee Inv_2})^\star(t) = (\mathcal{R}^{Inv_1})^\star(t) \cup (\mathcal{R}^{Inv_2})^\star(t)$$

5. *$(Inv_1 \wedge Inv_2)(t)$ avec $Inv_1(t)$ et $Inv_2(t)$ sont vraies :*

$$(\mathcal{R}^{Inv_1 \wedge Inv_2})^\star(t) = (\mathcal{R}^{Inv_1})^\star(t) \cap (\mathcal{R}^{Inv_2})^\star(t)$$

6. *Inv_α*

$$\mathcal{R}^{Inv_\alpha} = \{\ \alpha \to \alpha, \alpha \in \mathcal{T}(\mathcal{F}_{bin}, \mathcal{X})\ \}.$$

Une propriété de sûreté intéressante ϕ à vérifier sur le problème TAP est qu'il existe toujours un unique jeton circulant dans l'arbre de hiérarchie. Ainsi, nous pouvons formuler cette propriété comme un prédicat vérifier sur chaque configuration accessible.

Exemple 29 *Considérons les symboles $\perp$ et $\emptyset$ respectivement comme les valeurs TRUE et FALSE, Soit OneToken le prédicat défini comme le suivant pour un terme $\alpha \in \mathcal{T}(\mathcal{F}_{bin})$:*

$$OneToken(\alpha) = \begin{cases} \perp & si\ \forall p, p' \mathcal{P}os(\alpha), \\ & \alpha(p) = \alpha(p') = t \Rightarrow p = p' \\ \emptyset & sinon. \end{cases}$$

Par conséquent,

$$TAP\ satisfait\ \phi \Leftrightarrow \forall \alpha \in \mathcal{R}^\star(\{u\}), OneToken = \perp$$

Autrement dit,

$$TAP\ satisfait\ \phi \Leftrightarrow \forall \alpha \in \mathcal{R}^{\star}(\{u\}), OneToken \neq \emptyset$$

avec u *est le terme* $t(i(i(\bot,\bot),i(\bot,\bot)),i(i(\bot,\bot),i(\bot,\bot)))$.

Nous venons de définir la propriété ϕ *: il existe toujours un unique jeton circulant dans l'arbre de hiérarchie. Considérons* $\neg\phi$ *qui signifie que il existe peut-être une configuration ne contenant pas jeton ou au moins deux jetons.*

Nous montrons également comment nous pouvons prouver cette propriété ϕ *en TRS. Nous commençons par les feuilles en montant pas à pas de bas en haut, nous remplaçons chaque noeud par*

- $conflict$ *si tous ces enfants sont onlyt ou au moins un est* $conflict$*,*
- not *si tous ces enfants sont not,*
- $onlyt$ *sinon.*

L'ensemble de règles de TRS $\mathcal{R}^{OneToken}$ *est listées ci-dessus :*

$$a(\bot,\bot) \rightarrow not(\bot,\bot)\ with\ a \in \{i,r,b\} \tag{7.1}$$
$$t(\bot,\bot) \rightarrow onlyt(\bot,\bot) \tag{7.2}$$
$$a(conflict(x,y),z) \rightarrow conflict(conflict(x,y),z) \tag{7.3}$$
$$a(x,conflict(y,z)) \rightarrow conflict(x,conflict(y,z)) \tag{7.4}$$
$$a(onlyt(x,y),onlyt(z,w)) \rightarrow conflict(onlyt(x,y),onlyt(z,w)) \tag{7.5}$$
$$t(onlyt(x,y),onlyt(z,w)) \rightarrow conflict(onlyt(x,y),onlyt(z,w)) \tag{7.6}$$
$$a(not(x,y),not(z,w)) \rightarrow not(not(x,y),not(z,w)) \tag{7.7}$$
$$t(not(x,y),not(z,w)) \rightarrow onlyt(not(x,y),not(z,w)) \tag{7.8}$$
$$a(not(x,y),onlyt(z,w)) \rightarrow onlyt(not(x,y),onlyt(z,w)) \tag{7.9}$$
$$a(onlyt(x,y),not(z,w)) \rightarrow onlyt(onlyt(x,y),not(z,w)) \tag{7.10}$$
$$t(not(x,y),onlyt(z,w)) \rightarrow conflict(not(x,y),onlyt(z,w)) \tag{7.11}$$
$$t(onlyt(x,y),not(z,w)) \rightarrow conflict(onlyt(x,y),not(z,w)) \tag{7.12}$$

Pour vérifier que le $TAP\ satisfait\ \phi$*, cela suffit de tester s'il existe une configuration étiqueté onlyt à la racine :*

$$TAP\ satisfait\ \phi \Leftrightarrow$$
$$(Inv^{OneToken}) \wedge Inv^{onlyt}_{Inv_{all},Inv_{all}} \Leftrightarrow$$
$$(\mathcal{R}^{OneToken})^{\star}(\{u\}) \cap (Inv^{onlyt}_{Inv_{all},Inv_{all}})^{\star}(\{u\}) \neq \emptyset$$

où $Inv^{onlyt}_{Inv_{all},Inv_{all}}$ *est l'ensemble de règles pour filtrer une configuration étiqueté onlyt à la racine :*

$$onlyt(x,y) \rightarrow onlyt(x,y)$$

avec $\mathcal{P}os(t_l) = 00, \mathcal{P}os(t_r) = 01$ *et* $(\mathcal{R}^{Inv_{all}})^{\star}(t_l) \cap (\mathcal{R}^{Inv_{all}})^{\star}(t_r) \neq \emptyset$.

Nous observons que nous devons utiliser en TRS classique des marqueurs qui amènent souvent à une explosion d'états du système. Cependant, les systèmes fonctionnels d'invariance présentés ci-dessous peuvent éviter des marqueurs en utilisant

des non-terminaux. Les systèmes fonctionnels d'invariance primitifs sont définis comme suit :

Conjecture 2 *Soit t un terme de $\mathcal{T}(\mathcal{F}_{bin})$. Nous dirons que ce terme t satisfait ϕ définie dans la définition 18 s'il existe un TRS modélisant une invariance tel que $(\mathcal{R}_{\lambda^e}^{Inv})^\star(\{Inv(t)\})$ n'est pas vide dans une des formes ci-dessous :*

1. *Inv_{all} avec n'importe quel terme t*

$$\mathcal{R}_{\lambda^e}^{Inv_{all}} = \left\{ \begin{array}{l} Inv(a(x,y)) \to a(x,y)\ \forall a \in \mathcal{F}_{bin} \\ Inv(\bot) \to \bot \end{array} \right\}.$$

2. *$Inv_\bot$ avec $t = \bot$*

$$\mathcal{R}_{\lambda^e}^{Inv_\bot} = \left\{ \ Inv(\bot) \to \bot \ \right\}.$$

3. *$Inv^a_{Inv_l,Inv_r}$ avec $t = a(t_l, t_r)$, $a \in \mathcal{F}_{bin}$, $Inv_l(t_l)$ et $Inv_r(t_r)$ sont vraies.*

$$\mathcal{R}_{\lambda^e}^{Inv^a_{Inv_l,Inv_r}} = \left\{ \ Inv(a(x,y)) \to a(Inv_l(x), Inv_r(y)) \ \right\}.$$

4. *$Inv_1 \vee Inv_2$ avec $Inv_1(t)$ ou $Inv_2(t)$ est vraies.*

$$\mathcal{R}_{\lambda^e}^{Inv_1 \vee Inv_2} = \left\{ \begin{array}{l} Inv(a(x,y)) \to Inv_1(a(x,y))\ \forall a \in \mathcal{F}_{bin} \\ Inv(a(x,y)) \to Inv_2(a(x,y)), \\ Inv(\bot) \to Inv_1(\bot), \\ Inv(\bot) \to Inv_2(\bot) \end{array} \right\}.$$

5. *$Inv_1 \wedge Inv_2$ avec $Inv_1(t)$ et $Inv_2(t)$ sont vraies.*

$$\mathcal{R}_{\lambda^e}^{Inv_1 \wedge Inv_2} = \left\{ \begin{array}{l} Inv(a(x,y)) \to Inv_1(Inv_2(a(x,y)))\ \forall a \in \mathcal{F}_{bin}, \\ Inv(\bot) \to Inv_1(Inv_2(\bot)) \end{array} \right\}.$$

6. *Inv_α Nous avons besoin de l'EFTRS $\mathcal{R}_{check}^{\alpha \to \beta}$ donné dans la section 4.2 qui nous permet de vérifier si un terme $t$de $\mathcal{T}(\mathcal{F}_{bin})$ concorde avec α.*

En fait, nous pouvons vérifier si l'ensemble de mauvais états $Bad = \neg\phi = \neg Inv(\mathcal{R}^\star(init))$ soit toujours faux. Un état $\neg\phi$ vrai est compté comme un exemple.

Définition 19 *Soit t un terme de $\mathcal{T}(\mathcal{F}_{bin})$. Une invariance négative $\neg Inv(t)$ est vraie seulement si $\neg Inv$ est une des formes ci-dessous :*

1. *$\neg Inv_{all} = \emptyset$*
2. *$\neg Inv_\bot$ avec $t = a(t_1, t_2)$*
3. *$\neg Inv^a_{Inv_l,Inv_r}$*
   - *avec $t = b(t_l, t_r)$, $b \neq a \mid b \in \mathcal{F}_{bin}$, ou*
   - *avec $t = a(t_l, t_r)$ $Inv_l(t_l) \vee Inv_r(t_r)$ est faux.*
4. *$\neg(Inv_1 \vee Inv_2)$ avec $\neg Inv_1(t) \wedge \neg Inv_2(t)$ est vraie.*
5. *$\neg(Inv_1 \wedge Inv_2)$ avec $\neg Inv_1(t) \vee \neg Inv_2(t)$ est vraie.*

6. $\neg Inv_\alpha$ *avec* $\mathcal{R}^{\alpha\to\beta}_{check}$ *sur* t *est vide*

En fait, il existe une autre solution pour calculer $\neg Inv$ en utilisant le calcul des états accessibles $\mathcal{R}^\star$. Elle est présentée dans la conjecture suivante :

Conjecture 3 *Négation d'invariance.*

$$\neg Inv(\alpha) = \mathcal{R}^\star(init) \backslash Inv(\alpha)$$

avec α *un terme clos.*

Revenons à notre étude de cas TAP.

Exemple 30 *L'EFTRS suivant* $\mathcal{R}^{\neg\phi}_{\lambda^e}$ *spécifie la propriété* $\neg\phi$.

$$No\_t(r(x,y)) \to r(No\_t(x), No\_t(y)) \tag{7.13}$$
$$No\_t(b(x,y)) \to b(No\_t(x), No\_t(y)) \tag{7.14}$$
$$No\_t(i(x,y)) \to i(No\_t(x), No\_t(y)) \tag{7.15}$$
$$No\_t(\bot) \to \bot \tag{7.16}$$
$$Two\_t(r(x,y)) \to r(Two\_t(x), y) \tag{7.17}$$
$$Two\_t(r(x,y)) \to r(x, Two\_t(y)) \tag{7.18}$$
$$Two\_t(b(x,y)) \to b(Two\_t(x), y) \tag{7.19}$$
$$Two\_t(b(x,y)) \to b(x, Two\_t(y)) \tag{7.20}$$
$$Two\_t(i(x,y)) \to i(Two\_t(x), y) \tag{7.21}$$
$$Two\_t(i(x,y)) \to i(x, Two\_t(y)) \tag{7.22}$$
$$Two\_t(t(x,y)) \to i(x, Two\_t(y)) \tag{7.23}$$

Sémantiquement dit, le symbole $No\_t \in \mathcal{F}_{NT}$ *est introduit pour vérifier qu'un terme ne contient pas le symbole* t. *Effectivement, il n'y a aucune règle de la forme* $No\_t(t(x,y)) \to \beta$. *En conséquence, pour un terme* $\alpha \in \mathcal{T}(\mathcal{F}_{bin})$ *pour lequel il existe* $p \in \mathcal{P}os(\alpha)$ *tel que* $\alpha(p) = t$, $(\mathcal{R}^{\neg\phi}_{\lambda^e})^*(\{No\_t(\alpha)\}) = \emptyset$.

Le symbole $Two\_t \in \mathcal{F}_{NT}$ *est introduit pour vérifier que le symbole* t *apparaît dans un terme au moins une fois. Ainsi, pour un terme* $\alpha \in \mathcal{T}(\mathcal{F}_{bin})$,

- $(\mathcal{R}^{\neg\phi}_{\lambda^e})^\star(\{Two\_t(Two\_t(\alpha))\}) = \emptyset$ *signifie que le symbole* t *apparaît dans* α *une seule fois ou jamais ;*
- $(\mathcal{R}^{\neg\phi}_{\lambda^e})^\star(\{Two\_t(Two\_t(\alpha))\}) \neq \emptyset$ *signifie que le symbole* t *apparaît dans* α *au moins deux fois.*

Par conséquent,

$$\text{le TAP satisfait } \phi \Leftrightarrow$$
$$(\mathcal{R}^{\neg\phi}_{\lambda^e})^\star((\mathcal{R}^{TAP}_{\lambda^e})^\star(\{Two\_t(Two\_t(t'_0))\})) \wedge (\mathcal{R}^{\neg\phi}_{\lambda^e})^\star((\mathcal{R}^{TAP}_{\lambda^e})^\star(\{No\_t(t'_0)\})) = \emptyset$$

quand $t'_0 = Arbiter(t_0)$ *et* t_0 *est la configuration initiale du TAP.*

Nous présentons une autre solution en calculant $\mathcal{R}^{\star}_{\lambda^e}(\{Inv_{OnlyT}(\alpha)\})$ *ci-dessous :*

$$Inv_{OnlyT}(a(x,y)) \rightarrow a(Inv_{OnlyT}(x), Inv_{NoT}(y)) \tag{7.24}$$
$$Inv_{OnlyT}(a(x,y)) \rightarrow a(Inv_{NoT}(x), Inv_{OnlyT}(y)) \tag{7.25}$$
$$Inv_{OnlyT}(t(x,y)) \rightarrow t(Inv_{NoT}(x), Inv_{NoT}(y)) \tag{7.26}$$
$$Inv_{NoT}(a(x,y)) \rightarrow a(Inv_{NoT}(x), Inv_{NoT}(y)) \tag{7.27}$$
$$Inv_{NoT}(\bot) \rightarrow \bot \; with \; a \in \{i, r, b\} \tag{7.28}$$

Résultats expérimentaux 9 *En utilisant notre outil (En détail dans le chapitre 8), nous sommes capables de vérifier que* ϕ *est vrai pour le TAP traitant* 2^{400} *processeurs en* $2,933$ *secondes.*

7.2.1 Quelques modèles arborescents

7.2.1.1 Vérification d'invariants du PP

Considérons le protocole de Percolate en section 6.1.1. Nous nous intéressons uniquement aux propriétés de E_1 qui contiennent seulement des étiquettes n, p et $\bot$ (Voir la figure 6.2).

$$E_1 = ((\mathcal{R}^{PP})^{\star}(E_0) \cap \mathcal{T}(\{n, p, \bot\})).$$

Un ensemble de règles d'EFTRS $(\mathcal{R}^{Filter_{np\bot}}_{\lambda^e})^{\star}(\{Inv(\alpha)\})$ correspondant est décrit ci-dessous :

$$Inv(\bot) \rightarrow \bot \tag{7.29}$$
$$Inv(p(x,y)) \rightarrow p(Inv(x), Inv(y)) \tag{7.30}$$
$$Inv(n(x,y)) \rightarrow n(Inv(x), Inv(y)) \tag{7.31}$$

Finalement, nous considérons la propriété telle que sa racine est étiquetée par p :

$$E'_1 = (E_1 \cap \{p(t_1, t_2) | t_1, t_2 \in T\}).$$

Ainsi, $(\mathcal{R}^{P}_{\lambda^e})^{\star}(\{P(\alpha)\})$ avec $P(p(x,y)) \rightarrow p(x,y)$.

En conclusion, $E'_1 \neq \emptyset$ signifie qu'au moins une des feuilles possède une valeur p ; $E'_1 = \emptyset$ signifie que toutes les feuilles n'ont qu'une valeur n.

Résultats expérimentaux 10 *En utilisant notre outil (détaillé au chapitre 8), il est possible de traiter le protocole de Percolate ayant* 2^{500} *processus en moins d'une demie de seconde.*

7.2.1.2 Vérification d'invariants du protocole d'Election Arborescent

Considérons le protocole d'Election Arborescent en section 6.1.2.

Dans la deuxième étape, nous essayerons de descendre en sélectionnant le candidat (étiqueté par p) si possible.

A partir du résultat de E_2, nous nous intéressons aux propriétés ayant un chemin qui contient uniquement c de la racine à la feuille.

Un tel EFTRS $(\mathcal{R}_{\lambda^e}^{Path_c})^\star(\{Inv(\alpha)\})$ est décrit ci-dessous :

$$O(a(x,y)) \to a(x,y) \text{ avec } a \in \{n,p\} \tag{7.32}$$
$$Inv(\bot) \to \bot \tag{7.33}$$
$$O(\bot) \to \bot \tag{7.34}$$
$$Inv(c(x,y)) \to c(O(x), Inv(y)) \tag{7.35}$$
$$Inv(c(x,y)) \to c(Inv(x), O(y)) \tag{7.36}$$

L'ensemble E_2' est illustré en figure 6.4.

Nous montrons également comment nous pouvons modéliser ce EFTRS en TRS : Nous pouvons dénombrer le nombre de feuilles étiquetées par c.

Nous proposons une autre solution : Nous pouvons trouver ce résultat final en lançant $(\mathcal{R}_{\lambda^e}^{Path_c})^\star(\{Inv(\alpha)\})$ sur $E"_1$ (Voir la figure 6.4).

Résultats expérimentaux 11 *En calculant $(\mathcal{R}_{\lambda^e}^{Path_c})^\star(\{Inv(\alpha)\})$ sur $E"_1$ par notre outil (détaillé au chapitre 8), il est possible de traiter le protocole d'Election Arborescent ayant 2^{450} processus en moins d'une seconde.*

TRS $\mathcal{R}^{Path_o}$:

$$c(\bot,\bot) \to o(\bot,\bot) \tag{7.37}$$
$$c(o(x,y),z) \to o(o(x,y),z) \tag{7.38}$$
$$o(x,o(y,z)) \to o(x,o(y,z)) \tag{7.39}$$

Un tel EFTRS $\mathcal{R}_{\lambda^e}^{Path_o}$ est décrit ci-dessous :

– *Règles générées*

$$O(o(x,y)) \to o(x,y) \tag{7.40}$$
$$O(\bot) \to \bot \tag{7.41}$$
$$H(c(x,y)) \to o(O(x),y) \tag{7.42}$$
$$H(c(x,y)) \to o(x,O(y)) \tag{7.43}$$

– *Règles de circulation*

$$H(c(x,y)) \to c(H(x),y) \tag{7.44}$$
$$H(c(x,y)) \to c(x,H(y)) \tag{7.45}$$

– *Règles de point fixe*

$$Path_o(x) \rightarrow x \tag{7.46}$$
$$Path_o(x) \rightarrow Path_o(H(x)) \tag{7.47}$$

Nous commençons à partir du bottom en remplaçant chaque étiquette c par o et monter noeud par noeud. La propriété ayant une étiquette o à la racine est celle qui nous intéresse.

$$E_2 = (\mathcal{R}^{Path_o})^\star(E"_1) = (\mathcal{R}^{Path_o}_{\lambda^e})^\star(Inv(E"_1)).$$

Ainsi, nous présentons les systèmes ci-dessous pour chercher une propriété ayant une étiquette o à la racine :

$$E"_2 = (E'_2 \cap \{o(t_1, t_2) | t_1, t_2 \in T\}).$$

Ainsi, $(\mathcal{R}^{O}_{\lambda^e})^\star(\{O(\alpha)\})$ avec $O(o(x,y)) \rightarrow o(x,y)$.

Cette propriété est exactement comme en figure 6.4 sauf que le chemin des étiquettes c est remplacé par celui des étiquettes o.

Résultats expérimentaux 12 *En calculant $(\mathcal{R}^{Path_o}_{\lambda^e})^\star(\{Inv(\alpha)\})$ sur $E"_1$ par notre outil (détaillé au chapitre 8), il est possible de traiter le protocole d'Election Arborescent ayant 2^{400} processus en plus de deux minutes.*

7.2.2 Quelques modèles de réseaux de Petri

Dans les modèles de réseaux de Petri, il faut bien maîtriser les règles de circulation pour aller à l'endroit de la place indiquée et tester le marquage souhaité.

7.2.2.1 Vérification d'invariants des Philosophes

Test d'invariance pour vérifier qu'il y a aucune configuration telle que deux voisins prennent la même fourchette. Le problème désormais devient de lancer un système de ré-écriture $\mathcal{R}^{Conflit}_{\lambda^e}$ pour tester les deux voisins. Si $(\mathcal{R}^{Conflit}_{\lambda^e})^\star(\{Conflit(\alpha)\})$ fournit un résultat non vide, alors il existe une telle configuration.

$$Conflit(system(x,y)) \rightarrow Sync1N(system(x,y)) \tag{7.48}$$
$$Conflit(system(x,y)) \rightarrow SyncIIPP(system(x,y)) \tag{7.49}$$
$$SyncIIPP(system(x,y)) \rightarrow system(SyncIIPP(x), y) \tag{7.50}$$
$$SyncIIPP(system(x,y)) \rightarrow system(x, SyncIIPP(y)) \tag{7.51}$$
$$SyncIIPP(system(x,y)) \rightarrow system(TestI(x), TestIPP(y)) \tag{7.52}$$
$$Sync1N(system(x,y)) \rightarrow system(TestIPP(x), TestI(y)) \tag{7.53}$$
$$TestI(system(x,y)) \rightarrow system(x, TestI(y)) \tag{7.54}$$
$$TestIPP(system(x,y)) \rightarrow system(TestIPP(x), y) \tag{7.55}$$
$$TestI(local(x,y)) \rightarrow local(x, LocalTestI(y)) \tag{7.56}$$
$$TestIPP(local(x,y)) \rightarrow local(x, LocalTestIPP(y)) \tag{7.57}$$
$$LocalTestI(x) \rightarrow Testhasright(x) \tag{7.58}$$
$$LocalTestIPP(x) \rightarrow Testhasleft(x) \tag{7.59}$$

Deux tests primitifs pour vérifier si ce philosophe a bien pris une fourchette dans le main gauche (droite) :

$$Testhasleft(hasleft(x,y)) \to hasleft(Test(x),y) \tag{7.60}$$
$$Testhasleft(fork(x,y)) \to fork(x,Testhasleft(y)) \tag{7.61}$$
$$Testhasleft(hasright(x,y)) \to hasright(x,Testhasleft(y)) \tag{7.62}$$
$$Testhasleft(waitright(x,y)) \to waitright(x,Testhasleft(y)) \tag{7.63}$$
$$Testhasleft(waitleft(x,y)) \to waitleft(x,Testhasleft(y)) \tag{7.64}$$
$$Testhasleft(idle(x,y)) \to idle(x,Testhasleft(y)) \tag{7.65}$$
$$Testhasleft(fork(x,y)) \to fork(x,Testhasleft(y)) \tag{7.66}$$
$$Testhasright(hasright(x,y)) \to hasright(Test(x),y) \tag{7.67}$$
$$Testhasright(fork(x,y)) \to fork(x,Testhasright(y)) \tag{7.68}$$
$$Testhasright(hasleft(x,y)) \to hasleft(x,Testhasright(y)) \tag{7.69}$$
$$Testhasright(waitright(x,y)) \to waitright(x,Testhasright(y)) \tag{7.70}$$
$$Testhasright(waitleft(x,y)) \to waitleft(x,Testhasright(y)) \tag{7.71}$$
$$Testhasright(idle(x,y)) \to idle(x,Testhasright(y)) \tag{7.72}$$
$$Testhasright(fork(x,y)) \to fork(x,Testhasright(y)) \tag{7.73}$$
$$Test(1) \to 1 \tag{7.74}$$

Résultats expérimentaux 13 *En utilisant notre outil (détaillé au chapitre 8), nous sommes capables de vérifier que parmi* 2^{35} *philosophes, il n'y a aucune configuration telle que deux voisins qui prennent la même fourchette en une dizaine de minutes.*

7.2.2.2 Vérification des invariants du protocole de Round-Robin Mutex

Nous intéressons également à un test d'invariance φ pour vérifier qu'il y a une seule ressource dans le modèle. Ainsi, nous essayons un test de négation d'invariance $\neg\varphi$ pour chercher une configuration dans la quelle :

– soit au moins deux processus qui utilisent cette ressource en même temps

$$H(x) \to TestR(TestR(x)) \tag{7.75}$$
$$TestR(systemGlobal(x,y)) \to systemGlobal(x,TestR(y)) \tag{7.76}$$
$$TestR(system(x,y)) \to system(TestR(x),y) \tag{7.77}$$
$$TestR(system(x,y)) \to system(x,TestR(y)) \tag{7.78}$$
$$TestR(local(x,y)) \to local(x,TestR(y)) \tag{7.79}$$
$$TestR(r(x,y)) \to r(Test(x),y) \tag{7.80}$$
$$TestR(psend(x,y)) \to psend(x,TestR(y)) \tag{7.81}$$
$$TestR(pload(x,y)) \to pload(x,TestR(y)) \tag{7.82}$$
$$TestR(pok(x,y)) \to pok(x,TestR(y)) \tag{7.83}$$
$$TestR(buffull(x,y)) \to buffull(x,TestR(y)) \tag{7.84}$$
$$TestR(bufidle(x,y)) \to bufidle(x,TestR(y)) \tag{7.85}$$
$$TestR(pask(x,y)) \to pask(x,TestR(y)) \tag{7.86}$$
$$TestR(pwait(x,y)) \to pwait(x,TestR(y)) \tag{7.87}$$
$$Test(1) \to 0 \tag{7.88}$$

– soit aucun la touche.

$$H(x) \rightarrow TestZ(x) \quad (7.89)$$
$$TestZ(systemGlobal(x,y)) \rightarrow systemGlobal(TestZ(x), TestZ(y)) \quad (7.90)$$
$$TestZ(system(x,y)) \rightarrow system(TestZ(x), TestZ(y)) \quad (7.91)$$
$$TestZ(res(x,y)) \rightarrow res(Test_0(x), y) \quad (7.92)$$
$$TestZ(local(x,y)) \rightarrow local(x, TestZ(y)) \quad (7.93)$$
$$TestZ(r(x,y)) \rightarrow r(Test(x), y) \quad (7.94)$$
$$TestZ(psend(x,y)) \rightarrow psend(x, TestZ(y)) \quad (7.95)$$
$$TestZ(pload(x,y)) \rightarrow pload(x, TestZ(y)) \quad (7.96)$$
$$TestZ(pok(x,y)) \rightarrow pok(x, TestZ(y)) \quad (7.97)$$
$$TestZ(buffull(x,y)) \rightarrow buffull(x, TestZ(y)) \quad (7.98)$$
$$TestZ(bufidle(x,y)) \rightarrow bufidle(x, TestZ(y)) \quad (7.99)$$
$$TestZ(pask(x,y)) \rightarrow pask(x, TestZ(y)) \quad (7.100)$$
$$TestZ(pwait(x,y)) \rightarrow pwait(x, TestZ(y)) \quad (7.101)$$
$$Test_0(0) \rightarrow 0 \quad (7.102)$$

Remarques 14 *Nous observons que ces systèmes ci-dessus sont une version de Two_t et No_t du TAP pour le protocole de Round Robin Mutex. Intuitivement, les règles de circulation de TestR (TestZ) sont quasiment identiques à celles de Two_t (No_t).*

Résultats expérimentaux 14 *En utilisant notre outil (détaillé au chapitre 8), il est possible de vérifier que parmi* 2^8 *processus, il n'y a aucune configuration dans la quelle, deux processus utilisent cette ressource en même temps en une vingtaine de minutes.*

7.3 Logiques temporelles

7.3.1 Logique temporelle linéaire LTL

Nous avons rappelé en section 2.2 les opérateurs primitifs la logique temporelle linéaire LTL pour décrire des événements suivis un seul chemin de calcul. Nous avons dit comment [Wolper 2001, Couvreur 1999] peuvent traduire une formule LTL en automate de Büchi. Le problème ici est de construire des structures de vérification symbolique en EFTRS à partir des spécifications en LTL.

Dans un premier temps, nous construisons un automate de Büchi en EFTRS.

Définition 20 *Automate de Büchi*
Soit une table de vérité de l'automate à transitions $\mathcal{A}$*. Ainsi,* $\mathcal{R}^{\omega}_{\lambda^e}$ *est un ensemble de règles d'EFTRS comme suit :*

$$TestChange_t(\varphi(x,y)) \rightarrow \varphi(TestChange_{s \xrightarrow{\varphi} s'}(x), y), t \in T_{\mathcal{A}}$$

où $T_{\mathcal{A}}$ *est l'ensemble de transition de l'automate donné.*

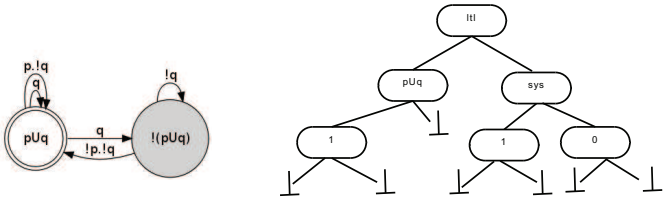


FIG. 7.1 – a) Automate de Büchi de la formule LTL $f = p\ U\ q$ b) Produit synchronisé de l'automate et le système

Considérons l'exemple ci-dessous :

Exemple 31 *Supposons que nous avons déjà un automate de Büchi (déjà discuté en chapitre 2). L'automate de cette formule est parue dans la figure 7.1 (a)). Le calcul de l'automate à transitions pour la formule $p\ U\ q$ est construit par $now(pUq) = q \vee (p \wedge next(pUq))$ et l'Acceptation $= q \vee \neg now(pUq)$ dans la table de vérité comme suivant :*

p	q	$next(pUq)$	$now(pUq)$	Acceptation
0	0	0	0	1
1	0	0	0	1
0	1	0	1	1
1	1	0	1	1
0	0	1	0	1
1	0	1	1	0
0	1	1	1	1
1	1	1	1	1

Nous construisons un système d'EFTRS pour l'automate de Büchi de la formule $f = p\ U\ q$ à partir de cette table. Cet automate est représenté dans la figure 7.1 (a). et est interprété par un ensemble de règles $\mathcal{R}^{\omega}_{\lambda^e}$ indiqués ce-dessous :

$$TestChange_{t_0}(pUq(x,y)) \rightarrow pUq(Test_1(x), y) \tag{7.103}$$
$$TestChange_{t_1}(pUq(x,y)) \rightarrow pUq(TestChange_{10}(x), y) \tag{7.104}$$
$$TestChange_{t_2}(pUq(x,y)) \rightarrow pUq(Test_0(x), y) \tag{7.105}$$
$$TestChange_{t_3}(pUq(x,y)) \rightarrow pUq(TestChange_{01}(x), y) \tag{7.106}$$
$$TestChange_{t_4}(pUq(x,y)) \rightarrow pUq(Test_1(x), y) \tag{7.107}$$

Le problème désormais, est d'utiliser le codage de cet automate en EFTRS, puis de construire le produit synchronisé de l'automate de Büchi et de celui du système.

Définition 21 *Terme du produit synchronisé*
Un terme du produit synchronisé est de la forme

$$ltl(t_\omega, t_M)$$

où t_ω est le codage de l'automate, t_M est le codage du système et $t_\omega, t_M \in \mathcal{T}(\mathcal{F}_{bin})$.

Un codage de cet exemple avec l'automate de Büchi du côté gauche, et le système du côté droit dans la figure 7.1 (b).

Définition 22 *Règles du produit synchronisé*
Soit un terme de la définition précédente. L'ensemble des règles du produit synchronisé $\mathcal{R}_{\lambda^e}^{Sync}$ *est défini pour chaque propriété* φ *ci-dessous :*

$$Sync_\varphi(ltl(\alpha_\omega, \beta_M)) \rightarrow ltl(TestChange_{s \xrightarrow{\varphi} s'}(\alpha_\omega), Sys_\varphi(\beta_M))$$

où

- $\mathcal{R}_{\lambda^e}^{\omega}$ *est* $TestChange_{s \xrightarrow{\varphi} s'}(\alpha_\omega) \rightarrow \alpha'_\omega$
- $\mathcal{R}_{\lambda^e}^{M}$ *est une composition de quelques opérations de ce système :*

$$Sys_\varphi(\beta_M) \rightarrow Succ(Inv_\varphi(\beta'_M)$$

avec Succ le calcul des états suivants du système Inv_φ *le filtrage des états satisfaisant* φ *et* $\alpha_\omega, \alpha'_\omega, \beta_M, \beta'_M \in \mathcal{T}(\mathcal{F}_{bin}, \mathcal{X})$.

Nous illustrons le comportement du produit synchronisé par l'exemple suivant.

Exemple 32 *Revenons à l'exemple précédent. Après avoir vérifié des propriétés en utilisant* Inv_{t_i}*, nous construisons l'ensemble de règles* $\mathcal{R}_{\lambda^e}^{Sync}$ *pour le produit synchronisé :*

$$Sync_0(ltl(x,y)) \rightarrow ltl(TestChange_{t_0}(x), Inv_{t_0}(y)) \quad (7.108)$$
$$Sync_1(ltl(x,y)) \rightarrow ltl(TestChange_{t_1}(x), Inv_{t_1}(y)) \quad (7.109)$$
$$Sync_2(ltl(x,y)) \rightarrow ltl(TestChange_{t_2}(x), Inv_{t_2}(y)) \quad (7.110)$$
$$Sync_3(ltl(x,y)) \rightarrow ltl(TestChange_{t_3}(x), Inv_{t_3}(y)) \quad (7.111)$$
$$Sync_4(ltl(x,y)) \rightarrow ltl(TestChange_{t_4}(x), Inv_{t_4}(y)) \quad (7.112)$$

Test d'invariance $\mathcal{R}_{\lambda^e}^{Inv} \subseteq \mathcal{R}_{\lambda^e}^{M}$ *dans la figure 7.1 (a) :*

$$Inv_{t_0}(sys(x,y)) \rightarrow Test_1^p(Test_0^q(sys(x,y)) \quad (7.113)$$
$$Inv_{t_1} sys(x,y)) \rightarrow Test_1^q(sys(x,y) \quad (7.114)$$
$$Inv_{t_2}(sys(x,y)) \rightarrow Test_0^q(sys(x,y) \quad (7.115)$$
$$Inv_{t_3}(sys(x,y)) \rightarrow Test_0^p(Test_0^q(sys(x,y)) \quad (7.116)$$
$$Inv_{t_4}(sys(x,y)) \rightarrow Test_1^q(sys(x,y) \quad (7.117)$$

Proposition 9 *Logique LTL. Soit* f *une formule LTL. Soit l'automate de Büchi de cette formule* $\mathcal{R}_{\lambda^e}^{\omega}$*. Soit* $\mathcal{R}_{\lambda^e}^{M}$ *un système. Ainsi, s'il existe un système de ré-écriture* $\mathcal{R}_{\lambda^e}^{Sync}$ *tel que*

$$(\mathcal{R}_{\lambda^e}^{Sync} \cup \mathcal{R}_{\lambda^e}^{\omega} \cup \mathcal{R}_{\lambda^e}^{M})^\star(\{Sync(init)\}) \neq \emptyset$$

alors M *est vérifié par la formule* f.

Preuve 13 (Preuve succincte)
Selon la construction d'un tel système de ré-écriture $\mathcal{R}_{\lambda^e}^{Sync}$, il va suivre le comportement du système en synchronisant avec le chemin LTL de l'automate de Büchi. Ainsi,

$$(\mathcal{R}_{\lambda^e}^{Sync} \cup \mathcal{R}_{\lambda^e}^{\omega} \cup \mathcal{R}_{\lambda^e}^{M})^{\star}(\{Sync(init)\}) \neq \emptyset$$

c.à.d. M est vérifié par la formule f.

Nos systèmes de ré-écriture pour LTL sont applicables pour les modèles de réseaux de Petri. Comme déjà mentionné dans la section précédente, les réseaux de Petri comprend de bien maîtriser les règles de circulation pour chercher les places indiquées et tester leur marquage.

Nous revenons aux formules du problème des Philosophes de l'exemple 3 de la section 2.2.1 :

$$f_1 = \mathbf{F}(hasLeft_1 \wedge \mathbf{G}(\neg hasRight_1))$$
$$f_2 = (hasRight_1 \cup (Fork_1 \cup hasLeft_2))$$

Nous allons montrer ces propriétés de LTL pour les modèles de grande taille dans la partie expérimentale.

Résultats expérimentaux 15 *En utilisant notre outil (détaillé au chapitre 8), il est possible de prouver ces deux propriétés LTL du problème de 32 philosophes en environ d'une heure et demie pour la formule f_1 et en $251,871$ secondes pour la formule f_2.*

7.3.2 Logique temporelle arborescente CTL

Nous avons présenté dans la section 2.2 la logique temporelle arborescente CTL ainsi que ses opérateurs traditionnels de la logique booléenne, et des opérateurs temporels quantifiés tels que **EX**, **AX**, **EU**, **AU**... Le problème ici est de construire des structures de vérification symbolique en EFTRS à partir des spécifications en CTL.

Nous avons besoin des opérateurs primitifs $inv_{\varphi}, succ, pred$ pour des propriétés de CTL comme suit :

- Calcul des états accessibles $\mathcal{R}_{\lambda^e}^{\star}$: discuté dans la section précédente.
- Calcul arrière $\mathcal{R}_{\lambda^e}^{-1}$: calculable en changeant la direction des transitions des réseaux de Petri donnés (Post devient Pre et Pre devient Post). Ce calcul est déjà défini dans la définition 17 à la section 7.1.

Les formules CTL telles que EX et AX sur un terme u de $\mathcal{F}_{bin}$ sont calculées ci-dessous :

- $\mathbf{EX}_{\mathcal{R}_{\lambda^e}}(u) = \mathcal{R}_{\lambda^e}^{-1}(u)$
- $\mathbf{AX}_{\mathcal{R}_{\lambda^e}}(u) = \mathcal{R}_{\lambda^e}^{\star}(u) \setminus (\mathcal{R}_{\lambda^e}{}^{-1}(\mathcal{R}_{\lambda^e}^{\star}(u) \setminus u))$

Proposition 10 *Opérateurs primitifs de la logique CTL.*

Algorithme 9 Calcul de $EU_{\mathcal{R}_{\lambda^e}}(u, v)$

Précondition : $u, v \in \mathcal{T}(\mathcal{F}_{bin})$, $H \in \mathcal{F}_{NT}$ et $\mathcal{R}_{\lambda^e}$ un ensemble de règles d'EFTRSs

1: $a \leftarrow \emptyset$;
2: $b \leftarrow v$;
3: **tantque** $a \neq b$ **faire**
4: $\quad a \leftarrow b$;
5: $\quad b+ \leftarrow (b \cup (u \cap \mathbf{EX}_{\mathcal{R}_{\lambda^e}}(b)))$
6: **fin tanque**
7: **retourner** b

- $\mathbf{EX}_{\mathcal{R}_{\lambda^e}}(u) = \mathcal{R}_{\lambda^e}^{-1}(u)$
- $\mathbf{AX}_{\mathcal{R}_{\lambda^e}}(u) = \mathcal{R}^{\star}_{\lambda^e}(u) \backslash ({\mathcal{R}_{\lambda^e}}^{-1}(\mathcal{R}^{\star}_{\lambda^e}(u) \backslash u))$

Preuve 14 (Preuve succincte)

- $M, s \vDash EX\phi_1 \Longleftrightarrow \exists \pi, s_0 = s \Rightarrow M, s_1 \vDash \phi_1$. *Intuitivement, nous obtenons*

$$EX_{\mathcal{R}_{\lambda^e}}(u) = \mathcal{R}_{\lambda^e}^{-1}(u)$$

- $M, s \vDash AX\phi_1 \Longleftrightarrow \forall \pi, s_0 = s \Rightarrow M, s_1 \vDash \phi_1$. *Nous pouvons calculer autrement en utilisant* ***EX*** *comme suivant :*

$$\begin{aligned} & M, s \vDash AX\phi_1 \\ \Longleftrightarrow\ & s \neq s' \text{ avec } M, s \vDash EX\neg\phi_1 \\ \Longleftrightarrow\ & s \notin S = \{s' | \exists \pi, s_0 = s' \Rightarrow M, s_1 \vDash \neg\phi_1\} \end{aligned}$$

Effectivement, nous obtenons

$$AX_{\mathcal{R}_{\lambda^e}}(u) = \mathcal{R}^{\star}_{\lambda^e}(u) \backslash ({\mathcal{R}_{\lambda^e}}^{-1}(\mathcal{R}^{\star}_{\lambda^e}(u) \backslash u))$$

De plus, nous utilisons le calcul de point fixe pour les formules CTL telles que EU et AU.

- $\mathbf{EU}_{\mathcal{R}_{\lambda^e}}(u, v)$ est la petite solution de l'équation $x = v \cup (u \cap \mathbf{EX}_{\mathcal{R}_{\lambda^e}}(x))$. Voir l'algorithme 9.
- $\mathbf{AU}_{\mathcal{R}_{\lambda^e}}(u, v)$ est la petite solution de l'équation $x = v \cup (u \cap \mathbf{AX}_{\mathcal{R}_{\lambda^e}}(x))$. Voir l'algorithme 10.

Conjecture 4 *Les algorithmes 9, 10 et les formules CTL* **AU** *et* **EU**.

- $\mathbf{EU}_{\mathcal{R}_{\lambda^e}}(u, v)$ *calculé dans l'algorithme 9 est la petite solution de l'équation* $x = v \cup (u \cap \mathbf{EX}_{\mathcal{R}_{\lambda^e}}(x))$.
- $\mathbf{AU}_{\mathcal{R}_{\lambda^e}}(u, v)$ *calculé dans l'algorithme 10 est la petite solution de l'équation* $x = v \cup (u \cap \mathbf{AX}_{\mathcal{R}_{\lambda^e}}(x))$.

Algorithme 10 Calcul de $AU_{\mathcal{R}_{\lambda^e}}(u, v)$

Précondition : $u, v \in \mathcal{T}(\mathcal{F}_{bin})$, $H \in \mathcal{F}_{NT}$ et $\mathcal{R}_{\lambda^e}$ un ensemble de règles d'EFTRSs

1: $a \leftarrow \emptyset$;
2: $b \leftarrow v$;
3: **tantque** $a \neq b$ **faire**
4: $\quad a \leftarrow b$;
5: $\quad b+ \leftarrow (b \cup (u \cap \mathbf{AX}_{\mathcal{R}_{\lambda^e}}(b)))$
6: **fin tanque**
7: **retourner** b

Remarques 15 *Il faut noter que nous n'arrivons pas à trouver un système de réécriture fonctionnel pour modéliser les formules CTL* **AU** *et* **EU**.

Revenons au problème des Philosophes.

Exemple 33 *Problème à deux philosophes.*
Nous n'avons qu'un codage ayant un seul niveau pour deux philosophes. Comme déjà mentionné dans l'exemple 28, ce modèle contient deux états bloquants, l'un où chaque philosophe prenait la fourchette de gauche en attendant la fourchette de droite (qui étant également celle à gauche de son voisin droite, ne sera jamais libérée), et l'état symétrique où chaque philosophe prenait la fourchette de droite. Une autre propriété intéressante est l'état précédent de l'inter-blocage présenté par la formule CTL $\mathbf{EX}(deadlock)$.

EX *sur des états accessibles est des états bloquants :*

$$t_{sys_1} = sys(t^0_{local}, t^0_{local})$$

où

$$t^0_{local} = local(idle(0, \mathbf{waitleft}(\mathbf{1}, hasleft(0, waitright(0, \mathbf{hasright}(\mathbf{1}, fork(0, \bot))))))$$

$$t^1_{local} = local(idle(0, \mathbf{waitleft}(\mathbf{1}, hasleft(0, waitright(0, \mathbf{hasright}(\mathbf{1}, fork(0, \bot))))))$$

Et l'état symétrique :

$$t^0_{local} = local(idle(0, waitleft(0, \mathbf{hasleft}(\mathbf{1}, \mathbf{waitright}(\mathbf{1}, hasright(0, fork(0, \bot))))))$$

$$t^1_{local} = local(idle(0, waitleft(0, \mathbf{hasleft}(\mathbf{1}, \mathbf{waitright}(\mathbf{1}, hasright(0, fork(0, \bot))))))$$

$\mathbf{EX}^2$ *sur des états accessibles est un ensemble de quatre états listés ci-dessous : Le premier :*

$$t^0_{local} = local(idle(0, \mathbf{waitleft}(\mathbf{1}, hasleft(0, \mathbf{waitright}(\mathbf{1}, hasright(0, \mathbf{fork}(\mathbf{1}, \bot))))))$$

$$t^1_{local} = local(idle(0, \mathbf{waitleft}(\mathbf{1}, hasleft(0, waitright(0, \mathbf{hasright}(\mathbf{1}, fork(0, \bot))))))$$

Le deuxième :

$$t^0_{local} = local(idle(0, \mathbf{waitleft}(\mathbf{1}, hasleft(0, waitright(0, \mathbf{hasright}(\mathbf{1}, fork(0, \bot))))))$$

$$t^1_{local} = local(idle(0, \mathbf{waitleft}(\mathbf{1}, hasleft(0, \mathbf{waitright}(\mathbf{1}, hasright(0, \mathbf{fork}(\mathbf{1}, \bot))))))$$

Et le troisième :

$$t^0_{local} = local(idle(0, waitleft(0, \mathbf{hasleft}(\mathbf{1}, \mathbf{waitright}(\mathbf{1}, hasright(0, \mathbf{fork}(\mathbf{1}, \bot))))))$$

$$t^1_{local} = local(idle(0, \mathbf{waitleft}(\mathbf{1}, hasleft(0, \mathbf{waitright}(\mathbf{1}, hasright(0, fork(0, \bot))))))$$

Enfin le quatrième :

$$t^0_{local} = local(idle(0, \mathbf{waitleft}(\mathbf{1}, hasleft(0, \mathbf{waitright}(\mathbf{1}, hasright(0, fork(0, \bot))))))$$

$$t^1_{local} = local(idle(0, waitleft(0, \mathbf{hasleft}(\mathbf{1}, \mathbf{waitright}(\mathbf{1}, hasright(0, \mathbf{fork}(\mathbf{1}, \bot))))))$$

Nous allons montrer cette propriété de CTL avec des modèles de grande taille dans la partie expérimentale.

Résultats expérimentaux 16 *En utilisant notre outil (En détail dans le chapitre 8), nous sommes capables de détecter deux états bloquants ainsi que les états avant l'inter-blocage du problème de* 32.000 *philosophes en moins d'une heure en utilisant la formule CTL* **EX** *et* $\mathbf{EX}^2$ *sur des états accessibles.*

BILAN DU CHAPITRE 7

Finalement, nous pouvons conclure que dans le domaine de vérification, des EFTRSs sont capables de calculer des états accessibles du modèle, de calculer des propriétés de sûreté, de détecter des inter-blocages ainsi que de calculer des propriétés de la logique CTL ou de la logique LTL.

Quelques implémentations de modélisation et vérification par EFTRS seront listés sous forme de tableaux de résultat au chapitre suivant.

Chapitre 8

Benchmarks

RÉSUMÉ DU CHAPITRE 8

Dans le chapitre 4, nous avons présenté EFTRS préservant la puissance d'expression des systèmes fonctionnels et des techniques d'accélération des calculs aboutissant à un outil de vérification symbolique efficace. Dans le chapitre 5, nous avons montré comment nous pouvons évaluer les EFTRSs.

Dans le chapitre 6, nous avons montré la puissance des EFTRSs dans la modélisation de quelques protocoles pour que dans le chapitre 7, nous puissions vérifier certaines propriétés intéressantes de ces modèles par l'EFTRS.

Dans ce chapitre, nous allons comparer notre outil, d'une part avec des outils de ré-écriture tels que Timbuk, Maude et TOM sur des modèles arborescents, d'autre part avec des outils de vérification tels que SPIN, NuSMV, SMART et HSDD sur des modèles de réseaux de Petri, afin de montrer nos performances compétitives.

Voir ces benchmarks dans la figure 8.1. Les benchmarks sont divisés par les modèles arborescents et les réseaux de Petri. Nous nous focalisons sur les réseaux de Petri paramétrés soit par un nombre de jetons N, soit par un nombre de sous réseaux identiques (Voir cf. la section 2.1.2). Pour l'instant, notre technique est exclusivement définie pour des systèmes finis (et ainsi les systèmes de ré-écriture terminants).

FIG. 8.1 – Benchmarks traités avec nos outils

Enfin, certaines propriétés intéressantes du chapitre 7 sont bien vérifiées par notre outil.

Sommaire

8.1 Application sur des modèles arborescents

Dans cette section, nous comparons notre outil avec Timbuk, Maude et TOM. Les benchmarks sont les modèles arborescents tels que le protocole d'Arbitrage Arborescent (TAP), le protocole de Percolate (PP) et le protocole d'Election Arborescent (LEP). Les résultats sont obtenus sur un ordinateur 1.7GHz 1GB RAM.

8.1.1 Résultats expérimentaux en général

Pour ces expérimentations, nous avons adapté les techniques de LFP dans notre formalisme d'EFTRS. Bien sûr, les techniques d'accélération sont bien adaptées pour ce type de protocole où le nombre de configurations peut être très grand mais décidable. Les techniques d'accélération peuvent être implémentées automatiquement pour tous les exemples en utilisant les algorithmes optimisés au chapitre 4.

Ainsi, cela explique la différence du temps de calcul des accessibles entre les outils. Cependant, Maude et Timbuk peuvent être utilisés pour la vérification de systèmes infinis. Pour l'instant, notre technique est exclusivement définie pour des systèmes finis (et ainsi les systèmes de ré-écriture terminant).

Dans le tableau 8.1, Σ dénote l'étude de cas, N le nombre de processeurs aux arbres et $\#Confs$ la taille d'espace d'état d'explorer [1].

Σ	N	$\#Confs$	TRS			EFTRS	
			Timbuk	Maude	TOM	NoLFP	LFP
TAP	2^5	8,5 x 10^8	$> 3h$	$> 3h$	$> 3h$	$> 3h$	0,11
	2^{10}	1,9 x 10^{273}	-	-	-	-	0,20
	2^{20}	5,3 x 10^{278807}	-	-	-	-	0,26
	2^{400}	?	-	-	-	-	2,87
PP	2^5	1,1 x 10^4	0,013	1,479	610,19	0,128	0,02
	2^{10}	8,4 x 10^{135}	0,146	$> 3h$	$> 3h$	$> 3h$	0,05
	2^{20}	4,5 x 10^{139402}	0,268	-	-	-	0,10
	2^{500}	?	216,2	-	-	-	0,39
LEP	2^5	16	0,1	0,457	0.23	0,108	0,09
	2^{10}	512	0,2	$> 3h$	99,75	0,195	0,16
	2^{20}	5,2 x 10^5	0,3	-	$> 3h$	0,347	0,25
	2^{400}	1,2 x 10^{120}	333,8	-	-	393,9	138,7

TAB. 8.1 – Résumé

[1] Le nombre de configurations est indiqué par " ?" quand la bibliothèque *java.math.BigDecimal* que nous avons utilisé n'est pas capable de retourner le nombre de configurations dans un *time out* de trois heures. Effectivement, on ne le compte pas dans le temps de calcul des accessibles

Dans le dernière ligne du tableau ci-dessus, dans le protocole LEP, notre outil est obligé d'utiliser le calcul d'invariance $\mathcal{R}_{\lambda^e}^{Path_o}$ comme Maude et Timbuk. Cependant, nous avons besoin de moins d'une seconde pour le calcul d'invariance $\mathcal{R}_{\lambda^e}^{Path_c}$ de 2^{400} candidats.

8.1.2 TRSs, FTRSs et EFTRSs sur des modèles arborescents

Dans cette section, nous montrons en détail les résultats des benchmarks de notre EFTRS model-checker avec Timbuk, Maude. Les descriptions et modèles du TAP sont présentés dans les sections précédentes. Les descriptions et modèles du PP sont présentés en section 6.1.1. Les descriptions et modèles du LEP sont présentés en section 6.1.2. Les fichiers entrés pour Timbuk et pour Maude qui contient les systèmes de ré-écriture indiqués ci-dessus en ajoutant quelques options supplémentaires peuvent être trouvé à [Boichut 2010].

La technique de saturation lutte contre l'effet de la taille des données intermédiaires pour des structures très compactes à la BDD tels que MDD de [Ciardo 2003, Ciardo 2006] DDD de [Couvreur 2002], SDD de [Couvreur 2005], HSDD de [Thierry-Mieg 2008], [Thierry-Mieg 2004]. C'est la raison pour laquelle, nous décidons de reprendre cette idée pour les EFTRSs sous forme des techniques de point fixe locaux (LFPs).

1. **Protocole d'Arbitrage Arborescent.** Les descriptions et modèles sont présentés dans les sections précédentes.

   Nous commençons avec le TAP sans règle de demande en calculant le système classique $\mathcal{R}^{TAP}\backslash\mathcal{R}^{req}$, puis le système fonctionnel $\mathcal{R}_{\lambda}^{TAP}\backslash\mathcal{R}_{\lambda}^{req}$, et enfin le système élémentaire $\mathcal{R}_{\lambda^e}^{TAP}\backslash\mathcal{R}_{\lambda^e}^{req}$ sur le terme *total requesting* v.

   Dans le tableau 8.2 le temps d'attente est calculé en seconde. Il faut noter que l'augmentation exponentielle de la taille de modèle N nous conduit à une augmentation dramatique des états atteignables du modèle $\#Confs$.

N	#Confs	Timbuk	Maude	vérificateur EFTRS
2^5	43.264	2,931	3,672	N/A
2^{10}	-	$> 3h$	$> 3h$	N/A

TAB. 8.2 – TRS $\mathcal{R}^{TAP}\backslash\mathcal{R}^{req}$

N/A dans le tableau 8.2 et les tableaux suivants signifie que notre EFTRS model-checker ne supporte pas des systèmes classiques ainsi que des FTRSs.

N	#Confs	Timbuk	Maude	vérificateur EFTRS
2^5	43.264	$> 3h$	82,373	N/A
2^{10}	-	-	$> 3h$	N/A

TAB. 8.3 – FTRS $\mathcal{R}_{\lambda}^{TAP} \backslash \mathcal{R}_{\lambda}^{req}$

Le tableau 8.2 montre que l'outil Timbuk est efficace en TRS mais dans les tableaux 8.3 et 8.4, il n'est plus capable de reconnaître FTRSs et EFTRSs comme Maude.

N	Confs	Conversion		vérificateur EFTRS	
		Timbuk	Maude	NoLFP	LFP
2^5	43.264	$> 3h$	379,816	242,425	0,139
2^{10}	4,758 x 10^{16}	-	$> 3h$	$> 3h$	0,184
2^{20}	2,173 x 10^{63}	-	-	-	0,218
2^{50}	8,591 x 10^{383}	-	-	-	0,308
2^{150}	1,929 x 10^{3409}	-	-	-	0,634
2^{250}	8,648 x 10^{9444}	-	-	-	1,075
2^{350}	7,732 x 10^{18490}	-	-	-	1,418
2^{450}	1,379 x 10^{30547}	-	-	-	1,809

TAB. 8.4 – EFTRS $\mathcal{R}_{\lambda^e}^{TAP} \backslash \mathcal{R}_{\lambda^e}^{req}$

Dans le tableau 8.4 pour des EFTRSs, Maude a échoué très vite quand le temps d'attente du vérificateur EFTRS reste quasiment constant. Cela peut être expliqué par l'accélération de l'approche de LFP pour les EFTRSs

D'autre part, dans les tableaux 8.5, 8.6 et 8.7 nous réalisons également les benchmarks du modèle TAP avec les règles de demande en calculant le système classique $\mathcal{R}^{TAP}$, puis le système fonctionnel $\mathcal{R}_{\lambda}^{TAP}$, et enfin le système élémentaire $\mathcal{R}_{\lambda^e}^{TAP}$ sur le terme *total requesting* v.

N	#Confs	Timbuk	Maude	vérificateur EFTRS
2^5	8,539 x 10^8	$> 3h$	$> 3h$	N/A
2^{10}	-	-	-	N/A

TAB. 8.5 – TRS $\mathcal{R}^{TAP}$

Ce problème désormais devient trop difficile pour que ni Maude ni Timbuk puisse résoudre le modèle de taille $N = 2^5$. Voir cf. dans les tableaux 8.5, 8.6 et 8.7.

N	#Confs	Timbuk	Maude	vérificateur EFTRS
2^5	8,539 x 10^8	$> 3h$	$> 3h$	N/A
2^{10}	-	-	-	N/A

TAB. 8.6 – FTRS $\mathcal{R}_{\lambda}^{TAP}$

Dans le tableau 8.7, la technique de LFP toujours offre à notre vérificateur une performance surprise pour résoudre le modèle de taille $N = 2^{400}$.

N	Confs	Conversion		vérificateur EFTRS	
		Timbuk	Maude	NoLFP	LFP
2^5	8,539 x 10^8	$> 3h$	$> 3h$	$> 3h$	0,109
2^{10}	1,989 x 10^{273}	-	-	-	0,196
2^{20}	5,350 x 10^{278807}	-	-	-	0,256
2^{50}	?	-	-	-	0,430
2^{100}	?	-	-	-	0,789
2^{200}	?	-	-	-	1,453
2^{300}	?	-	-	-	2,137
2^{400}	?	-	-	-	2,866

TAB. 8.7 – EFTRS $\mathcal{R}_{\lambda^e}^{TAP}$

Enfin, nous présentons le tableau 8.8 pour le calcul d'invariance $(\mathcal{R}_{\lambda^e}^{\neg\phi})^\star((\mathcal{R}_{\lambda^e}^{TAP})^\star(\{Two\_t(Two\_t(t'_0))\})) \wedge (\mathcal{R}_{\lambda^e}^{\neg\phi})^\star((\mathcal{R}_{\lambda^e}^{TAP})^\star(\{No\_t(t'_0)\})) = \emptyset$. La technique de LFP offre également à notre vérificateur une performance surprise pour résoudre le modèle de taille $N = 2^{400}$.

N	Confs	vérificateur EFTRS	
		NoLFP	LFP
2^5	8,539 x 10^8	$> 3h$	0,173
2^{10}	1,989 x 10^{273}	-	0,227
2^{20}	5,350 x 10^{278807}	-	0,263
2^{50}	?	-	0,512
2^{100}	?	-	0,862
2^{200}	?	-	1,513
2^{300}	?	-	2,275
2^{400}	?	-	2,933

TAB. 8.8 – $(\mathcal{R}_{\lambda^e}^{\neg\phi})^\star((\mathcal{R}_{\lambda^e}^{TAP})^\star(\{Two\_t(Two\_t(t_0'))\})) \wedge (\mathcal{R}_{\lambda^e}^{\neg\phi})^\star((\mathcal{R}_{\lambda^e}^{TAP})^\star(\{No\_t(t_0')\})) = \emptyset$

2. **Protocole de Percolate.** Les descriptions et modèles sont présentés en section 6.1.1. N/A dans le tableau 8.9 signifie que notre vérificateur EFTRS ne supporte pas des systèmes classiques ainsi que des FTRSs.

N	#Confs	Timbuk	Maude	vérificateur EFTRS
2^5	11.047	0,138	1,479	N/A
2^{10}	8,445 x 10^{135}	24,179	$> 3h$	N/A
2^{20}	-	$> 3h$	-	N/A

TAB. 8.9 – TRS $\mathcal{R}^{PP}$

Dans cet exemple, Timbuk possède la meilleure performance soit TRSs soit EFTRSs. Voir cf. dans les tableaux 8.9 et 8.10.

N	#Confs	Timbuk	Maude	vérificateur EFTRS
2^5	11.047	0,380	19,041	N/A
2^{10}	8,445 x 10^{135}	2,956	$> 3h$	N/A
2^{20}	-	$> 3h$	-	N/A

TAB. 8.10 – FTRS $\mathcal{R}_\lambda^{PP}$

Comme le TAP, le tableau 8.11 nous dit que le temps d'attente de le vérificateur EFTRS reste quasiment constant quand les autres ont échoués très vite. Cela peut être expliqué seulement par l'accélération de l'approche de LFP pour les EFTRSs.

N	#Confs	Conversion		vérificateur EFTRS	
		Timbuk	Maude	NoLFP	LFP
2^5	11.047	0,165	19,041	0,128	0,018
2^{10}	8,445 x 10^{135}	0,555	$> 3h$	$> 3h$	0,053
2^{20}	4,508 x 10^{139402}	248,830	-	-	0,102
2^{50}	?	$> 1G$	-	-	0,139
2^{100}	?	-	-	-	0,183
2^{200}	?	-	-	-	0,293
2^{500}	?	-	-	-	0,396

TAB. 8.11 – EFTRS $\mathcal{R}^{PP}_{\lambda^e}$

3. **Protocole d'Election Arborescent.** Les descriptions et modèles sont présentés en section 6.1.2. La performance de l'approche LFP est montrée dans les tableaux 8.12, 8.13 et 8.14. Il faut noter qu'ici nous traitons le pire des cas, *c.*à.*d* le problème de taille ($N = 2^n$) aura $N/2 = 2^{(n-1)}$ possibilités d'élection. N/A dans ce tableau signifie que notre vérificateur EFTRS ne supporte pas les systèmes classiques.

N	#Confs	Timbuk	Maude	vérificateur EFTRS
2^5	16	0,159	0,014	N/A
2^{10}	512	7,595	71,981	N/A
2^{20}	5,243 x 10^5	160,661	$> 3h$	N/A
2^{50}	-	$> 3h$	-	N/A

TAB. 8.12 – TRS $\mathcal{R}^{LEP}$

Comme le TAP, les tableaux 8.12, 8.13 et 8.14 montrent que l'outil Timbuk est efficace en TRS mais il n'est plus capable de reconnaître FTRSs et EFTRSs comme Maude.

N	#Confs	Timbuk	Maude	vérificateur EFTRS
2^5	16	$> 3h$	0,457	N/A
2^{10}	512	-	$> 3h$	N/A
2^{20}	5,243 x 10^5	-	-	N/A

TAB. 8.13 – FTRS $\mathcal{R}^{LEP}_{\lambda}$

Les tableaux 8.12, 8.13 et 8.14 montrent également que notre EFTRS model-checker est toujours plus efficace que Maude.

N	Confs	EFTRS'		vérificateur EFTRS	
		Timbuk	Maude	NoLFP	LFP
2^5	16	$> 3h$	0,920	0,108	0,093
2^{10}	512	-	$> 3h$	0,195	0,159
2^{20}	5,243 x 10^5	-	-	0,347	0,256
2^{50}	5,629 x 10^{14}	-	-	1,728	0,956
2^{100}	6,338 x 10^{29}	-	-	8,820	3,194
2^{200}	8,035 x 10^{59}	-	-	56,233	18,355
2^{300}	1,019 x 10^{90}	-	-	170,067	57,430
2^{400}	1,291 x 10^{120}	-	-	393,969	138,699

TAB. 8.14 – EFTRS $\mathcal{R}_{\lambda^e}^{LEP}$

Nous comparons les deux systèmes d'invariance $\mathcal{R}_{\lambda^e}^{Path_o}$ et $\mathcal{R}_{\lambda^e}^{Path_c}$ pour dénombrer le nombre de possibilité d'élection dans le tableau 8.15 :

N	Confs	vérificateur EFTRS	
		$Path_o$	$Path_c$
2^5	16	0,093	0,039
2^{10}	512	0,159	0,078
2^{20}	5,243 x 10^5	0,256	0,129
2^{50}	5,629 x 10^{14}	0,956	0,241
2^{100}	6,338 x 10^{29}	3,194	0,245
2^{200}	8,035 x 10^{59}	18,355	0,302
2^{300}	1,019 x 10^{90}	57,430	0,428
2^{400}	1,291 x 10^{120}	138,699	0,529
2^{450}	1,453 x 10^{135}	$> 1G$	0,547

TAB. 8.15 – EFTRSs

Pour conclusion de cette section, les tableaux de résultats des benchmarks arborescents nous indiquent que notre outil est plus performant que Timbuk et Maude grâce aux techniques LFP.

8.2 Application sur des réseaux de Petri

Dans cette section, les benchmarks sont les codages de réseaux de Petri. Au début, nous comparons les codages de réseaux de Petri 1, 2 et 3 présentés dans la section 6.2. Ensuite, nous comparons notre outil avec des autres outils en utilisant le codage de réseaux de Petri hiérarchiques.

8.2.1 Simulations des réseaux de Petri

Après avoir analysé les codages de réseaux de Petri 1, 2 et 3 de la section 6.2, nous décidons d'implémenter les trois codages de réseaux de Petri pour les tester sur les cinq problèmes tels que le système de Kanban, le système de FMS, le problème des Philosophes, le protocole de Round Robin et le protocole de Slotted Ring. Les tests en EFTRS sont générés automatique pas notre générateur de manière indiquée dans le chapitre 6. Par conséquent, ils sont assez illisibles et donc assez difficiles à comprendre. Nous décidons de ne pas les mettre dans ce mémoire mais ils sont toujours disponibles dans notre site web [Boichut 2010].

8.2.1.1 Modèles paramétrés par un nombre de jetons N.

Deux représentants des modèles paramétrés par un nombre de jetons N sont le système de Kanban et le système de FMS.

Σ	N	#Confs	Codage 1	Codage 2	Codage 3
Kanban	2^1	18	0,612	2,715	1,349
	1	160	13,884	4,121	4,008
	2	4.600	27,484	9,392	10,606
	3	58.400	56,124	26,718	29,542
	4	454.475	240,279	754,193	291,359
	5	2,546 x 10^6	$> 3h$	$> 3h$	1837,512
	6	-	-	-	$> 3h$
FMS	1	120	57,134	32,882	6,541
	2	3.444	114,544	54,861	9,982
	3	48.590	$> 3h$	$> 3h$	297,462
	4	4,386 x 10^5	-	-	4.567,954
	5	-	-	-	$> 3h$

Tab. 8.16 – Modèles paramétrés par un nombre de jetons

Selon les résultats de deux benchmarks du tableau 8.16, le codage de réseau de Petri 1 a l'avantage avec le système de Kanban par rapport au codage 2 mais avec le système de FMS, il ne l'a plus. D'autre part, avec les deux systèmes, le codage 3 est effectivement plus efficace que les autres. Cela peut être expliqué par la structure

compacte du codage de réseau de Petri 3 et par le partage des règles de circulation ainsi que les règles primitives (pour retirer ou ajouter des jetons pour des places) par rapport aux codages de Petri 1 et 2.

8.2.1.2 Modèles paramétrés par un nombre de sous réseaux identiques.

Trois représentants des modèles paramétrés par un nombre de sous réseaux identiques sont le problème des Philosophes, le protocole de Slotted-Ring et le protocole de Round-Robin Mutex.

Σ	N	#Confs	Codage 1	Codage 2	Codage 3
Philosophes	2^1	18	0,612	2,715	1,349
	2^2	322	22,803	63,632	3,153
	2^3	1,036 x 10^5	2.666,321	$> 3h$	323,775
	2^4	-	$> 3h$	-	$> 3h$
Slotted-Ring	2^1	52	9,791	9,831	1,238
	2^2	5.136	$> 3h$	$> 3h$	19,421
	2^3	-	-	-	$> 3h$
Round-Robin Mutex	2^1	18	4,983	6,498	1,891
	2^2	144	520,180	219,508	3,658
	2^3	4.608	$> 3h$	$> 3h$	30,162
	2^4	-	-	-	$> 3h$

TAB. 8.17 – Modèles paramétrés par un nombre de sous réseaux identiques

Selon les résultats de deux benchmarks du tableau 8.17, le codage 1 a l'avantage avec le problème des Philosophes et le protocole de Slotted-Ring par rapport au codage 2, mais avec le protocole de Round-Robin Mutex, il ne l'a plus. D'autre part, avec les trois exemples, le codage 3 est effectivement plus efficace que les autres. Cela peut être expliqué par la structure compacte du codage de réseau de Petri 3 ainsi que par les partages des opérateurs des règles de retirer ou d'ajouter des jetons pour des places (les opérateurs *Minus* et *Plus*) par rapport au codage de réseau de Petri 2 (les opérateurs *Post* et *Pre*).

8.2.2 Simulation des réseaux de Petri hiérarchiques

Cette section contient les résultats dans la simulation des réseaux de Petri hiérarchiques. Après avoir analysé les codages de réseaux de Petri 1, 2 et 3, nous nous décidons d'implémenter les codages de réseaux de Petri 2 et 3 en intégrant avec le codage hiérarchique à la section 6.3 pour le tester sur le protocole de Slotted Ring, le protocole de Round Robin Mutex et le problème de Philosophes.

Nous comparons également notre outil avec quelques vérificateurs tels que SPIN [Holzmann 1986], NuSMV [Cimatti 2002], SMART [Ciardo 2002a, Ciardo 2002b,

Ciardo 2007] et HSDD [Thierry-Mieg 2008, Thierry-Mieg 2009]. Ces valeurs concernant les autres sont prises directement de [Ciardo 2005, Ciardo 2004, Couvreur 2005, Thierry-Mieg 2008, Thierry-Mieg 2009]. Quelques valeurs manquantes sont indiquées par "N/A".

Dans le tableau 8.18 Notre implémentation montre que l'outil de transformation automatique du réseau de Petri vers les EFTRSs fournit le même nombre de configurations pour des codages réseaux de Petri étudiés par rapport aux autres [2].

N	$\#Confs$	SPIN	NuSMV	SMART	HSDD	vérificateur EFTRS
Problème des Philosophes.						
2^5	1,15 x 10^{20}	N/A	0,4	0,01	0,00	0,65
2^{10}	1,02 x 10^{642}	N/A	-	1,8	0,00	1,25
2^{15}	2,1 x 10^{20544}	N/A	-	65,5	0,00	1,96
2^{20}	1,8 x 10^{657418}	N/A	-	-	0,01	2,77
2^{35}	?	N/A	-	-	0,02	135,4
Protocole de Slotted Ring.						
2^2	5,1 x 10^3	0,0	0,0	0,0	0,0	1,13
2^3	6,8 x 10^7	8,2	0,13	0,06	0,0	18,53
2^4	1,6 x 10^{16}	-	2.853	0,18	0,03	2.398
Protocole de Round Robin Mutex.						
2^4	2,3 x 10^6	43,0	0,34	0,01	0,0	3,32
2^6	2,6 x 10^{21}	-	11,7	0,09	0,2	30,91
2^8	6,6 x 10^{79}	-	-	7,04	1,0	1.115

TAB. 8.18 – Résumé

La comparaison entre des codages de réseaux de Petri hiérarchiques 2 et 3 ainsi qu'entre notre EFTRS model-checker et d'autres vérificateurs sont listées en détail ci-dessous.

1. **Problème des Philosophes.**

   Dans le tableau 8.19, le codage hiérarchique est capable de résoudre un problème de taille jusqu'à $N = 2^{35}$.

[2] Ces valeurs n'étaient pas mesurées par nous, mais elles sont directement reprises de [Ciardo 2005, Couvreur 2005, Ciardo 2004]. Ces résultats sont implémentés sur des machines plus performantes que le notre (*c.à.d.* soit plus de mémoire soit la vitesse de processeur plus rapide).

N	#Confs	Codage Petri hiérarchiques + Codage 2	+ Codage 3
2^5	1,115 x 10^{20}	0,839	0,654
2^{10}	1,023 x 10^{642}	2,366	1,251
2^{15}	2,094 x 10^{20544}	6,263	1,966
2^{20}	1,868 x 10^{657418}	12,306	2,779
2^{25}	?	15,365	3,661
2^{30}	?	18,941	4,782
2^{35}	?	137,094	135,407

TAB. 8.19 – Calcul des accessibles du problème des Philosophes

Dans le tableau 8.20, nous comparons également la taille N maximale possible de notre outil avec les autres vérificateurs [3] :

vérificateur EFTRS	SPIN	NuSMV	SDD	SMART	HSDD
3,43 x 10^{10}	N/A	200	5000	5000	2^{20000}

TAB. 8.20 – Comparaison des outils pour le problème des Philosophes

Dans le tableau 8.21, le test d'invariance pour vérifier qu'il n'y a aucune configuration telle que deux voisins prennent la même fourchette.

N	#Confs	Réseau de Petri hiérarchique
2^5	1,115 x 10^{20}	1,454
2^{10}	1,023 x 10^{642}	2,431
2^{15}	2,094 x 10^{20544}	6,931
2^{20}	1,868 x 10^{657418}	14,007
2^{25}	?	25,698
2^{30}	?	28,050
2^{35}	?	657,316

TAB. 8.21 – Test d'invariance pour le problème des Philosophes

2. **Protocole de Slotted Ring.**

   Dans le tableau 8.22, le codage hiérarchique est capable de résoudre un problème de taille jusqu'à $N = 2^4$.

   Dans le tableau 8.23, nous comparons également la taille N maximale possible de notre outil avec les autres vérificateurs :

N	#Confs	Réseau de Petri hiérarchique	
		+ Codage 2	+ Codage 3
2^1	52	1,412	1,020
2^2	5136	1,590	1,131
2^3	6,802 x 10^7	18,537	300,339
2^4	1,651 x 10^{16}	2398,754	$> 3h$

TAB. 8.22 – Calcul des accessibles du protocole de Slotted Ring

vérificateur EFTRS	SPIN	NuSMV	SDD	SMART	HSDD
16	6	15	50	300	200

TAB. 8.23 – Comparaison des outils pour le protocole de Slotted Ring

3. **Protocole de Round Robin Mutex** (présenté par [Graf 1996]).
   Dans le tableau 8.24, le codage hiérarchique est capable de résoudre un problème de taille jusqu'à $N = 2^8$.

N	#Confs	Réseau de Petri hiérarchique	
		+ Codage 2	+ Codage 3
2^1	18	1,419	1,228
2^2	144	1,722	1,435
2^3	4608	2,108	1,922
2^4	2,359 x 10^6	3,316	2,217
2^5	3,092 x 10^{11}	8,003	5,327
2^6	2,656 x 10^{21}	30,914	24,282
2^7	9,800 x 10^{40}	171,321	205,275
2^8	6,696 x 10^{79}	1115,704	1278,272

TAB. 8.24 – Calcul des accessibles du protocole de Round Robin Mutex

Dans le tableau 8.25, nous comparons également la taille N maximale possible de notre outil avec les autres vérificateurs :

Nous intéressons également à un test d'invariance φ pour vérifier qu'il y a une seule ressource dans le codage dans le tableau 8.26. Nous donc essayons un test de négation d'invariance $\neg\varphi$ pour chercher une configuration dans la quelle, au moins deux processus utilisent cette ressource en même temps ou aucun la touche.

[3] Quelques cases manquants sont indiqués par "N/A".

vérificateur EFTRS	SPIN	NuSMV	SDD	SMART	HSDD
256	16	60	N/A	1100	1000

TAB. 8.25 – Comparaison des outils pour le protocole de Round Robin Mutex

Les résultats des benchmarks nous indiquent également que notre vérificateur est plus performant que SPIN et NuSMV grâce aux techniques LFP. Quant à SMART, nous sommes plus performants sur le problème des Philosophes mais moins bon pour les deux autres protocoles. Enfin, notre vérificateur EFTRS est encore moins performant que HSDD.

N	#Confs	Réseau de Petri hiérarchiques
2^1	18	1,727
2^2	144	1,775
2^3	4608	2,177
2^4	2,359 x 10^6	3,386
2^5	3,092 x 10^{11}	9,138
2^6	2,656 x 10^{21}	32,143
2^7	9,800 x 10^{40}	190,291
2^8	6,696 x 10^{79}	1.293,921

TAB. 8.26 – Test d'invariance pour le protocole de Round Robin Mutex

8.3 Application sur l'inter-blocage et la vérification des logiques temporelles

8.3.1 Application sur l'inter-blocage

Considérons le problème des Philosophes. Comme déjà discuté en chapitre précédent, ce modèle a deux états bloquants, l'un où chaque philosophe prenait la fourchette de gauche en attendant la fourchette de droite (qui étant également celle de gauche de son voisin de droite, ne sera jamais libérée), et l'état symétrique où chaque philosophe prenait la fourchette de droite.

Notre outil est capable de détecter l'ensemble de ces deux états bloquants (dénoté *deadlock*) de ce problème ayant une taille N j'usqu'à 16 pour les codages ordinaires et sans surprise j'usqu'à 32 mille pour le codage hiérarchique. Les résultats en détail sont listés dans le tableau 8.27.

Par cet exemple, nous montrons encore une fois l'efficacité du codage de réseau de Petri hiérarchique par rapport aux codages ordinaires.

N	Codage du réseau de Petri			
	1	2	3	hiérarchique
2^1	0,780	2,976	1,372	1,275
2^2	29,040	66,970	3,261	1,303
2^3	2.945,884	$> 3h$	326,235	1,607
2^4	$> 3h$	-	$> 3h$	1,808
2^5	-	-	-	2,325
2^{10}	-	-	-	76,826
2^{15}	-	-	-	2.243,702
2^{20}	-	-	-	$> 3h$

TAB. 8.27 – Inter-blocage

8.3.2 Application sur la logique LTL

Dan le problème des Philosophes, nous nous sommes intéressés à un comportement tel que : *A un moment dans le futur, un philosophe prend la fourchette de gauche mais il n'aura jamais celle de droite* en utilisant la formule de LTL ci-dessous :

$$f_1 = \mathbf{F}(p_1.hasL \wedge \ \mathbf{G}(\neg p_1.hasR))$$

Voir l'automate de Büchi de la figure 2.4a).

Nous nous sommes également intéressés à un comportement tel que : *A un moment dans le futur, au début, un philosophe prend la fourchette à droite, puis il la libère, et finalement, cette fourchette est prise par son voisin de droite.* en utilisant la formule de LTL suivante :

$$f_2 = (p_1.hasR \cup (p_1.Fork \cup p_N.hasL))$$

Voir l'automate de Büchi de la figure 2.4b).

Une première implémentation de vérificateur de LTL basée sur EFTRS pour quelques formules de LTL est montrée dans le tableau 8.28.

Formule	N	Confs	Temps de calcul (s)
f_1	2^3	28.657	3,139
	2^4	2,971 x 10^9	46,584
	2^5	3,194 x 10^{19}	5.486,060
f_2	2^3	1,036 x 10^5	1,777
	2^4	1,074 x 10^{10}	8,936
	2^5	1,155 x 10^{20}	251,871

TAB. 8.28 – Quelques formules de LTL

8.3.3 Application sur la logique CTL

Dans le problème des Philosophes, une propriété intéressante est l'état précédent de l'inter-blocage en utilisant calculer la formule CTL **EX**(*deadlock*).

Notre outil est capable de calculer la formule CTL **EX**(*deadlock*) du problème ayant une taille N j'usqu'à 16 pour les codages ordinaires et sans surprise j'usqu'à 32 mille pour le codage hiérarchique. Les résultats détaillés sont listés dans le tableau 8.29 :

N	Avant Inter-blocage	Réseau de Petri			
		1	2	3	hiérarchique
2^1	4	0,820	3,505	1,674	1,431
2^2	8	30,023	67,856	3,290	1,532
2^3	16	3.175,431	$> 3h$	329,942	1,802
2^4	32	$> 3h$	-	$> 3h$	2,145
2^5	64	-	-	-	2,580
2^{10}	2.048	-	-	-	77,430
2^{15}	65.536	-	-	-	2.351,461
2^{20}	-	-	-	-	$> 3h$

Tab. 8.29 – Formule CTL **EX**(*deadlock*)

Notre outil est également capable de calculer la formule CTL **E**(*deadlock* **U** *init*) du problème ayant une taille N j'usqu'à 16 pour les codages ordinaires et sans surprise j'usqu'à 32 mille pour le codage hiérarchique. Les résultats détaillés sont listés dans le tableau 8.30 :

N	Codage du réseau de Petri			
	1	2	3	hiérarchique
2^1	0,815	2,997	2,508	1,503
2^2	29,313	70,282	3,988	1,641
2^3	2.991,374	$> 3h$	346,726	1,783
2^4	$> 3h$	-	$> 3h$	2,721
2^5	-	-	-	2,965
2^{10}	-	-	-	83,644
2^{15}	-	-	-	2.658,787
2^{20}	-	-	-	$> 3h$

Tab. 8.30 – Formule CTL **E**(*deadlock* **U** *init*)

Par cet exemple, nous montrons encore une fois l'efficacité du codage de réseau Petri hiérarchique par rapport aux codages de réseaux de Petri ordinaires.

BILAN DU CHAPITRE 8

Dans ce chapitre, nous avons comparé notre outil, d'une part avec des outils de ré-écriture tels que Timbuk, Maude et TOM, d'autre part avec des outils de vérification tels que SPIN, NuSMV, SMART, HSDD. Pour l'instant, notre technique est exclusivement définie pour des systèmes finis (et ainsi les systèmes de ré-écriture terminés).

Dans la modélisation, d'une part, les benchmarks sous forme d'un modèle arborescent nous démontrent que notre outil est plus performant que Timbuk, Maude et TOM grâce aux points fixe locaux. D'autre part, les benchmarks d'origine d'un réseau de Petri nous démontrent également que notre vérificateur est plus performant que SPIN et NuSMV. Quant à SMART, nous sommes plus performants sur le problème des Philosophes mais moins bien pour les deux autres protocoles. Cependant, notre EFTRS vérificateur est encore moins performant que l'outil de HSDD.

Dans la vérification, notre outil est capable de vérifier des invariants ainsi que des propriétés temporelles. Actuellement, l'implémentation du vérificateur de LTL n'est pas encore bonne comme ceux de CTL, de l'inter-blocage et d'invariance parce que le système élémentaire concernant au produit synchronisé entre l'automate de Büchi et le système n'est pas encore bien optimisé.

Quatrième partie

Partie IV : Vers un langage pour la Vérification par EFTRS

Chapitre 9

Langage pour la Vérification

RÉSUMÉ DU CHAPITRE 9

Ce chapitre est avant tout une perspective sur le thème de la vérification de modèles de haut niveau. Nous proposons un *Langage pour la Vérification* (Language for Verification, LfV) : un langage formel plus simple et plus efficace pour la vérification des systèmes répartis.

Sommaire

9.1 Introduction

Le développement du LfV s'est déroulé dans le cadre du projet MORSE (*Méthodes et Outils pour la Vérification Formelle de Systèmes Interopérables Embarqués critiques*) [Bréant 2004], un projet RNTL de coopération entre l'industrie et les universités. Le projet MORSE définit une démarche méthodologique centrée sur UML pour aider des ingénieurs à construire une application dont le comportement sera déterministe, puis à produire l'application correspondante.

Le projet MORSE couvre donc les points suivants :

1. la mise en place d'une méthodologie adaptée au domaine d'application considéré : les systèmes de drones (avions sans pilotes).
2. la définition d'un langage de spécification, adapté au domaine et aux besoins de génération de code et de vérification en aval : le *Langage de spécification pivot* LfP permet de spécifier le comportement du système à l'aide de processus séquentiels communiquant via des médias.
3. la mise en place de techniques de vérification du bon comportement du système : le LfV et la structure symbolique DDD.
4. la réalisation d'un générateur automatique de programmes pour produire le système rapidement et sans dérive par rapport à la spécification : le LfP et le langage de programmation C^{++}.

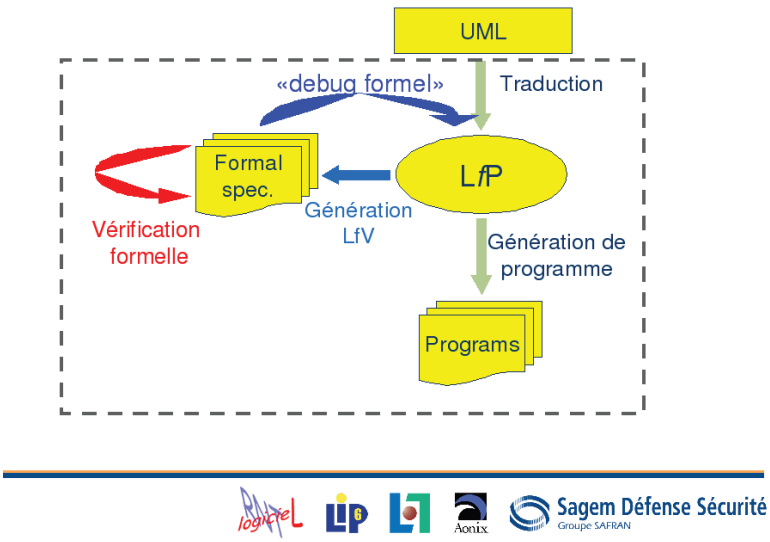


FIG. 9.1 – Projet MORSE : a). Méthodologie b). Partenaires

Notre ambition est de développer LfV, un langage formel plus simple et plus efficace pour la vérification des systèmes répartis basés sur le *Langage de spécification pivot* LfP. Un système de LfV est représenté par le comportement de processus simples communiquant via des *buffers* de messages partagés.

9.2 Descriptions générales du LfV

On simplifie le problème de la vérification en utilisant la même forme d'interaction pour la communication des données et l'appel de procédure distante. Cette approche est celle de SPIN [Holzmann 1991, Holzmann 1986] mais en plus simple car l'interaction entre des processus de LfV n'est que la transmission des messages élémentaire.

Chaque processus de LfV est composé de deux éléments principaux : la liste des variables (A) et la liste des transitions (B). L'index de chaque processus est généré automatiquement par le système.

(A) Tous les types de variable sont définis dans la figure 9.2. Chaque variable locale de LfV a un type $STRING, BOOL, INT$ ou $PROCESS$ et est utilisée uniquement dans ce processus. Les variables globales de LfV entre deux processus ou *buffer* des messages ayant une type $Fifo$ (corresponde aux *binders* en LfP) sont partagés par des processus de LfV. D'autre part, nous avons réservé une type spéciale pour le variable d'état de LfV.

(B) Une transition est sous forme :

```
state1 |-  séquence des instructions; -> state2;
```

où $state_1$ et $state_2$ sont des variables d'état. Dans le corps de la transition, on n'accepte que une séquence des instructions. Alors avec la structure de sélection et de répétition, nous devons un ensemble de transitions. Une instruction spéciale est un commande d'interaction ou une création dynamique des processus. La création dynamique des processus est représentée comme suivante :

```
i |- p := new Process( ... ); -> j;
```

L'interaction d'écrire et de lire un message msg dans les *buffers* messages itf : $itf@send(msg)$ ou $itf@recv(msg)$. Le message msg est une liste d'éléments de données ayant une taille indéfinie. Chaque élément a n'import quel type. Cette technique donne la même forme d'interaction pour la communication des données et l'appel de procédure en distance.

On a deux mécanismes d'interaction : synchronique and asynchronique :

Le mécanisme synchronique : l'opération *send* est bloquée jusqu'à la disponibilité du fifo. Le processus demande cette opération sera donc bloqué et reviendra quand le fifo est disponible et *send* peut être exécuté.

```
i |- [ itf.size <  itf.max ] ;  itf@send (msg);  -> j;
```

Le mécanisme asynchronique : l'opération *send* est toujours possible (*c.à.d.* l'opération est jamais en inter-blocage), mais quand le fifo est pleine, les messages envoyés sont simplement perdu.

```
i |- [ itf.size <  itf.max ] ;  itf@send (msg);  -> j;
i |- [ itf.size >= itf.max ] ;                   -> j;
```

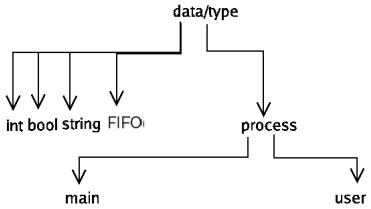


FIG. 9.2 – Types de données de LfV

Par exemple, l'interaction d'écriture un message est représenté comme suivante :

```
i|- itf@recv (msg);
    [ (msg.size = 5)              && (msg[0].dataType = PROCESS) && ...
      (msg[2].dataType = STRING) && (msg[2] = "rpc")             && ... ] -> j;
```

L'état du processus est changé de $state_i$ à $state_j$, tous les instructions sont exécutable seulement si toutes les conditions sont satisfaites. Par exemple, le processus reçoit un message msg (ligne code 1) seulement si la condition dans les ligne code 2 et 3 est vraie.

Un système de LfV a un serveur (s) et deux clients (c_1 and c_2) communiqués par deux *fifos* (itf_cs ayant direction du client au serveur et itf_sc du serveur au client).

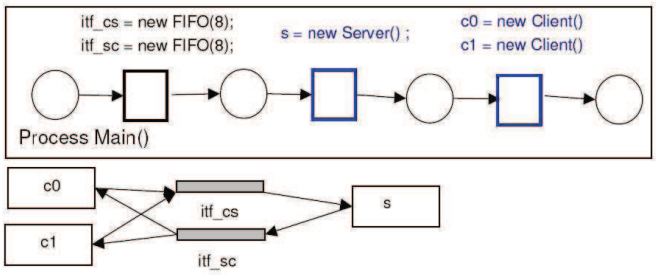


FIG. 9.3 – Création des processus en LfV

Toute la communication de données de LfV est codée par le processus de source et puis décodée par le processus de destination. Le reste du corps de message est transparent aux processus intermédiaires qui fonctionne comme un transporteur de données entre le processus de source et le processus de destination. L'interaction des processus de LfV est dessinée dans la figure 9.4.

En bref, nous avons réussit à construire le LfV-DDD vérificateur qui fonctionne sur le *prototype 1* du projet MORSE. Notre vérificateur fonctionne bien également sur des études de cas simple mais typique (Par exemple : le problème *Exclusive*, le problème *Alternative bit*, etc). D'autre part, nous présentons un langage d'expression des propriétés pour analyser des états accessibles et vérifier si ces résultats sont satisfaits au modèle.

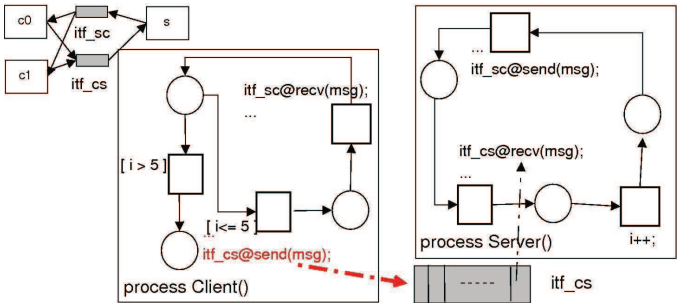


FIG. 9.4 – Interaction entre des processus en LfV

Les résultats sont présentés en [Nguyen 2006]. Voir cf. notre étude de cas de LfV dans l'annexe B.

Le principe du LfV est comme le langage PROMELA de l'outil SPIN [Holzmann 1986], [Holzmann 1991]. Cependant, LfV plus simple que SPIN parce que l'interaction processus de LfV n'est que la transmission des messages élémentaire. Malgré sa simplicité, LfV est capable de simuler énormément de modèles en réalité.

9.3 LfV pour la vérification symbolique

Les DDDs sont utilisés pour l'implémentation d'un vérificateur du langage de modélisation de PROMELA [Beaudenon 2005]. Nous les avons également choisi pour LfV.

Les données en DDD doivent être organisées raisonnablement pour la présentation des états accessibles du système global.

Dans notre modèle, un état du système de LfV est codé comme la concaténation d'une liste des processus et une liste des fifos comme dans l'annexe C.

Nous pouvons coder tous les types de données en utilisant une liste de valeurs d'entier. Par exemple : Une variable BOOL est codé par un seul entier mais une chaîne de caractère (type STRING) est codée par une liste des entiers (un entier pour chaque caractère de la chaîne).

Les homomorphismes (opérations) en DDD doivent être capables de générer des états accessibles du système global. Chaque comportement du système est représenté par une homomorphisme ou un ensemble de homomorphismes. Quelques homomorphismes statiques ordinaires (tels que *Affectation* des valeurs aux variables, *Test* si une variable satisfait une telle conditions, *Interaction* des processus, etc).

De plus, nous somme réussis à construire quelques homomorphismes dynamiques tels que la création des processus et Fifos, la manipulation des messages, etc. Par exemple : Le homomorphisme de création des processus est comme le suivant (Où d est une structure DDD du nouveau processus).

D'autre part, on propose également un homomorphisme spécial *Exec* qui exécute

dans tous les processus ayant type t (on appelle un groupe ayant type t) en traversant les DDDs une seule fois pour chaque groupe par rapport une pour chaque processus. Cette technique peut économiser une énorme espace de mémoire et accélérer le calcul.

Les résultats sont présentés en [Nguyen 2007]. Actuellement, cet outil supporte seulement des variables type INT, STRING, BOOL.

BILAN DU CHAPITRE 9

Ce chapitre est avant tout une perspective sur le thème de la vérification de modèles de haut niveau. Après avoir proposé un *Langage pour la Vérification* (Language for Verification, LfV) : un langage formel plus simple et plus efficace pour la vérification des systèmes répartis, nous allons montrer comment traduire LfV en EFTRS dans le chapitre suivant.

Chapitre 10

Vers un langage pour la Vérification par EFTRS

RÉSUMÉ DU CHAPITRE 10

Dans le chapitre 9, sur le thème de la vérification de modèles de haut niveau, nous avons proposé un *Langage pour la Vérification* (Language for Verification, LfV) : un langage formel plus simple et plus efficace pour la vérification des systèmes répartis.

Dans ce chapitre, nous développons les principes de la traduction d'un modèle du LfV en EFTRS. Cela nous conduit à l'idée d'un système fonctionnel élémentaire paramétré, appelé PEFTRS. Nous montrons que PEFTRS ont le même pouvoir d'expression que les EFTRS mais faciliter l'écriture de grands FTRS.

Nous nous focalisons sur deux aspects que sont la simulation de PEFTRS par des règles d'EFTRSs à l'aide des invariants simples et le pouvoir d'expression du langage de PEFTRS.

Sommaire

10.1 Introduction

Un terme de EFTRS est un arbre dont le côté gauche est réservé pour des processus et le côté gauche est pour des structures fifos.

$$model(t_{process}, t_{fifo})$$

où $t_{process}$ et t_{fifo} sont respectivement le codage des processus et le codage des fifos. Voir un exemple dans la figure 10.1.

Ces codages sont plus intéressants que DDD grâce à la structure hiérarchique qui permet les opérations sont plus flexibles et efficaces.

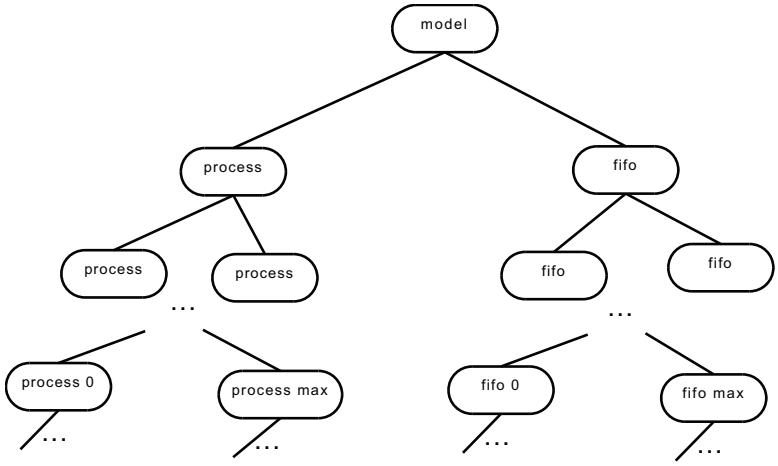


Fig. 10.1 – Modèle de LfV en FTRS

Nous allons présenter deux modèles de LfV en EFTRS :

1. Systèmes élémentaires simples : L'utilisation des règles d'EFTRSs.
2. Systèmes élémentaires paramétrés PFTRSs : Nous focalisons deux aspects qui sont la simulation de PEFTRS par des règles d'EFTRSs à l'aide des invariants et l'expression du langage de PEFTRS.

10.2 Systèmes élémentaires simples

Nous utilisons des règles d'EFTRSs pour simuler des opérateurs primitifs de LfV, pour traiter des structures de taille infinie et pour la communication des messages entre des processus.

EFTRS règles pour simuler des opérateurs primitifs de LfV :

– TestValue $(a == i)?$:
$Test_{(a==i)?}(b(x,y)) \rightarrow b(x, Test_{(a==i)?}(y))$,
$Test_{(a==i)?}(b(x,y)) \rightarrow b(Test_{(a==i)?}(x), y)$,

$Test_{(a==i)?}(a(x,y)) \rightarrow a(Test_i(x), y)$.

- TestVar $(a == b)?$:
  $Test_{(a==b)?}(x) \rightarrow Test_{(a==i)?}(Test_{(b==i)?}(x))$ pour tout $i \in DOM(a) \cap DOM(b)$.

- SetValue $a = i | i \in DOM(a)$:
  $Set_{a=i}(b(x,y)) \rightarrow b(x, Set_{a=i}(y))$,
  $Set_{a=i}(b(x,y)) \rightarrow b(Set_{a=i}(x), y)$,
  $Set_{a=i}(a(x,y)) \rightarrow a(TestChange_{ji}(x), y)$ pour tout $j \in DOM(a)$.

- Value $a++$:
  $Set_{a++}(b(x,y)) \rightarrow b(x, Set_{a++}(y))$,
  $Set_{a++}(b(x,y)) \rightarrow b(Set_{a++}(x), y)$,
  $Set_{a++}(a(x,y)) \rightarrow a(TestChange_{i \rightarrow i+1}(x), y)$ pour tout $i, i+1 \in DOM(a)$.

- Value $a--$:
  $Set_{a--}(b(x,y)) \rightarrow b(x, Set_{a--}(y))$,
  $Set_{a--}(b(x,y)) \rightarrow b(Set_{a--}(x), y)$,
  $Set_{a--}(a(x,y)) \rightarrow a(TestChange_{i \rightarrow i-1}(x), y)$ pour tout $i, i-1 \in DOM(a)$.

- SetVar $a = b$:
  $Set_{a=b}(x) \rightarrow Set_{a=i}(Test_{(b==i)?}(x))$ pour tout $i \in DOM(b)$.

- Permute $a <-> b$:
  $Set_{a<->b}(x) \rightarrow Set_{b=i}(Set_{a=b}(Test_{(a==i)?}(x)))$ pour tout $i \in DOM(a)$.

- SetVar $a = b$ *op* c où *op* $\in \{+, -, *, /\}$:
  $Set_{a=(b\ op\ c)}(x) \rightarrow Set_{a=(i\ op\ j)}(Test_{(b==i)?}(Test_{(c==j)?}(x)))$ pour tout $i \in DOM(b)$ et $j \in DOM(c)$.

Les règles de traitement des structures de taille infinie telles que des fifos et des messages sont réalisées au côté droit, au niveau de feuille du processus au côté gauche (pour des messages).

- CreateFifo
- TestFifo
- InsertFifo
- DeleteFifo

Les règles de communication des messages entre des processus sont comptées comme les déplacement des messages (codés sous forme des sous-termes) entre les côtés gauche et droit.

- *send* : Déplacement des sous-termes du côté gauche au côté droit.
- *recv* : Déplacement des sous-termes du côté droit au côté gauche.

Nous voulons construire les règles de traitement des structures de taille infinie et les règles de communication dans notre vérificateur en les testant sur nos prototypes.

Remarques 16 *La capacité d'expressivité du FTRS n'est pas puissante comme DDD. Nous sommes obligés de générer de nombreuses règles autant de symboles apparaissent.*

10.3 Systèmes élémentaires paramétrés PFTRSs

Nous montrons que les systèmes paramétrés PFTRSs conviennent très bien au LfV et ils peuvent nous offrir des opérations pour manipuler des données mieux qu'en EFTRS et DDD.

Nous focalisons deux aspects qui sont la simulation de PEFTRS par des règles d'EFTRSs à l'aide des invariants et l'expression du langage de PEFTRS.

10.3.1 Systèmes élémentaires paramétrés PFTRSs à l'aide des invariants

Nous définissons un système fonctionnel spécial expressive comme les fonctionnels en utilisant quelques paramètres supplémentaires avec l'aide des invariants simples. Chaque règle paramétrée est équivalente à un ensemble de règles fonctionnels. Autrement dit, une règle paramétrée est une bibliothèque de règles d'EFTRS.

C'est évident que l'on n'avance pas beaucoup avec cette représentation mais non plus si on ajoute des caractères magiques.

Par exemple, $F(f(x,y)) \rightarrow \alpha \; si \; f = a$ et $F(f(x,y)) \rightarrow \beta \forall f \in \mathcal{F}_{bin}, f \neq a$ où f est la représentation pour n'importe quel symbole binaire dans $\mathcal{F}_{bin}$:

$$\mathcal{R}_{\lambda^e} = \left\{ \begin{array}{l} F(f(x,y)) \rightarrow \alpha \; si \; f = a \\ F(f(x,y)) \rightarrow \beta \; si \; f = b \\ F(f(x,y)) \rightarrow \beta \; si \; f = c... \end{array} \right\}.$$

peut être écrit par des invariants comme suivant en FTRS :

$$\mathcal{R}_{\lambda}^{par} = \left\{ \begin{array}{l} F(x) \rightarrow F_1(Inv^a_{Inv_{all},Inv_{all}}(x)) \\ F_1(x) \rightarrow \alpha \\ F(x) \rightarrow F_2(\neg Inv^a_{Inv_{all},Inv_{all}}(x)) \\ F_2(x) \rightarrow \beta \end{array} \right\}.$$

peut être écrit par des invariants comme suivant en EFTRS :

$$\mathcal{R}_{\lambda^e}^{par} = \left\{ \begin{array}{l} F(f(x,y)) \rightarrow F_1(Inv^a_{Inv_{all},Inv_{all}}(f(x,y))) \\ F_1(f(x,y)) \rightarrow \alpha \\ F(f(x,y)) \rightarrow F_2(\neg Inv^a_{Inv_{all},Inv_{all}}(f(x,y))) \\ F_2(f(x,y)) \rightarrow \beta \end{array} \right\}.$$

De plus, nous définissons également la sémantique des symboles...

Définition 23 *Soit un ordre $\gg$ sur $\mathcal{F}_{bin}$ tel que*

$$f_{max} \gg ... \gg f_i \gg f_j \gg ... \gg f_1 \gg f_0$$

.

f^+ (f^-) sont définis comme le symbole suivant (précédant) dans l'ordre :
– si $f = f_j, j < max$ alors $f^+ = f_i$
– si $f = f_i, j > 0$ alors $f^- = f_j$

Par exemple : $... \gg 2 \gg 1 \gg 0$ ou $... \gg c \gg b \gg a...$
– $0^+ = 1$, $1^+ = 2$, $a^+ = b$, $b^+ = c...$
– $1^- = 0$, $2^- = 1$, $b^- = a$, $c^- = b...$
Nous avons défini des éléments nécessaires pour des Systèmes élémentaires paramétrés.

10.3.2 Systèmes élémentaires paramétrés (PFTRSs)

Dans, cette section, nous présentons les systèmes paramétrés PFTRSs. Nous montrons qu'ils conviennent très bien au LfV et ils peuvent nous offrir des opérations pour manipuler des données mieux qu'en DDD.

Définition 24 *Soit $\mathcal{F}_{par}$ un ensemble disjoint de $\mathcal{F}_{bin}$ et de $\mathcal{F}_{NT}$. Une règle de FTRS paramétrée est définie comme une fonction $\mathcal{F} : \mathcal{T}(\mathcal{F}_{bin}) \mapsto 2^{\mathcal{R}_{par}}$ avec $\mathcal{F} \in \mathcal{F}_{par}$ et $\mathcal{R}_{par} = \mathcal{T}(\mathcal{F}_{bin} \cup \mathcal{F}_{NT}, \mathcal{X}) \times \mathcal{T}(\mathcal{F}_{bin} \cup \mathcal{F}_{NT} \cup \mathcal{F}_{par}, \mathcal{X})$.*
Nous notons cette règle par $\mathcal{F}_{\mathcal{T}(\mathcal{F}_{bin})}$.

Nous remarquons que $\mathcal{R}_{par}$ est $\mathcal{R}$ de FTRS si cet ensemble de règles ne contient aucun symbole de $\mathcal{F}_{par}$.

Règles de PFTRS pour des opérateurs primitifs de LfV :
– $TEST_{a(\perp,\perp)}$ (ou $Inv^a_{Inv_{all},Inv_{all}}$) est un ensemble de règles comme suivants :

$$\mathcal{R}^{par}_{\lambda^e} = \left\{ \; Test(a(x,y)) \rightarrow a(x,y) \; \right\}.$$

– $SET_{a(\perp,\perp)}$ est un ensemble de règles comme suit :

$$\mathcal{R}^{par}_{\lambda^e} = \left\{ \; Set(f(x,y)) \rightarrow a(x,y) \; \right\}.$$

– $TESTVALUE_{a(\perp,i(\perp,\perp))}$ est un ensemble de règles comme suit :

$$\mathcal{R}^{par}_{\lambda^e} = \left\{ \begin{array}{l} TestValue(f(x,y)) \rightarrow A(\mathbf{TEST_{a(\perp,\perp)}}(f(x,y))) \\ A(f(x,y)) \rightarrow a(\mathbf{TEST_{i(\perp,\perp)}}(x), y) \\ \\ TestValue(f(x,y)) \rightarrow B(\neg\mathbf{TEST_{a(\perp,\perp)}}(f(x,y))) \\ B(f(x,y)) \rightarrow f(x, TestValue(y)) \\ B(f(x,y)) \rightarrow f(TestValue(x), y) \end{array} \right\}.$$

On fait un appel d'un autre système d'EFTRS paramétré $\mathbf{TEST}_{\mathbf{i}(\bot,\bot)}$

– $TESTVAR_{a(\bot,b(\bot,\bot))}$ est un ensemble de règles comme suit :

$$\mathcal{R}^{par}_{\lambda^e} = \left\{ \begin{array}{l} TestVar(f(x,y)) \to A(\mathbf{TEST}_{\mathbf{a}(\bot,\bot)}(f(x,y))) \\ A(f(x,y)) \to a(x, \mathbf{TESTVALUE}_{\mathbf{b}(\bot,\mathbf{Top(x)})}(y)) \\ \\ TestVar(f(x,y)) \to B(\mathbf{TEST}_{\mathbf{b}(\bot,\bot)}(f(x,y))) \\ B(f(x,y)) \to b(x, \mathbf{TESTVALUE}_{\mathbf{a}(\bot,\mathbf{Top(x)})}(y)) \\ \\ TestVar(f(x,y)) \to C(\mathbf{TEST}_{\neg(\mathbf{b}\vee\mathbf{a})}(f(x,y))) \\ C(f(x,y)) \to f(x, TestVar(y)), \\ C(f(x,y)) \to f(TestVar(x), y) \end{array} \right\}.$$

On fait un appel d'un autre EFTRS paramétré $\mathbf{TESTVALUE}_{\mathbf{a}(\bot,\mathbf{i}(\bot,\bot))}$ avec la règle $Top(i(x,y)) \to i(\bot,\bot)$ ne garde que la lettre la plus haute du terme.

– $SETVALUE_{a(\bot,i(\bot,\bot))}$ est un ensemble de règles comme suit :

$$\mathcal{R}^{par}_{\lambda^e} = \left\{ \begin{array}{l} SetValue(f(x,y)) \to A(\mathbf{TEST}_{\mathbf{a}(\bot,\bot)}(f(x,y))) \\ A(a(x,y)) \to a(SET_{i(\bot,\bot)}(x), y) \\ \\ SetValue(f(x,y)) \to B(\neg\mathbf{TEST}_{\mathbf{a}(\bot,\bot)}(f(x,y))) \\ B(f(x,y)) \to f(x, SetValue(y)), \\ B(f(x,y)) \to f(SetValue(x), y) \end{array} \right\}.$$

– $PLUSPLUS_{a(\bot,\bot)}$ est un ensemble de règles comme suit :

$$\mathcal{R}^{par}_{\lambda^e} = \left\{ \begin{array}{l} Plusplus(f(x,y)) \to A(\mathbf{TEST}_{\mathbf{a}(\bot,\bot)}(f(x,y))) \\ A(f(x,y)) \to f(Plus(x), y) \\ Plus(f(x,y)) \to f^{+}(x,y) \\ \\ Plusplus(f(x,y)) \to B(\neg\mathbf{TEST}_{\mathbf{a}(\bot,\bot)}(f(x,y))) \\ B(f(x,y)) \to f(x, Plusplus(y)), \\ B(f(x,y)) \to f(Plusplus(x), y) \end{array} \right\}.$$

– $MINUSMINUS_{a(\bot,\bot)}$ est un ensemble de règles comme suit :

$$\mathcal{R}^{par}_{\lambda^e} = \left\{ \begin{array}{l} Minusminus(f(x,y)) \to A(\mathbf{TEST}_{\mathbf{a}(\bot,\bot)}(f(x,y))) \\ A(f(x,y)) \to f(Minus(x), y) \\ Minus(f(x,y)) \to f^{-}(x,y) \\ \\ Minusminus(f(x,y)) \to B(\neg\mathbf{TEST}_{\mathbf{a}(\bot,\bot)}(f(x,y))) \\ B(f(x,y)) \to f(x, Minusminus(y)), \\ B(f(x,y)) \to f(Minusminus(x), y) \end{array} \right\}.$$

Remarques 17 *Les règles paramétrées pour LfV sont incomplètes mais le reste peut être construit de la même manière :*

- *SetVar* $a = b$
- *Permute* $a <-> b$
- *SetVar* $a = b$ *op* c *où* $op \in \{+, -, *, /\}$...

Remarques 18 *Nous montrons que les systèmes paramétrés PFTRSs conviennent très bien au LfV et ils peuvent nous offrir des opérations pour manipuler des données mieux qu'en EFTRS et DDD.*

BILAN DU CHAPITRE 10

Nous avons également montré comment traduire un modèle du LfV en EFTRS. Cela nous conduit à l'idée d'un système fonctionnel élémentaire paramétré, appelé PEFTRS, plus expressif que les élémentaires. Chaque règle paramétrée est équivalente à un ensemble de règles élémentaires.

Dans ce chapitre, nous avons développé les principes de la traduction d'un modèle du LfV en EFTRS. Cela nous conduit à l'idée d'un PEFTRS. Nous avons montré que PEFTRS ont le même pouvoir d'expression que les EFTRS mais faciliter l'écriture de grands FTRS. Nous nous sommes focalisés sur deux aspects que sont la simulation de PEFTRS par des règles d'EFTRSs à l'aide des invariants simples et le pouvoir d'expression du langage de PEFTRS.

Nos ambitions est de développer une extension de PEFTRS avec des règles d'EFTRSs à l'aide des invariants complexes ainsi que la négation des invariants complexes.

Chapitre 11

Conclusion générale

Dans la partie théorique, cette thèse propose les *systèmes de ré-écriture fonctionnels* (FTRS), un nouveau type de systèmes de ré-écriture pour la vérification symbolique de modèles décidables. Nous montrons que notre modèle a la puissance d'expression des systèmes de ré-écriture et qu'il est bien adapté à l'étude de propriétés de sûreté et de propriétés de logique temporelle de modèles.

Nous avons mis en évidence une sous classe de systèmes fonctionnels, les *élémentaires* (EFTRS) et les *élémentaires à droite* (REFTRS), préservant la puissance d'expression des systèmes fonctionnels. De plus, nous avons établi des techniques d'accélération des calculs aboutissant à un outil de vérification symbolique efficace. Ces techniques d'accélération sont effectivement applicables non seulement pour notre modèle mais également pour les autres formalismes basés sur les systèmes de ré-écriture et les automates d'arbres.

En calculant la clôture réflexive et transitive des termes accessibles, surtout pour des systèmes de ré-écriture de grandes tailles, le temps de calcul devient très long. Nous devons fixer un délai (les résultats de taille exponentielle dans le chapitre 8 sont réalisés avec un délai de 3 heures). Dans ce cas-là, nous ne savons pas exactement d'où vient le problème : Les ressources (la mémoire et le processeur) ne sont pas suffisantes pour lutter contre l'explosion combinatoire. Au delà des puissance de calcul, un autre problème peut provenir d'une spécification bogueé résultant ainsi sur des calculs infinis. Pour pallier cette difficulté, des résultats généraux de terminaison en ré-écriture sont applicables sur nos modèles. De bons états de l'art des recherches peuvent être trouvées en [Zantema 2003, Ohlebusch 2002].

Dans la partie expérimentale, nous avons implanté un ensemble d'outils tels qu'un vérificateur d'EFTRS, un outil de transformation automatique des modèles de réseaux de Petri ordinaires et hiérarchiques vers des EFTRSs et un outil permettant de décrire et vérifier des propriétés de logiques temporelles et des propriétés de sûreté. Nous avons comparé notre vérificateur, d'une part avec des outils de ré-écriture tels que Timbuk, Maude et TOM sur des modèles arborescents, d'autre part avec des outils de vérification tels que SPIN, NuSMV, SMART ou HSDD sur des modèles de réseaux de Petri. Les expérimentations rendent à démontrer que notre outil est très compétitif.

Pour les *modèles arborescents* tels que les protocoles TAP, PP et LEP, notre vérificateur possède une performance significative. Pour les *réseaux de Petri paramétrés par un nombre de sous réseaux identiques* tels que le problème des Philosophes, le protocole de Slotted-Ring et le protocole de Round-Robin Mutex, malgré la génération automatique des règles de ré-écriture, nous avons des premiers

succès. Cela peut être expliqué que notre modèle hiérarchique pour le pliage récursif des sous réseaux identiques convient très bien avec les techniques d'accélération (c.à.d. LFPs). Cependant, pour les *réseaux de Petri paramétrés par un nombre de jetons* tel que le système de Kanban et le système de FMS, où le pliage récursif de notre modèle hiérarchique n'est pas applicable, les techniques de LFPs ne sont donc pas efficaces. Nous sommes à la recherche d'une solution efficace tels que l'utilisation intelligent du codage des termes à l'aide de systèmes paramétré (PEFTRS).

Actuellement, dans l'étude des propriétés des logiques temporelles, notre vérificateur reste encore de limite, en particulier pour les propriétés de LTL (Voir cf. des résultats de chapitre 8). Cela peut être expliqué que le calcul du produit synchronisé entre l'automate de Büchi et le système (présenté dans la section 7.3.1) n'est pas bien optimisé. Nous avons donc besoin d'étudier profondément le système élémentaire concernant à ce produit pour que ce système élémentaire adapte mieux aux techniques d'accélération.

Bibliographie

[Abdulla 2002] P. A. Abdulla, B. Jonsson, P. Mahata et J. d'Orso. *Regular Tree Model Checking*. In CAV'02, volume 2404/2002 of *Lecture Notes in Computer Science*, pages 452–466. Springer-Verlag, 2002. 24

[Abdulla 2006] P. A. Abdulla, A. Legay, J. d'Orso et A. Rezine. *Tree Regular Model Checking : A Simulation-based Approach*. The Journal of Logic and Algebraic Programming, 2006. iv, 18, 24, 29, 85

[Arnold 1988] André Arnold et Paul Crubille. *A linear algorithm to solve fixed-point equations on transition systems*. Inf. Process. Lett., vol. 29, no. 2, pages 57–66, 1988. 4, 12

[Baader 1998] F. Baader et T. Nipkow. Term rewriting and all that. Cambridge University Press, 1998. iv, 18

[Bahar 1997] R. Iris Bahar, Erica A. Frohm, Charles M. Gaona, Gary D. Hachtel, Enrico Macii, Abelardo Pardo et Fabio Somenzi. *Algebraic Decision Diagrams and Their Applications*. In ICCAD'93, volume 10 of *Formal Methods in System Design*, pages 171–206, 1997. iii, 14

[Balland 2007] Emilie Balland, Paul Brauner, Radu Kopetz, Pierre-Etienne Moreau et Antoine Reilles. *Tom : Piggybacking rewriting on java*. In Conference on Rewriting Techniques and Applications - RTA'07 Proceedings of the 18th Conference on Rewriting Techniques and Applications, volume 4533 of *LNCS*, pages 36–47, Paris/France France, 06 2007. Springer-Verlag. 30

[Beaudenon 2005] V. Beaudenon. *Diagrammes de Decision de Données pour la Vérification de Système Matériels*. Thèse de doctorat en Informatique, LIP6, Univ. Paris VI, 2005. 148

[Boichut 2007] Y. Boichut, T. Genet, T. Jensen et L. Le Roux. *Rewriting Approximations for Fast Prototyping of Static Analyzers*. In Proceedings of the 18th Conference on Rewriting Techniques and Applications, volume 4533 of *Lecture Notes in Computer Science*, pages 48–62, 2007. 30

[Boichut 2008] Yohan Boichut, Pierre-Cyrille Heam et Olga Kouchnarenko. *Approximation based tree regular model checking*. Nordic Journal of Computing, vol. 14, pages 216–241, 2008. iv

[Boichut 2010] Y. Boichut, J.-M. Couvreur et D-T Nguyen. *Functional Term Rewriting Systems*. Research report, LIFO, 2010. iv, 125, 131

`http ://eftrs.svn.sourceforge.net/viewvc/eftrs/`.

[Bouajjani 2000] Ahmed Bouajjani, Bengt Jonsson, Marcus Nilsson et Tayssir Touili. *Regular Model Checking*. In CAV'00, volume 1855 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000. iv, 24

[Bouajjani 2002] A. Bouajjani et T. Touili. *Extrapolating Tree Transformations*. In CAV'02, volume 2404/2002 of *Lecture Notes in Computer Science*, pages 349–363. Springer-Verlag, 2002. 18, 24

[Bréant 2004] F. Bréant, J.-M. Couvreur, F. Gilliers, F. Kordon, I. Mounier, E. Paviot-Adet, D. Poitrenaud, D. Regep et G. Sutre. *Modeling and verifying behavioral aspects.* In Formal methods for embedded distributed systems : how to master the complexity, pages 171–211. Kluwer Academic Publishers, 2004. iv, 15, 145

[Bryant 1986] R. E. Bryant. *Graph-Based Algorithms for Boolean Function Manipulation.* IEEE Transactions on computers, vol. C-35, no. 8, pages 677–691, Août 1986. iii, 4, 12, 13, 69

[Bryant 1992] R. E. Bryant. *Symbolic Boolean manipulation with ordered binary-decision diagrams.* ACM Comput. Surv., vol. 24, no. 3, pages 293–318, 1992. iii, 4, 12, 13, 69

[Burch 1992] J.R. Burch, E.M. Clarke et K.L. McMillan. *Symbolic Model Checking : 10^{20} States and Beyond.* Information and Computation (Special issue for best papers from LICS90), vol. 98, no. 2, pages 153–181, 1992. iii, 13, 15

[Ciardo 2000] G. Ciardo, G. Lüttgen et R. Siminiceanu. *Efficient Symbolic State-Space Construction for Asynchronous Systems.* In ICATPN'2000, volume 1825 of *Lecture Notes in Computer Science*, pages 103–122. Springer Verlag, 2000. iii, 14

[Ciardo 2002a] G. Ciardo, R. L. Jones III, Marmorstein R. M., A. S. Miner et R. Siminiceanu. *SMART : Stochastic Model-checking Analyzer for Reliability and Timing.* In DSN, page 545, 2002. 133

`http ://www.cs.ucr.edu/ ciardo/SMART/`.

[Ciardo 2002b] G. Ciardo et R. Siminiceanu. *Using Edge-Valued Decision Diagrams for Symbolic Generation of Shortest Paths.* In FMCAD '02 : Proceedings of the 4th International Conference on Formal Methods in Computer-Aided Design, pages 256–273, London, UK, 2002. Springer-Verlag. 133

[Ciardo 2003] G. Ciardo, R. M. Marmorstein et R. Siminiceanu. *Saturation Unbound.* In TACAS, pages 379–393, 2003. 14, 70, 71, 125

[Ciardo 2004] G. Ciardo. *Reachability Set Generation for Petri Nets : Can Brute Force Be Smart ?* In ICATPN'04, volume 3099 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004. 133

[Ciardo 2005] G. Ciardo et A. J. Yu. *Saturation-Based Symbolic Reachability Analysis Using Conjunctive and Disjunctive Partitioning.* In CHARME, pages 146–161, 2005. 133

[Ciardo 2006] G. Ciardo, R. Marmorstein et R. Siminiceanu. *The saturation algorithm for symbolic state-space exploration.* Int. J. Softw. Tools Technol. Transf., vol. 8, no. 1, pages 4–25, 2006. 125

[Ciardo 2007] G. Ciardo, G. Lüttgen et A. S. Miner. *Exploiting interleaving semantics in symbolic state-space generation.* Formal Methods in System Design, vol. 31, no. 1, pages 63–100, 2007. 133

[Cimatti 2002] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani et A. Tacchella. *NuSMV Version 2 : An OpenSource Tool for Symbolic Model Checking*. In Proc. International Conference on Computer-Aided Verification (CAV 2002), volume 2404 of *Lecture Notes in Computer Science*, Copenhagen, Denmark, July 2002. Springer-Verlag. 15, 132

`http ://nusmv.irst.itc.it/`.

[Clarke 1986] Edmund M. Clarke, E. Allen Emerson et A. Prasad Sistla. *Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications*. ACM Trans. Program. Lang. Syst., vol. 8, no. 2, pages 244–263, 1986. 4, 11, 12

[Clarke 1989] E. Clarke, D. Long et K. McMillan. *Compositional model checking*. In Proceedings of the Fourth Annual Symposium on Logic in computer science, pages 353–362, Piscataway, NJ, USA, 1989. IEEE Press. 18, 25

[Clarke 2000] E. M. Clarke, O. Grumberg et D. A. Peled. Model checking. MIT Press, 2000. iii, 4

[Clavel 2001] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer et J. F. Quesada. *Maude : Specification and Programming in Rewriting Logic*. Theoretical Computer Science, 2001. 30

`http ://maude.cs.uiuc.edu`.

[Comon 2002] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison et M. Tommasi. Tree automata techniques and applications. `http ://www.grappa.univ-lille3.fr/tata/`, 2002. 18

[Couvreur 1999] J.-M. Couvreur. *On-the-fly Verification of Linear Temporal Logic*. In Springer Verlag, editeur, Proc. of FM'99, pages 253–271, 1999. Lecture Notes in Computer Science. 9, 113

[Couvreur 2002] J.-M. Couvreur, E. Encrenaz, E. PaviotAdet, D. Poitrenaud et P. Wacrenier. *Data Decision Diagram for Petri Net Analysis*. In ICATPN, volume 2360, pages 1–101. Springer Verlag, 2002. iii, 14, 15, 71, 125

[Couvreur 2004] J.-M. Couvreur. *Contribution à l'algorithme de la vérification*. Mémoire d'habilitation à diriger des recherches, Université Bordeaux 1, LaBRI, 2004. 9

[Couvreur 2005] J.-M. Couvreur et Y. Thierry-Mieg. *Hierarchical Decision Diagrams to Exploit Model Structure*. In FORTE, pages 443–457, 2005. 14, 15, 71, 125, 133

[Couvreur 2008] J.-M. Couvreur et D.-T. Nguyen. *Tree Data Decision Diagrams*. In VeCOS. Leeds, UK, 2008. iv

[Denker 1998] G. Denker, J. Meseguer et C. Talcott. *Protocol Specification and Analysis in Maude*. In Proc. 2nd WRLA Workshop, Pont à Mousson (France), 1998. 30

[Dershowitz 1990] N. Dershowitz et J.-P. Jouannaud. Handbook of theoretical computer science, volume B, chapitre 6 : Rewrite Systems, pages 244–320. Elsevier Science Publishers B. V. (North-Holland), 1990. Also as : Research report 478, LRI. iv, 18

[Dill 1989] D. L. Dill. Trace theory for automatic hierarchical verification of speed-independent circuits. MIT Press, Cambridge, MA, USA, 1989. 18, 25

[Emerson 1980] E. Emerson et Edmund Clarke. *Characterizing correctness properties of parallel programs using fixpoints.* Automata, Languages and Programming, pages 169–181, 1980. 4, 12

[Feuillade 2004] G. Feuillade, T. Genet et V. Viet Triem Tong. *Reachability Analysis over Term Rewriting Systems.* JAR, vol. 33 (3-4), pages 341–383, 2004. iv, 25, 30

[Gastin 2001] P. Gastin et D. Oddoux. *Fast LTL to Büchi Automata Translation.* In G. Berry, H. Comon et A. Finkel, editeurs, Proceedings of the 13th Conference on Computer Aided Verification (CAV'01), numéro 2102 de Lecture Notes in Computer Science, pages 53–65. Springer Verlag, 2001. 9

[Genet 2000] T. Genet et F. Klay. *Rewriting for Cryptographic Protocol Verification.* In Proc. 17th CADE Conf., Pittsburgh (Pen., USA), volume 1831 of *LNAI*. Springer-Verlag, 2000. 30

[Genet 2001] T. Genet et V. Viet Triem Tong. *Reachability Analysis of Term Rewriting Systems with Timbuk.* In Proc. 8th LPAR Conf., Havana (Cuba), volume 2250 of *LNAI*, pages 691–702. Springer-Verlag, 2001. iv, 30

`http ://www.irisa.fr/celtique/genet/timbuk/`.

[Genet 2003] T. Genet, Y.-M. Tang-Talpin et V. Viet Triem Tong. *Verification of Copy Protection Cryptographic Protocol using Approximations of Term Rewriting Systems.* In In Proceedings of Workshop on Issues in the Theory of Security, 2003. 30

[Gerth 1995] R. Gerth, D. Peled, M. Y. Vardi et P. Wolper. *Simple on-the-fly automatic verification of linear temporal logic.* In PSTV, pages 3–18, 1995. 4, 11

[Gilleron 1995] R. Gilleron et S. Tison. *Regular Tree Languages and Rewrite Systems.* Fundamenta Informaticae, vol. 24, pages 157–175, 1995. 18

[Gilliers 2004] F. Gilliers, J.-P. Velu et F. Kordon. *Generation of Distributed Programs in Their Target Execution Environment.* In IEEE International Workshop on Rapid System Prototyping, pages 127–134, 2004. iv, 15

[Graf 1996] S. Graf, B. Steffen et G. Lüttgen. *Compositional Minimisation of Finite State Systems Using Interface Specifications.* Formal Asp. Comput., vol. 8, no. 5, pages 607–616, 1996. 8, 135

[Gupta 1993] A. Gupta et A. L. Fisher. *Representation and symbolic manipulation of linearly inductive Boolean functions.* In ICCAD'93, pages 111–116, 1993. iii, 14

[Gupta 1994] A. Gupta. *Inductive Boolean Function Manipulation.* PhD thesis, Carnegie Mellon University, 1994. iii, 14

[Holzmann 1986] G.J. Holzmann. *The Model Checker SPIN.* IEEE Trancastions on Software Engineering, vol. 23, no. 5, may 1986. 4, 14, 132, 146, 148

`http ://spinroot.com/spin/whatispin.html`.

[Holzmann 1991] G.J. Holzmann. Design and validation of computer protocols. Prentice Hall, 1991. 4, 14, 146, 148

[Hulgaard 1999] H. Hulgaard, P. F. Williams et H. R. Andersen. *Equivalence Checking of Combinational Circuits using Boolean Expression Diagrams.* IEEE Transactions of Computer-Aided Design, vol. 18, no. 7, 1999. iii, 13

[Kesten 1997] Y. Kesten, O. Maler, M. Marcus, A. Pnueli et E. Shahar. *Symbolic Model Checking with Rich Assertional Languages.* In THEORETICAL COMPUTER SCIENCE, pages 424–435. Springer-Verlag, 1997. iv, 24, 83

[Kolks 1993] T. Kolks, B. Lin et H. De Man. *Sizing and Verification of Communication Buffers for Communicating Processes.* In ICCAD'93, volume 1825, pages 660–664, 1993. 14

[Mauborgne 1999] L. Mauborgne. *Binary Decision Graphs.* In SAS'99, volume 1694 of *Lecture Notes in Computer Science*, pages 101–116. Springer-Verlag, 1999. iii, 14

[Mauborgne 2000] L. Mauborgne. *An Incremental Unique Representation for Regular Trees.* Nordic Journal of Computing, vol. 7, no. 4, pages 290–311, 2000. iii, 14

[Meseguer 2003] J. Meseguer, M. Palomino et N. Martí-Oliet. *Equational Abstractions.* In Proc. 19th CADE Conf., Miami Beach (Fl., USA), volume 2741 of *Lecture Notes in Computer Science*, pages 2–16. Springer-Verlag, 2003. 30

[Minato 1990] S. Minato, N. Ishiura et S. Yajima. *Shared Binary Decision Diagrams with Attributed Edges for Efficient Boolean Function Manipulation.* In DAC'90, pages 52–57. ACM/IEEE, IEEE Computer Society Press, 1990. iii, 13

[Miner 1999] A. S. Miner et G. Ciardo. *Efficient Reachability Set Generation and Storage Using Decision Diagrams.* In Proceedings of the 20th International Conference on Application and Theory of Petri Nets, pages 6–25, London, UK, 1999. Springer-Verlag. iii, 6, 14, 15, 96

[Nguyen 2006] D.-T. Nguyen. *LfV, Language for Verification.* In Student session, 7th School on MOdelling and VErifying of parallel Processes, pages 336–341. Bordeaux Computer Science Laboratory (LaBRI) and CNRS, 2006. iv, 15, 148

[Nguyen 2007] D.-T. Nguyen. *LfV-DDD Checker.* In Doctoral Symposium, 5th IEEE International Conference on Research, Innovation and Vision for the Future, pages 165–166. Hanoi, Vietnam, 2007. iv, 15, 149

[Ohlebusch 2002] E. Ohlebusch. Advanced topics in term rewriting. Springer-Verlag, London, UK, 2002. iv, 161

[Pnueli 1977] A. Pnueli. *The Temporal Logic of Programs.* In FOCS, pages 46–57, 1977. 4, 8

[Reffel 1999] F. Reffel. *BDD-Nodes Can Be More Expressive.* In ASIAN'99, volume 1742 of *Lecture Notes in Computer Science*, pages 294–307. Springer Verlag, 1999. iii, 14

[Sistla 1982] A. P. Sistla et E. M. Clarke. *The complexity of propositional linear temporal logics.* In STOC '82 : Proceedings of the fourteenth annual ACM symposium on Theory of computing, pages 159–168, New York, NY, USA, 1982. ACM. 4, 12

[Thierry-Mieg 2004] Y. Thierry-Mieg. *Techniques pour le Model-Checking de spécifications de Haut Niveau.* Thèse de doctorat en Informatique, LIP6, Univ. Paris VI, 2004. 14, 15, 71, 125

[Thierry-Mieg 2008] Y. Thierry-Mieg, A. Hamez et F. Kordon. *Building efficient model checkers using Hierarchical Set decidion diagrams and automatic saturation.* Fundamenta Inforaticae Petri Nets, vol. 1-25, 2008. 14, 15, 71, 125, 133

[Thierry-Mieg 2009] Y. Thierry-Mieg, D. Poitrenaud, A. Hamez et F. Kordon. *Hierarchical Set Decision Diagrams and Regular Models.* In TACAS, pages 1–15, 2009. 133

`http ://move.lip6.fr/ software/DDD/`

`http ://sourceforge.net/projects/buddy/`

`http ://vlsi.colorado.edu/ fabio/CUDD/` .

[Vardi 1986] M. Y. Vardi et P. Wolper. *Automata-Theoretic Techniques for Modal Logics of Programs.* J. Comput. Syst. Sci., vol. 32, no. 2, pages 183–221, 1986. 4, 9

[Wolper 1998] Pierre Wolper et Bernard Boigelot. *Verifying systems with infinite but regular state spaces.* In CAV'98, VOLUME 1427 OF Lecture Notes in Computer Science, pages 889–7. Springer-Verlag, 1998. iv, 24

[Wolper 2001] P. Wolper. *Constructing Automata from Temporal Logic Formulas : A Tutorial.* In Lectures on Formal Methods in Performance Analysis (First EEF/Euro Summer School on Trends in Computer Science), volume 2090 of *Lecture Notes in Computer Science*, pages 261–277. Springer-Verlag, July 2001. 10, 113

[Zantema 2003] H. Zantema. Term rewriting system, chapter termination. Cambridge University Press, UK, 2003. iv, 161

Annexe A

Preuves

A.1 Chapitre 4. Systèmes de Ré-écriture Fonctionnels

A.1.1 Section 4.1. FTRSs

Preuve 1 *Proposition 1*

- $\alpha \to_{\{l\to r\}} \beta \Rightarrow F_{l\to r}(\alpha) \to^{\star}_{\mathcal{R}_\lambda} \beta$: *Selon la définition, il existe une position p de α et une substitution $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}_{bin})$ telles que $\alpha|_p = l\sigma$ et $\beta = \alpha[r\sigma]_p$. Soit α' un terme construit comme le suivant : $\alpha' = \alpha[F_{l\to r}(\alpha|_p)]_p$. Clairement, en utilisant seulement les règles de circulation, c.à.d. les règles de la forme $F_{l\to r}(a(x,y)) \to a(F_{l\to r}(x), y)$ et $F_{l\to r}(a(x,y)) \to a(x, F_{l\to r}(y))$, nous avons*

$$F_{l\to r}(\alpha) \to^{\star}_{\mathcal{R}_\lambda} \alpha'. \tag{A.1}$$

Selon la hypothèse, il existe une substitution $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}_{bin})$ telle que $\alpha|_p = l\sigma$. Par conséquent, nous avons $F_{l\to r}(\alpha|_p) = F_{l\to r}(l\sigma)$. Ainsi, $F_{l\to r}(\alpha|_p) = F_{l\to r}(l)\sigma$. Selon la construction de α', il existe une règle générée de $\mathcal{R}_\lambda$, c.à.d $F_{l\to r}(l) \to r$, une substitution $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}_{bin})$ et une position p telles que $\alpha'|_p = F_{l\to r}(l)\sigma$. En conséquence, nous pouvons construire le terme $\alpha'[r\sigma]$ qui est égal à β selon la construction de α'. Nous avons donc :

$$\alpha' \to_{\mathcal{R}_\lambda} \beta. \tag{A.2}$$

Ainsi, en utilisant (A.1) et (A.2), nous pouvons déduire que $F_{l\to r}(\alpha) \to^{\star}_{\mathcal{R}_\lambda} \beta$. Par conséquent, $\alpha \to_{\{l\to r\}} \beta \Rightarrow F_{l\to r}(\alpha) \to^{\star}_{\mathcal{R}_\lambda} \beta$.
- $\alpha \to_{\{l\to r\}} \beta \Leftarrow F_{l\to r}(\alpha) \to^{\star}_{\mathcal{R}_\lambda} \beta$: *Soit $\mathcal{R}'_\lambda$ le FTRS tel que $\mathcal{R}'_\lambda = \mathcal{R}_\lambda \setminus \{F_{l\to r}(l) \to r \mid l \to r \in \mathcal{R}\}$. Effectivement, nous pouvons montrer que $\mathcal{R}'^{\star}_\lambda(\{F_{l\to r}(\alpha)\}) = \emptyset$.*
En conséquence, cela implique que, pour obtenir le terme $\beta \in \mathcal{R}^{\star}_\lambda(\{F_{l\to r}(\alpha)\})$, la règle $F_{l\to r}(l) \to r$ doit être appliquée au moins une fois. Actuellement, nous demandons que cette règle est appliquée exactement une fois. Effectivement, poursuivons en induction sur le chemin de ré-écriture qui nous mène à partir de $\mathcal{F}_{l\to r}(\alpha)$ à β c.à.d il existe $t_0, \ldots, t_n \in \mathcal{T}(\mathcal{F}_{bin} \cup \mathcal{F}_{NT})$ tel que $t_0 = F_{l\to r}(\alpha) \to_{\mathcal{R}_\lambda} t_1 \to_{\mathcal{R}_\lambda} \ldots t_{n-1} \to_{\mathcal{R}_\lambda} t_n = \beta$. Nous montrons d'abord que chaque t_i a une position p au plus telle que $t_i(p) = F_{l\to r}$. Soit P_n la proposition suivante : pour tout $p, p' \in \mathcal{P}os(t_n)$, si $t_n(p) = t_n(p') = F_{l\to r}$ alors $p = p'$.
  - *P_0 : Puisque $t_0 = F_{l\to r}(\alpha)$ et $\alpha \in \mathcal{T}(\mathcal{F}_{bin})$, ε est l'unique position de t_0 où $F_{l\to r}$ apparaît. P_0 donc est vrai.*

- $P_n \Rightarrow P_{n+1}$: *Supposons que* P_n *soit vrai. Par conséquent,* t_n *satisfait à la propriété :* $p, p' \in \mathcal{P}os(t_n)$*, si* $t_n(p) = t_n(p') = F_{l\to r}$*. De plus, puisque* $t_n \to_{\mathcal{R}_\lambda} t_{n+1}$*, il existe nécessairement une unique position* p *de* t_n *telle que* $t_n(p) = F_{l\to r}$*. Poursuivons en ré-écrivant l'analyse de cas :*
  - $F_{l\to r}(a(x,y)) \to a(F_{l\to r}(x), y)$ *est appliquée à la position* p *: Il donc existe une substitution* $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}_{bin})$ *telle que* $t_n|_p = F_{l\to r}(a(x,y))\sigma = F_{l\to r}(a(\sigma(x), \sigma(y)))$*. Par conséquent,* $t_{n+1} = t_n[a(F_{l\to r}(x), y)\sigma] = t_n[a(F_{l\to r}(\sigma(x)), \sigma(y))]$*. Puisque* $\sigma(x), \sigma(y) \in \mathcal{T}(\mathcal{F}_{bin})$ *et* $\forall p' \in \mathcal{P}os(t_n)$ *telles que* $p \neq p'$*,* $t_n(p') \notin \mathcal{F}_{NT}$*,* t_{n+1} *satisfait également à* $\forall p, p' \in \mathcal{P}os(t_{n+1})$*, si* $t_{n+1}(p) = t_{n+1}(p') = F_{l\to r}$*. Plus précisément, l'unique position* p' *de* t_{n+1} *telle que* $t_{n+1}(p) = F_{l\to r}$ *est* $p' = p.0$*.*
  - $F_{l\to r}(a(x,y)) \to a(x), F_{l\to r}(y)$ *est appliquée à la position* p *: le cas de preuve est similaire au précédent.*
  - $F_{l\to r}(l) \to r$ *est appliquée à la position* p *: Il donc existe une substitution* $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}_{bin})$ *telle que* $t_n|_p = F_{l\to r}(l)\sigma = F_{l\to r}(l\sigma)$*. En conséquence,* $t_{n+1} = t_n[r\sigma]$*. Puisque* $r \in \mathcal{T}(\mathcal{F}_{bin}, \mathcal{X})$ *et* $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}_{bin})$*,* $r\sigma$ *est ainsi un terme de* $\mathcal{T}(\mathcal{F}_{bin})$ *et* $t_{n+1} \in \mathcal{T}(\mathcal{F}_{bin})$*. Ainsi,* t_{n+1} *satisfait à la propriété :* $\forall p, p' \in \mathcal{P}os(t_{n+1})$*, si* $t_{n+1}(p) = t_{n+1}(p') = F_{l\to r}$*.*

Nous venons justement de montrer que chaque terme dans le chemin de ré-écriture contient exactement un symbole $F_{l\to r}$ *et dès que la règle* $F_{l\to r}(l) \to r$ *est appliquée, un terme de* $\mathcal{T}(\mathcal{F}_{bin})$ *est obtenu. Le FTRS* $\mathcal{R}_\lambda$ *ne peut donc plus être appliquée. Cela montre la revendication.*
Ainsi, le chemin de ré-écriture nous menant à partir de $F_{l\to r}(\alpha)$ *à* β *est de la forme suivante :* $t_0 = F_{l\to r}(\alpha) \to_{\mathcal{R}_\lambda \setminus \{F_{l\to r}(l)\to r\}} t_1 \to_{\mathcal{R}_\lambda \setminus \{F_{l\to r}(l)\to r\}} \dots t_{n-1} \to_{\{F(l)\to r\}} t_n = \beta$*. Avec une induction très proche au précédent, nous pouvons montrer qu'il existe une position* p *de* α *et une substitution* $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}_{bin})$ *telles que* $t_{n-1} = \alpha[F(l)\sigma]_p$*. Nous pouvons également montrer que* $t_{n-1}|_{p.0} = \alpha|_p$*. En conséquence,*

$$l\sigma = \alpha|_p. \tag{A.3}$$

De plus, $t_n = \beta = t_{n-1}[r\sigma]_p$*. Ainsi,*

$$\beta = \alpha[r\sigma]_p. \tag{A.4}$$

Selon la construction de $\mathcal{R}_\lambda$*,* $l \to r \in \mathcal{R}$*. Ainsi, d'après (A.3) et (A.4), il existe une règle* $l \to r \in \mathcal{R}$*, une position* $p \in \mathcal{P}os(\alpha)$ *et une substitution* $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}_{bin})$ *telles que* $l\sigma = \alpha|_p$ *et* $\beta = \alpha[r\sigma]_p$*. Par conséquent,* $\alpha \to_{\{l\to r\}} \beta$*. En résumé, nous avons montré que* $\alpha \to_{\{l\to r\}} \beta \Leftarrow F_{l\to r}(\alpha) \to^\star_{\mathcal{R}_\lambda} \beta$*, concluant la preuve.*

Preuve 2 *Théorème 1*

- $\mathcal{R}^\star(E) \subseteq \mathcal{R}^\star_\lambda(E')$: *Soit* $\alpha \in E$ *et* $\beta \in \mathcal{R}^\star(\{\alpha\})$. *Selon la définition de* $\mathcal{R}^\star$, *il existe* $l_1 \to r_1, \dots l_n \to r_n \in \mathcal{R}$ *et* $t_0, \dots, t_n$ *telles que* $\alpha \to_{\{l_0 \to r_0\}} t_0 \to_{\{l_1 \to r_1\}} \dots \to_{\{l_{n-1} \to r_{n-1}\}} t_{n-1} \to_{\{l_n \to r_n\}} t_n = \beta$. *Le FTRS* $\mathcal{R}_{l_0 \to r_0}$ *remplit les conditions attendues spécifiées à la proposition 1. Par conséquent, selon la proposition 1,* $t_0 \to_{l_0 \to r_0} t_1$ *implique que* $F_{l_0 \to r_0}(t_0) \to^\star_{\mathcal{R}_{l_0 \to r_0}} t_1$. *Malheureusement,* $F_{l_0 \to r_0}(t_0) \notin E'$. *Cependant,* $t_0 = \alpha$ *et* $\alpha \in E$. *Ainsi,* $G(t_0) \in E'$. *Effectivement,* $G(t_0) \to_{G(x) \to G(F_{l_0 \to r_0}(x))} G(F_{l_0 \to r_0}(\alpha))$. *En conséquence,* $G(t_0) \to^\star_{\mathcal{R}_\lambda} G(t_1) \to_{G(x) \to x} t_1$ *implique que* $t_1 \in \mathcal{R}^\star_\lambda(E')$. *En généralisant ce processus, nous obtenons le chemin de ré-écriture ci-dessous :* $G(t_0) \to^\star_{\mathcal{R}_\lambda} G(t_1) \dots \to^\star_{\mathcal{R}_\lambda} G(t_n) \to_{G(x) \to x} t_n = \beta$. *Par conséquent,* $\beta \in \mathcal{R}^\star_\lambda(E')$.
  Ainsi, nous pouvons déduire que

$$\mathcal{R}^\star(\{\alpha\}) \subseteq \mathcal{R}^\star_\lambda(\{G(\alpha)\}). \tag{A.5}$$

- $\mathcal{R}^\star_\lambda(E') \subseteq \mathcal{R}^\star(E)$: *Étudions* $\mathcal{R}^\star_\lambda(\{G(\alpha)\})$.
  - *Supposons qu'une règle de la forme* $G(x) \to G(F_{l \to r}(x))$ *est appliquée. Ainsi,* $G(\alpha) \to_{G(x) \to G(F_{l \to r}(x))} G(F_{l \to r}(\alpha)))$. *Notez que* $G(x) \to G(F_{l \to r}(x))$ *ne peut pas être appliquée dès que le symbole* $F_{l \to r}$ *apparaît. Si* $\mathcal{R}^\star_{l \to r}(F_{l \to r}(\alpha)) = \emptyset$ *alors aucun terme de* $\mathcal{T}(\mathcal{F}_{bin})$ *peut être atteint de* $F_{l \to r}(\alpha)$. *Ainsi, puisque* $\mathcal{R}_{l \to r}$ *permet la définition de* $\mathcal{R}_\lambda$ *dans le proposition 1, nous pouvons déduire que la règle* $l \to r \in \mathcal{R}$ *ne peut pas être appliquée sur* α.
    Si $\mathcal{R}^\star_{l \to r}(F_{l \to r}(\alpha)) \neq \emptyset$ *alors il existe un terme de* $t_1 \in \mathcal{T}(\mathcal{F}_{bin})$ *qui peut être atteint de* $F_{l \to r}(\alpha)$. *Puisque* $F_{l \to r}(\alpha) \to^\star_{\mathcal{R}_{l \to r}} t_1$, *nous avons* $G(\alpha) \to^+_{\mathcal{R}_\lambda} G(t_1)$. *De plus, d'après la proposition 1,* $\alpha \to_{\{l \to r\}} t_1$. *Notez que* $t_1 \in \mathcal{R}^\star_\lambda(E')$ *puisque la règle* $G(x) \to x$ *peut être appliquée sur* $G(t_1)$. *Ainsi, ce processus peut être itéré et nous pouvons construire une séquence telle que* $G(\alpha) \to^+_{\mathcal{R}_\lambda} G(t_1) \to^+_{\mathcal{R}_\lambda} G(t_2) \dots \to^+_{\mathcal{R}_\lambda} G(t_n)$ *avec* $t_i \in \mathcal{R}^\star_\lambda(E')$ *si la règle* $G(x) \to x$ *est appliquée sur chaque terme* $G(t_i)$ *(c'est possible puisque* $t_i \in \mathcal{T}(\mathcal{F}_{bin})$*). De plus, selon la proposition 1, nous avons* $t_i \to_{\{l \to r\}} t_{i+1}$ *pour* $i = 1, \dots, n-1$. *Nous pouvons donc déduire que* $t_n \in \mathcal{R}^\star(\{\alpha\})$. *Finalement, nous pouvons déduire que*

$$\mathcal{R}^\star(\{\alpha\}) \supseteq \mathcal{R}^\star_\lambda(\{G(\alpha)\}). \tag{A.6}$$

  - *Supposons qu'aucune règle de la forme* $G(x) \to G(F_{l \to r}(x))$ *est appliquée. Par conséquent, aucune règle de* $\bigcup_{l \to r \in \mathcal{R}} \mathcal{R}_{l \to r}$ *peut être appliquée .*
    L'unique terme atteint de $\mathcal{T}(\mathcal{F}_{bin})$ *est donc obtenu en appliquant la règle* $G(x) \to x$. *Ainsi,* $\alpha \in \mathcal{R}^\star_\lambda(\{G(\alpha)\})$. *D'après la définition,* $\alpha \in \mathcal{R}^\star(\{\alpha\})$.

Pour conclusion, pour tous $\alpha \in E$, *de (A.5) et (A.6), nous obtenons que* $\mathcal{R}^\star(E) = \mathcal{R}^\star_\lambda(E')$.

Preuve 3 *Lemme 1*
Supposons un tel TRS construit comme dans le lemme 1. Selon le théorème 1, nous obtenons $\mathcal{R}^{\star}_{\lambda}(E') = \mathcal{R}^{\star}(E)$.

A.1.2 Section 4.2. EFTRSs

Preuve 4 (Preuve succincte de la proposition 2)

Le point clé est de construire un EFTRS $\mathcal{R}_{\lambda^e}$ *qui implémente le processus de ré-écriture.*

Le EFTRS $\mathcal{R}_{\lambda^e}$ *est construit comme le suivant :*

$\mathcal{R}_{\lambda^e} = \mathcal{R}^{l\to r}_{visit} \cup \mathcal{R}^{l\to r}_{TV} \cup \mathcal{R}^{l\to r}_{check} \cup \mathcal{R}^{l\to r}_{\sigma} \cup \mathcal{R}^{l\to r}_{GS} \cup \mathcal{R}^{l\to r}_{\sigma-apply}$

- $\alpha \to_{\{l\to r\}} \beta \Rightarrow F_{l\to r}(\alpha) \to^*_{\mathcal{R}_{\lambda^e}} \beta$: *Effectivement, si* $\alpha \to_{\{l\to r\}} \beta$ *alors il existe une position* p *de* α *et une substitution* $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}_{bin})$ *telles que* $\alpha|_p = l\sigma$ *and* $\beta = \alpha[r\sigma]_p$. *A partir de la définition de* $\mathcal{R}_{\lambda^e}$, *nous pouvons déduire que* $\mathcal{F}_{l\to r}(\alpha) \to^*_{\mathcal{R}_{\lambda^e}} \alpha[F_{l\to r}(\alpha|_p)]_p$. *Ainsi, en appliquant de la règle* $F_{l\to r}(x) \to F^{l\to r}_{rewrite}(F^{l\to r}_{check}(F^{l\to r}_{copy}(F^{\varepsilon}_{l\to r}(x))))$ *sur* $\alpha[F_{l\to r}(\alpha|_p)]_p$, *nous obtenons* $\alpha[F^{l\to r}_{rewrite}(F^{l\to r}_{check}(\ F^{l\to r}_{copy}(F^{\varepsilon}_{l\to r}(\alpha|_p))))]_p$. *Puisque* σ *est une substitution de* $\mathcal{X}$ *to* $\mathcal{T}(\mathcal{F}_{bin})$, *et d'après l'ensemble des règles* $\mathcal{R}^{l\to r}_{TV} \cup \mathcal{R}^{l\to r}_{check} \cup \mathcal{R}^{l\to r}_{\sigma} \cup \mathcal{R}^{l\to r}_{GS}$, $\alpha[F^{l\to r}_{rewrite}(F^{l\to r}_{check}(F^{l\to r}_{copy}(F^{\varepsilon}_{l\to r}(\alpha|_p))))]_p$ *peut être ré-écrits en* $\alpha[F^{l\to r}_{rewrite}(t)]_p$ *où* t *est défini comme le suivant :*
  - $t(p) = \sigma(x_i)$, $t(p') = \top_{\mathcal{X}} \in \mathcal{F}_{bin}$ *avec* $p' = (2.)^i\ p = p'.1$ *et* x_i *est la* i^{th} *variable de* l *lu du côté gauche de* l
  - $t(p_{rmp}) = \bot_{\mathcal{X}} \in \mathcal{F}_{bin}$ *avec* $p_{rmp} = (2.)^m 2$ *et* m *est le nombre de variables apparaissant dans* l

  Ainsi, le terme t *représente comme un terme la substitution* σ. *Finalement, le EFTRS* $\mathcal{R}^{l\to r}_{\sigma-apply}$ *nous permet de reconstruire* r *et la valeur des variables est copiée symbole par symbole. Les uniques termes accessibles donc sont* $r\sigma$. *Ainsi, en résumé,* $F^{l\to r}_{rewrite}(t) \to^*_{\mathcal{R}^{l\to r}_{\sigma-apply}} r\sigma \in \mathcal{T}(\mathcal{F}_{bin})$. *En conséquence,* $\alpha[F^{l\to r}_{rewrite}(t)]_p \to^*_{\mathcal{R}^{l\to r}_{\sigma-apply}} \alpha[r\sigma]_p \in \mathcal{T}(\mathcal{F}_{bin})$ *et puisque* $\alpha[r\sigma]_p = \beta$, $\beta \in \mathcal{R}^*_{\lambda^e}(\{F_{l\to r}(\alpha)\})$.

- $\alpha \to_{l\to r} \beta \Leftarrow F_{l\to r}(\alpha) \to^*_{\mathcal{R}_{\lambda^e}} \beta$: *La preuve pour ce cas est proche au cas similaire traité dans la preuve de la proposition 1 dans le sens que Des pas de ré-écrire sont ordonnés et dépendants du terme initial c.à.d* $F_{l\to r}(\alpha)$. *Finalement, puisque la hypothèse* $F_{l\to r}(\alpha) \to^*_{\mathcal{R}_{\lambda^e}} \beta$, *dû aux définitions de* $\mathcal{R}^{l\to r}_{visit}$ *et* $\mathcal{R}^{l\to r}_{TV}$, *il existe une position* $p \in \mathcal{P}os(\alpha)$ *telle que* $F_{l\to r}(\alpha) \to^*_{\mathcal{R}^{l\to r}_{visit}\cup\mathcal{R}^{l\to r}_{TV}} \alpha[F^{l\to r}_{rewrite}(F^{l\to r}_{check}(F^{l\to r}_{copy}(F^{\varepsilon}_{l\to r}(\alpha|_p))))]_p$ *et* $F^{l\to r}_{rewrite}(F^{l\to r}_{check}(F^{l\to r}_{copy}(F^{\varepsilon}_{l\to r}(\alpha|_p)))) \to^*_{\mathcal{R}_{\lambda^e}} \beta$. *Dû au pile des symboles de* $\mathcal{F}_{NT}$ *sur* $\alpha|_p$, *nous pouvons montrer que* $F^{l\to r}_{rewrite}(F^{l\to r}_{check}(F^{l\to r}_{copy}(F^{\varepsilon}_{l\to r}(\alpha|_p)))) \to^*_{\mathcal{R}_{\lambda^e}} \beta \Leftrightarrow (\mathcal{R}^{l\to r}_{\sigma-apply})^*((\mathcal{R}^{l\to r}_{GS})^*((\mathcal{R}^{l\to r}_{\sigma})^*((\mathcal{R}^{l\to r}_{check})^*(F^{l\to r}_{rewrite}(F^{l\to r}_{check}(F^{l\to r}_{copy}(F^{\varepsilon}_{l\to r}(\alpha|_p))))))) = \{\beta\}$. *Par conséquent, selon définitions de* $\mathcal{R}^{l\to r}_{\sigma}$, $\mathcal{R}^{l\to r}_{GS}$ *et* $\mathcal{R}^{l\to r}_{check}$, *pour un terme* $t \in \mathcal{T}(\mathcal{F}_{bin})$, *nous pouvons montrer*

que si $(\mathcal{R}^{l\to r}_{check})^*(\{F^{l\to r}_{check}(F^{l\to r}_{copy}(F^{\varepsilon}_{l\to r}(t)))\}) \neq \emptyset$ *alors il existe une substitution* $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}_{bin})$ *telle que* $l\sigma = t$. *En appliquant comme un résultat de notre étude de cas, il existe* $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}_{bin})$ *tel que* $l\sigma = \alpha|_p$. *Pas seulement* $(\mathcal{R}^{l\to r}_{check})^*(\{F^{l\to r}_{check}(F^{l\to r}_{copy}(F^{\varepsilon}_{l\to r}(t)))\}) \neq \emptyset$, *mais il existe également un terme* t' *de* $\mathcal{T}(\mathcal{F}_{bin})$ *tel que* $(\mathcal{R}^{l\to r}_{check})^*(\{F^{l\to r}_{check}(F^{l\to r}_{copy}(F^{\varepsilon}_{l\to r}(t)))\}) = \{t'\}$. *Le terme* t' *est une représentation de* σ, *et l'exécution de l'EFTRS* $\mathcal{R}^{l\to r}_{\sigma-apply}$ *sur* t' *nous permet de reconstruire* r *en substituant chaque variable* x_i *de* r *par* $\sigma(x_i)$. *L'unique terme accessible de* t' *donc est* $r\sigma$. *Ainsi, en résumé, il existe une position de* α *et une substitution* $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}_{bin})$ *telles que* $\alpha|_p = l\sigma$, $\beta = \alpha[r\sigma]_p$ *et* $F_{l\to r}(\alpha) \to^*_{\mathcal{R}_{\lambda^e}} \beta$. *Nous pouvons déduire que* $\alpha \to_{l\to r} \beta \Rightarrow F_{l\to r}(\alpha) \to^*_{\mathcal{R}_{\lambda^e}} \beta$ *qui conclut la preuve.*

Preuve 5 *Théorème 2*

Étant donné l'EFTRS $\mathcal{R}_{\lambda^e} = \bigcup_{l\to r\in\mathcal{R}}(\mathcal{R}^{l\to r}_{visit} \cup \mathcal{R}^{l\to r}_{TV} \cup \mathcal{R}^{l\to r}_{check} \cup \mathcal{R}^{l\to r}_{\sigma} \cup \mathcal{R}^{l\to r}_{GS} \cup \mathcal{R}^{l\to r}_{\sigma-apply}) \cup \mathcal{R}_{FP}$ *avec* $\mathcal{R}_{FP} = \{G(a(x,y)) \to G(F_{l\to r}(a(x,y))) \mid l \to r \in \mathcal{R} \wedge a \in \mathcal{F}_{bin}\} \cup \{G(a(x,y)) \to a(x,y) \mid a \in \mathcal{F}_{bin}\}$, *la preuve est très proche à celle du théorème 1.*

Preuve 6 (Preuve succincte de la proposition 3)
Étant donné l'EFTRS $\mathcal{R}_{\lambda^e}$, *la preuve est très proche à celle de la proposition 2.*

Le point clé est de construire un EFTRS $\mathcal{R}_{\lambda^e}$ *qui implémente le processus de ré-écriture en considérant* H *comme un symbole spécial en* $\mathcal{F}_{bin}$ *et en ignorant les règles de circulation.*

Le EFTRS $\mathcal{R}_{\lambda^e}$ *est construit comme le suivant :*

$$\mathcal{R}_{\lambda^e} = \mathcal{R}^{H(t)\to\alpha}_{TV} \cup \mathcal{R}^{H(t)\to\alpha}_{check} \cup \mathcal{R}^{H(t)\to\alpha}_{\sigma} \cup \mathcal{R}^{H(t)\to\alpha}_{GS} \cup \mathcal{R}^{H(t)\to\alpha}_{\sigma-apply}$$

A.1.3 Section 4.3. REFTRSs

Preuve 7 (Preuve succincte de la proposition 4)
Étant donné les REFTRSs $\mathcal{R}_{\lambda^e\to I}$, $\mathcal{R}_{\lambda^e\to II}$, $\mathcal{R}_{\lambda^e\to III}$, $\mathcal{R}_{\lambda^e\to A}$ *et* $\mathcal{R}_{\lambda^e\to B}$, *la preuve est très proche à celle de la proposition 2.*

Le point clé est de construire un REFTRS qui implémente le processus de ré-écriture.

Le REFTRS est construit comme le suivant :

$$\mathcal{R}_{\lambda^e\to\cup} = \mathcal{R}^{H\ (a(x,y))\to\alpha}_{TV} \cup \mathcal{R}^{H\ (a(x,y))\to\alpha}_{check} \cup \mathcal{R}^{H\ (a(x,y))\to\alpha}_{\sigma} \cup \mathcal{R}^{H\ (a(x,y))\to\alpha}_{\sigma-apply}$$

où

- $\mathcal{R}^{H\ (a(x,y))\to\alpha}_{TV} = \{F_{H\ (a(x,y))\to\alpha}(x) \to F^{H\ (a(x,y))\to\alpha}_{rewrite}($
  $F^{H\ (a(x,y))\to\alpha}_{check}(F^{H\ (a(x,y))\to\alpha}_{copy}(F^{\varepsilon}_{H\ (a(x,y))\to\alpha}(x))))\}$ *va être traduit en* **REFTRSs** *comme suit :*

$$\mathcal{R}_{TV}^{H\ (a(x,y))\to\alpha} = \left\{ \begin{array}{l} F_{H\ (a(x,y))\to\alpha}(a(x,y)) \to F^1_{TV}(F^{\varepsilon}_{H\ (a(x,y))\to\alpha}(a(x,y))) \\ F^1_{TV}(a(x,y)) \to F^2_{TV}(F^{H\ (a(x,y))\to\alpha}_{copy}(a(x,y))) \\ F^2_{TV}(a(x,y)) \to F^3_{TV}(F^{H\ (a(x,y))\to\alpha}_{check}(a(x,y))) \\ F^3_{TV}(a(x,y)) \to F^4_{TV}(F^{H\ (a(x,y))\to\alpha}_{rewrite}(a(x,y))) \\ F^4_{TV}(a(x,y)) \to a(x,y) \end{array} \right\}.$$

pour $\forall a \in \mathcal{F}_{bin}$

- $\mathcal{R}_{check}^{H\ (a(x,y))\to\alpha}$ *est un* **REFTRS** *nous permettant de vérifier si un terme de* $\mathcal{T}(\mathcal{F}_{bin})$ *concorde avec* $H\ (a(x,y))$.
  L'application de cet **REFTRS** *est renvoyé par la présence du symbole non-terminal* $F^{\varepsilon}_{H\ (a(x,y))\to\alpha}$. *Étant donné un terme* $t \in \mathcal{T}(\mathcal{F}_{bin})$ *concordant avec* $H\ (a(x,y))$, $(\mathcal{R}_{check}^{H\ (a(x,y))\to\alpha})^*(\{F^{\varepsilon}_{H\ (a(x,y))\to\alpha}(t)\})$ *est* $\{t'\}$ *où* $\mathcal{P}os_{\mathcal{X}}(H(a(x,y))) = \{11, 12\}$ *et* $\oplus_{x,11}, \oplus_{y,12} \in \mathcal{F}_{bin}$ *;*

$$\mathcal{R}^{=}_{check} \left\{ \begin{array}{l} F^{\varepsilon}_{H\ (a(x,y))\to\alpha}(H(x)) \to H(F^1_{H\ (a(x,y))\to\alpha}(x)) \\ F^1_{H\ (a(x,y))\to\alpha}(a(x,y)) \to a(F^{11}_{H\ (a(x,y))\to\alpha}(x), F^{12}_{H\ (a(x,y))\to\alpha}(y)) \\ F^{11}_{H\ (a(x,y))\to\alpha}(x) \to \oplus_x(x) \\ F^{12}_{H\ (a(x,y))\to\alpha}(y) \to \oplus_y(y) \end{array} \right\}.$$

- $\mathcal{R}_{\sigma}^{H\ (a(x,y))\to\alpha}$ *est un* **REFTRS** *nous permettant de construire une liste ordonnée de termes indexés par des variables de* $H(a(x,y))$. *Cette liste est représentée par un terme et sémantiquement au courant, nous pouvons considérer cette liste comme la substitution résultant du pas concordance. Soit* t' *le terme résultant de* $(\mathcal{R}_{check}^{H\ (a(x,y))\to\alpha})^*(\{F^{\varepsilon}_{H\ (a(x,y))\to\alpha}(t)\})$. $(\mathcal{R}_{\sigma}^{H\ (a(x,y))\to\alpha})^*(F^{H\ (a(x,y))\to\alpha}_{copy}(t'))$ *mène à un unique terme* t'' *représentant la substitution résultant de la concordance entre* t *et* $H(a(x,y))$.

- $\mathcal{R}_{\sigma-apply}^{H\ (a(x,y))\to\alpha}$ *est un* **REFTRS** *spécifiant l'application de la substitution bien-formée résultant de* t *et* $H(a(x,y))$ *sur le terme* α. *Ainsi,* $(\mathcal{R}_{\sigma-apply}^{H\ (a(x,y))\to\alpha})^*(F^{H\ (a(x,y))\to\alpha}_{rewrite}(t'')) = \{\alpha\sigma\}$.

Preuve 8 *Théorème 3*
Selon les propositions 4 et le théorème 2.

A.2 Chapitre 5. Évaluation des systèmes élémentaires

Preuve 9 (Preuve de la proposition 5)

$$(\mathcal{R}_{\lambda^e} \cup (\mathcal{R}'_{\lambda^e \to I} \cup \mathcal{R}'_{\lambda^e \to II}))^{\star}(E') = \mathcal{R}^{\star}_{\lambda^e}(E').$$

avec $E \subseteq \mathcal{T}(\mathcal{F}_{bin})$ *et* $E' = \{H(t) | t \in E\}$.

- $\mathcal{R}^{\star}_{\lambda^e}(E') \subseteq (\mathcal{R}_{\lambda^e} \cup (\mathcal{R}'_{\lambda^e \to I} \cup \mathcal{R}'_{\lambda^e \to II}))^{\star}(E')$, *c'est évident selon* $\mathcal{R}_{\lambda^e} \subseteq (\mathcal{R}_{\lambda^e} \cup (\mathcal{R}'_{\lambda^e \to I} \cup \mathcal{R}'_{\lambda^e \to II}))$
- $(\mathcal{R}_{\lambda^e} \cup (\mathcal{R}'_{\lambda^e \to I} \cup \mathcal{R}'_{\lambda^e \to II}))^{\star}(E') \subseteq \mathcal{R}^{\star}_{\lambda^e}(E')$. *Poursuivons en traitant l'analyse de cas :*
  - $(\mathcal{R}_{\lambda^e} \cup \mathcal{R}'_{\lambda^e \to I})^{\star}(E') \subseteq \mathcal{R}^{\star}_{\lambda^e}(E')$:
    - ${\mathcal{R}'_{\lambda^e}}^{\star}_{H^{\star}\ (a(x,y)) \to a(x,y)}(E') \subseteq {\mathcal{R}_{\lambda^e}}^{\star}_{H\ (a(x,y)) \to a(x,y)}(E')$
    - ${\mathcal{R}'_{\lambda^e}}^{\star}_{H^{\star}\ (a(x,y)) \to a(G^{\star}\ (x),y)}(E') \subseteq {\mathcal{R}_{\lambda^e}}^{\star}_{H\ (a(x,y)) \to a(G\ (x),y)}(E')$ *car* $H^n\ (a(x,y)) \to^{\star}_{H\ (a(x,y)) \to a(G\ (x),y)} a(G^n\ (x), y)$ *avec* n *suffisamment grand.*
    - *Similairement pour le reste.*
  - $(\mathcal{R}_{\lambda^e} \cup \mathcal{R}'_{\lambda^e \to II})^{\star}(E') \subseteq \mathcal{R}^{\star}_{\lambda^e}(E')$:
    - *Similairement pour les autres cas.*

Preuve 10 (Preuve succincte de la proposition 6)

$$(\mathcal{R}_{\lambda^e} \cup (\mathcal{R}'_{\lambda^e \to I} \cup \mathcal{R}'_{\lambda^e \to II}) \setminus (\mathcal{R}_{\lambda^e \to I} \cup \mathcal{R}_{\lambda^e \to II}))^{\star}(E') = \mathcal{R}^{\star}_{\lambda^e}(E').$$

avec $E \subseteq \mathcal{T}(\mathcal{F}_{bin})$ *et* $E' = \{H(t) | t \in E\}$.

D'après la proposition 5 nous obtenons

$$(\mathcal{R}_{\lambda^e} \cup (\mathcal{R}'_{\lambda^e \to I} \cup \mathcal{R}'_{\lambda^e \to II}))^{\star}(E') = \mathcal{R}^{\star}_{\lambda^e}(E').$$

avec $E \subseteq \mathcal{T}(\mathcal{F}_{bin})$ *et* $E' = \{H(t) | t \in E\}$.

D'autre part, au moment donné, nous pouvons prouver qu'une règle en $\mathcal{R}'_{\lambda^e \to I}$ *est applicable seulement si la règle correspondante en* $\mathcal{R}_{\lambda^e \to I}$ *est applicable. De plus le résultat d'un pas de ré-écrire de la règle en* $\mathcal{R}'_{\lambda^e \to I}$ *est identique à celui de la règle correspondante en* $\mathcal{R}_{\lambda^e \to I}$. *Similaire pour* $\mathcal{R}'_{\lambda^e \to II}$ *avec* $\mathcal{R}_{\lambda^e \to II}$. *Cela nous permet d'obtenir le résultat d'un pas de ré-écrire sans utiliser la règle en* $(\mathcal{R}_{\lambda^e \to I} \cup \mathcal{R}_{\lambda^e \to II})$.

Cela montre la revendication.

A.3 Chapitre 6. Modélisation par EFTRS

Preuve 11 (Preuve succincte de la proposition 7)
Au moment donné, nous pouvons prouver qu'une règle en $\mathcal{R}_{\lambda^e}$ *est applicable sur* t_m *seulement si la transition correspondante en* T *est franchissable sur* m. *Effectivement, nous pouvons prouver qu'il existe une fonction* f *de la clôture transitive* $\mathcal{R}^{\star}_{\lambda^e}$ *sur* t_{m_0} *à l'ensemble des marquages accessibles de ce réseau Petri.*

A.4 Chapitre 7. Vérification par EFTRS

Preuve 12 (Preuve succincte de la proposition 8)

$$deadlock = \mathcal{R}^{\star}_{\lambda^e}(init) \backslash \mathcal{R}^{-1}_{\lambda^e}(\mathcal{R}^{\star}_{\lambda^e}(init))$$

- $deadlock \subseteq \mathcal{R}^{\star}_{\lambda^e}(init) \backslash \mathcal{R}^{-1}_{\lambda^e}(\mathcal{R}^{\star}_{\lambda^e}(init))$: *Intuitivement selon* $deadlock \subseteq \mathcal{R}^{\star}_{\lambda^e}(init)$ *et* $deadlock \cap \mathcal{R}^{-1}_{\lambda^e}(\mathcal{R}^{\star}_{\lambda^e}(init)) = \emptyset$

- $\mathcal{R}^{\star}_{\lambda^e}(init) \backslash \mathcal{R}^{-1}_{\lambda^e}(\mathcal{R}^{\star}_{\lambda^e}(init)) \subseteq deadlock$: *Effectivement parce que s'il existe t accessible et il n'est pas dans* $\mathcal{R}^{-1}_{\lambda^e}(\mathcal{R}^{\star}_{\lambda^e}(init))$, *alors il doit être un état bloquant.*

Preuve 13 (Preuve succincte de la proposition 9)
Selon la construction d'un tel système de ré-écriture $\mathcal{R}^{Sync}_{\lambda^e}$, *il va suivre le comportement du système en synchronisant avec le chemin LTL de l'automate de Büchi. Ainsi,*

$$(\mathcal{R}^{Sync}_{\lambda^e} \cup \mathcal{R}^{\omega}_{\lambda^e} \cup \mathcal{R}^{M}_{\lambda^e})^{\star}(\{Sync(init)\}) \neq \emptyset$$

c.à.d. M *est vérifié par la formule* f.

Preuve 14 (Preuve succincte de la proposition 10)

- $M, s \vDash EX\phi_1 \Longleftrightarrow \exists \pi, s_0 = s \Rightarrow M, s_1 \vDash \phi_1$. *Effectivement, nous obtenons*

$$EX_{\mathcal{R}_{\lambda^e}}(u) = \mathcal{R}^{-1}_{\lambda^e}(u)$$

- $M, s \vDash AX\phi_1 \Longleftrightarrow \forall \pi, s_0 = s \Rightarrow M, s_1 \vDash \phi_1$. *Nous pouvons calculer autrement en utilisant* ***EX*** *comme suivant :*

$$\begin{aligned} & M, s \vDash AX\phi_1 \\ \Longleftrightarrow\ & s \neq s' \text{ avec } M, s \vDash EX\neg\phi_1 \\ \Longleftrightarrow\ & s \notin S = \{s' | \exists \pi, s_0 = s' \Rightarrow M, s_1 \vDash \neg\phi_1\} \end{aligned}$$

Effectivement, nous obtenons

$$AX_{\mathcal{R}_{\lambda^e}}(u) = \mathcal{R}^{\star}_{\lambda^e}(u) \backslash ({\mathcal{R}_{\lambda^e}}^{-1}(\mathcal{R}^{\star}_{\lambda^e}(u) \backslash u))$$

Annexe B

Etude de cas du LfV

B.1 Processus principal

La partie déclarative est dans le processus **Main** au dessous où on crée des processus utilisés dans le système 9.3, on utilise donc seulement les pointeurs des processus dans le code de l'autre processus.

```
process Main() {
vars:
     Fifo itf_Server_Rpc_mc, itf_Server_Rpc_cm;
     Fifo itf_Rpc_Client_mc0, itf_Rpc_Client_cm0;...

     Server s0, s1;
     Rpc rpc0, rpc1, rpc2, rpc3;
     Client c0, c1, c2, c3;

trans:
0 |- itf_Server_Rpc_mc  := new FIFO(6); //Fifo from media to class
     itf_Server_Rpc_cm  := new FIFO(6); //Fifo from class to media
-> 1;

1 |- itf_Rpc_Client_mc0 := new FIFO(8);
     itf_Rpc_Client_cm0 := new FIFO(8);
-> 2;

...

5 |- s0                := new Server( itf_cm    <- itf_Server_Rpc_cm,
                                       itf_mc    <- itf_Server_Rpc_mc   );
-> 6;

6 |- s1                := new Server( itf_cm    <- itf_Server_Rpc_cm,
                                       itf_mc    <- itf_Server_Rpc_mc   );
-> 7;

7 |- c0                := new Client( itf_cm    <- itf_Rpc_Client_cm0,
                                       itf_mc    <- itf_Rpc_Client_mc0  );
-> 8;

...

11 |- rpc0              := new Rpc(    input_cm  <- itf_Rpc_Client_cm0,
                                       input_mc  <- itf_Rpc_Client_mc0,
                                       target_cm <- itf_Server_Rpc_cm,
                                       target_mc <- itf_Server_Rpc_mc   );
-> 12;...
};//process Main
```

B.2 Processus Serveur

La classe **Server** n'a qu'une seule méthode *handle_ request* ayant un paramètre formel *num*. Le serveur attend des appels, exécute des méthodes, rend des résultats obtenus aux clients, et revient à l'état initial. Le code du processus Serveur est au-dessous. Quand la méthode est appelée, le paramètre actuel est incrémenté (Ligne code 16).

```
process Server() {
vars:
      Fifo itf_cm, itf_mc;
      Process client;

      int num;
      data msg[];

trans:
0 |- itf_mc@recv (msg);
     [ (msg.size        = 4)       && (msg[0].dataType = PROCESS) &&
       (msg[1].dataType = PROCESS) && (msg[2].dataType = STRING)  &&
       (msg[2]          = "rpc")   && (msg[3].dataType = INT)         ] ;
       //condition for receiving a message
-> 1;

1 |- client          := msg[0];    num    := msg[3];
-> 2;

2 |- num             := num + 1;
-> 3;

3 |- msg.size        := 4;
     msg[0].dataType := PROCESS;   msg[0] := NULL;
     msg[1].dataType := PROCESS;   msg[1] := client;
     msg[2].dataType := STRING;    msg[2] := "rpc";
     msg[3].dataType := INT;       msg[3] := num;
-> 4;

4 |- [ itf_cm.size < itf_cm.max ] ; //synchronous condition for sending a message
     itf_cm@send (msg);
-> 0;

};//process Server
```

B.3 Processus Client

La classe **Client** fait une procédure *handle_request* au serveur (Ligne code 18) si le paramètre *count* est inférieur à 5. Le processus Client est codé comme suivant :

```
process Client(){
vars: Fifo itf_cm, itf_mc;
      int count;
      data msg[];
trans:

0 |- count           := 0;
-> 1;

1 |- [ count >  5 ] ;
-> 6;

1 |- [ count <= 5 ] ;
-> 2;

2 |- msg.size        := 4;
     msg[0].dataType := PROCESS; msg[0] := this;
     msg[1].dataType := PROCESS; msg[1] := NULL;
     msg[2].dataType := STRING;  msg[2] := "rpc";
     msg[3].dataType := INT;     msg[3] := count;
-> 3;

3 |- [ itf_cm.size < itf_cm.max ];
     itf_cm@send (msg);
-> 4;

4 |- itf_mc@recv (msg);
     [ (msg.size = 4)              && (msg[0].dataType = PROCESS) &&
       (msg[1].dataType = PROCESS) && (msg[1]          = this)    &&
       (msg[2].dataType = STRING)  && (msg[2]          = "rpc")   &&
       (msg[3].dataType = INT)                                        ] ;
-> 5;

5 |- count           := msg[3];
-> 0;

};//process Client
```

B.4 Processus Média

Dans cet étude de cas, le média **Rpc** est responsable pour la communication entre les classes Client et Server. Dans un premier temps, il lit le message d'activation dans le *binder* de client puis l'envoie au serveur. Dans un seconde temps il lit le message retourné correspondant (parmi l'autres messages) et l'envoie au client. Le média Rpc est en même forme pour la communication des données utilisant le protocole Rpc. Le processus Rpc est codé comme suivant :

```
process Rpc(){

vars:
      Fifo input_cm, input_mc, target_cm, target_mc;
      Process inputProcess, targetProcess;
      data msg[];
trans:
0 |- input_cm@recv (msg);
     [ (msg[0].dataType = PROCESS) && (msg[1].dataType = PROCESS) &&
       (msg[2].dataType = STRING)  && (msg[2]          = "rpc")       ] ;
-> 1;

1 |- inputProcess  := msg[0];   targetProcess := msg[1];
-> 2;

2 |- [ target_mc.size < target_mc.max ];
     target_mc@send (msg);
-> 3;

3 |- target_cm@recv (msg);
     [ (msg[0].dataType = PROCESS) && (msg[0] = targetProcess) &&
       (msg[1].dataType = PROCESS) && (msg[1] = inputProcess)  &&
       (msg[2].dataType = STRING)  && (msg[2] = "rpc")            ] ;
-> 4;

4 |- [ input_mc.size < input_mc.max ];
     input_mc@send (msg);
-> 0;

};//process Rpc
```

Annexe C

DDD pour LfV

C.1 Codage du modèle en DDD

Nous proposons un codage d'un état du système de LfV comme la concaténation d'une liste des processus et une liste des fifos comme suivant :

$$proc0 \rightarrow proc1 \rightarrow ... \rightarrow procN \rightarrow procMAX \rightarrow$$
$$fifoM \rightarrow ... \rightarrow fifo1 \rightarrow fifo0 \rightarrow fifoMAX$$

où $proc ::= PROC(type); IDENT; var0; ...varL;$ et $fifo ::= FIFO(ident); FIRST; ...$

Les processus et les fifos particulières sont comme les suivants :

$$proc0 ::= procMAIN;$$
$$procMAX ::= PROC(MAX); IDENT(MAX); PNUM;$$
$$fifoMAX ::= FIFO(MAX); FIFONUM;$$

Noms réservés pour des variables DDD : PROC, FIFO, IDENT, NEXT, STATE, PROCNUM, FIFONUM. Les variables du message ont un type non-définie et son nom est différent comme un variable local :

$$var ::= nom; NEXT(val1); NEXT(val2); ...NEXT(valP);$$

C.2 Opérations du modèle en DDD

Après quelques homomorphismes statiques ordinaires tels que l'affectation des valeurs aux variables, les Tests et l'interaction des processus, nous sommes réussi à construire quelques homomorphismes dynamiques tels que la création des processus et Fifos, la manipulation des messages, etc. Par exemple : Le homomorphisme de création des processus est comme le suivant (Où d est une structure DDD du nouveau processus) :

$$ProcCreate(d)(e,x) = \begin{cases} d \hat{} e \xrightarrow{x} Id & \text{si } (e = \text{PROC}) \\ & \& \ (x = \text{MAX}) \\ e \xrightarrow{x} ProcCreate(d) & \text{sinon} \end{cases}$$

La structure de DDD de l'exemple est parue dans les codages au dessous : L'état initial du système LfV :

```
PROC(0) IDENT(0) STATE(0) //main process
PROC(MAX) IDENT(MAX) PROCNUM(1) //default
FIFO(MAX) FIFONUM(-100) //default
```

Un état suivant après la création dynamique des processus et fifos :

```
PROC(0) IDENT(0) STATE(1) //main process
PROC(1) IDENT(1) STATE(UNDEF) X(UNDEF) MSG(UNDEF) //server s
PROC(2) IDENT(2) STATE(UNDEF) Y(UNDEF) MSG(UNDEF) //client c1
PROC(2) IDENT(3) STATE(UNDEF) Y(UNDEF) MSG(UNDEF) //client c2
PROC(MAX) IDENT(MAX) PROCNUM(4) //default
FIFO(-101) //fifo itf_cs
FIFO(-100) //fifo itf_sc
FIFO(MAX) FIFONUM(-102) //default
```

D'autre part, on propose également un homomorphisme spécial *Exec* qui exécute dans tous les processus ayant type t (on appelle un groupe ayant type t) en traversant les DDDs une seule fois pour chaque groupe par rapport une pour chaque processus. Cette technique peut économiser une énorme espace de mémoire et accélérer le calcul.

$$Exec(t, gh)(e, x) = \begin{cases} e \xrightarrow{x} Exec(t, gh) \& gh & \text{si } (e = \text{PROC}) \\ & \quad \& \ (x = t) \\ e \xrightarrow{x} Exec(t, gh) & \text{sinon} \end{cases}$$

Puis on lance le homomorphisme pour le système global en utilisant la composition des homomorphismes *Exec* de tous les groupes.

Duy-Tùng NGUYÊN

Vérification symbolique de modèles à l'aide de systèmes de ré-écritures dédiés

Résumé :

Cette thèse propose un nouveau type de systèmes de ré-écriture, appelé les *systèmes de ré-écriture fonctionnels*. Nous montrons que notre modèle a la puissance d'expression des systèmes de ré-écriture et qu'il est bien adapté à l'étude de propriétés de sûreté et de propriétés de logique temporelle de modèles.
Nous avons mis en évidence une sous classe de systèmes fonctionnels, les *élémentaires* et les *élémentaires à droite*, préservant la puissance d'expression des systèmes fonctionnels et des techniques d'accélération des calculs aboutissant à un outil de vérification symbolique efficace.
Dans la partie expérimentale, nous avons comparé notre outil, d'une part avec des outils de ré-écriture tels que Timbuk, Maude et TOM, d'autre part avec des outils de vérification tels que SPIN, NuSMV, SMART, HSDD. Nos benchmarks démontrent l'efficacité des systèmes fonctionnels élémentaires pour la vérification de modèles.

Mots clés : vérification symbolique, systèmes de ré-écriture des termes, diagrammes de décisions binaires, algorithme de saturation, logique temporelle linéaire.

Symbolic model-checking based on rewriting systems

Abstract :

This PhD thesis proposes the theoretical foundations of a new formal tool for symbolic verification of finite models. Some approaches reduce the problem of system verification to the reachability problem in term rewriting systems (TRSs).
In our approach, states are encoded by terms in a BDD-like manner and the transition relation is represented by a new rewriting relation so called *Functional Term Rewriting Systems* (FTRSs).
First, we show that FTRSs are as expressive as TRSs. Then, we focus on a subclass of FTRSs, so called *Elementary Functional Term Rewriting Systems* (EFTRSs), and we show that EFTRSs preserve the FTRSs expressiveness. The main advantage of EFTRSs is that they are well adapted for acceleration techniques usually used in saturation algorithms on BDD-like data structures.
Our experiments show that for well-known protocols (e.g. Tree Arbiter, Percolate, Round Robin Mutex protocols,...) our tool is not only better than other rewriting tools such as Timbuk, Maude or TOM, but also competitive with other model-checkers such as SPIN, NuSMV or SMART. Moreover, it can also be applied to model-checking invariant properties which are a particular subclass of linear temporal logic formula (LTL).

Keywords : symbolic verification, term rewriting system, binary decision diagram, saturation algorithm, linear temporal logic

Laboratoire d'Informatique Fondamentale d'Orléans
Bat. 3IA, Université d'Orléans
Rue Léonard de Vinci, B.P. 6759
F-45067 ORLEANS Cedex 2