THÈSE

présentée par

Anne Labroue

Méthodes algébriques pour la vérification des systèmes infinis

Thèse soutenue le 14 janvier 2002 devant le jury :

Philippe Schnoebelen,	Directeur de thèse
Brigitte Rozoy ,	Présidente
Luca Aceto, Ahmed Bouajjani ,	Rapporteurs
Pierre Rety, David Janin, Alain Finkel,	Examinateurs

Thèse préparée au sein du Laboratoire Spécification et Vérification de l'ENS Cachan.

Remerciements

Je remercie Philippe Schnoebelen d'avoir encadré mon travail de recherche, depuis mon stage de DEA jusqu'à cette thèse. Je lui suis reconnaissant pour sa patience.

Je remercie Béatrice Bérard pour son soutien et sa disponibilité.

Je remercie tous ceux qui ont participé à la relecture de cette thèse en particuliere Xavier Urbain pour sa minutie.

Je remercie Luca Aceto, Ahmed Bouajjani, Alain Finkel, David Janin, Pierre Rety et Brigitte Rozoy d'avoir accepté de faire partie de mon jury.

Je remercie les membres du LSV et tous les thésards que j'ai côtoyés de m'avoir supportée.

Je remercie ma famille pour m'avoir soutenue et encouragée durant toutes mes études.

Je remercie Lucie pour son amitié.

Enfin, je remercie Pierre-Cyrille et Arthur pour toutes les joies qu'ils m'apportent quotidiennement.

Table des matières

1. Présentation générale

Les systèmes informatiques sont présents dans la plupart des secteurs d'activités : télécommunications, finance, internet, transports, énergie, santé, etc. Les tâches qu'ils gèrent sont de plus en plus complexes et sont parfois critiques, c'est-à-dire qu'une défaillance peut avoir des conséquences humaines et/ou économiques graves. Nous citerons à titre d'exemples le système de contrôle-commande d'une centrale nucléaire, le système de surveillance du réseau électrique d'un pays, le système de commande de vol d'un avion ou le système opérant des transactions d'argent électronique.

Il est donc crucial de disposer de méthodes de conception rigoureuses, automatiques et efficaces pour ces systèmes. Il est clair que plus tôt une erreur sera détectée, plus simple sera sa correction et moindre sera son coût. Il en résulte un grand besoin d'utiliser des outils de vérification lors de la conception d'un logiciel.

Cette thèse s'inscrit dans le cadre de la vérification des systèmes infinis c'est-à-dire ceux qui ne peuvent pas être modélisés par des automates à nombre fini d'états. C'est un domaine important car il regroupe, entre autres, les systèmes logiciels manipulant des structures de données complexes et des variables parcourant des domaines infinis (canaux de communication, compteurs) ainsi que les systèmes temps-réels tels que les protocoles de communication utilisant des horloges, les applications multimédia sur les réseaux.

Dans ce domaine, le défi actuel est de proposer des modèles qui soient suffisamment expressifs pour décrire la réalité des systèmes tout en restant analysables. Ces dernières années ont vu l'émergence de nombreux résultats de la forme : telle propriété est décidable pour tel système sous telles hypothèses. Ceci a permis de proposer des outils de vérification pour des systèmes infinis comme Kronos [Yov97] et UPPAAL [LPY97] pour les automates temporisés ou HyTech [HHW97] pour les systèmes hybrides.

Une manière consensuelle de modéliser la sémantique des systèmes informatiques est de la présenter sous forme d'un graphe étiqueté orienté, appelé **système de transitions étiquetées**, dont les nœuds représentent les différents états possibles du système et où la présence d'un arc entre deux nœuds s et s' étiqueté par a signifie que le système s'il est dans l'état représenté par s peut faire une action désignée par a et être ensuite dans l'état représenté par s'. Plus formellement,

Définition 1.1 *Un* **système de transitions étiquetées** *est un quadruplet* $(S, \Sigma, \rightarrow, s_0)$ *où*

- S *est un ensemble d'états,*
- Σ *est un ensemble fini de noms d'actions,*
- $\rightarrow \subseteq S \times \Sigma \times S$ *est une relation de transitions, nous noterons* $s \xrightarrow{a} s'$ *pour* $(s, a, s') \in \rightarrow$,
- $s_0 \in S$ *est l'état initial.*

Dans la suite, nous notons $\rightarrow^*$ la clôture transitive de $\rightarrow$.

Un système est dit **fini** lorsque l'ensemble des états du système de transitions qui lui est associé est fini. Sinon, il est **infini**. Cette thèse porte sur certains systèmes infinis.

1.1. Les systèmes infinis

1.1.1. Différentes causes d'infini

Les raisons pour lesquelles un système informatique est infini sont multiples, nous allons présenter celles qui nous semblent les plus fréquentes.

La récursivité

Une raison qui amène naturellement un nombre d'états infinis est la récursivité.

Elle est présente dans les réseaux de Petri [Pet62] qui constituent un des modèles les plus utilisés du parallélisme et des systèmes distribués (depuis plus de quarante ans). Ils sont formés d'un nombre fini de places dans lesquelles se trouvent des jetons et d'un nombre fini de transitions qui ont en entrée et en sortie des places. Un état du système de transitions associé à un réseau de Petri est formé du nombre de jetons de chacune des places. Lorsqu'une transition est tirée, le nombre total de jetons des places situées en entrée et en sortie est modifié. En particulier, le nombre de jetons peut augmenter. De plus, de nouvelles transitions deviennent tirables. La récursivité vient du fait que les transitions se rendent tirables les unes les autres.

On trouve également la récursivité dans la plupart des algèbres de processus. Une algèbre de processus représentative est CCS (Calculus of Communicating Systems) [Mil89], elle fournit un langage qui permet de décrire très naturellement les systèmes concurrents en utilisant des constructions dont l'interprétation est intuitive comme la composition parallèle et le renommage. Un processus CCS est défini à l'aide d'une famille finie d'équations de processus récursives. Ces équations sont de la forme $X = E$ où X est une variable de processus et E est un terme CCS donné, c'est-à-dire un élément de l'ensemble donné par la syntaxe :

$$E, F ::= Nil \mid Y \mid aE \mid E + F \mid E \parallel F \mid E \setminus L \mid E[f]$$

où Y est une variable de processus, a un nom d'action, L un ensemble d'actions et f une fonction de l'ensemble des noms d'actions dans lui-même appelée **fonction de renommage**. Le symbole Nil désigne le processus qui ne fait aucune action, aE représente le processus qui fait une action a et se comporte comme le processus E, $E + F$ représente le choix entre se comporter comme E ou comme F, $E \parallel F$ représente la composition entre E et F qui permet aux processus d'évoluer indépendamment l'un de l'autre ou de se synchroniser sur des noms d'actions complémentaires, $E \setminus L$ représente le processus qui se comporte comme E auquel on aurait interdit les actions non présentes dans L, enfin, $E[f]$ représente le processus E où tous les noms d'actions ont été renommés en fonction de f. Le système de transitions associé est donné par des règles de sémantique opérationnelle structurelle (SOS).

Ce modèle étant très expressif – on peut réduire le problème de l'arrêt d'une machine de Turing en un problème de décision de la bisimulation entre deux processus CCS [Mil89] (nous y reviendrons plus tard) – de nombreux fragments de CCS ont été étudiés, notamment l'algèbre BPP (Basic Parallel Process) [Chr93] qui interdit le renommage, la restriction et surtout la synchronisation.

Une autre algèbre très employée est l'algèbre PA (Process Algebra) [BW90] qui désigne l'algèbre interdisant le renommage, la restriction et la synchronisation mais qui utilise un opérateur séquentiel ; $E.F$ désigne alors le processus qui se comporte comme E jusqu'à ce qu'il ne puisse plus faire d'action, puis se comporte comme F. L'algèbre BPA (Basic Process Algebra) [BK85] est un fragment de PA obtenu en interdisant la composition parallèle.

La manipulation de variables parcourant des domaines infinis

La manipulation de variables parcourant des domaines infinis conduit également à considérer des systèmes infinis. En effet, les états du système de transitions associé à ce type de modèles sont composés de la valeur de chacune des variables et éventuellement d'une information de contrôle. Si une variable n'est pas bornée, alors le nombre d'états est infini.

Les machines de Minski [Min67] (les machines à compteurs) appartiennent à cette classe de modèles. Les états de leur système de transitions sont composés d'un état de contrôle et de la valeur de chacun des compteurs. Lors des transitions les compteurs peuvent être incrémentés, décrémentés ou remis à zéro.

Elle regroupe également les machines utilisant des piles ou des files comme les automates à pile dont les états du système de transitions associé sont composés d'un état de contrôle et du contenu de la pile.

Dans cette classe, figurent également les systèmes d'automates communicants [Boc78] qui se composent d'un ensemble d'automates, représentant chacun un processus et d'un ensemble de canaux, de taille non bornées, à travers lesquels ces processus communiquent en s'envoyant des messages. Ils sont utilisés de façon plus ou moins explicite dans les langages de spécification d'algorithmes répartis comme ESTELLE ou SDL [Tur93] et permettent de vérifier des protocoles de communication comme le protocole du bit alterné [BSW69].

Une variable particulière : le temps

De nombreux modèles permettent d'étudier les systèmes temps-réel qui sont des systèmes devant satisfaire des contraintes temporelles strictes sur des délais séparant certains événements. Là encore, il n'y a pas une façon standard de représenter le temps. Les points principaux de choix de modélisation sont la nature du domaine du temps –discret ou dense– et l'existence d'une référence de temps absolue telle qu'une horloge globale.

Un exemple très étudié est celui des automates temporisés [AD94]. Ils se composent de deux éléments : un automate fini au sens habituel qui décrit les états de contrôle du système et les transitions supposées instantanées entre ces états et un ensemble d'horloges utilisé pour spécifier des contraintes quantitatives de temps qui peuvent être associées aux transitions de l'automate fini. Ici le temps est dense, représenté par les réels positifs ou nuls, il y a une référence du temps absolue et les actions peuvent être différées. Une extension proposée dans l'outil UPPAAL [LPY97] permet de considérer des actions urgentes.

Un autre domaine où le temps a été introduit est celui des algèbres de processus, un bilan des extensions de l'algèbre ACP [BV01] (une extension de l'algèbre PA qui considère les blocages et permet plusieurs formes de synchronisation) est proposé par J.C.M. Baeten et C.A. Middelburg [BM01]. Nous remarquons simplement que ces extensions ont un domaine de temps discret ou dense, manipulent des actions urgentes ou non [BB91, BB96]. Ces algèbres sont des systèmes dont le caractère infini a deux origines : le temps et la récursivité.

La paramétrisation de la taille du système

Les systèmes paramétrés sont généralement considérés comme des systèmes infinis. Ils désignent des systèmes composés d'un nombre n fini mais arbitrairement grand de machines identiques que l'on cherche à étudier pour toutes les valeurs de n ou comprenant une constante qui est un paramètre. Par exemple, les systèmes de protocole broadcast [EN98] sont des systèmes composés d'un nombre arbitrairement grand de processus indissociables qui communiquent par rendez-vous (deux processus échangent un message) ou par broadcast (un processus envoie un message à tous les autres).

1.1.2. Comment classer tous ces modèles ?

Nous venons de voir qu'il y a de nombreux modèles de systèmes infinis. Il est naturel de chercher à les ordonner. À notre connaissance, il n'existe pas de classification qui regroupe tous ces modèles.

T.A. Henzinger et R. Majumdar [HM00] proposent une classification des systèmes infinis. Elle regroupe les systèmes en fonction des équivalences comportementales pour lesquelles le nombre de classes d'équivalence de ces systèmes est fini. Il manque beaucoup de résultats sur les équivalences comportementales pour pouvoir associer à un modèle sa classe.

Toutefois, certaines études présentent un ordonnancement de sous-groupes. C'est notamment le cas de la thèse de R. Mayr [May98] qui fait suite aux travaux de D. Caucal [Cau92] et F. Moller [Mol96]. Elle propose une unification du modèle des réseaux de Petri [Pet62] et de plusieurs algèbres de processus en les voyant comme des sous-classes des systèmes de réécriture de processus (PRS). La figure 1.1 présente cette classification.

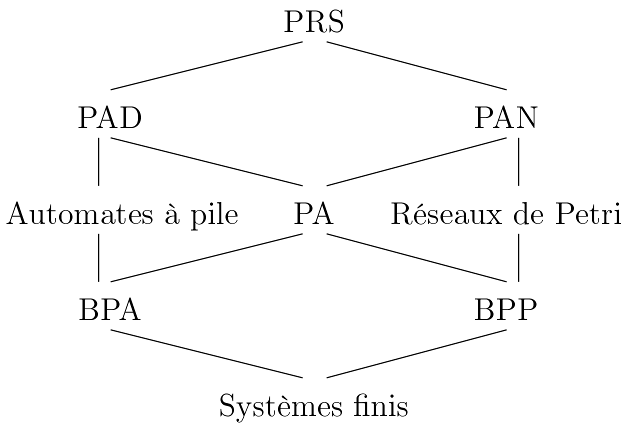


FIG. 1.1. – La hiérarchie PRS.

Les modèles sont de moins en moins expressifs quand on lit la figure de haut en bas, les modèles situés sur un même niveau sont incomparables.

1.2. La vérification

La vérification des systèmes infinis porte sur trois types de propositions qui ne sont pas disjoints. Il y a tout d'abord les questions classiques de terminaison, d'accessibilité, de caractère borné, etc. Ensuite, le **model-checking** temporel désigne la vérification des propriétés qui s'expriment dans une logique temporelle (les questions classiques peuvent souvent s'exprimer dans une logique temporelle). Enfin, pour vérifier qu'un système a un certain comportement, il est courant, notamment dans le domaine des algèbres de processus, d'exprimer le comportement attendu dans le même modèle et de regarder si les deux systèmes sont équivalents pour une certaine **équivalence comportementale**. Un bilan de la vérification des systèmes infinis dans le cadre « algèbre de processus » est proposé par O. Bukart, D. Caucal, F. Moller et B. Steffen [BCMS01].

1.2.1. Les propriétés classiques

La hiérarchie habituelle des propriétés classiques est la suivante :

- Une propriété d'**accessibilité** énonce qu'une certaine situation peut être atteinte ;
- Une propriété de **sûreté** énonce que, sous certaines conditions, quelque chose ne se produit jamais ;
- Une propriété de **vivacité** énonce que, sous certaines conditions, quelque chose finira par avoir lieu ;
- Une propriété d'**équité** énonce que, sous certaines conditions, quelque chose aura lieu (ou n'aura pas lieu) un nombre infini de fois.

L. Lamport a été le premier à proposer d'introduire une distinction entre propriétés de sûreté et propriétés de vivacité [Lam73]. Nous nous arrêtons surtout sur le problème de l'accessibilité, la seconde partie du livre de Ph. Schnoebelen, B. Bérard, M. Bidoit, F. Laroussinie et A. Petit [SBB+99, BBF+01] propose une présentation plus complète de des propriétés classiques.

Les propriétés d'accessibilité

Le problème de l'accessibilité prend en entrée un système de transitions et un état et indique si on peut atteindre cet état.

L'accessibilité est décidable pour les systèmes d'automates communicants avec pertes de messages [AJ96b], indécidable pour les systèmes hybrides [HKPV95].

R. Mayr propose un bilan du problème de l'accessibilité pour les algèbres de processus et les réseaux de Petri qui sont des sous-classes des systèmes de réécriture de processus [May98], il est rappelé dans la figure 1.2.

Systèmes d'états finis	Polynomial
BPA	Polynomial
Automates à pile	Polynomial
BPP	NP-complet [Esp97b]
PA	NP-complet
PAD	NP-complet
Réseau de Petri	Décidable [May84], expspace dur
PAN	Décidable, expspace dur
PRS	Décidable, expspace dur

Fig. 1.2. – L'accessibilité pour les systèmes de réécriture de processus.

Les propriétés de sûreté

Il est difficile de donner une définition formelle des propriétés de sûreté dans le cas général, il existe toutefois des propositions [LS85, ADS86, AS87, Kin94].

M. Maidl propose un algorithme de model-checking pour les propriétés de sûreté des systèmes paramétrés [Mai01]. Celui-ci permet de caractériser une sous-classe de systèmes paramétrés pour laquelle il termine. Elle contient les anneaux à jetons et les systèmes broadcast [EN98] ; la restriction porte sur la façon dont les systèmes communiquent.

Les propriétés de vivacité

De nouveau, il est difficile de donner une définition formelle des propriétés de vivacité dans le cas général, il existe toutefois des propositions [AS85, AS87, Kin94].

Les propriétés d'équité

Un exemple de propriété d'équité est le suivant :

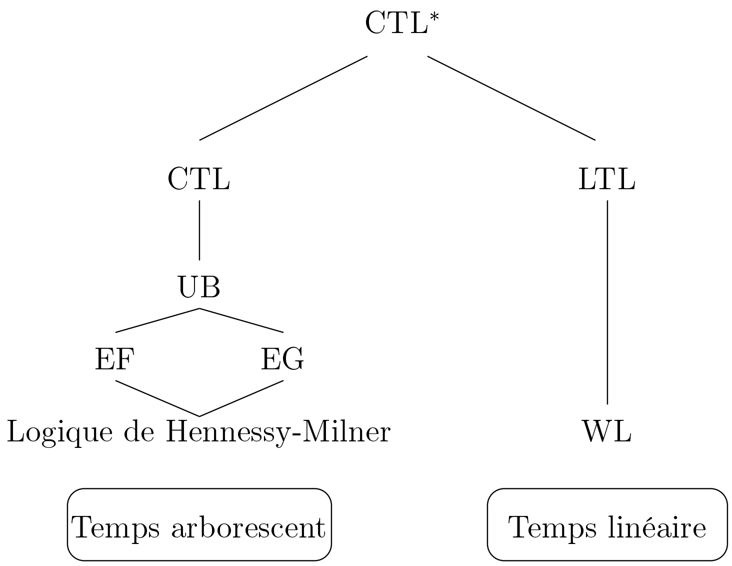
Le problème de l'état récurrent prend en entrée un système de transitions et un ensemble d'états de contrôle et indique s'il existe une exécution partant de l'état initial qui passe une infinité de fois par l'un de ces états de contrôle.

Ce problème est indécidable pour les systèmes d'automates communicants avec perte de messages [AJ96a].

1.2.2. Le model checking

Le problème du model checking prend en entrée un système donné par son système de transitions étiquetées et une propriété donnée par une formule de logique temporelle. L'algorithme de model checking répond à la question : le système de transition satisfait la propriété ? De nombreuses études de cas ont montré que cette approche permet de détecter les erreurs très tôt [CW96].

Il existe deux classes de logiques temporelles, les **logiques de temps arborescent** dont la plus utilisée pour le model checking est la logique CTL (Computation Tree Logic) [CE81] et les **logiques de temps linéaire** dont la plus utilisée pour le model checking est la logique LTL (Linear-Time Logic) [Pnu77]. Les logique CTL et LTL peuvent être vues comme des fragments de la logique CTL* [EH86]. La figure 1.3 reprend la classification en fonction de l'expressivité des logiques de temps linéaire et de temps arborescent. Nous allons présenter les logiques CTL*, CTL et LTL.



Lorsque deux logiques sont reliées par un trait, celle du dessus est plus expressive que celle du dessous.

FIG. 1.3. – Logiques de temps arborescent et linéaire.

La logique CTL*

Étant donné un systèmes de transitions $(S, \Sigma, \rightarrow, s_0)$, nous considérons un ensemble de **propositions atomiques** $Prop = \{P_1, P_2, \ldots\}$, dont la sémantique est donnée par des ensembles d'états.

Définition 1.2 *La **logique CTL*** a la syntaxe suivante :*

$$\varphi ::= P \mid \neg\varphi \mid \varphi \wedge \varphi' \mid \mathsf{X}\varphi \mid \mathsf{F}\varphi \mid \mathsf{G}\varphi \mid \varphi\mathsf{U}\varphi' \mid \mathsf{E}\varphi \mid \mathsf{A}\varphi.$$

En plus des combinateurs booléens, on distingue deux types de combinateurs : $\mathsf{X}, \mathsf{F}, \mathsf{G}, \mathsf{U}$ sont les **combinateurs temporels** et E, A sont les **quantificateurs de chemin**. Nous ne donnerons ici qu'une présentation informelle de ces combinateurs (la logique EF est présentée plus amplement dans le paragraphe 4.4.1).

Les combinateurs temporels permettent de parler de l'enchaînement des états le long d'une exécution. $\mathsf{X}\varphi$ énonce que «l'état suivant vérifie φ», $\mathsf{F}\varphi$ énonce qu' «un état futur vérifie φ», $\mathsf{G}\varphi$ énonce que «tous les états futurs vérifient φ» et $\varphi\mathsf{U}\varphi'$ que «φ est vérifiée jusqu'à ce que φ' le soit».

Les quantificateurs de chemin permettent de parler de l'ensemble des exécutions. $\mathsf{A}\varphi$ énonce que «toutes les exécutions partant de l'état courant satisfont φ» et $\mathsf{E}\varphi$ énonce qu' «il existe une exécution partant de l'état courant qui satisfait φ».

Les logiques LTL, CTL et EF

La logique LTL est le fragment obtenu à partir de CTL* lorsque l'on retire les quantificateurs de chemin. Une formule de LTL ne considère qu'une exécution donnée et pas un ensemble d'exécution. C'est une logique de temps linéaire.

La logique CTL est le fragment obtenu à partir de CTL* quand on exige que chaque utilisation d'un combinateur temporelle soit immédiatement dans la portée d'un quantificateur de chemin. Une formule de CTL porte sur l'état courant, pas sur une exécution particulière. C'est une logique de temps arborescent.

Bien que moins expressive que la logique CTL*, il arrive que la logique CTL soit encore trop complexe. Par exemple, le model checking de la logique CTL est indécidable pour l'algèbre PA [EK95]. On peut alors considérer un fragment de la logique CTL pour pouvoir faire du model checking comme par exemple la logique EF dont la syntaxe est $\varphi ::= P \mid \neg\varphi \mid \varphi \wedge \varphi' \mid \mathsf{EX}\varphi \mid \mathsf{EF}\varphi$.

R. Mayr [May98] fait le point sur le model-checking des algèbres de processus pour les logiques temporelles. Il est rappelé dans la figure 1.4 et la figure 1.5.

	EF	CTL
Systèmes d'états finis	Polynomial	Polynomial [CGL94]
BPA	Pspace-complet [BEM97]	Exptime, Pspace-dur
Automates à pile	Pspace-complet	Exptime [I. 96], Pspace-dur
BPP	Pspace-complet [Esp97a]	Indécidable [EK95]
PA	Pspace-dur [May97]	Indécidable [EK95]
Réseaux de Petri	Indécidable [Esp97a]	Indécidable [EK95]

FIG. 1.4. – Model-checking pour les logiques EF et CTL.

	LTL
Systèmes d'états finis	Pspace complet [SC85]
BPA	exptime-complet [May98]
Automates à pile	exptime-complet [BEM97]
BPP	expspace-dur [Esp94]
PA	indécidable [BH96]
Réseaux de Petri	expspace-dur [Esp94]

FIG. 1.5. – Model-checking pour la logique LTL.

1.2.3. Les équivalences comportementales

Une question fondamentale est : quand peut-on dire que deux systèmes ont le même comportement ? Pour répondre à cette question un très grand nombre d'équivalences comportementales ont été définies. R. Van Glabbeek [Gla90] les a ordonnées dans la hiérarchie de temps linéaire-arborescent des équivalences illustrée par la figure 1.6 (la figure est en anglais car certains noms d'équivalences se traduisent difficilement).

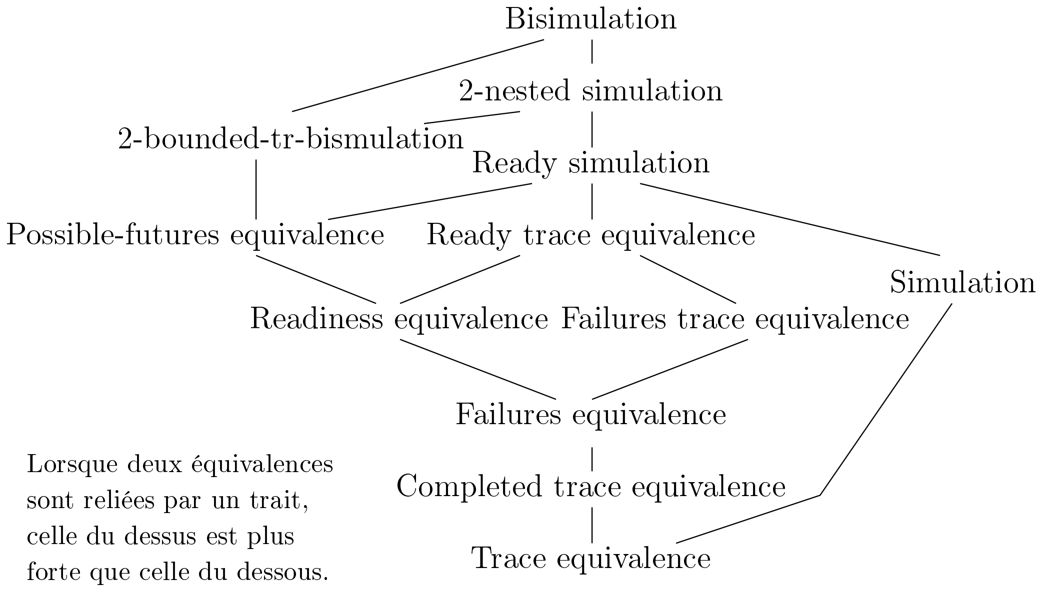


FIG. 1.6. – La hiérarchie de R. Van Glabbeek.

Nous ne définirons que la bisimulation forte, introduite par D.M.R. Park [Par81] et l'équivalence de traces qui vient de la théorie des automates finis. Une hiérarchie analogue est obtenue quand on considère des actions inobservables (τ), les équivalences sont alors qualifiées de faibles ; ici, nous ne parlerons que de la bisimulation faible.

Définition 1.3 *Deux systèmes de transitions étiquetées sont équivalents pour l'**équivalence de traces** s'ils ont le même ensemble de traces maximales.*

Nous allons maintenant définir la bisimulation :

Définition 1.4 *Soit $(S, \Sigma, \rightarrow, s_0)$ un système de transitions étiquetées. Une relation $\mathcal{R} \subseteq S \times S$ est appelée une **relation de bisimulation** si $u \mathcal{R} v$ entraîne*

1. *Pour tout $u \stackrel{a}{\rightarrow} u'$, il existe $v \stackrel{a}{\rightarrow} v'$ tel que $u' \mathcal{R} v'$,*
2. *Et vice versa : pour tout $v \stackrel{a}{\rightarrow} v'$, il existe $u \stackrel{a}{\rightarrow} u'$ avec $u' \mathcal{R} v'$.*

*La **bisimulation** est la plus grande des relations de bisimulation (elle est unique et existe).*

Soit $(S, \Sigma, \rightarrow, s_0)$ un système de transitions étiquetées. Nous supposons qu'il existe un nom d'action τ qui représente une action inobservable, nous définissons une nouvelle relation de transition $\Rightarrow$ par $\stackrel{\tau}{\Rightarrow} = \stackrel{\tau}{\rightarrow}^*$ et pour tout $a \in \Sigma \setminus \{\tau\}$, $\stackrel{a}{\Rightarrow} = \stackrel{\tau}{\rightarrow}^* \stackrel{a}{\rightarrow} \stackrel{\tau}{\rightarrow}^*$. La définition de la bisimulation faible pour $(S, \Sigma, \rightarrow, s_0)$ est la même que celle de la bisimulation forte pour $(S, \Sigma, \Rightarrow, s_0)$.

	Bisimulation	Bisimulation avec un système fini
Systèmes d'états finis	P-complet [BGS92]	
BPA	2exptime [BCS97b] Polynomial dans le cas normé [HJM94]	Donné : Polynomial [KM99] Non Donné : 2exptime [BCS97a]
Automates à pile	Décidable [Sén98] Pspace-dur [May01]	Donné : exptime [JKM00] Non Donné : pspace-dur [May01]
BPP	co-NP dur [May00] Polynomial dans le cas normé [HJM96]	Donné : pspace [JKM00] Non Donné : co-NP dur [May00]
PA	co-NP dur [Jan95] Décidable dans le cas normé [HJ99]	Donné : décidable [JKM00] Non Donné : co-NP dur [May00]
PAD		Décidable [JKM00]
réseaux de Petri	Indécidable [Jan95]	Expspace-dur [May01]
Automates à un compteur	Décidable [Jan97]	Décidable [Jan97]
Automates communicants avec perte de messages	Indécidable [Sch01]	

FIG. 1.7. – La bisimulation forte.

O. Bukart, D. Caucal, F. Moller et B. Steffen [BCMS01] ont rassemblés un grand nombre de résultats de décidabilité de la bisimulation forte et la bisimulation faible, nous les avons complétés. La figure 1.7 regroupe les résultats pour la bisimulation forte et la figure 1.8 pour la bisimulation faible.

	Bisimulation	Bisimulation avec un système fini donné
BPA	Pspace-dur [Str00]	Polynomial [KM99]
Automates à pile	Pspace-dur [Str00]	Exptime [JKM00] Pspace-dur [May01]
BPP	NP-dur [Str00] Π_2^p dure [May00]	Pspace [JKM00]
PA	Pspace dur [Str00]	Décidable [JKM00]
Réseaux de Petri	Indécidable [Jan95]	Indécidable [EJM99]

FIG. 1.8. – La bisimulation faible

Nous ajouterons que toutes les équivalences figurant entre la bisimulation et l'équivalence de langage dans la hiérarchie de Van Glabbek [Gla90] sont indécidables pour BPP et BPA [HT95, GH94, Hüt94].

1.2.4. La vérification des systèmes temporisés

Comme nous venons de le voir, dans le domaine des sémantiques de concurrence, une approche traditionnelle et bien acceptée pour comparer deux processus consiste à déterminer s'ils sont liés par une équivalence comportementale. Afin d'étendre cette démarche au cas temporisé, une notion robuste de **performance** a été introduite pour comparer l'efficacité de systèmes qui ont le même comportement (ils font les mêmes actions), mais qui ont des vitesses différentes (ils ne les font pas à la même vitesse). Elle a donné naissance à plusieurs préordres [MT91, AH92, GRS95, CGR97] et équivalences [FM95, AM96, Cor98, CD01] dont notamment l'équivalence de performance [GRS95, CGR97] qui n'est autre que la bisimulation pour un système de transitions où les étiquettes ont comme deuxième composante une date.

Les préordres et les équivalences d'efficacité considérés par F. Corradini, R. Gorrieri, M. Roccetti, E. Stancampiano [GRS95, CGR97, Cor98] d'une part et L. Aceto et D. Murphy [AM96] d'autre part concernent des algèbres de processus où les composantes parallèles ont chacune leur propre horloge locale et où les actions ont une **durée** et sont **urgentes**. Une action urgente a lieu aussitôt que possible, sauf dans le cas où le processus qui la produit doit attendre pour se synchroniser avec un autre, auquel cas elle est retardée. Si l'algèbre de processus n'autorise pas la synchronisation, l'équivalence de performance est une congruence pour tous les constructeurs de processus [CGR97].

Une étude plus récente [Cor98, CD01, CP01] porte sur des algèbres plus larges où les actions peuvent être retardées en dehors de toute synchronisation. Dans ce cas, l'équivalence de performance n'est pas une congruence, c'est pourquoi les auteurs ont introduit une congruence de performance.

Tous ces travaux portent principalement sur la sémantique. Toutefois, certains problèmes de vérification ont été étudiés :

1. F. Corradini et M. Pistore [CP96] ont montré que l'**équivalence de performance sans urgence** est décidable pour une classe de systèmes ayant un contrôle fini, mais ayant un nombre infini de configurations quand on tient compte des valeurs des horloges locales.
2. Xiao Jun Chen et F. Corradini [CC97] ont prouvé que le problème du model checking pour TAL (une logique modale temporisée) est décidable pour une autre classe de systèmes ayant un contrôle fini.

1.3. Différentes techniques de vérification

– Les systèmes de tableaux sont un outil classique de la logique. L'application des systèmes de tableaux pour des problèmes de vérification a été initiée par C. Stirling et J. Bradfield [BS90] pour vérifier des propriétés

temporelles. Ensuite, ces tableaux ont permis de montrer des résultats de décidabilité [CHM93, May98].
Un tableau est un arbre de preuves dont les nœuds sont étiquetés par une expression logique ou un ensemble d'expressions logiques. Le but du tableau est de prouver la correction de la formule qui est à la racine du tableau. Chaque nœud doit vérifier une règle de tableau avec ses fils.

- On peut montrer que certains systèmes infinis ont une relation d'accessibilité reconnaissable. C'est le cas pour BPP, elle est semilinéaire [Esp97b] ; pour les automates temporisés, elle s'exprime dans la théorie additive des réels [CJ99] ; pour des systèmes de réécriture de caractères comprenant les automates à pile, c'est une relation de mots reconnaissable [Cau00] ; pour les termes PA sans congruence structurelle, elle est reconnaissable par des automates d'arbres [LS01]. On en déduit que l'accessibilité est décidable, mais plus encore, par exemple que la bisimulation faible est semi-décidable dans le cas de BPP [Esp97b].

- Les **systèmes de transitions bien structurés** sont les systèmes infinis qui sont munis d'un beau préordre $\leq$ (c'est-à-dire que dans toute suite infinie d'états, il existe deux indices $i < j$ tels que l'état d'indice i soit plus petit que celui d'indice j) qui est compatible avec la relation de transitions (c'est-à-dire que si $s \leq t$ et $s \to s'$, il existe t' tel que $t \overset{*}{\to} t'$ et $s' \leq t'$). Cette classe de systèmes infinis a été introduite par A. Finkel [Fin87] et P. A. Abdulla, K. Čerāns, B. Jonsson et T. Yih-Kuen [AČJY96].
A. Finkel et Ph. Schnoebelen [FS01] recensent les propriétés, telles que la décidabilité de l'accessibilité, qui sont vérifiées par les systèmes de transitions bien structurés en fonction de la compatibilité du beau préordre avec la relation de transition, de la décidabilité du beau préordre et de la calculabilité de *Pre*.
Ils présentent également un certain nombre de modèles que l'on peut munir d'un beau préordre décidable tels que les réseaux de Petri [Pet62], BPA [BK85], BPP [Chr93], les automates communicants avec perte de messages [Boc78].

- Pour montrer que l'accessibilité des régions est décidable pour les automates temporisés, R. Alur et D. L. Dill [AD94] ont construit un automate fini appelé **automate des régions** qui reconnaît un langage non temporisé. Cette méthode est reprise par P. Bouyer, C. Dufourd, E. Fleury et A. Petit [BDFP00], pour des automates temporisés plus généraux où il est possible de mettre à jour les horloges lors des transitions en leur donnant la valeur d'un réel, d'une autre horloge ou les décrémentant.

- Dans le domaine des algèbres temporisées, les méthodes de décision de l'équivalence comportementale [CP96] ou du model checking pour une certaine logique temporisée [CC97] consistent jusqu'à présent à construire une approximation finie du système pour laquelle le problème original peut être résolu avec des méthodes standard pour les systèmes d'états finis. Cela induit des algorithmes qui sont en temps exponentiel car l'approximation finie est de taille exponentielle. Elle consiste à ne pas tenir compte des valeurs exactes des horloges (ou les différences entre elles) quand elles sont assez grandes, ces travaux ont des points communs avec la technique du graphe des régions de R. Alur, C. Courcoubetis et D. Dill [ACD93].

1.4. Les travaux réalisés dans cette thèse

Cette thèse propose des résultats de deux sortes.

1.4.1. Sur les schémas de programmes récursif-parallèles

Dans la première partie de cette thèse, nous nous intéressons à la vérification des schémas RPPS (Recursive Parallel Program Schemes) qui ont été introduits par O. Kouchnarenko et Ph. Schnoebelen [KS97b, KS97a] pour modéliser de façon abstraite le contrôle de flot des langages de programmation ayant des sous-programmes parallèles récursifs. L'analyse de l'accessibilité de ce type de modèle a de nombreuses applications dans l'analyse statique des langages de programmation parallèle [EK99, EP00].

Dans leurs travaux, O. Kouchnarenko et Ph. Schnoebelen voient les schémas RPPS comme une variante des réseaux de Petri. Ici nous les présentons comme une algèbre de processus, ce qui nous permet de reprendre l'approche de D. Lugiez et Ph. Schnoebelen [LS01] qui utilisent, comme nous l'avons vu, des automates d'arbres pour reconnaître l'accessibilité entre deux termes PA [BW90].

Notre contribution.

La première partie de cette thèse propose trois résultats. Tout d'abord, nous définissons RPA une nouvelle algèbre de processus qui permet coder les schémas RPPS de façon précise. Ensuite, nous montrons que l'atteignabilité entre des termes RPA est une relation reconnaissable, nous utilisons des automates d'arbres alternés pour que la preuve soit plus lisible. Enfin, nous montrons comment l'accessibilité entre deux états emboîtés RPPS peut se réduire à des questions d'accessibilité entre des termes RPA et nous présentons un algorithme qui résout ce problème. Une conséquence de ce résultat est que l'accessibilité entre deux états emboîtés RPPS est un problème NP-complet.

La difficulté de ce travail vient du fait que lorsque l'on code naturellement les états emboîtés RPPS par des processus, il devient compliqué de définir la relation de transitions de l'algèbre par des règles SOS tout en ayant une relation d'accessibilité reconnaissable. En particulier, nous n'avons pas vu comment réutiliser l'algèbre PA.

Jusqu'ici, les travaux de décidabilité sur les schémas RPPS reposaient sur des méthodes de tableaux adhoc [KS97b, KS97a] ou sur le fait que les schémas RPPS sont bien structurés [FS01]. Nous proposont des résultats plus forts à la fin de la première partie.

Par ailleurs, nos résultats permettent d'ajouter l'algèbre RPA à la liste, présentée dans la section 1.3, des systèmes dont l'accessibilité est reconnaissable. Notre approche est plus forte que l'approche de model-checking régulier [BJNT00] dans la mesure où nous pouvons calculer les successeurs et prédécesseurs itérés et non pas seulement les successeurs et prédécesseurs immédiats.

1.4.2. Sur une extension temporisée de BPP

Dans la deuxième partie de cette thèse, nous nous intéressons à la décidabilité de l'**équivalence de performance** introduite par F. Corradini, R. Gorrieri et M. Roccetti [CGR97] quand il n'est fait aucune restriction sur le nombre d'états. Dans ce cadre, comme aucune synchronisation n'est autorisée, les systèmes considérés semblent être obtenus à partir de l'algèbre BPP (Basic Parallel Processes [Chr93]) en ajoutant du temps par l'intermédiaire d'horloges locales. C'est pourquoi nous désignons cette algèbre TBPP (Timed BPP).

La bisimulation pour BPP est un problème décidable [CHM93] et co-NP dur [May00]. Nous voulions voir ce qu'il en était pour BPP plus des horloges locales.

Notre contribution.

Notre résultat principal est que l'équivalence de performance est décidable pour TBPP et qu'elle peut être décidée en temps polynomial (le problème est en fait Ptime-complet). De façon surprenante, l'ajout d'horloges locales ne rend pas le problème plus compliqué : ces horloges permettent de décomposer le système d'une manière proche du cas normé [HJM96].

À notre connaissance, aucun papier ne traite de problème de vérification pour les algèbres de processus ayant un contrôle infini et des horloges locales.

Dans le cadre d'une horloge globale, P. A. Abdulla et B. Jonsson [AJ98] s'intéressent à des systèmes qui peuvent avoir un nombre arbitraire de processus. Il existe également de nombreux travaux sur les réseaux de Petri temporisés mais la plupart d'entre eux ne fournissent pas de résultat de décidabilité pour les réseaux non bornés.

Ces problèmes concernent l'intersection de deux domaines récents : la vérification de systèmes temporisés ayant un contrôle fini et la vérification de systèmes infinis non temporisés. Nous devons donc faire face à deux difficultés.

Première partie .

Analyse de l'atteignabilité des RPPS par des automates d'arbres

2. Introduction

Dans cette partie, nous nous intéressons à la vérification des schémas RPPS (Recursive Parallel Program Schemes) introduits par O. Kouchnarenko et Ph. Schnoebelen [KS97a, KS97b]. Ces schémas constituent une modélisation abstraite du contrôle des langages de programmation ayant des sous-programmes récursifs et parallèles. J. Esparza a montré dans [EK99, EP00] que la décision de l'accessibilité dans ce type de modèles a de nombreuses applications dans l'analyse statique des programmes parallèles. C'est pourquoi nous étudions ce problème de l'accessibilité pour les schémas RPPS.

Récemment, D. Lugiez et Ph. Schnoebelen ont montré [LS00] que le problème de l'accessibilité pour l'algèbre de processus PA ([BW90]) pouvait être résolu simplement et élégamment en utilisant des techniques d'automates d'arbres. Dans cette démarche, une des idées majeures est de commencer par traiter les termes sans la congruence structurelle.

Nous nous sommes inspirés de ce travail pour étudier le problème de l'accessibilité pour les schémas RPPS. Or, contrairement au cas de PA, lorsque nous avons commencé, il n'existait aucune algèbre de processus correspondant aux schémas RPPS modulo une certaine congruence. Nous avons donc construit une algèbre de processus, appelée RPA, répondant aux critères suivants :

- L'accessibilité de termes RPA est reconnaissable par un automate d'arbres ;
- Il existe une congruence sur les termes RPA telle que chaque schéma RPPS corresponde à RPA modulo cette congruence ;
- Cette congruence est en grande partie reconnaissable par un automate d'arbres.

Nous avons enfin montré que l'accessibilité est un problème NP-complet pour les schémas RPPS.

Dans le premier chapitre, nous présentons les schémas RPPS. Dans le deuxième, nous définissons l'algèbre RPA et montrons que l'accessibilité de termes RPA est reconnaissable par un automate d'arbres. Dans le troisième, nous expliquons comment coder un schéma RPPS par une algèbre RPA. Enfin dans le dernier, nous montrons comment relier le problème de l'accessibilité dans un schéma RPPS au problème de l'accessibilité pour RPA, ce qui nous permet de faire du model checking des schémas RPPS.

3. Les schémas de programmes récursifs-parallèles

Sommaire

Les schémas de programmes récursifs-parallèles (RPPS) constituent une modélisation abstraite des programmes écrits dans le langage RPC, un langage développé à l'Institut des Problèmes d'Informatique (IPI) de l'académie des sciences de Russie [VEKM94]. Ce modèle, introduit par O. Kouchnarenko et Ph. Schnoebelen, est décrit en détail dans [Kou97]. Il permet de donner une sémantique aux langages récursifs-parallèles comme Concurrent Prolog [Sha87], d'évaluer l'expressivité de RPC [KS97b] et de développer des outils de vérification pour les programmes RPC [SK95, KS97a].

Dans ce chapitre, nous présentons les aspects spécifiques de la programmation RP (récursive-parallèle) qui sont modélisés par les RPPS, puis nous définissons les RPPS et leur sémantique opérationnelle. Ces définitions sont extraites des articles de O. Kouchnarenko et Ph. Schnoebelen. Pour faciliter leur compréhension, nous les avons illustrées sur un exemple, celui du fonctionnement d'un fast-food.

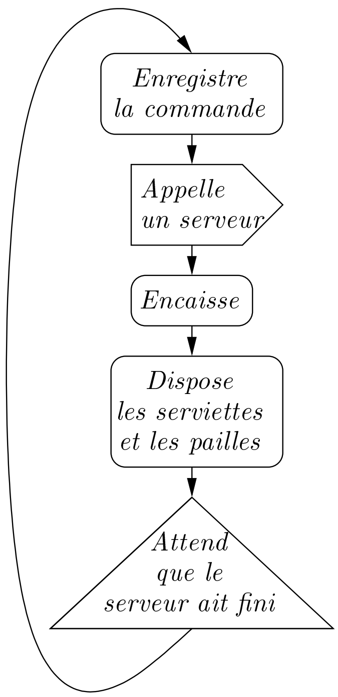
3.1. L'approche récursive-parallèle

La programmation récursive-parallèle (RP) propose une approche particulière de l'organisation du calcul en parallèle. Un **système RP** est un ensemble de sous-systèmes parallèles qui peuvent s'invoquer récursivement. Les spécificités de cette programmation tiennent dans le choix des primitives de synchronisation : `call` et `wait`.

3.1.1. Un exemple : le fast-food

Pour essayer de faire comprendre les fonctionnalités de la programmation RP, nous présentons un exemple qui modélise le fonctionnement très simplifié d'un fast-food.

Ce fast-food emploie cinq types de *travailleurs* : les **caissiers**, les **serveurs**, les **cuisiniers**, les **pâtissiers** et le **distributeur de boissons**. Pour chaque type, nous décrivons les tâches qu'il doit exécuter. Celles des caissiers sont présentées dans la figure 3.1, celles des serveurs dans la figure 3.2 et celles des autres *travailleurs* dans la figure 3.3.



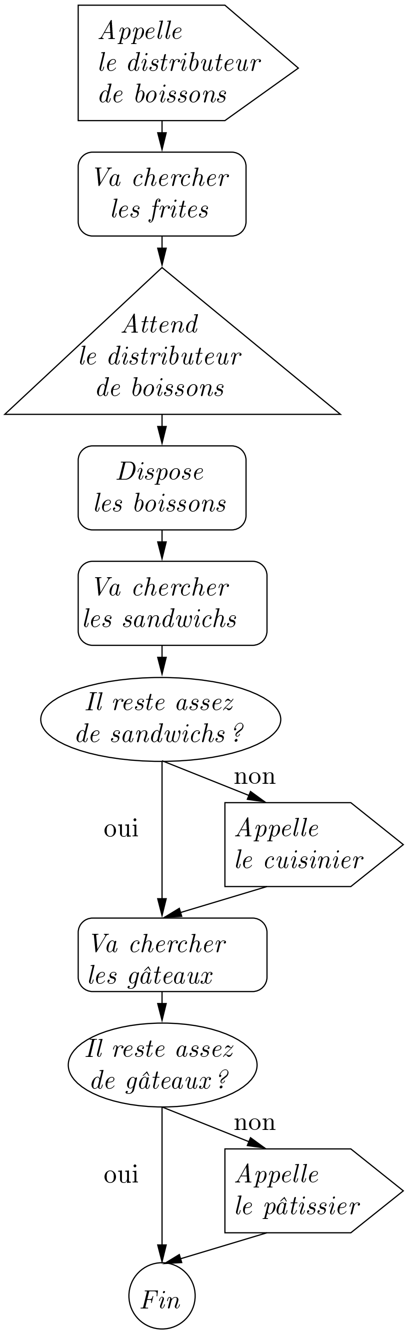
Les tâches d'un caissier sont les suivantes : il enregistre la commande du client, puis il demande à un serveur de commencer son travail.
Pendant qu'un serveur accomplit ses tâches, le caissier encaisse, puis dipose les serviettes et les pailles sur le plateau.
Enfin, le caissier s'assure que le serveur a fini les tâches qu'il lui a commandées et passe à la nouvelle commande.

FIG. 3.1. – Le caissier.

Les schémas qui sont utilisés dans ces figures sont composés de boîtes qui contiennent les tâches à exécuter et de flèches qui permettent de suivre leur ordonnancement. Les boîtes ont différentes formes. Chaque forme a une signification particulière qui va correspondre à une fonctionnalité RP :

- Les rectangles dont les coins sont arrondis contiennent des tâches qui n'ont rien de particulier et qui sont appelées **actions non spéciales**.
- Les ellipses contiennent des **choix** qui sont également des actions non spéciales.
- Les pentagones signalent les tâches spéciales correspondant à l'action spéciale `call`. Quand dans sa liste de tâches, un *travailleur A* rencontre un pentagone cela signifie qu'il va passer commande des tâches d'un travailleur d'un certain type x. À ce moment là, l'un des *travailleurs* du type x, le *tra-*

vailleur B par exemple, commence à faire les tâches de sa liste. Le *travailleur A* et le *travailleur B* exécutent alors leur tâches en parallèle. Cependant le *travailleur A* garde en mémoire qu'il a activé le *travailleur B*.

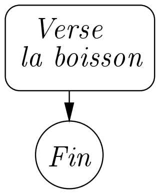


Les tâches d'un serveur sont les suivantes : il commence par lancer le distributeur de boissons. Pendant que le distributeur remplit les verres, il va chercher les frites. Quand les boissons sont prêtes, il les place sur le plateau.
Puis, il va chercher les sandwichs ; s'il prend les derniers sandwichs, il demande à un cuisinier d'en faire de nouveau. Enfin, il va chercher les gâteaux ; s'il prend les derniers, il demande à un pâtissier d'en faire de nouveaux. Les tâches du serveur sont alors terminées.

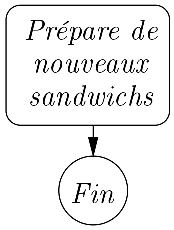
FIG. 3.2. – Le serveur.

– Les triangles signalent les tâches spéciales correspondant à l'action spéciale `wait`. Lorsqu'un *travailleur* rencontre un triangle, il regarde si les

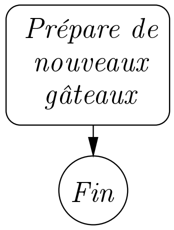
travailleurs qu'il a activés ont fini leurs tâches. Si c'est le cas, il passe à la tâche suivante, sinon il attend qu'ils aient terminé.
– Les cercles indiquent la fin d'une série de tâche, ils correspondent à l'action spéciale `end`.



Le distributeur de boissons remplit les verres.



Le cuisinier confectionne des sandwichs.



Le pâtissier fait des gâteaux.

FIG. 3.3. – Le distributeur de boissons, le cuisinier et le pâtissier.

Le fonctionnement de ce fast-food est le suivant : l'un des caissiers enregistre un commande, active l'un des serveurs. Ces deux *travailleurs* effectuent certaines tâches en même temps. Puis le caissier vérifie que le serveur a fini avant de passer à la commande suivante.

Pour effectuer ses tâches, le serveur a activé le distributeur de boissons, s'est occupé des frites, puis a attendu que le distributeur ait rempli les verres avant de les disposer sur le plateau. Il a ensuite activé un cuisinier s'il prenait les derniers sandwichs et un pâtissier s'il prenait les derniers gâteaux. À ce moment là, son travail est fini indépendamment de ce que font les *travailleurs* qu'il a activés car il n'y a aucune tâche marquée d'un triangle.

Nous remarquons dans cet exemple qu'un même *travailleur* peut activer plusieurs *travailleurs* et que ceux-ci sont totalement autonomes (les uns par rapport aux autres).

Par contre, le *travailleur* qui active et le *travailleur* qui est activé n'ont pas les mêmes libertés. Le *travailleur* A qui active un *travailleur* B dépend de B. En effet, si A rencontre une tâche entourée d'un triangle, il doit attendre que B ait fini. B, pour sa part, travaille totalement indépendamment de A et n'a aucun moyen de savoir quand A a fini. Il y a donc une **hiérarchie** entre les travailleurs.

Enfin, nous constatons que l'utilisation des triangles après un pentagone n'est pas obligatoire. En effet, les serveurs activent les cuisiniers et les pâtissiers, mais ils n'attendent pas que les uns ou les autres aient fini pour terminer leur tâche (il n'y a aucun triangle après les deux derniers pentagones).

Cet exemple nous a permis d'introduire la façon dont la programmation RP modélise le parallélisme à l'aide d'**activations** de tâches et d'attentes éventuelles de leur terminaison.

3.1.2. Les actions spéciales de la programmation RP

Nous revenons dans ce paragraphe sur les spécificités de la programmation RP, en particulier sur les actions spéciales.

Un système RP est formé d'un ensemble de sous-systèmes parallèles. Chaque sous-système est lui-même composé d'un ensemble de tâches qui sont de trois types : les actions non spéciales, les choix et les actions spéciales `call`, `wait` et `end`.

Les tâches d'un sous-système sont graphiquement reliées par des traits pleins qui indiquent dans quel ordre elles sont exécutées.

Les sous-systèmes peuvent s'appeler entre eux. Pour cela, ils utilisent l'instruction spéciale `call`. Graphiquement, l'action `call` est encadrée par un pentagone, un trait plein la lie à l'action suivante du sous-système appelant et un trait pointillé la lie à la première action du sous-système appelé.

Exemple 3.1
Le système RP de la figure 3.4 modélise l'exemple du fast-food. Ce système est obtenu :

- En recollant les figures 3.1, 3.2 et 3.3 ;
- En remplaçant toutes les actions non spéciales (celles entourées par un rectangle aux bords arrondis ou par une ellipse) par les noms d'actions abstraits $a, b, c, \ldots$ d'un alphabet non-interprété A ;
- En inscrivant l'action spéciale `call` à l'intérieur des pentagones, l'action spéciale `wait` à l'intérieur des triangles et l'action `end` à l'intérieur des cercles ;
- En liant par des pointillés les pentagones `call` aux sous-systèmes qu'elles activent.

Le fonctionnement d'un système RP

Un processus exécute la liste des tâches inscrites dans le sous-système principal. Dans l'exemple de la figure 3.4, le sous-système principal qui correspond au caissier comprend une boucle de cinq tâches a, `call`, b, c, `wait`. Nous appelons A ce processus.

Lorsqu'un processus rencontre une instruction `call`, un nouveau processus est activé et exécute la liste des tâches du sous-système appelé. Dans l'exemple de la figure 3.4, A active un nouveau processus, appelé B, pour exécuter les actions du serveur. De façon traditionnelle, lorsqu'un sous-système s'appelle lui-même par un `call`, il est qualifié de **récursif**.

À chaque fois qu'il y a un appel par l'intermédiaire d'une instruction `call`, un nouveau processus est activé. Il peut donc y avoir au même instant plusieurs processus qui exécutent les tâches d'un même sous-système si celui-ci a été appelé plusieurs fois.

Un processus est désactivé lorsqu'il rencontre l'action spéciale `end`.

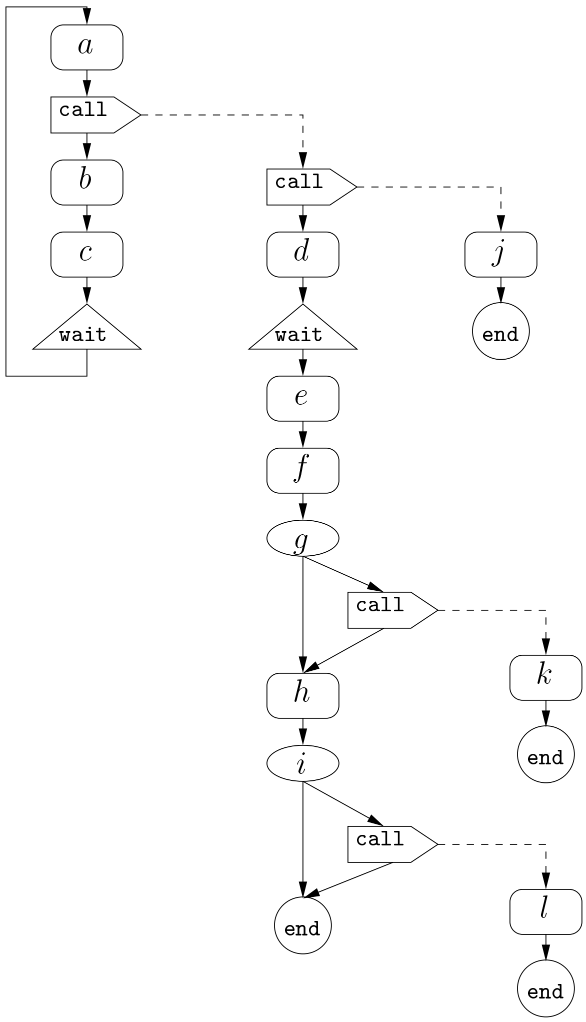


FIG. 3.4. – Le système RP abstrait.

Lorsqu'un processus rencontre une action `wait`, il doit attendre que tous les processus qu'il a activés aient fini avant d'exécuter l'action suivante. Ainsi dans notre exemple, A doit attendre que B ait fini avant de refaire une action a.

La hiérarchie des processus

Il existe une hiérarchie entre les processus activés. Un processus qui appelle un autre processus en est le **père**. De manière duale, le processus appelé est son **fils**. Ainsi A est le père de B et B le fils de A. Cette hiérarchie est induite par la hiérarchie statique donnée par le graphe mais elle en est différente : dans l'exemple du fast-food, un serveur n'a de fils que s'il appelle un cuisinier parce qu'il prend les derniers sandwichs ou un pâtissier parce qu'il prend les derniers gâteaux.

Un père dépend de ses fils par l'intermédiaire de l'instruction `wait`. Cependant, cette instruction n'est pas obligatoirement présente : le processus père peut choisir de terminer tout en laissant ses fils en marche : dans l'exemple du fast-food, le serveur s'arrête sans s'occuper de ce que fait le cuisinier.

Les différents processus fils d'un même processus père s'exécutent librement : dans l'exemple de la figure 3.4, s'il manque des sandwichs et des gâteaux, B va appeler un processus K pour exécuter les actions du cuisinier et un processus L pour celles du pâtissier. Les actions k et l s'exécutent en parallèle, indépendamment l'une de l'autre. Cependant, K et L sont tous deux les fils de B qui est lui-même le fils de A. Par conséquent, pour que A puisse effectuer une nouvelle action a, comme elle est précédée par l'action spéciale `wait`, il faut que K et L aient achevé leurs actions k et l ainsi que les actions `end`.

3.2. Les schémas de programmes RP

La figure 3.5 présente le schéma de programme récursif qui permet d'étudier la structure de contrôle du système de la figure 3.4. Nous remarquons que ce schéma est associé de façon évidente au système RP. Les seules différences sont que les actions apparaissent maintenant sur les arcs et les boîtes contiennent des noms d'états. Ce graphisme permet d'étudier plus facilement le système considéré. Un schéma de programme récursif-parallèle (RPPS) est donc un graphe fini enraciné.

Plus formellement, soit $A = \{a, b, \ldots\}$ un ensemble fini de noms d'actions qui ne contient aucune des actions spéciales `call`, `wait` et `end`. Nous notons $\tilde{A} = \{\alpha, \beta, \ldots\}$ l'ensemble formé par l'union de A et des actions spéciales.

Définition 3.2 *Un **schéma de programme récursif-parallèle (RPPS)** est un graphe fini enraciné $G = (Q, q_{init}, \Lambda)$ où :*
- $Q = \{q, \ldots\}$ *est un ensemble fini de* nœuds*,*
- $q_{init} \in Q$ *est le* nœud initial*,*
- Λ *est une fonction de transitions, elle associe à chaque nœud q un n-uplet dans* $(A \times Q) \cup (A \times Q \times Q) \cup (\{\texttt{call}\} \times Q \times Q) \cup (\{\texttt{wait}\} \times Q) \cup \{\texttt{end}\}$.

La fonction Λ est formellement lourde mais elle est facile à comprendre graphiquement : chaque nœud est relié par une action en général à un nœud (c'est le cas pour les actions de A et `wait`), parfois à une paire de nœuds (`call`) et parfois à aucun nœud (`end`). Dans l'exemple de la figure 3.5, $\Lambda(q_0) = (a, q_1)$, $\Lambda(q_1) = (\texttt{call}, q_2, q_7)$,...., $\Lambda(q_{23}) = \texttt{end}$.

3.3. Sémantique opérationnelle

La sémantique opérationnelle d'un schéma G est donnée via un système de transitions étiquetées $\mathcal{M}_G$ présenté par O. Kouchnarenko et Ph. Schnoebe-

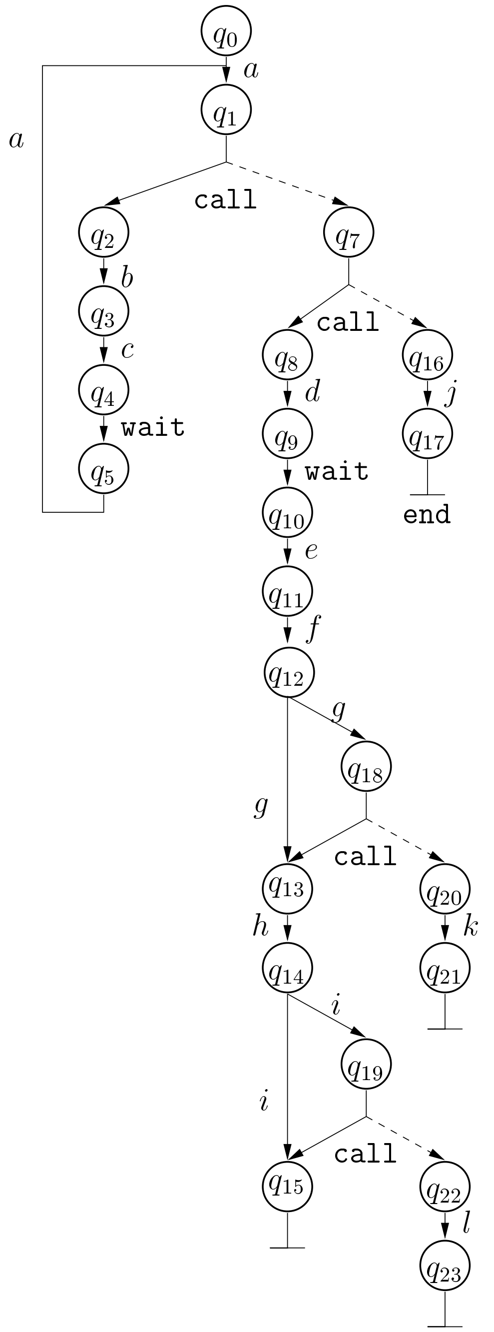


FIG. 3.5. – RPPS associé au système de la figure 3.4.

len [KS97b, KS97a] en suivant l'approche classique de G. D. Plotkin [Plo81]. Elle utilise le concept d'état emboîté.

3.3.1. Les états emboîtés

Définition 3.3 *L'ensemble des* ***états emboîtés*** $M(G)$ *d'un schéma* G *est le plus petit ensemble tel que pour tout n-uplet* $(q_1, \ldots, q_n)$ *de nœuds de* G *et tout n-uplet* $(s_1, \ldots, s_n)$ *d'états emboîtés de* $M(G)$ *, le multi-ensemble* $s = \{(q_1, s_1), \ldots, (q_n, s_n)\}$ *appartient à* $M(G)$.

En particulier, $\emptyset$ appartient à $M(G)$. L'état emboîté $s = \{(q_1, s_1), \ldots, (q_n, s_n)\}$ correspond à l'activation de n processus indépendants. Chacun de ces processus (q_i, s_i) a une **composante mère** dans l'état q_i et un ensemble s_i de **sous-composantes filles** parallèles, desquelles dépend l'exécutabilité d'une instruction `wait`.

Définition 3.4 *La* ***taille*** *de l'état emboîté* s*, notée* $|s|$*, est définie inductivement par :*

- $|\emptyset| = 0$,
- $|(q_1, s_1), \ldots, (q_n, s_n)| = \sum_{i=1,\ldots,n}(1 + |s_i|)$.

Exemple 3.5 Un état emboîté associé au schéma de la figure 3.5 est :

$$\{(q_2, \{(q_{15}, \{(q_{20}, \emptyset), (q_{22}, \emptyset)\})\})\}.$$

Sa taille est 4, sa composante mère est dans l'état de contrôle q_2 et sa composante fille est un état emboîté qui a deux composantes filles $(q_{20}, \emptyset)$ et $(q_{22}, \emptyset)$ indépendantes.

Nous utilisons les notations usuelles « $s + s'$ », « $s \subseteq s'$ », … pour noter la somme, l'inclusion, … de multi-ensembles et donc d'états emboîtés.

Dans toute la suite, nous écrirons (q, s) au lieu du singleton $\{(q, s)\}$.

Nous représentons graphiquement les états emboîtés par des forêts d'arbres généraux de la façon suivante :

- $\emptyset$ est représenté par l'arbre vide ;
- $s = \{(q_1, s_1), \ldots, (q_n, s_n)\}$ est représenté par la forêt formée de l'arbre représentant (q_1, s_1), de l'arbre représentant (q_2, s_2), etc. ;
- L'arbre représentant $(q, \{(q_1, s_1), \ldots, (q_n, s_n)\})$ a pour racine q et n sous-arbres fils dont le premier représente (q_1, s_1), le second représente (q_2, s_2), etc.

Exemple 3.6 La figure 3.6 est une représentation graphique de l'état emboîté $\{(q_2, \{(q_{15}, \{(q_{20}, \emptyset), (q_{22}, \emptyset)\})\})\}$ présenté dans l'exemple 3.5.

3.3.2. Transitions entre états emboîtés

Une transition est un triplet (s, a, s'), noté aussi $s \xrightarrow{a} s'$ où s et s' sont deux états emboîtés et a appartient à $\tilde{A}$. Formellement, nous avons :

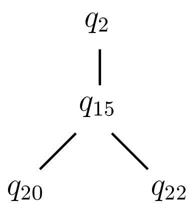


FIG. 3.6. – Un état emboîté.

Définition 3.7 *La relation $\rightarrow \subseteq M(G) \times \tilde{A} \times M(G)$ est définie inductivement comme le plus petit ensemble de transitions (s, a, s') vérifiant :*

$$\begin{aligned}
&\textbf{\textit{action}}: \textit{si } \Lambda(q) = (a, q') \textit{ ou } \Lambda(q) = (a, q', q'') \textit{ ou } \Lambda(q) = (a, q'', q'), \\
&\qquad \textit{alors } (q, s) \xrightarrow{a} (q', s) \textit{ pour tout } s \textit{ et tout } a, && \text{(Ga)} \\
&\textbf{\textit{end}}: \textit{si } \Lambda(q) = \texttt{end}, \\
&\qquad \textit{alors } (q, s) \xrightarrow{\texttt{end}} s \textit{ pour tout } s, && \text{(Ge)} \\
&\textbf{\textit{call}}: \textit{si } \Lambda(q) = (\texttt{call}, q', q''), \\
&\qquad \textit{alors } (q, s) \xrightarrow{\texttt{call}} (q', s + (q'', \emptyset)) \textit{ pour tout } s, && \text{(Gc)} \\
&\textbf{\textit{wait}}: \textit{si } \Lambda(q) = (\texttt{wait}, q'), \\
&\qquad \textit{alors } (q, \emptyset) \xrightarrow{\texttt{wait}} (q', \emptyset), && \text{(Gw)} \\
&\textbf{\textit{paral1}}: \textit{si } s \xrightarrow{\alpha} s', \\
&\qquad \textit{alors } s + s'' \xrightarrow{\alpha} s' + s'' \textit{ pour tout } s'', && \text{(Gp1)} \\
&\textbf{\textit{paral2}}: \textit{si } s \xrightarrow{\alpha} s', \\
&\qquad \textit{alors } (q, s) \xrightarrow{\alpha} (q, s') \textit{ pour tout } q \in Q. && \text{(Gp2)}
\end{aligned}$$

Les règles **paral1** et **paral2** pour le parallélisme signifient qu'une action $s \xrightarrow{\alpha} s'$ peut encore avoir lieu si s a des frères (i.e. dans $s + s''$) ou un père (i.e. (q, s)).

La règle **wait** montre que l'on peut faire une action `wait` dans l'état de contrôle q uniquement si tous les fils invoqués ont fini (et donc ne sont plus présents).

Les autres règles montrent comment sont créées les invocations de fils et comment elles sont gérées.

Pour conclure, la sémantique opérationnelle de G est donnée par le système de transitions étiquetées $\mathcal{M}_G = (M(G), \tilde{A}, \rightarrow, s_0)$ où l'état initial est $s_0 \stackrel{\text{def}}{=} (q_{init}, \emptyset)$.

Exemple 3.8 Une exécution d'un RPPS.

La séquence suivante est une exécution du système $\mathcal{M}_G$ associé au schéma de la figure 3.5 qui permet d'atteindre l'état proposé dans l'exemple 3.5.

$$
\begin{aligned}
&(q_0, \emptyset) \xrightarrow{a} (q_1, \emptyset) \xrightarrow{\texttt{call}} (q_2, (q_7, \emptyset)) \xrightarrow{\texttt{call}} (q_2, (q_8, (q_{16}, \emptyset))) \xrightarrow{d} (q_2, (q_9, (q_{16}, \emptyset)))\\
\xrightarrow{j}\ & (q_2, (q_9, (q_{17}, \emptyset))) \xrightarrow{\texttt{end}} (q_2, (q_9, \emptyset)) \xrightarrow{\texttt{wait}} (q_2, (q_1 0, \emptyset)) \xrightarrow{e} (q_2, (q_{11}, \emptyset))\\
\xrightarrow{f}\ & (q_2, (q_{12}, \emptyset)) \xrightarrow{g} (q_2, (q_{18}, \emptyset)) \xrightarrow{\texttt{call}} (q_2, (q_{13}, (q_{20}, \emptyset))) \xrightarrow{h} (q_2, (q_{14}, (q_{20}, \emptyset)))\\
\xrightarrow{i}\ & (q_2, (q_{19}, (q_{20}, \emptyset))) \xrightarrow{\texttt{call}} (q_2, (q_{15}, \{(q_{20}, \emptyset), (q_{22}, \emptyset)\})).
\end{aligned}
$$

Une représentation graphique de cette exécution à l'aide d'arbres est proposée dans la figure 3.7.

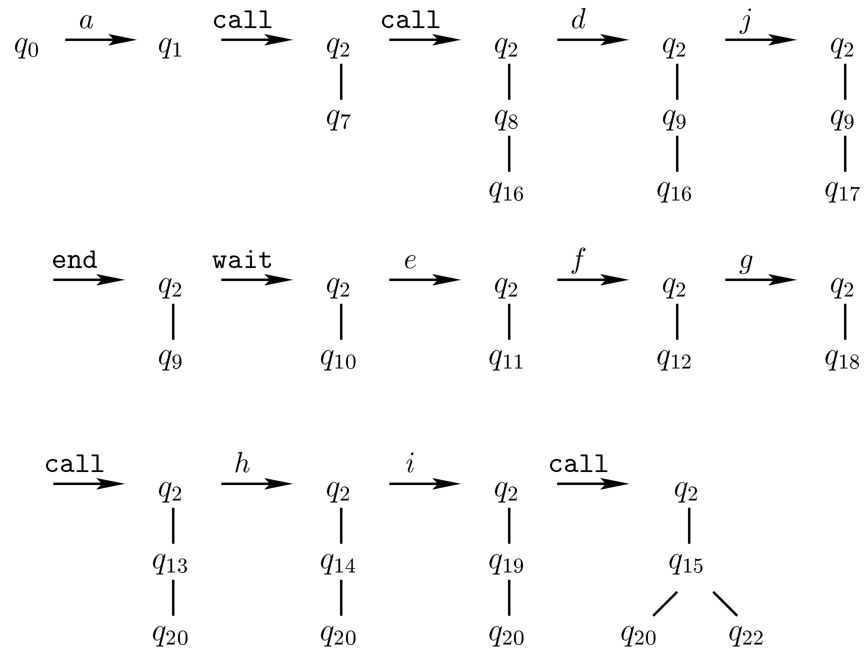


FIG. 3.7. – Une exécution.

Pour ces schémas RPPS, nous examinerons la relation d'accessibilité entre deux états emboîtés. Cette étude passe par la définition d'une nouvelle algèbre de processus que nous appelons RPA.

4. Une algèbre de processus où l'accessibilité est reconnaissable : RPA

Sommaire

Comme nous l'avons expliqué dans l'introduction de cette partie, nous introduisons une algèbre de processus appelée RPA pour étudier le problème de l'accessibilité des schémas RPPS. Le codage d'un schéma RPPS par une algèbre RPA n'est pas immédiat, il fait l'objet du prochain chapitre.

Pour l'instant, nous définissons l'algèbre RPA indépendamment de tout schéma RPPS. Nous présentons ensuite les langages d'arbres, les automates d'arbres et la notion de reconnaissabilité par un automate d'arbres. Nous proposons alors un automate d'arbres qui reconnaît l'accessibilité pour une algèbre RPA donnée (c'est le premier critère que nous exigeons pour l'algèbre qui code les schémas RPPS). Nous terminons avec les applications de ce résultat.

4.1. L'algèbre de processus RPA

4.1.1. Les termes RPA

Soit Σ un ensemble fini de noms d'**actions** dont nous notons les éléments α, β, etc.

Soit $\mathcal{X}$ un ensemble fini de **constantes de processus** dont nous notons les éléments X, Y, etc. $\mathcal{X}$ est partitionné en deux ensembles disjoints $\mathcal{X}^!$ et $\mathcal{X}^?$. L'ensemble $\mathcal{X}^!$ regroupe des processus qui peuvent faire une action sans tenir compte de leur contexte et l'ensemble $\mathcal{X}^?$ ceux qui peuvent être empêchés d'agir par leur entourage.

Soit *Nil* un symbole particulier qui n'appartient pas à $\mathcal{X}$. Il représente le processus fini, celui qui ne peut plus faire d'action. Nous notons *Atomes* l'union de $\mathcal{X}$ et de $\{Nil\}$ et C, C', etc. ses éléments.

Définition 4.1 *L'ensemble des* ***termes RPA****, noté $T_{\mathcal{X}}$ ou T, dont les éléments sont notés t, u, v, etc., est donné par la syntaxe suivante :*

$$t, u ::= C \mid t \blacktriangleright u.$$

Le terme $t \blacktriangleright u$ peut se lire comme le processus t entouré par u. Une interprétation plus précise du symbole $\blacktriangleright$ est donnée dans le chapitre 5. Pour l'instant, $\blacktriangleright$ permet surtout de donner un squelette aux termes RPA.

Soit t un terme, nous désignons par $Constante(t)$ l'ensemble de toutes les constantes de $\mathcal{X}$ qui ont une occurrence dans t.

La **taille** de t, notée $|t|$, est égale au nombre de symboles de t. Elle est définie inductivement par :

$$\begin{aligned} |C| &\stackrel{\text{def}}{=} 1, \\ |t \blacktriangleright u| &\stackrel{\text{def}}{=} 1 + |t| + |u|. \end{aligned}$$

Exemple 4.2 Le terme RPA $t = (X \blacktriangleright Nil) \blacktriangleright (Nil \blacktriangleright (Y \blacktriangleright X))$ vérifie $Constante(t) = \{X, Y\}$ et $|t| = 9$.

Définition 4.3 *Une* ***déclaration (gardée) RPA*** *est un ensemble fini $\Delta \subseteq \mathcal{X} \times \Sigma \times Atomes \times T_{\mathcal{X}}$ de règles de réécriture, notée $\{X_i \xrightarrow{\alpha_i}_{\Delta} C_i, t_i \mid i = 1, \ldots, n\}$.*

Les X_i ne sont pas nécessairement distincts. Pour des raisons techniques, nous imposons que tous les $X \in \mathcal{X}$ apparaissent dans le membre gauche d'au moins une règle (ce n'est pas une restriction et on pourrait remplacer toutes les occurrences des X qui n'apparaissent pas dans le membre gauche d'une règle par *Nil*).

Le sens informel d'une règle $X_i \xrightarrow{\alpha_i}_{\Delta} C_i, t_i$ est que le processus X_i peut faire une action α_i et ensuite agir comme le processus C_i entouré de t_i.

Exemple 4.4 Une déclaration RPA.

Voici un exemple de déclaration RPA gardée :

$$\Delta \stackrel{\text{def}}{=} \left\{ \begin{array}{l} X \xrightarrow{\alpha} Y, Nil \ ; \\ Y \xrightarrow{\beta} Nil, X \ . \end{array} \right\}$$

Dans cet exemple, il y a deux processus X et Y, le premier peut faire une action α et agir comme Y sans avoir rien ajouté au contexte et le deuxième peut faire une action β et avoir fini tout en ajoutant un processus X au contexte.

4.1.2. La sémantique de RPA

Nous allons maintenant donner la sémantique de RPA. Nous notons Act l'ensemble $\Sigma \times \{!, ?\}$. Afin de préserver la lisibilité, nous écrirons $\alpha^!$ et $\alpha^?$ au lieu de $(\alpha, !)$ et $(\alpha, ?)$.

Définition 4.5 *Une déclaration Δ définit un système de transitions étiquetées $\rightarrow \subseteq T_{\mathcal{X}} \times Act \times T_{\mathcal{X}}$, donné par les règles de SOS suivantes :*

$$R1 \quad \frac{}{X \xrightarrow{\alpha^!} C \blacktriangleright t} \ \textit{si } (X \xrightarrow{\alpha}_{\Delta} C, t) \in \Delta \textit{ et } X \in \mathcal{X}^!$$

$$R2 \quad \frac{}{X \xrightarrow{\alpha^?} C \blacktriangleright t} \ \textit{si } (X \xrightarrow{\alpha}_{\Delta} C, t) \in \Delta \textit{ et } X \in \mathcal{X}^?$$

$$R3 \quad \frac{t \xrightarrow{\alpha^!} t'}{u \blacktriangleright t \xrightarrow{\alpha^!} u \blacktriangleright t'} \qquad R4 \quad \frac{t \xrightarrow{\alpha^?} t'}{u \blacktriangleright t \xrightarrow{\alpha^!} u \blacktriangleright t'}$$

$$R5 \quad \frac{t \xrightarrow{\alpha^!} t'}{t \blacktriangleright u \xrightarrow{\alpha^!} t' \blacktriangleright u}$$

$$R6 \quad \frac{t \xrightarrow{\alpha^?} t'}{t \blacktriangleright u \xrightarrow{\alpha^?} t' \blacktriangleright u} \ \textit{si Constante}(u) = \emptyset$$

L'étiquette ! ou ? des transitions est introduite par les règles R1 et R2 ; elle dépend directement de l'appartenance de X à $\mathcal{X}^!$ ou $\mathcal{X}^?$. Cette étiquette n'est ? que dans le cas où l'action est faite par un processus qui appartient à $\mathcal{X}^?$ et qui n'est le contexte d'aucun sous-terme. Dans ce cas, l'action ne peut avoir lieu que si son contexte est vide, c'est-à-dire s'il ne contient que des processus *Nil* (règle R6).

Par la suite, nous utiliserons les notations et les abréviations suivantes : $u \xrightarrow{!} v$ (resp. $u \xrightarrow{?} v$) quand $u \xrightarrow{\alpha^!} v$ (resp. $u \xrightarrow{\alpha^?} v$) où $\alpha \in Act$ et $u \rightarrow v$ quand $u \xrightarrow{!} v$ ou

$u \xrightarrow{?} v$. Si $n \in \mathbb{N}$, $\xrightarrow{n}$ et $\xrightarrow{n,!}$ désignent respectivement les relations itérées $(\rightarrow)^n$ et $\left(\xrightarrow{!}\right)^n$. De plus, nous noterons $\rightarrow^*$ la clôture transitive de $\rightarrow$. Enfin, $u \rightarrow$ et $u \not\rightarrow$ signifient respectivement qu'il existe (resp. il n'existe pas) un v tel que $u \rightarrow v$.

Exemple 4.6 Une exécution RPA.

Voici un exemple d'exécution pour la déclaration de l'exemple 4.4 en supposant que $X^? = \{X, Y\}$:

$$X \xrightarrow{\alpha?} Y \blacktriangleright Nil \xrightarrow{\beta?} (Nil \blacktriangleright X) \blacktriangleright Nil \xrightarrow{\alpha!} (Nil \blacktriangleright (Y \blacktriangleright Nil)) \blacktriangleright Nil)$$

Nous remarquons que la deuxième transition peut avoir lieu car le contexte de Y est vide. La figure 4.1 montre les trois étapes de la dérivation de la troisième transition.

$$R5\;\dfrac{R4\;\dfrac{R2\;\dfrac{}{X \xrightarrow{\alpha?} Y \blacktriangleright Nil}}{Nil \blacktriangleright X \xrightarrow{\alpha!} Nil \blacktriangleright (Y \blacktriangleright Nil)}}{(Nil \blacktriangleright X) \blacktriangleright Nil \xrightarrow{\alpha!} (Nil \blacktriangleright (Y \blacktriangleright Nil)) \blacktriangleright Nil}$$

FIG. 4.1. – La transition $(Nil \blacktriangleright X) \blacktriangleright Nil \xrightarrow{\alpha!} (Nil \blacktriangleright (Y \blacktriangleright Nil)) \blacktriangleright Nil)$.

4.1.3. Quelques propriétés des transitions RPA

Nous allons maintenant présenter quelques lemmes sur les transitions entre deux termes RPA qui servent dans les chapitres suivants.

Lemme 4.7 *Si $u \blacktriangleright v \rightarrow w$, alors w est de la forme $u' \blacktriangleright v'$ et*
- *Soit $u \rightarrow u'$ et $v = v'$,*
- *Soit $v \rightarrow v'$ et $u = u'$.*

Démonstration. La dernière étape de la dérivation de $u \blacktriangleright v \rightarrow w$ utilise l'une des règles R3, R4, R5 ou R6. Si cette règle est R3 ou R4, alors $u = u'$ et $v \rightarrow v'$ et si cette règle est R5 ou R6, nous avons $u \rightarrow u'$ et $v = v'$. ⌟

Ce lemme 4.7 montre qu'un opérateur $\blacktriangleright$ ne peut pas disparaître lors d'une transition. Les trois lemmes suivants en découlent :

Lemme 4.8 *Si $u \blacktriangleright v \rightarrow^* w$, alors w est de la forme $u' \blacktriangleright v'$ avec $u \rightarrow^* u'$ et $v \rightarrow^* v'$.*

Démonstration. Nous montrons par induction sur la longueur de la dérivation $u \blacktriangleright v \xrightarrow{n} w$, que w est de la forme $u' \blacktriangleright v'$ avec $u \rightarrow^* u'$ et $v \rightarrow^* v'$.

- Si $n = 0$, alors la propriété est vérifiée.
- Sinon, $u \blacktriangleright v \xrightarrow{n-1} w' \rightarrow w$. Par hypothèse d'induction, w' est de la forme $u' \blacktriangleright v'$ avec $u \rightarrow^* u'$ et $v \rightarrow^* v'$. D'après le lemme 4.7, $w = u'' \blacktriangleright v''$ avec soit $u' \rightarrow u''$ et $v' = v''$, soit $v' \rightarrow v''$ et $u' = u''$. Dans tous les cas, w' est de la forme $u'' \blacktriangleright v''$ avec $u \rightarrow^* u''$ et $v \rightarrow^* v''$.

$\lrcorner$

Lemme 4.9 *Si $u \rightarrow v$, alors $|v| > |u|$.*

Démonstration. Nous montrons par induction structurelle sur la dérivation $u \rightarrow v$.

- Les cas de bases sont l'application des règles R1 et R2. Nous avons alors $u = X$ et $v = C \blacktriangleright t$ et la propriété est vérifiée.
- Sinon, la dernière étape de dérivation est l'application de l'une des règles R3, R4, R5 ou R6. Tous ces cas sont similaires, nous allons traiter, par exemple, le cas où cette dernière est la règle R3. Nous avons alors $u = w \blacktriangleright t$ et $v = w \blacktriangleright t'$ avec $t \rightarrow t'$ et cette dernière dérivation est de longueur n. Par hypothèse de récurrence, $|t'| > |t|$. Il s'ensuit que $|v| > |u|$.

$\lrcorner$

Lemme 4.10 *Soient X et Y deux constantes de processus, $X \rightarrow^* Y$ si et seulement si $X = Y$.*

Démonstration. Si $X = Y$, alors $X \rightarrow^* Y$. Si $X \xrightarrow{n} Y$, le lemme 4.9 entraîne que $n = 0$ et donc $X = Y$. $\lrcorner$

Le lemme qui suit affirme qu'un terme composé uniquement de processus *Nil* ne peut faire aucune action et surtout que, réciproquement, un terme qui contient une constante de processus peut faire une action.

Lemme 4.11 *Soit u un terme RPA, $u \rightarrow$ si et seulement $Constante(u) \neq \emptyset$.*

Démonstration.

($\Rightarrow$) Nous supposons que $u \rightarrow$ et nous allons montrer par induction structurelle sur la dérivation $u \rightarrow$ que $Constante(u) \neq \emptyset$.

- Les cas de base sont l'application des règles R1 et R2. Nous avons alors $u = X$ et la propriété est vérifiée.
- Sinon la dernière étape est l'application de l'une des règles R3, R4, R5 ou R6. Tous ces cas sont similaires, nous allons traiter, par exemple, le cas où cette dernière est la règle R5. Nous avons alors $u = w \blacktriangleright t$ et $v = w' \blacktriangleright t$ avec $w \rightarrow w'$ et cette dernière dérivation est de longueur n. Par hypothèse de récurrence, $Constante(w) \neq \emptyset$. Il s'ensuit que $Constante(u) \neq \emptyset$. Une preuve similaire permet de conclure pour les règles R3, R4 et R6.

($\Leftarrow$) Nous supposons que $Constante(u) \neq \emptyset$ et nous allons montrer par induction structurelle sur u que $u \rightarrow$.

- Le cas $u = Nil$ ne se présente pas car $Constante(u) \neq \emptyset$.
- Si $u = X \in \mathcal{X}$, alors nous avons supposé qu'il y a au moins une règle de Δ dont X est le membre gauche. Soit $X \xrightarrow{\alpha} C, v \in \Delta$. Les règles R1 et R2 entraînent que $u \rightarrow C \blacktriangleright v$.
- Si u est de la forme $u_1 \blacktriangleright u_2$, alors $Constante(u_1) \neq \emptyset$ ou $Constante(u_2) \neq \emptyset$.
  - Si $Constante(u_2) \neq \emptyset$, alors $u_2 \rightarrow$ par hypothèse d'induction. Il s'ensuit que $u \rightarrow$ par application de la règle R3 ou de la règle R4.
  - Si $Constante(u_2) = \emptyset$ alors $Constante(u_1) \neq \emptyset$ et $u_1 \rightarrow$ par hypothèse d'induction. Il s'ensuit que $u \rightarrow$ par application de la règle R5 ou de la règle R6. Cette dernière est applicable car $Constante(u_2) = \emptyset$. ⌟

4.2. Les langages et les automates d'arbres

Lors des démonstrations, nous allons utiliser deux familles de propriétés des termes RPA. La première est obtenue en considérant les termes RPA comme des arbres binaires, c'est cet aspect que nous présentons dans cette section. La seconde, développée dans le chapitre 5, considère les termes RPA comme des peignes gauches. Dans cette section, nous présentons les notions de langage d'arbres, d'automate d'arbres, d'automate d'arbres alternant et de relation reconnaissable sur les arbres. Nous utilisons pour cela les définitions de H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison et M. Tommasi [CDG+99] et M. Vardi [Var97].

4.2.1. Les langages d'arbres

Définition 4.12 *Soit $\mathcal{F}$ un alphabet muni d'une fonction d'arité. Nous notons $T_\mathcal{F}$ l'ensemble des termes finis construits sur $\mathcal{F}$ et nous appelons ces termes des **arbres**.*

Nous pouvons partitionner $\mathcal{F}$ en fonctions des arités : $\mathcal{F} = \mathcal{F}_0 \cup \mathcal{F}_1 \cup \ldots \cup \mathcal{F}_m$.

Exemple 4.13 Soient $\mathcal{F}_0 = \{a, b\}$, $\mathcal{F}_1 = \{g, h\}$ et $\mathcal{F}_2 = \{f\}$, $T_\mathcal{F}$ contient des arbres tels que a, $f(a, b)$ et $f(g(f(h(b), a)), b)$, ce dernier est illustré par la figure 4.2.

Définition 4.14 *Un **langage d'arbres** est un sous-ensemble de $T_\mathcal{F}$.*

4.2.2. Les automates d'arbres

Définition 4.15 *Un **automate descendant d'arbres** est un quadruplet $\mathcal{A} = (\mathcal{F}, \mathcal{Q}, F, \delta)$ où :*

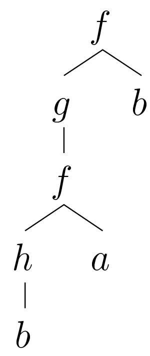


FIG. 4.2. – L'arbre $f(g(f(h(b), a)), b)$.

- *$\mathcal{F}$ est un alphabet fini muni d'une fonction d'arité,*
- *$\mathcal{Q} = \{p, p' \ldots\}$ est un ensemble fini d'**états de contrôle**,*
- *$F \subset \mathcal{Q}$ est un ensemble d'**états acceptants**,*
- *$\delta \subseteq \cup_{n\in\mathbb{N}}(\mathcal{Q} \times \mathcal{F}_n \times \mathcal{Q}^n)$ est ensemble fini de **règles de transitions**.*

Un automate descendant d'arbres peut être vu comme une grammaire qui engendre des arbres construits sur l'alphabet $\mathcal{F} \cup \mathcal{Q}$ où les éléments de $\mathcal{Q}$ ont une arité nulle.

L'ensemble des transitions $\mapsto_{\mathcal{A}} \subset T_{\mathcal{F}\cup\mathcal{Q}} \times T_{\mathcal{F}\cup\mathcal{Q}}$ de $\mathcal{A}$ est le plus petit ensemble de paires (T, T'), notées $T \mapsto_{\mathcal{A}} T'$, qui satisfont les règles suivantes :

- Si $f \in \mathcal{F}$ est d'arité n et si $(p, f, p_1, \ldots, p_n) \in \delta$, alors $p \mapsto_{\mathcal{A}} f(p_1, \ldots, p_n)$;
- Si $f \in \mathcal{F}$ est d'arité n, si $T_1, \ldots, T_n \in T_{\mathcal{F}\cup\mathcal{Q}}$ et si $T_i \mapsto_{\mathcal{A}} T'_i$, alors $f(T_1, \ldots, T_n) \mapsto_{\mathcal{A}} f(T_1, \ldots, T_{i-1}, T'_i, T_{i+1}, \ldots, T_n)$.

Par abus de notation, nous écrirons la règle $(p, f, p_1, \ldots, p_n)$ de δ, $p \mapsto_{\mathcal{A}} f(p_1, \ldots, p_n)$.

Définition 4.16 *Soient $p \in \mathcal{Q}$ et $T \in T_{\mathcal{F}\cup\mathcal{Q}}$.*

*Un **calcul** $p \overset{*}{\mapsto}_{\mathcal{A}} T$ de l'automate $\mathcal{A}$ est une suite $(T_i)_{0\leq i\leq n}$ d'arbres de $T_{\mathcal{F}\cup\mathcal{Q}}$ telle que $T_0 = p$, $T_n = T$ et pour tout $0 \leq i \leq n-1$, $T_i \mapsto_{\mathcal{A}} T_{i+1}$.*

*Un calcul est **réussi** si $p \in F$ et $T \in T_{\mathcal{F}}$.*

Exemple 4.17

Soit $\mathcal{F} = \{a, g, f\}$ avec $\mathcal{F}_0 = \{a\}$, $\mathcal{F}_1 = \{g\}$ et $\mathcal{F}_2 = \{f\}$ un alphabet ordonné.

Nous considérons l'automate $\mathcal{A}$ dont les états de contrôle sont p et p', dont l'état acceptant est p et dont les règles de transitions sont :

$$\begin{array}{lllllll}
p & \mapsto_{\mathcal{A}} & a, & & & & \\
p & \mapsto_{\mathcal{A}} & g(p'), & \quad & p' & \mapsto_{\mathcal{A}} & g(p), \\
p & \mapsto_{\mathcal{A}} & f(p, p), & \quad & p' & \mapsto_{\mathcal{A}} & f(p', p), \\
p & \mapsto_{\mathcal{A}} & f(p', p'), & \quad & p' & \mapsto_{\mathcal{A}} & f(p, p').
\end{array}$$

Un calcul réussi de cet arbre est :

$$p \mapsto_{\mathcal{A}} f(p,p) \mapsto_{\mathcal{A}} f(p,g(p')) \mapsto_{\mathcal{A}} f(p,g(g(p))) \mapsto_{\mathcal{A}} f(p,g(g(a)))$$
$$\mapsto_{\mathcal{A}} f(a,g(g(a))).$$

Définition 4.18 *Un arbre t de $T_{\mathcal{F}}$ est **reconnu** par un automate $\mathcal{A}$ s'il existe un état acceptant p de $\mathcal{A}$ et un calcul réussi $p \overset{*}{\mapsto}_{\mathcal{A}} t$.*

Définition 4.19
Pour tout $p \in \mathcal{Q}$, $L(p)$ désigne $\{t \mid p \overset{*}{\mapsto}_{\mathcal{A}} t\}$.
*$L(\mathcal{A}) \overset{\text{def}}{=} \bigcup_{p \in F} L(p)$ est le langage d'arbres **reconnu** par $\mathcal{A}$.*

L'automate de l'exemple 4.17 reconnaît les arbres qui ont un nombre pair d'occurrences de g.

4.2.3. Les automates d'arbres alternants

Si E est un ensemble donné, $\mathcal{B}^+(E)$ désigne l'ensemble de toutes les formules booléennes positives sur E (c'est-à-dire les formules booléennes construites à partir des éléments de E en utilisant les symboles $\wedge$ et $\vee$), ainsi que les formules *vrai* et *faux*.

Définition 4.20 *Un **automate d'arbres alternant** est un quadruplet $\mathcal{A} = (\mathcal{F}, \mathcal{Q}, F, \delta)$ où δ désigne (maintenant) une famille de fonctions de transitions de $\mathcal{Q} \times \mathcal{F}_n$ dans $\mathcal{B}^+(\{1,\dots,n\} \times \mathcal{Q})$ indexée par n.*

La relation de satisfaisabilité $\models_{\mathcal{A}} \subset \mathcal{Q} \times T_{\mathcal{F}}$ de $\mathcal{A}$ est définie inductivement par : $p \models_{\mathcal{A}} f(t_1,\dots,t_n)$ si la valeur de $\delta(p,f)$ où nous avons remplacé (p',i) par $\begin{cases} \textit{vrai}, \text{ si } p' \models_{\mathcal{A}} t_i \\ \textit{faux}, \text{ si } p' \not\models_{\mathcal{A}} t_i \end{cases}$ est *vrai*.

Définition 4.21 *Un arbre t de $T_{\mathcal{F}}$ est **reconnu** par un automate alternant $\mathcal{A}$ s'il existe un état acceptant p de $\mathcal{A}$ tel que $p \models_{\mathcal{A}} t$.*

Définition 4.22
Pour tout $p \in \mathcal{Q}$, $L(p)$ désigne $\{t \mid p \models_{\mathcal{A}} t\}$.
*$L(\mathcal{A}) \overset{\text{def}}{=} \bigcup_{p \in F} L(p)$ est le langage d'arbres **reconnu** par $\mathcal{A}$.*

Exemple 4.23
Nous obtenons un automate alternant reconnaissant le même langage que celui présenté dans l'exemple 4.17 en considérant la fonction de transitions δ suivante :

$$\begin{aligned}
&\delta(p,a) = \textit{vrai},\\
&\delta(p,g) = (1,p')\\
&\delta(p',g) = (1,p),\\
&\delta(p,f) = ((1,p) \wedge (2,p)) \vee ((1,p') \wedge (2,p')),\\
&\delta(p',f) = ((1,p') \wedge (2,p)) \vee ((1,p) \wedge (2,p')).
\end{aligned}$$

Il est bien connu qu'un automate d'arbres standard peut être vu comme un automate d'arbres alternant utilisant uniquement des disjonctions et que la classe des langages d'arbres reconnaissables par des automates d'arbres alternants est exactement la classe des langages reconnaissables par des automates d'arbres non alternants (une construction analogue à celle proposée dans [CKS81] pour les mots finis convient).

4.2.4. Une notion de reconnaissabilité pour les relations sur les arbres

Avant de montrer que la relation d'accessilité entre deux termes RPA est reconnaissable, nous avons besoin de définir ce qu'est une relation reconnaissable sur des arbres. Il existe plusieurs façons de définir une notion de reconnaissabilité pour les relations sur les arbres [CDG+99]. Celle qui nous intéresse utilise la notion d'entrelacement d'arbres et a permis à D. Lugiez et Ph. Schnoebelen de montrer [LS00] que l'accessilité entre deux termes PA est reconnaissable. Nous la présentons dans le cas d'une relation binaire.

À un couple d'arbres t, t' de $T_{\mathcal{F}}$ peut être associé un arbre unique produit, noté $[t, t']$ sur un alphabet produit $\mathcal{F}' \overset{\text{def}}{=} (\mathcal{F} \cup \{\perp\}) \times (\mathcal{F} \cup \{\perp\})$ où $\perp$ est un nouveau symbole d'arité 0. L'arité d'un symbole fg de $\mathcal{F}'$ est le maximum entre l'arité de f et celle de g. Plus formellement, $[t, t']$ est défini inductivement par :

Si $n \geq m$,

$$[f(u_1, \ldots, u_n), g(v_1, \ldots, v_m)] \overset{\text{def}}{=} fg([u_1, v_1], \ldots, [u_m, v_m], [u_{m+1}, \perp], \ldots, [u_n, \perp],$$

et si $n \leq m$,

$$[f(u_1, \ldots, u_n), g(v_1, \ldots, v_m)] \overset{\text{def}}{=} fg([u_1, v_1], \ldots, [u_n, v_n], [\perp, v_{n+1}], \ldots, [\perp, v_m].$$

Exemple 4.24 $[f(a, g(b)), f(f(a, a), b)] = ff(af(\perp a, \perp a), gb(b\perp))$ comme illustré par la figure 4.3.

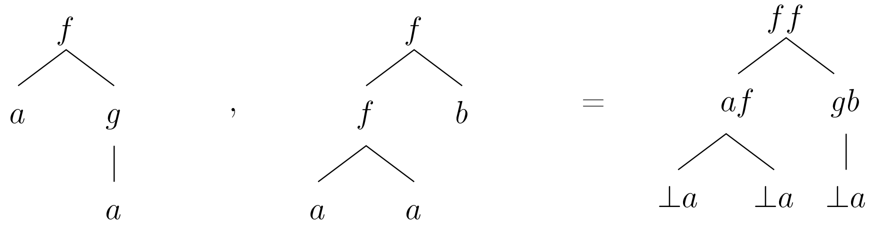


FIG. 4.3. – Un produit de deux arbres.

Cette définition s'étend aux n-uplets d'arbres de façon évidente.

Définition 4.25 *Une relation n-aire $R \subseteq T_{\mathcal{F}} \times \cdots \times T_{\mathcal{F}}$ est* ***reconnaissable*** *si et seulement si l'ensemble de tous les arbres $[t_1, \ldots, t_n]$ où $(t_1, \ldots, t_n) \in R$ est un langage d'arbres reconnaissable sur $(\mathcal{F} \cup \{\perp\})^n \setminus (\perp, \ldots, \perp)$.*

Exemple 4.26 La relation $Id = \{(t,t) \mid s \in T_{\mathcal{F}}\}$ est une relation reconnaissable et un automate qui accepte Id n'a qu'un seul état q qui est final et ses règles sont $ff(q, \ldots, q) \mapsto_{\mathcal{A}} q$ pour tout f.

Lemme 4.27 *L'intersection, l'union et le complément de relations n-aires reconnaissables sont aussi reconnaissables.*

Définition 4.28
Pour $1 \leq i \leq n$, la $i^{\textbf{ième}}$ ***projection*** *de la relation n-aire $\mathcal{R}$ est la relation $n-1$-aire obtenue en supprimant la $i^{\text{ième}}$ composante de chaque n-uple appartenant à $\mathcal{R}$.*
Pour $0 \leq i \leq n$, la $i^{\textbf{ième}}$ ***cylindrification*** *de $\mathcal{R}$ est la plus grande relation $n+1$-aire dont la $i+1^{\text{ième}}$ projection est $\mathcal{R}$.*

Exemple 4.29 Soit $\mathcal{R}$ la relation de $\mathbb{N} \times \mathbb{N}$ définie par :

$$(i, j) \in \mathcal{R} \text{ si et seulement si } i = 2j.$$

La première projection de $\mathcal{R}$ est l'ensemble des entiers pairs et la deuxième l'ensemble des entiers.

La première cylindrification de $\mathcal{R}$ est la relation $\mathcal{R}'$ de $\mathbb{N}^3$ définie par :

$$(i, k, j) \in \mathcal{R}' \text{ si et seulement si } i = 2j.$$

Lemme 4.30 *La $i^{ième}$ projection et la $i^{ième}$ cylindrification d'une relation reconnaissable sont reconnaissables.*

Un corollaire très important des lemmes 4.27 et 4.30 est que la théorie du premier ordre des relations reconnaissables sur les arbres finis est décidable, plus précisemment nous avons :

Théorème 4.31 *[TW68] Soit $\varphi(x_1, \ldots, x_n)$ une formule du premier ordre comprenant des relations reconnaissables $\mathcal{R}_1, \mathcal{R}_2$, etc. et soit $Sol(\varphi)$ l'ensemble des n-uplets décrivant φ, $Sol(\varphi) = \{(t_1, \ldots, t_n) \mid \varphi(t_1, \ldots, t_n)\}$. Alors $Sol(\varphi)$ est un ensemble reconnaissable. De plus, un automate reconnaissant $Sol(\varphi)$ peut être construit à partir des automates reconnaissant les relations $\mathcal{R}_1, \mathcal{R}_2$, etc.*

4.3. Résultats d'accessibilité

4.3.1. Les termes RPA vus comme des arbres binaires

Les termes RPA sont des arbres binaires, $T_{\mathcal{X}}$ est tout simplement $T_{\mathcal{F}}$ où $\mathcal{F} = \mathcal{F}_0 \cup \mathcal{F}_2$ avec $\mathcal{F}_0 = Atomes$ et $\mathcal{F}_2 = \{\blacktriangleright\}$ et où nous notons les termes construits avec $\blacktriangleright$ de façon infixe.

Exemple 4.32 Par exemple, le terme $(X_1 \blacktriangleright X_2) \blacktriangleright (((X_3 \blacktriangleright X_4) \blacktriangleright Nil) \blacktriangleright (X_5 \blacktriangleright X_6))$ est présenté dans la figure 4.4 sous la forme d'un arbre binaire.

Exemple 4.33 La figure 4.5 présente l'exécution de l'exemple 4.6 :

$$X \xrightarrow{\alpha?} Y \blacktriangleright Nil \xrightarrow{\beta?} (Nil \blacktriangleright X) \blacktriangleright Nil \xrightarrow{\alpha!} (Nil \blacktriangleright (Y \blacktriangleright Nil)) \blacktriangleright Nil)$$

où les termes sont représentés comme des arbres binaires.

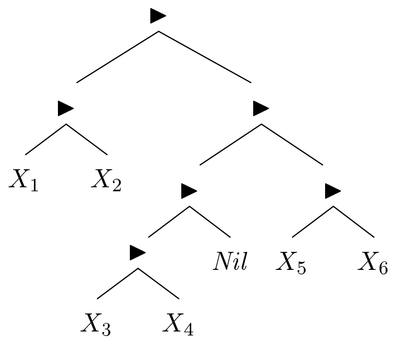


FIG. 4.4. – Le terme $(X_1 \blacktriangleright X_2) \blacktriangleright (((X_3 \blacktriangleright X_4) \blacktriangleright Nil) \blacktriangleright (X_5 \blacktriangleright X_6))$ vu comme un arbre binaire.

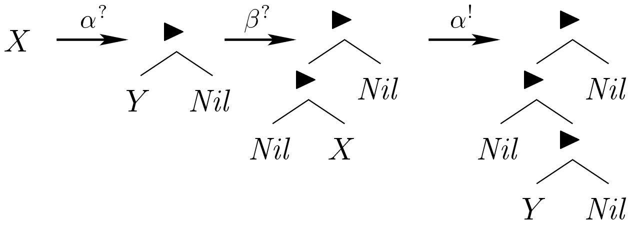


FIG. 4.5. – Une exécution RPA.

Le fait de voir les termes RPA comme des arbres nous permet d'utiliser des automates d'arbres pour définir des ensembles de termes.

Exemple 4.34 Nous considérons l'automate descendant $\mathcal{A}$ dont les états de contrôle sont p et p', dont l'état acceptant est p et dont les règles de transitions sont :

$$\begin{array}{lllllll} p' & \mapsto_{\mathcal{A}} & Nil, & & & & \\ p & \mapsto_{\mathcal{A}} & X, & \qquad & p' & \mapsto_{\mathcal{A}} & Y, \text{ si } Y \neq X, \\ p & \mapsto_{\mathcal{A}} & p \blacktriangleright p, & & p & \mapsto_{\mathcal{A}} & p' \blacktriangleright p, \\ p & \mapsto_{\mathcal{A}} & p \blacktriangleright p', & & p' & \mapsto_{\mathcal{A}} & p' \blacktriangleright p'. \end{array}$$

Le langage reconnu par $\mathcal{A}$ est l'ensemble des termes RPA qui contiennent la constante de processus X.

Nous considérons l'automate alternant $\mathcal{A}'$ dont les états de contrôle sont p_g, p_d et p_n, dont les états acceptants sont p_g et dont la fonction de transitions est :

$$\begin{array}{l} \delta(p_g, Y) = \mathit{vrai},\ Y \in \mathit{Atomes}, \\ \delta(p_g, \mathit{Nil}) = \mathit{vrai}, \\ \delta(p_g, \blacktriangleright) = (1, p_g) \wedge (2, p_d), \\ \delta(p_d, X) = \mathit{vrai}, \\ \delta(p_d, Y) = \mathit{faux}, \text{ si } Y \neq X, \\ \delta(p_d, \mathit{Nil}) = \mathit{faux}, \\ \delta(p_d, \blacktriangleright) = ((1, p_d) \wedge (2, p_n)) \vee ((1, p_n) \wedge (2, p_d)), \\ \delta(p_n, \mathit{Nil}) = \mathit{vrai}, \\ \delta(p_n, X) = \mathit{faux}, \\ \delta(p_n, Y) = \mathit{vrai}, \text{ si } Y \neq X, \\ \delta(p_n, \blacktriangleright) = (1, p_n) \wedge (2, p_n). \end{array}$$

Le langage reconnu par $\mathcal{A}$ est l'ensemble des termes RPA $C \blacktriangleright^n (u_1, \ldots, u_n)$ tels que chaque u_i contient exactement une occurrence de X. Par exemple, $X \blacktriangleright^2 (\mathit{Nil} \blacktriangleright^2 (X, \mathit{Nil}), X \blacktriangleright \mathit{Nil})$ est reconnu comme illustré par la figure 4.6.

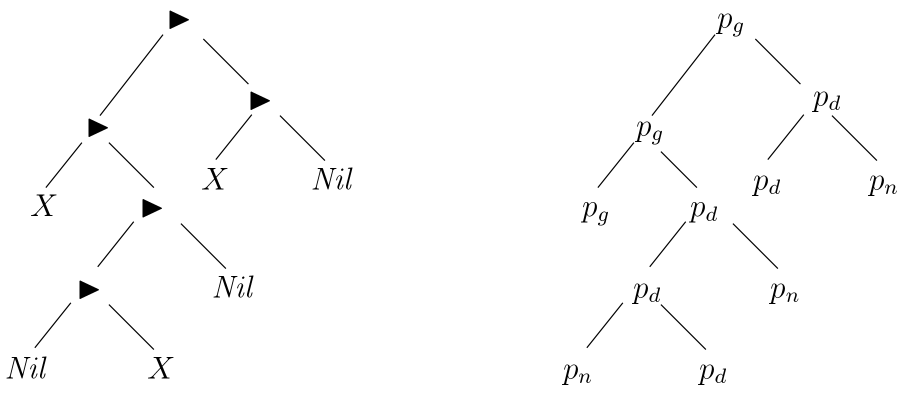


FIG. 4.6. – $X \blacktriangleright^2 (\mathit{Nil} \blacktriangleright^2 (X, \mathit{Nil}), X \blacktriangleright \mathit{Nil})$ reconnu par $\mathcal{A}$.

4.3.2. Nouvelles propriétés des transitions RPA

Nous allons maintenant présenter un certain nombre de lemmes sur les transitions entre deux termes RPA. Ces propriétés expliquent comment décomposer une exécution entre des termes en transitions sur leurs sous-termes. Ceci est la base des constructions des automates d'arbres du paragraphe suivant.

Dans toute la suite du chapitre, nous supposons donnés un ensemble de constantes de processus $\mathcal{X}$ et une déclaration RPA Δ.

Le lemme 4.35 donne une caractérisation de $\xrightarrow{!}^*$: si quel que soit t, u ayant pour frère t peut atteindre v ayant pour frère t, alors u peut atteindre v en ne faisant que des actions étiquetées par !.

Lemme 4.35 *Soient u et v deux termes RPA, $u \xrightarrow{!}^* v$ si et seulement si pour tout $t \in T$, $u \blacktriangleright t \to^* v \blacktriangleright t$.*

Démonstration.

($\Rightarrow$) Si $u \xrightarrow{!}^* v$, soit $t \in T_G$, en utilisant la règle R5 pour chaque transition liant u à v, nous obtenons $u \blacktriangleright t \to^* v \blacktriangleright t$. La règle R5 n'impose aucune condition sur t.

($\Leftarrow$) Nous supposons que pour tout $t \in T_G$, $u \blacktriangleright t \to^* v \blacktriangleright t$. En particulier, si $t = X$, nous allons montrer par induction sur $n \in \mathbb{N}$ que $u \blacktriangleright X \xrightarrow{n} v \blacktriangleright X$ entraîne $u \xrightarrow{!}^* v$.

- Si $n = 0$, alors $u \blacktriangleright X = v \blacktriangleright X$. Par conséquent, $u = v$, il s'ensuit $u \xrightarrow{!}^* v$.
- On suppose que la propriété est vraie pour $n \geq 0$. Supposons que $u \blacktriangleright X \xrightarrow{n+1} v \blacktriangleright X$, alors $u \blacktriangleright X \xrightarrow{n} t \to v \blacktriangleright X$. D'après le lemme 4.8, t est donc nécessairement de la forme $t_1 \blacktriangleright t_2$. De plus, le lemme 4.8 entraîne que $t_2 \xrightarrow{i} X$ avec $0 \leq i \leq 1$. Nécessairement $i = 0$ et donc $t_2 = X$, sinon d'après le lemme 4.9, $|X| > |t_2|$ ce qui est impossible.
  Comme $Constante(X) \neq \emptyset$, la transition $t_1 \blacktriangleright X \to v \blacktriangleright X$ ne peut être donnée par la règle R6, elle est donc une conséquence de la règle R5 et donc $t_1 \xrightarrow{!} v$. Par hypothèse d'induction, $u \xrightarrow{!}^* t_1$, par conséquent, nous avons $u \xrightarrow{!}^* v$.

⌟

Les deux lemmes qui suivent donnent les conditions portant sur v, v', t et t' pour que $v' \blacktriangleright t'$ soit atteignable à partir de $v \blacktriangleright t$. Ils permettent de décomposer une exécution entre $v \blacktriangleright t$ et $v' \blacktriangleright t'$ en une exécution entre v et v' d'une part, et une exécution entre t et t' d'autre part.

Lemme 4.36 *Soient v, v', t et t' quatre termes RPA, $v \blacktriangleright t \xrightarrow{!}^* v' \blacktriangleright t'$ si et seulement si $v \xrightarrow{!}^* v'$ et $t \to^* t'$.*

Démonstration.

($\Leftarrow$) Nous supposons que $v \xrightarrow{!}^* v'$ et $t \rightarrow^* t'$. En appliquant aux transitions successives de l'exécution $t \rightarrow^* t'$ la règle R3 quand l'action a une étiquette ! et la règle R4 quand l'action a une étiquette ?, nous obtenons que $v \blacktriangleright t \xrightarrow{!}^* v \blacktriangleright t'$. Nous pouvons alors appliquer à chaque transition la règle R5 (il n'y en effet plus aucune étiquette ?) et nous obtenons $v \blacktriangleright t' \xrightarrow{!}^* v' \blacktriangleright t'$. Par conséquent, $v \blacktriangleright t \xrightarrow{!}^* v' \blacktriangleright t'$.

($\Rightarrow$) Nous allons montrer par induction sur n que $v \blacktriangleright t \xrightarrow{n,!} v' \blacktriangleright t'$ entraîne $v \xrightarrow{!}^* v'$ et $t \rightarrow^* t'$.

- Si $n = 0$, la propriété est vérifiée.
- Si $n = 1$ et $v \blacktriangleright t \xrightarrow{!} v' \blacktriangleright t'$. Alors l'une des règles R3, R4 et R5 a été appliquée (la règle R6 ne l'a pas été sinon l'étiquette de la transition serait ?). Si c'est la règle R3 ou la règle R4, alors $v = v'$ et $t \rightarrow t'$. Si c'est la règle R5, $v \xrightarrow{!} v'$ et $t = t'$. Dans tous les cas, $v \xrightarrow{!}^* v'$ et $t \rightarrow^* t'$.
- Nous supposons maintenant que la propriété est vraie pour n et que $v \blacktriangleright t \xrightarrow{n+1,!} v' \blacktriangleright t'$. Alors $v \blacktriangleright t \xrightarrow{n,!} w \xrightarrow{!} v' \blacktriangleright t'$. Le lemme 4.8 entraîne que w est de la forme $v'' \blacktriangleright t''$. Par hypothèse d'induction, $v \xrightarrow{!}^* v''$ et $t \rightarrow^* t''$ et d'après le cas précédent, $v'' \xrightarrow{!}^* v'$ et $t'' \rightarrow^* t'$. Nous concluons donc que $v \xrightarrow{!}^* v'$ et $t \rightarrow^* t'$.

⌟

Le lemme suivant raffine le lemme 4.8 :

Lemme 4.37 *Soient, v, v', t et t' quatre termes RPA, $v \blacktriangleright t \rightarrow^* v' \blacktriangleright t'$ si et seulement si $t \rightarrow^* t'$ et* $\begin{cases} v \xrightarrow{!}^* v', \\ \textit{ou } t' \not\rightarrow \textit{ et } v \rightarrow^* v'. \end{cases}$

Démonstration.

($\Leftarrow$) Nous supposons que $t \rightarrow^* t'$ et $\begin{cases} v \xrightarrow{!}^* v', \\ \text{ou } t' \not\rightarrow \text{ et } v \rightarrow^* v'. \end{cases}$

Si $t \rightarrow^* t'$ et $v \xrightarrow{!}^* v'$, d'après le lemme précédent, $v \blacktriangleright t' \xrightarrow{!}^* v' \blacktriangleright t'$ et donc $v \blacktriangleright t' \rightarrow^* v' \blacktriangleright t'$.

Si $t \rightarrow^* t' \not\rightarrow$ et $v \rightarrow^* v'$. En appliquant aux transitions successives de l'exécution $t \rightarrow^* t'$ la règle R3 quand l'action a une étiquette ! et la règle R4 quand l'action a une étiquette ?, nous obtenons que $v \blacktriangleright t \xrightarrow{!}^* v \blacktriangleright t'$. En appliquant aux transitions successives de l'exécution $v \rightarrow^* v'$ la règle R5 quand l'action a une étiquette ! et la règle R6 quand l'action a une étiquette ?, nous obtenons que $v \blacktriangleright t' \rightarrow^* v \blacktriangleright t'$. La règle R6 est applicable car $t' \not\rightarrow$.

($\Rightarrow$) Nous supposons que $v \blacktriangleright t \rightarrow^* v' \blacktriangleright t'$ alors soit (a) $v \blacktriangleright t \xrightarrow{!}^* v' \blacktriangleright t'$ soit (b) $v \blacktriangleright t \rightarrow^* v_1 \blacktriangleright t_1 \xrightarrow{?} v_2 \blacktriangleright t_2 \rightarrow^* v' \blacktriangleright t'$.

(a) Le lemme 4.36 entraîne que $t \rightarrow^* t'$ et $v \xrightarrow{!}^* v'$.

(b) La transition $v_1 \blacktriangleright t_1 \xrightarrow{?} v_2 \blacktriangleright t_2$ est due à l'application de la règle R6 qui est la seule règle dont le membre gauche contient le signe $\blacktriangleright$ et dont l'étiquette est ?. t_1 doit donc satisfaire la condition de la règle R6 donc $State(t_1) = \emptyset$. Il s'ensuit, d'après le lemme 4.11 que $t_1 \not\to$. Par conséquent, $t' = t_1$ et $t' \not\to$. On a donc $t' \not\to$ et $v \to^* v'$.

$\lrcorner$

Les deux lemmes suivants donnent des conditions nécessaires et suffisantes pour qu'un terme soit issu d'une exécution commençant par une constante de processus appartenant à $\mathcal{X}$.

Lemme 4.38 *Soient X une constante de processus et v et t deux termes RPA, $X \to^* v \blacktriangleright t$ si et seulement s'il existe C et u tels que $(X \to_\Delta C, u)$ est une règle de Δ, $u \to^* t$ et* $\begin{cases} C \xrightarrow{!}{}^* v, \\ \textit{ou } t \not\to \textit{ et } C \to^* v. \end{cases}$

Démonstration.

($\Rightarrow$) Nous supposons que $X \to^* v \blacktriangleright t$. La première étape de cette exécution est obtenue par l'une des règles R1 ou R2. Donc, $X \to C \blacktriangleright u$ avec $X \to_\Delta C, u$ in Δ. Nous avons donc $C \blacktriangleright u \to^* v \blacktriangleright t$, le lemme 4.37 entraîne $u \to^* t$, et $\begin{cases} C \xrightarrow{!}{}^* v, \\ \text{ou } t \not\to \text{ et } C \to^* v. \end{cases}$.

($\Leftarrow$) S'il existe C et u tels que $(X \to_\Delta C, u)$ est une règle de Δ, $u \to^* t$ et $\begin{cases} C \xrightarrow{!}{}^* v, \\ \text{ou } t \not\to \text{ et } C \to^* v. \end{cases}$.
D'après le lemme 4.37, $C \blacktriangleright u \to^* v \blacktriangleright t$ et d'après les règles R1 et R2, $X \to C \blacktriangleright u$. D'où $X \to^* v \blacktriangleright t$.

$\lrcorner$

Lemme 4.39 *Soient X une constante de processus et v et t deux termes RPA, $X \xrightarrow{!}{}^* v \blacktriangleright t$ si et seulement si $X \in \mathcal{X}^!$ et il existe C et u tels que $X \to_\Delta C, u$ est une règle de Δ, $u \to^* t$ et $C \xrightarrow{!}{}^* v$.*

Démonstration.

($\Rightarrow$) Nous supposons que $X \xrightarrow{!}{}^* v \blacktriangleright t$. La première étape de cette exécution est obtenue par la règle R1. Donc, $X \in \mathcal{X}^!$ et $X \to C \blacktriangleright u$ avec $X \to_\Delta C, u$ in Δ. Nous avons donc $C \blacktriangleright u \xrightarrow{!}{}^* v \blacktriangleright t$, le lemme 4.36 entraîne $u \to^* t$ et $C \xrightarrow{!}{}^* v$.

($\Leftarrow$) Si $X \in \mathcal{X}^!$ et il existe C et u tels que $X \to_\Delta C, u$ est une règle de Δ, $u \to^* t$ et $C \xrightarrow{!}{}^* v$. D'après le lemme 4.36, $C \blacktriangleright u \xrightarrow{!}{}^* v \blacktriangleright t$ et d'après la règle R1, $X \to C \blacktriangleright u$. D'où $X \xrightarrow{!}{}^* v \blacktriangleright t$.

$\lrcorner$

Les transitions des automates d'arbres que nous rencontrerons dans le chapitre 4 sont des traductions immédiates des lemmes 4.36, 4.37, 4.38 et 4.39.

4.3.3. Un automate d'arbres reconnaissant $\rightarrow^*$ et $\xrightarrow{!}^*$

Dans ce paragraphe, nous prouvons que les relations d'accessibilité $\rightarrow^*$ et $\xrightarrow{!}^*$ entre des termes RPA sont reconnaissables par des automates d'arbres.

Lemme 4.40 *L'ensemble* $L_{\text{term}} \stackrel{def}{=} \{u \in T_G \mid u \not\rightarrow\}$ *des* ***termes terminés*** *est reconnaissable.*

Démonstration. $u \not\rightarrow$ si et seulement si $Constante(u) = \emptyset$ (Lemme 4.11). Donc, un terme appartient à L_{term} si et seulement s'il est produit par l'alphabet $\{Nil, \blacktriangleright\}$. Par conséquent, l'automate dont l'unique état $p_\downarrow$ est acceptant et dont les règles de transitions sont :

$$\delta(p_\downarrow, Nil) = vrai, \qquad \delta(p_\downarrow, X) = faux, \qquad \delta(p_\downarrow, \blacktriangleright) = (1, p_\downarrow) \wedge (2, p_\downarrow)$$

reconnaît L_{term}. ⌟

Nous considérons maintenant l'automate alternant $\mathcal{A}_{\xrightarrow{*}}$ dont les états sont p, $\bar{p}$, $p_\downarrow$ et tous les p_t et $\bar{p}_t$ où t est un sous-terme de l'un des termes qui apparaît dans Δ (par conséquent $|\mathcal{Q}|$ est en $O(|\Delta|)$). $\mathcal{A}_{\xrightarrow{*}}$ reconnaît des paires de termes et satisfait les propriétés de correction suivantes :

Lemme 4.41

$$L(p) = \{[u,v] \mid u \rightarrow^* v\}, \qquad L(\bar{p}) = \{[u,v] \mid u \xrightarrow{!}^* v\},$$
$$L(p_t) = \{[u,v] \mid t \rightarrow^* v\}, \qquad L(\bar{p}_t) = \{[u,v] \mid t \xrightarrow{!}^* v\},$$
$$L(p_\downarrow) = \{[u,v] \mid v \not\rightarrow\},$$

où t, u, v *sont des termes de* T.

Nous démontrerons ce lemme une fois défini $\mathcal{A}_{\xrightarrow{*}}$.

Voici la définition de la fonction de transition alternée δ où $X, Y \in \mathcal{X}$ et $f \in (Atomes \cup \{\perp\})$. Pour plus de lisibilité, nous n'avons pas écrit les transitions $\delta(p', fg) = faux$ avec $p' \in \mathcal{Q}$ et $f, g \in (Atomes \cup \{\perp\})$. Donc, quand $\delta(p', fg)$ n'est pas explicitement définie, $\delta(p', fg) = faux$.

$$\delta(p_\downarrow, f\ Nil) = vrai, \tag{4.1}$$
$$\delta(p_\downarrow, f\ \blacktriangleright) = (1, p_\downarrow) \wedge (2, p_\downarrow), \tag{4.2}$$

$$\delta(p, NilNil) = \delta(\bar{p}, NilNil) = vrai, \tag{4.3}$$

$$\delta(p, XY) = \delta(\bar{p}, XY) = \begin{cases} vrai \text{ si } X = Y, \\ faux \text{ sinon,} \end{cases} \tag{4.4}$$

$$\delta(p, \blacktriangleright\blacktriangleright) = (2, p) \wedge \big[(1, \bar{p}) \vee ((2, p_\downarrow) \wedge (1, p))\big], \tag{4.5}$$
$$\delta(\bar{p}, \blacktriangleright\blacktriangleright) = (1, \bar{p}) \wedge (2, p), \tag{4.6}$$

$$\delta(p, X \blacktriangleright) = \bigvee_{X \to_\Delta C, u} (2, p_u) \wedge \big[(1, \bar{p}_C) \vee ((2, p_\downarrow) \wedge (1, p_C))\big], \tag{4.7}$$
$$\delta(\bar{p}, X \blacktriangleright) = \begin{cases} \displaystyle\bigvee_{X \to_\Delta C, u} (2, p_u) \wedge (1, \bar{p}_C) \text{ si } X \in \mathcal{X}^!, \\ \mathit{faux} \text{ sinon,} \end{cases} \tag{4.8}$$

$$\delta(p_t, f\ \mathit{Nil}) = \delta(\bar{p}_t, f \mathit{Nil}) = \begin{cases} \mathit{vrai} \text{ si } t = \mathit{Nil}, \\ \mathit{faux} \text{ sinon,} \end{cases} \tag{4.9}$$
$$\delta(p_t, f X) = \delta(\bar{p}_t, f X) = \begin{cases} \mathit{vrai} \text{ si } t = X, \\ \mathit{faux} \text{ sinon,} \end{cases} \tag{4.10}$$

$$\delta(p_{t_1 \blacktriangleright t_2}, f \blacktriangleright) = (2, p_{t_2}) \wedge \big[(1, \bar{p}_{t_1}) \vee ((2, p_\downarrow) \wedge (1, p_{t_1}))\big], \tag{4.11}$$
$$\delta(\bar{p}_{t_1 \blacktriangleright t_2}, f \blacktriangleright) = (1, \bar{p}_{t_1}) \wedge (2, p_{t_2}), \tag{4.12}$$

$$\delta(p_X, f \blacktriangleright) = \delta(p, X \blacktriangleright) \tag{4.13}$$
$$\delta(\bar{p}_X, f \blacktriangleright) = \delta(\bar{p}, X \blacktriangleright) \tag{4.14}$$

Remarque 4.42 Nous avons $\delta(\bar{p}_t, f \blacktriangleright) = \mathit{faux}$ si t n'est pas de la forme $t_1 \blacktriangleright t_2$.

Démonstration de $\mathbf{L}(\mathbf{p}_\downarrow) = \{[\mathbf{u}, \mathbf{v}] \mid \mathbf{v} \not\to\}$.
La valeur de la fonction de transition de l'automate alterné $\delta(p_\downarrow, fg)$ est la même que celle de la fonction de l'automate d'arbres du lemme 4.40 $\delta(p_\downarrow, g)$. Par conséquent, $L(p_\downarrow) = \{[u, v] \mid v \not\to\}$. ⌟

Démonstration de $\mathbf{L}(\mathbf{p_t}) = \{[\mathbf{u}, \mathbf{v}] \mid \mathbf{t} \to^* \mathbf{v}\}$ *et de* $\mathbf{L}(\bar{\mathbf{p}}_\mathbf{t}) = \{[\mathbf{u}, \mathbf{v}] \mid \mathbf{t} \xrightarrow{!}{}^* \mathbf{v}\}$.
Nous allons faire une démonstration par double inclusion.

- $L(p_t) \subseteq \{[u, v] \mid t \to^* v\}$ et $L(\bar{p}_t) \subseteq \{[u, v] \mid t \xrightarrow{!}{}^* v\}$.
  Soient $[u, v]$ un terme produit et t un terme, nous montrons par induction sur v que si $[u, v] \in L(p_t)$, alors $t \to^* v$ et si $[u, v] \in L(\bar{p}_t)$, alors $t \xrightarrow{!}{}^* v$.
  - Supposons que $v = \mathit{Nil}$.

Si $[u, v] \in L(p_t)$ ou si $[u, v] \in L(\bar{p}_t)$, alors d'après la règle 4.9, $t = \mathit{Nil}$. Donc $t \rightarrow^* v$ et $t \xrightarrow{!}{}^* v$.

- Supposons que $v = X$.

  Si $[u, v] \in L(p_t)$ ou si $[u, v] \in L(\bar{p}_t)$, alors d'après la règle 4.10, $t = X$. Donc $t \rightarrow^* v$ et $t \xrightarrow{!}{}^* v$.

- Supposons que $v = v_1 \blacktriangleright v_2$.

  Nous avons alors $[u, v] = [u_1, v_1]f \blacktriangleright [u_2, v_2]$.

  - Si $[u, v] \in L(p_t)$, alors, d'après les règles 4.11 et 4.13, soit $t = t_1 \blacktriangleright t_2$, soit $t = X$.
    - Si $t = t_1 \blacktriangleright t_2$, alors $[u_2, v_2] \in L(p_{t_2}) \wedge \big[[u_1, v_1] \in L(\bar{p}_{t_1}) \vee ([u_2, v_2] \in L(p_\downarrow) \wedge [u_1, v_1] \in L(p_{t_1}))\big]$. D'après ce qui précède, si $[u_2, v_2] \in L(p_\downarrow)$, alors $v_2 \not\rightarrow$. De plus, par hypothèse d'induction, si $[u_2, v_2] \in L(p_{t_2})$, $t_2 \rightarrow^* v_2$, si $[u_1, v_1] \in L(\bar{p}_{t_1})$, $t_1 \xrightarrow{!}{}^* v_1$ et si $[u_1, v_1] \in L(p_{t_1})$, $t_1 \rightarrow^* v_1$. Nous avons donc $t_2 \rightarrow^* v_2 \wedge \big[t_1 \xrightarrow{!}{}^* v_1 \vee (v_2 \not\rightarrow \wedge t_1 \rightarrow^* v_1)\big]$. Donc d'après le lemme 4.37, $t \rightarrow^* v$.
    - Si $t = X$, alors il existe une règle $X \rightarrow_\Delta C, w$ et $[u_2, v_2] \in L(p_w) \wedge \big[[u_1, v_1] \in L(\bar{p}_C) \vee ([u_2, v_2] \in L(p_\downarrow) \wedge [u_1, v_1] \in L(p_C))\big]$. Par hypothèse d'induction, il existe une règle $X \rightarrow_\Delta C, w$ et $w \rightarrow^* v_2 \wedge \big[C \xrightarrow{!}{}^* v_1] \vee (v_2 \not\rightarrow \wedge C \rightarrow^* v_1)\big]$. Donc d'après le lemme 4.38, $t \rightarrow^* v$.
  - Si $[u, v] \in L(\bar{p}_t)$, alors, d'après les règles 4.12 et 4.14, soit $t = t_1 \blacktriangleright t_2$, soit $t = X$.
    - Si $t = t_1 \blacktriangleright t_2$, alors $[u_1, v_1] \in L(\bar{p}_{t_1}) \wedge [u_2, v_2] \in L(p_{t_2})$. Par hypothèse d'induction, $t_1 \xrightarrow{!}{}^* v_1$ et $t_2 \rightarrow^* v_2$. Donc d'après le lemme 4.36, $t \xrightarrow{!}{}^* v$.
    - Si $t = X$, alors il existe une règle $X \rightarrow_\Delta C, w$ et $[u_1, v_1] \in L(\bar{p}_C) \wedge [u_2, v_2] \in L(p_w)$. Par hypothèse d'induction, il existe une règle $X \rightarrow_\Delta C, w$ et $C \xrightarrow{!}{}^* v_1 \wedge w \rightarrow^* v_2$. Donc d'après le lemme 4.39, $t \xrightarrow{!}{}^* v$.

  Nous avons bien montré que pour tout v et tout u, si $[u, v] \in L(p_t)$, alors $t \rightarrow^* v$ et si $[u, v] \in L(\bar{p}_t)$, alors $t \xrightarrow{!}{}^* v$. Nous en concluons que $L(p_t) \subseteq \{[u, v] \mid t \rightarrow^* v\}$ et si $L(\bar{p}_t) \subseteq \{[u, v] \mid t \xrightarrow{!}{}^* v\}$.

- $\{[u, v] \mid t \rightarrow^* v\} \subseteq L(p_t)$ et $\{[u, v] \mid t \xrightarrow{!}{}^* v\} \subseteq L(\bar{p}_t)$.

  Soient $[u, v]$ un terme produit et t un terme, nous montrons par induction sur v que si $t \rightarrow^* v$, alors $[u, v] \in L(p_t)$ et si $t \xrightarrow{!}{}^* v$, alors $[u, v] \in L(\bar{p}_t)$.

  - Supposons que $v = \mathit{Nil}$.

    Si $t \rightarrow^* v$ ou si $t \xrightarrow{!}{}^* v$, alors $t = v$. En effet, si $t \xrightarrow{n} v$ avec $n > 0$ alors, d'après le lemme 4.9 $|v| > |t|$. Donc, d'après la règle 4.9, $[u, v] \in L(p_t)$ et $[u, v] \in L(\bar{p}_t)$.

- Supposons que $v = X$.

  Si $t \to^* v$ ou si $t \xrightarrow{!}{}^* v$, alors $t = v$ pour la même raison que dans le cas précédent. Donc, d'après la règle 4.10, $[u, v] \in L(p_t)$ et $[u, v] \in L(\bar{p}_t)$.

- Supposons $v = v_1 \blacktriangleright v_2$.

  Nous avons alors $[u, v] = [u_1, v_1]f \blacktriangleright [u_2, v_2]$.

  – Si $t \to^* v$ soit $t = t_1 \blacktriangleright t_2$, soit $t = X$.
    – Si $t = t_1 \blacktriangleright t_2$, alors d'après le lemme 4.37, $t_2 \to^* v_2 \wedge \big[t_1 \xrightarrow{!}{}^* v_1 \vee (v_2 \not\to \wedge t_1 \to^* v_1)\big]$. Comme nous l'avons vu tout au début de la démonstration du lemme si $v_2 \not\to$, alors $[u_2, v_2] \in L(p_\downarrow)$. De plus, par hypothèse d'induction, si $t_2 \to^* v_2$, $[u_2, v_2] \in L(p_{t_2})$, si $t_1 \xrightarrow{!}{}^* v_1$, $[u_1, v_1] \in L(\bar{p}_{t_1})$, et si $t_1 \to^* v_1$, $[u_1, v_1] \in L(p_{t_1})$. Nous avons donc $[u_2, v_2] \in L(p_{t_2}) \wedge \big[[u_1, v_1] \in L(\bar{p}_{t_1}) \vee ([u_2, v_2] \in L(p_\downarrow) \wedge [u_1, v_1] \in L(p_{t_1}))\big]$. Donc d'après la règle 4.11, $[u, v] \in L(p_t)$.
    – Si $t = X$, alors d'après le lemme 4.38, il existe une règle $X \to_\Delta C, w$ et $w \to^* v_2 \wedge \big[C \xrightarrow{!}{}^* v_1] \vee (v_2 \not\to \wedge C \to^* v_1)\big]$. Par hypothèse d'induction, il existe une règle $X \to_\Delta C, w$ et $[u_2, v_2] \in L(p_w) \wedge \big[[u_1, v_1] \in L(\bar{p}_C) \vee ([u_2, v_2] \in L(p_\downarrow) \wedge [u_1, v_1] \in L(p_C))\big]$. Donc d'après la règle 4.13, $[u, v] \in L(p_t)$.

  – Si $t \xrightarrow{!}{}^* v$, soit $t = t_1 \blacktriangleright t_2$, soit $t = X$.
    – Si $t = t_1 \blacktriangleright t_2$, alors, d'après le lemme 4.36, $t_1 \xrightarrow{!}{}^* v_1$ et $t_2 \to^* v_2$. Par hypothèse d'induction, $[u_1, v_1] \in L(\bar{p}_{t_1}) \wedge [u_2, v_2] \in L(p_{t_2})$. Donc d'après la règle 4.12, $[u, v] \in L(pb_t)$.
    – Si $t = X$, alors, d'après le lemme 4.39, il existe une règle $X \to_\Delta C, w$ et $C \xrightarrow{!}{}^* v_1 \wedge w \to^* v_2$. Par hypothèse d'induction, il existe une règle $X \to_\Delta C, w$ et $[u_1, v_1] \in L(\bar{p}_C) \wedge [u_2, v_2] \in L(p_w)$. Donc d'après la règle 4.14, $[u, v] \in L(pb_t)$.

  Nous avons bien montré que pour tout v et tout u, si $t \to^* v$, alors $[u, v] \in L(p_t)$ et si $t \xrightarrow{!}{}^* v$, alors $[u, v] \in L(\bar{p}_t)$. Nous en concluons que $\{[u, v] \mid t \to^* v\} \subseteq L(p_t)$ et si $\{[u, v] \mid t \xrightarrow{!}{}^* v\} \subseteq L(\bar{p}_t)$.

Par conséquent, $L(p_t) = \{[u, v] \mid t \to^* v\}$ et $L(\bar{p}_t) = \{[u, v] \mid t \xrightarrow{!}{}^* v\}$.

⌟

Démonstration de $\mathbf{L(p) = \{[u, v] \mid u \to^* v\}}$ *et de* $\mathbf{L(\bar{p}) = \{[u, v] \mid u \xrightarrow{!}{}^* v\}}$.

Nous allons faire une démonstration par double inclusion.

- $L(p) \subseteq \{[u, v] \mid u \to^* v\}$ et $L(\bar{p}) \subseteq \{[u, v] \mid u \xrightarrow{!}{}^* v\}$.

  Soit $[u, v]$ un terme produit, nous montrons par induction sur v que si $[u, v] \in L(p)$, alors $u \to^* v$ et si $[u, v] \in L(\bar{p}_t)$, alors $u \xrightarrow{!}{}^* v$.

  - Supposons que $v = Nil$.

Si $[u, v] \in L(p)$ ou si $[u, v] \in L(\bar{p})$, alors d'après la règle 4.3, $u = \mathit{Nil}$. Donc $u \rightarrow^* v$ et $u \xrightarrow{!}{}^* v$.

- Supposons que $v = X$.

  Si $[u, v] \in L(p)$ ou si $[u, v] \in L(\bar{p})$, alors d'après la règle 4.4, $u = X$. Donc $u \rightarrow^* v$ et $u \xrightarrow{!}{}^* v$.

- Supposons que $v = v_1 \blacktriangleright v_2$.
  - Si $[u, v] \in L(p)$, alors, d'après les règles 4.5 et 4.7, soit $u = u_1 \blacktriangleright u_2$, soit $u = X$.
    - Si $u = u_1 \blacktriangleright u_2$, alors $[u_2, v_2] \in L(p) \wedge \big[[u_1, v_1] \in L(\bar{p}) \vee ([u_2, v_2] \in L(p_\downarrow) \wedge [u_1, v_1] \in L(p))\big]$. Alors $v_2 \not\rightarrow$ et par hypothèse d'induction, si $[u_2, v_2] \in L(p)$, $u_2 \rightarrow^* v_2$, si $[u_1, v_1] \in L(\bar{p})$, $u_1 \xrightarrow{!}{}^* v_1$ et si $[u_1, v_1] \in L(p)$, $u_1 \rightarrow^* v_1$. Nous avons donc $u_2 \rightarrow^* v_2 \wedge \big[u_1 \xrightarrow{!}{}^* v_1 \vee (v_2 \not\rightarrow \wedge u_1 \rightarrow^* v_1)\big]$. D'où, d'après le lemme 4.37, $u \rightarrow^* v$.
    - Si $u = X$, alors il existe une règle $X \rightarrow_\Delta C, w$ et $[u_2, v_2] \in L(p_w) \wedge \big[[u_1, v_1] \in L(\bar{p}_C) \vee ([u_2, v_2] \in L(p_\downarrow) \wedge [u_1, v_1] \in L(p_C))\big]$. D'après les résultats précédents de cette démonstration, il existe une règle $X \rightarrow_\Delta C, w$ et $w \rightarrow^* v_2 \wedge \big[C \xrightarrow{!}{}^* v_1] \vee (v_2 \not\rightarrow \wedge C \rightarrow^* v_1)\big]$. Donc d'après le lemme 4.38, $u \rightarrow^* v$.
  - Si $[u, v] \in L(\bar{p})$, alors, d'après les règles 4.6 et 4.8, soit $u = u_1 \blacktriangleright u_2$, soit $u = X$.
    - Si $u = u_1 \blacktriangleright u_2$, alors $[u_1, v_1] \in L(\bar{p}) \wedge [u_2, v_2] \in L(p)$. Par hypothèse d'induction, $u_1 \xrightarrow{!}{}^* v_1$ et $u_2 \rightarrow^* v_2$. Donc d'après le lemme 4.36, $u \xrightarrow{!}{}^* v$.
    - Si $u = X$, alors il existe une règle $X \rightarrow_\Delta C, w$ et $[u_1, v_1] \in L(\bar{p}_C) \wedge [u_2, v_2] \in L(p_w)$. D'après les résultats précédents de cette démonstration, il existe une règle $X \rightarrow_\Delta C, w$ et $C \xrightarrow{!}{}^* v_1 \wedge w \rightarrow^* v_2$. Donc d'après le lemme 4.39, $u \xrightarrow{!}{}^* v$.

  Nous avons bien montré que pour tout v et tout u, si $[u, v] \in L(p)$, alors $u \rightarrow^* v$ et si $[u, v] \in L(\bar{p})$, alors $u \xrightarrow{!}{}^* v$. Nous en concluons que $L(p) \subseteq \{[u, v] \mid u \rightarrow^* v\}$ et si $L(\bar{p}) \subseteq \{[u, v] \mid u \xrightarrow{!}{}^* v\}$.

- $\{[u, v] \mid u \rightarrow^* v\} \subseteq L(p)$ et $\{[u, v] \mid u \xrightarrow{!}{}^* v\} \subseteq L(\bar{p})$.

  Soit $[u, v]$ un terme produit, nous montrons par induction sur v que si $u \rightarrow^* v$, alors $[u, v] \in L(p)$ et si $u \xrightarrow{!}{}^* v$, alors $[u, v] \in L(\bar{p})$.

  - Supposons que $v = \mathit{Nil}$.

    Si $u \rightarrow^* v$ ou si $u \xrightarrow{!}{}^* v$, alors $u = v$. Donc, d'après la règle 4.3, $[u, v] \in L(p)$ et $[u, v] \in L(\bar{p})$.

  - Supposons que $v = X$.

Si $t \rightarrow^* v$ ou si $t \xrightarrow{!}{}^* v$, alors $t = v$. Donc, d'après la règle 4.4, $[u, v] \in L(p)$ et $[u, v] \in L(\bar{p})$.

- Supposons $v = v_1 \blacktriangleright v_2$.
  - Si $u \rightarrow^* v$ soit $u = u_1 \blacktriangleright u_2$, soit $u = X$.
    - Si $u = u_1 \blacktriangleright u_2$, alors d'après le lemme 4.37, $u_2 \rightarrow^* v_2 \wedge \big[u_1 \xrightarrow{!}{}^* v_1 \vee (v_2 \not\rightarrow \wedge u_1 \rightarrow^* v_1)\big]$. Par hypothèse d'induction, $[u_2, v_2] \in L(p) \wedge \big[[u_1, v_1] \in L(\bar{p}) \vee ([u_2, v_2] \in L(p_\downarrow) \wedge [u_1, v_1] \in L(p))\big]$. Donc d'après la règle 4.5, $[u, v] \in L(p)$.
    - Si $u = X$, alors d'après le lemme 4.38, il existe une règle $X \rightarrow_\Delta C, w$ et $w \rightarrow^* v_2 \wedge \big[C \xrightarrow{!}{}^* v_1\big] \vee (v_2 \not\rightarrow \wedge C \rightarrow^* v_1)\big]$. D'après ce qui précède, il existe une règle $X \rightarrow_\Delta C, w$ et $[u_2, v_2] \in L(p_w) \wedge \big[[u_1, v_1] \in L(\bar{p}_C) \vee ([u_2, v_2] \in L(p_\downarrow) \wedge [u_1, v_1] \in L(p_C))\big]$. Donc d'après la règle 4.13, $[u, v] \in L(p)$.
  - Si $u \xrightarrow{!}{}^* v$, soit $u = u_1 \blacktriangleright u_2$, soit $u = X$.
    - Si $u = u_1 \blacktriangleright u_2$, alors, d'après le lemme 4.36, $u_1 \xrightarrow{!}{}^* v_1$ et $u_2 \rightarrow^* v_2$. Par hypothèse d'induction, $[u_1, v_1] \in L(\bar{p}) \wedge [u_2, v_2] \in L(p)$. Donc d'après la règle 4.6, $[u, v] \in L(pb)$.
    - Si $u = X$, alors, d'après le lemme 4.39, il existe une règle $X \rightarrow_\Delta C, w$ et $C \xrightarrow{!}{}^* v_1 \wedge w \rightarrow^* v_2$. D'après ce qui précède, il existe une règle $X \rightarrow_\Delta C, w$ et $[u_1, v_1] \in L(\bar{p}_C) \wedge [u_2, v_2] \in L(p_w)$. Donc d'après la règle 4.8, $[u, v] \in L(pb)$.

  Nous avons bien montré que pour tout v et tout u, si $u \rightarrow^* v$, alors $[u, v] \in L(p)$ et si $u \xrightarrow{!}{}^* v$, alors $[u, v] \in L(\bar{p})$. Nous en concluons que $\{[u, v] \mid u \rightarrow^* v\} \subseteq L(p)$ et si $\{[u, v] \mid u \xrightarrow{!}{}^* v\} \subseteq L(\bar{p})$.

Par conséquent, $L(p) = \{[u, v] \mid u \rightarrow^* v\}$ et $L(\bar{p}) = \{[u, v] \mid u \xrightarrow{!}{}^* v\}$.

⌟

Nous avons alors le corollaire important :

Théorème 4.43 *Les relations $\rightarrow^*$ et $\xrightarrow{!}{}^*$ sont reconnaissables. De plus, il existe un automate d'arbres qui les reconnaît et qui n'a que $O(|\Delta|)$ états.*

Démonstration. Notre construction utilise un automate d'arbres alternant pour plus de clarté : les clauses qui définissent δ sont ainsi des traductions immédiates des lemmes de la section 4.3.2. Il est cependant facile d'adapter cette construction et d'obtenir un automate descendant d'arbres ayant $O(|\Delta|)$ états et reconnaissant les relations d'accessibilité comme nous allons le voir.

Nous considérons l'automate descendant d'arbres $\mathcal{A}$ dont les états sont P, $P_\downarrow$, $\bar{P}$, $\bar{P}_\downarrow$ et tous les P_t, $P_{t\downarrow}$, $\bar{P}_t$ et $\bar{P}_{t\downarrow}$ où t est un sous-terme de l'un des termes qui apparaît dans Δ (par conséquent $|\mathcal{Q}|$ est en $O(|\Delta|)$). $\mathcal{A}$ reconnaît des paires de termes et ses états satisfont les propriétés de correction suivantes :

Lemme 4.44

$$
\begin{array}{ll}
L(P) = \{[u,v] \mid u \to^* v\}, & L(P_\downarrow) = \{[u,v] \mid u \to^* v \wedge v \not\to\}, \\
L(\bar{P}) = \{[u,v] \mid u \overset{!}{\to}{}^* v\}, & L(\bar{P}_\downarrow) = \{[u,v] \mid u \overset{!}{\to}{}^* v \wedge v \not\to\}, \\
L(P_t) = \{[u,v] \mid t \to^* v\}, & L(P_{t\downarrow}) = \{[u,v] \mid t \to^* v \wedge v \not\to\}, \\
L(\bar{P}_t) = \{[u,v] \mid t \overset{!}{\to}{}^* v\}, & L(\bar{P}_{t\downarrow}) = \{[u,v] \mid t \overset{!}{\to}{}^* v \wedge v \not\to\},
\end{array}
$$

où t, u, v sont des termes de $T_G \cup \{\bot\}$.

Voici la définition de la fonction de transition $\mapsto_{\mathcal{A}}$ où $X, Y \in \mathcal{X}$ et $f \in (Atomes \cup \{\bot\})$. Cette fonction est directement issue de δ.

- La règle $\delta(p, NilNil) = \delta(\bar{p}, NilNil) = vrai$ est remplacée par :
  $P \mapsto_{\mathcal{A}} NilNil,\quad P_\downarrow \mapsto_{\mathcal{A}} NilNil,\quad \bar{P} \mapsto_{\mathcal{A}} NilNil,\quad \bar{P}_\downarrow \mapsto_{\mathcal{A}} NilNil.$

- La règle $\delta(p, XY) = \delta(\bar{p}, XY) = vrai$ si $X = Y$ est remplacée par :
  $P \mapsto_{\mathcal{A}} XX,\qquad \bar{P} \mapsto_{\mathcal{A}} XX.$

- La règle $\delta(p, \blacktriangleright\blacktriangleright) = (2,p) \wedge \big[(1,\bar{p}) \vee ((2,p_\downarrow) \wedge (1,p))\big]$
  $= \big[(1,\bar{p}) \wedge (2,p)\big] \vee \big[(1,p) \wedge (2,p) \wedge (2,p_\downarrow)\big]$ est remplacée par :

  $P \mapsto_{\mathcal{A}} \blacktriangleright\blacktriangleright\ \bar{P}P,\quad P \mapsto_{\mathcal{A}} \blacktriangleright\blacktriangleright\ \bar{P}_\downarrow P,\quad P \mapsto_{\mathcal{A}} \blacktriangleright\blacktriangleright\ \bar{P}P_\downarrow,$
  $P \mapsto_{\mathcal{A}} \blacktriangleright\blacktriangleright\ PP_\downarrow,\quad P \mapsto_{\mathcal{A}} \blacktriangleright\blacktriangleright\ \bar{P}_\downarrow P_\downarrow,\quad P \mapsto_{\mathcal{A}} \blacktriangleright\blacktriangleright\ P_\downarrow P_\downarrow,$
  $P_\downarrow \mapsto_{\mathcal{A}} \blacktriangleright\blacktriangleright\ \bar{P}_\downarrow P_\downarrow,\quad P_\downarrow \mapsto_{\mathcal{A}} \blacktriangleright\blacktriangleright\ P_\downarrow P_\downarrow.$

- La règle $\delta(\bar{p}, \blacktriangleright\blacktriangleright) = (1,\bar{p}) \wedge (2,p)$ est remplacée par :
  $\bar{P} \mapsto_{\mathcal{A}} \blacktriangleright\blacktriangleright\ \bar{P}P,\quad \bar{P} \mapsto_{\mathcal{A}} \blacktriangleright\blacktriangleright\ \bar{P}_\downarrow P,\quad \bar{P} \mapsto_{\mathcal{A}} \blacktriangleright\blacktriangleright\ \bar{P}P_\downarrow,$
  $\bar{P} \mapsto_{\mathcal{A}} \blacktriangleright\blacktriangleright\ \bar{P}_\downarrow P_\downarrow,\quad \bar{P}_\downarrow \mapsto_{\mathcal{A}} \blacktriangleright\blacktriangleright\ \bar{P}_\downarrow P_\downarrow.$

- La règle $\delta(p, X\blacktriangleright) = \bigvee_{X \to_\Delta C,u} (2,p_u) \wedge \big[(1,\bar{p}_C) \vee ((2,p_\downarrow) \wedge (1,p_C))\big]$
  $= \bigvee_{X \to_\Delta C,u} \big[(1,\bar{p}_C) \wedge (2,p_u)\big] \vee \big[(1,p_C) \wedge (2,p_u) \wedge (2,p_\downarrow)\big]$ est remplacée par :

  Si $X \to_\Delta C, u$, alors
  $P \mapsto_{\mathcal{A}} X\blacktriangleright \bar{P}_C P_u,\quad P \mapsto_{\mathcal{A}} X\blacktriangleright \bar{P}_{C\downarrow} P_u,\quad P \mapsto_{\mathcal{A}} X\blacktriangleright \bar{P}_C P_{u\downarrow},$
  $P \mapsto_{\mathcal{A}} X\blacktriangleright P_C P_{u\downarrow},\quad P \mapsto_{\mathcal{A}} X\blacktriangleright \bar{P}_{C\downarrow} P_{u\downarrow},\quad P \mapsto_{\mathcal{A}} X\blacktriangleright P_{C\downarrow} P_{u\downarrow},$
  $P_\downarrow \mapsto_{\mathcal{A}} X\blacktriangleright \bar{P}_{C\downarrow} P_{u\downarrow},\quad P_\downarrow \mapsto_{\mathcal{A}} X\blacktriangleright P_{C\downarrow} P_{u\downarrow}.$

- La règle $\delta(\bar{p}, X\blacktriangleright) = \bigvee_{X \to_\Delta C,u} (2,p_u) \wedge (1,\bar{p}_C)$ si $X \in \mathcal{X}^!$ est remplacée par :
  Si $X \to_\Delta C, u$ et $X \in \mathcal{X}^!$, alors
  $\bar{P} \mapsto_{\mathcal{A}} X\blacktriangleright \bar{P}_C P_u,\quad \bar{P} \mapsto_{\mathcal{A}} X\blacktriangleright \bar{P}_{C,\downarrow} P_u,\quad \bar{P} \mapsto_{\mathcal{A}} X\blacktriangleright \bar{P}_C P_{u\downarrow},$
  $\bar{P} \mapsto_{\mathcal{A}} X\blacktriangleright \bar{P}_{C\downarrow} P_{u\downarrow},\quad \bar{P}_\downarrow \mapsto_{\mathcal{A}} X\blacktriangleright \bar{P}_{C\downarrow} P_{u\downarrow}.$

- La règle $\delta(p_t, f\ Nil) = \delta(\bar{p}_t, f Nil) = vrai$ si $t = Nil$ est remplacée par :
  $P_{Nil} \mapsto_{\mathcal{A}} f Nil, \quad P_{Nil\downarrow} \mapsto_{\mathcal{A}} f Nil, \quad \bar{P}_{Nil} \mapsto_{\mathcal{A}} f Nil, \quad \bar{P}_{Nil\downarrow} \mapsto_{\mathcal{A}} f Nil.$

- La règle $\delta(p_t, fX) = \delta(\bar{p}_t, fX) = vrai$ si $t = X$ est remplacée par :
  $P_X \mapsto_{\mathcal{A}} fX, \qquad \bar{P}_X \mapsto_{\mathcal{A}} fX.$

- La règle $\delta(p_{t_1 \blacktriangleright t_2}, f\ \blacktriangleright) = (2, p_{t_2}) \wedge \big[(1, \bar{p}_{t_1}) \vee ((2, p_{\downarrow}) \wedge (1, p_{t_1}))\big]$ est remplacée par :
  $P_{t_1 \blacktriangleright t_2} \mapsto_{\mathcal{A}} \blacktriangleright\blacktriangleright\ \bar{P}_{t_1} P_{t_2}, \qquad P_{t_1 \blacktriangleright t_2} \mapsto_{\mathcal{A}} \blacktriangleright\blacktriangleright\ \bar{P}_{t_1\downarrow} P_{t_2},$
  $P_{t_1 \blacktriangleright t_2} \mapsto_{\mathcal{A}} \blacktriangleright\blacktriangleright\ \bar{P}_{t_1} P_{t_2\downarrow}, \qquad P_{t_1 \blacktriangleright t_2} \mapsto_{\mathcal{A}} \blacktriangleright\blacktriangleright\ P_{t_1} P_{t_2\downarrow},$
  $P_{t_1 \blacktriangleright t_2} \mapsto_{\mathcal{A}} \blacktriangleright\blacktriangleright\ \bar{P}_{t_1\downarrow} P_{t_2\downarrow}, \qquad P_{t_1 \blacktriangleright t_2} \mapsto_{\mathcal{A}} \blacktriangleright\blacktriangleright\ P_{t_1\downarrow} P_{t_2\downarrow},$
  $P_{t_1 \blacktriangleright t_2\downarrow} \mapsto_{\mathcal{A}} \blacktriangleright\blacktriangleright\ \bar{P}_{t_1\downarrow} P_{t_2\downarrow}, \qquad P_{t_1 \blacktriangleright t_2\downarrow} \mapsto_{\mathcal{A}} \blacktriangleright\blacktriangleright\ P_{t_1\downarrow} P_{t_2\downarrow}.$

- La règle $\delta(\bar{p}_{t_1 \blacktriangleright t_2}, f\ \blacktriangleright) = (1, \bar{p}_{t_1}) \wedge (2, p_{t_2})$ est remplacée par :
  $\bar{P}_{t_1 \blacktriangleright t_2} \mapsto_{\mathcal{A}} \blacktriangleright\blacktriangleright\ \bar{P}_{t_1} P_{t_2}, \qquad \bar{P}_{t_1 \blacktriangleright t_2} \mapsto_{\mathcal{A}} \blacktriangleright\blacktriangleright\ \bar{P}_{t_1\downarrow} P_{t_2},$
  $\bar{P}_{t_1 \blacktriangleright t_2} \mapsto_{\mathcal{A}} \blacktriangleright\blacktriangleright\ \bar{P}_{t_1} Pt_2 \downarrow, \qquad \bar{P}_{t_1 \blacktriangleright t_2} \mapsto_{\mathcal{A}} \blacktriangleright\blacktriangleright\ \bar{P}_{t_1\downarrow} Pt_2 \downarrow,$
  $\bar{P}_{t_1 \blacktriangleright t_2\downarrow} \mapsto_{\mathcal{A}} \blacktriangleright\blacktriangleright\ \bar{P}_{t_1\downarrow} Pt_2 \downarrow.$

- La règle 4.13 est remplacée par :
  Si $X \rightarrow_{\Delta} C, u$, alors
  $P_X \mapsto_{\mathcal{A}}\ X \blacktriangleright \bar{P}_C P_u, \qquad P_X \mapsto_{\mathcal{A}}\ X \blacktriangleright \bar{P}_{C\downarrow} P_u, \qquad P_X \mapsto_{\mathcal{A}}\ X \blacktriangleright \bar{P}_C P_{u\downarrow},$
  $P_X \mapsto_{\mathcal{A}}\ X \blacktriangleright P_C P_{u\downarrow}, \qquad P_X \mapsto_{\mathcal{A}}\ X \blacktriangleright \bar{P}_{C\downarrow} P_{u\downarrow}, \qquad P_X \mapsto_{\mathcal{A}}\ X \blacktriangleright P_{C\downarrow} P_{u\downarrow},$
  $P_{X\downarrow} \mapsto_{\mathcal{A}}\ X \blacktriangleright \bar{P}_{C\downarrow} P_{u\downarrow}, \qquad P_{X\downarrow} \mapsto_{\mathcal{A}}\ X \blacktriangleright P_{C\downarrow} P_{u\downarrow}.$
- La règle 4.14 est remplacée par :
  Si $X \rightarrow_{\Delta} C, u$ et $X \in \mathcal{X}^!$, alors
  $\bar{P}_X \mapsto_{\mathcal{A}}\ X \blacktriangleright \bar{P}_C P_u, \qquad \bar{P}_X \mapsto_{\mathcal{A}}\ X \blacktriangleright \bar{P}_{C,\downarrow} P_u, \qquad \bar{P}_X \mapsto_{\mathcal{A}}\ X \blacktriangleright \bar{P}_C P_{u\downarrow},$
  $\bar{P}_X \mapsto_{\mathcal{A}}\ X \blacktriangleright \bar{P}_{C\downarrow} P_{u\downarrow}, \qquad \bar{P}_{X\downarrow} \mapsto_{\mathcal{A}}\ X \blacktriangleright \bar{P}_{C\downarrow} P_{u\downarrow}.$

Le lemme 4.44 découle directement du lemme 4.41. En effet, les règles de $\mapsto_{\mathcal{A}}$ reprennent exactement celles de δ en codant la règle 4.1 dans les autres règles. ⌟

Une conséquence du théorème 4.43 est que l'on peut calculer l'ensemble des prédécesseurs et l'ensemble des successeurs d'un ensemble.

Définition 4.45
Soient $(S, \Sigma, \rightarrow, s_0)$ un système de transitions étiquetées, s un état de S et S' un sous-ensemble de S.
*L'**ensemble des prédécesseurs** de v, noté $Pre^*(v)$ est l'ensemble $\{u \mid u \xrightarrow{*} v\}$.*
*L'**ensemble des successeurs** de v, noté $Post^*(v)$ est l'ensemble $\{u \mid v \xrightarrow{*} u\}$.*

Proposition 4.46 *Si L est un langage reconnaissable, $Pre^*(L)$ et $Post^*(L)$ sont également reconnaissables et les automates correspondants peuvent être obtenus en temps polynomial lorsqu'un automate pour L est connu.*

Démonstration. Comme $Pre^*(L) = 1^{\text{ère}}projection(\{u \times v \mid u \xrightarrow{*} v\} \cap T \times L)$ et $Post^*(L) = 2^{\text{ème}}projection(\{u \times v \mid u \xrightarrow{*} v\} \cap L \times T)$. Le théorème 4.31 permet de conclure. ⌟

4.4. Application : le model checking

Le problème du model-checking pour RPA est le suivant : étant donnés un terme RPA t et une formule logique φ, est-ce que t satisfait φ ?

Dans le cas de PA, D. Lugiez et Ph. Schnoebelen ont montré [LS00] que la décidabilité de l'accessibilité pour PA sans congruence entraînait la décidabilité du model-checking des logiques EF et LT pour PA. Dans ce paragraphe, nous allons montrer qu'il en est de même pour RPA : le théorème 4.43 nous permet de décider ce problème quand φ appartient à la logique EF ou à la logique LT.

4.4.1. Le model checking de la logique EF

La logique EF est un fragment de la logique CTL [CE81] dont le model checking est décidable pour l'algèbre PA [May97], alors qu'il ne l'est pas avec la logique CTL [EK95].

Pour définir la logique EF, nous commençons par considérer un ensemble de **propositions atomiques** $Prop = \{P_1, P_2, \ldots\}$, dont la sémantique est donnée par les ensembles de termes $Sol(P_1), Sol(P_2), \ldots$.

Nous ne considérons que des propositions P telles que $Sol(P)$ soit un langage d'arbres reconnaissable. Par exemple, P peut être la proposition « t peut faire une action a tout de suite » ou « il y a au moins deux occurrences de X dans t » etc.

Définition 4.47 *La **logique EF** a la syntaxe suivante :*

$$\varphi ::= P \mid \neg\varphi \mid \varphi \wedge \varphi' \mid \mathsf{EX}\varphi \mid \mathsf{EF}\varphi.$$

La sémantique de la logique EF est la suivante :

$$\begin{aligned}
t \models P &\overset{\text{def}}{\Leftrightarrow} t \in Sol(P),\\
t \models \neg\varphi &\overset{\text{def}}{\Leftrightarrow} t \not\models \varphi,\\
t \models \varphi \wedge \varphi' &\overset{\text{def}}{\Leftrightarrow} t \models \varphi \text{ et } t \models \varphi',\\
t \models \mathsf{EX}\varphi &\overset{\text{def}}{\Leftrightarrow} \text{ il existe } t' \text{ tel que } t' \models \varphi \text{ et } t \to t',\\
t \models \mathsf{EF}\varphi &\overset{\text{def}}{\Leftrightarrow} \text{ il existe } t' \text{ tel que } t' \models \varphi \text{ et } t \to^* t'.
\end{aligned}$$

Ceci explique que nous lisions $\mathsf{EX}\varphi$ comme « il est possible d'atteindre en une transition un terme tel que φ » et $\mathsf{EF}\varphi$ comme « il est possible d'atteindre (via une suite de transitions) un terme tel que φ ».

Définition 4.48 *Soit φ une formule de la logique EF, nous notons $Sol(\varphi)$ l'ensemble $\{t \in T \mid t \models \varphi\}$.*

Une autre façon de donner la sémantique de EF est la suivante :

Proposition 4.49 *Si φ est une formule de la logique EF, alors*

$$\begin{array}{rcl} Sol(\neg\varphi) & = & T \setminus Sol(\varphi), \\ Sol(\varphi \wedge \varphi') & = & Sol(\varphi) \cap Sol(\varphi'), \\ Sol(\mathsf{EX}\varphi) & = & Pre(Sol(\varphi)), \\ Sol(\mathsf{EF}\varphi) & = & Pre^*(Sol(\varphi)). \end{array}$$

Une conséquence de la propriété 4.46 est le théorème suivant :

Théorème 4.50 *Pour toute formule de la logique EF φ, $Sol(\varphi)$ est un langage d'arbres reconnaissable.*

Il s'ensuit que :

Conséquence 4.51 *Le problème du model-checking de la logique EF est décidable.*

Remarque 4.52 Le fait de calculer $Sol(\varphi)$ est plus général que de dire si un terme t donné lui appartient.

4.4.2. Le model checking de la logique de transitions

Pour décider du problème du model-checking de la logique EF, nous n'avons eu besoin que de la reconnaissance effective de $Pre^*(L)$ quand L est reconnaissable. La reconnaissabilité de $\xrightarrow{*}$ nous permet plus : elle donne un algorithme de model-checking simple pour la logique de transitions toute entière.

Définition 4.53 *Soit Δ une déclaration RPA donnée. La* ***logique de transitions du premier ordre*** *LT est la logique du premier ordre sur les variables de termes RPA notés u, v, etc. les prédicats binaires $=$ et $\rightarrow^*$ et toutes les relations reconnaissables.*

Exemple 4.54 La formule suivante :

$$\forall t_1, t_2, t_3 \left((t_1 \xrightarrow{*} t_2 \wedge t_1 \xrightarrow{*} t_3) \Rightarrow \exists t_4 (t_2 \xrightarrow{*} t_4 \wedge t_3 \xrightarrow{*} t_4)\right)$$

est une formule LT qui énonce la confluence de $\xrightarrow{*}$.

Définition 4.55 *Soit φ une formule de la logique LT ayant n variables libres, nous notons $Sol(\varphi)$ l'ensemble $\{(t_1, \ldots, t_n) \in T^n \mid (t_1, \ldots, t_n) \models \varphi\}$.*

Une conséquence du théorème 4.43 et du théorème 4.31 est le théorème suivant :

Théorème 4.56 *Pour toute formule de la logique LT φ, $Sol(\varphi)$ est une relation reconnaissable.*

Remarque 4.57 Il est possible d'utiliser simultanément plusieurs déclarations Δ différentes, simplement en utilisant des prédicats d'atteignabilité indexés. Par exemple, nous énonçons que tous les processus atteignables depuis X via Δ le sont aussi en utilisant Δ_1 puis Δ_2, au moyen de

$$\forall u \left(X \xrightarrow{*}_{\Delta} u \Rightarrow \exists v \ (X \xrightarrow{*}_{\Delta_1} v \wedge v \xrightarrow{*}_{\Delta_2} u) \right).$$

Cette formule est décidable, c'est encore une conséquence des théorèmes 4.43 et 4.31.

5. Le codage des RPPS dans une algèbre RPA

Sommaire

Dans ce chapitre, nous montrons comment coder un schéma de programme récursif-parallèle (RPPS) dans une algèbre RPA. Nous illustrons cette construction sur l'exemple du fast-food, introduit au chapitre 3. Enfin, nous énonçons formellement comment ce codage préserve fidèlement le comportement du schéma codé.

5.1. Les termes RPA vus comme des peignes

Dans notre codage des schémas RPPS par une algèbre RPA, les états emboîtés sont représentés par des termes RPA. Or les états emboîtés représentent un ensemble hiérarchisé d'états de contrôle, cette hiérarchie est codée dans les termes RPA ce qui explique que le côté gauche et le côté droit des termes RPA ne jouent pas le même rôle. C'est pourquoi il est plus naturel de lire les termes RPA comme des peignes gauches avec un élément de *Atomes* tout à gauche et une liste de sous-termes à droite de l'arête.

Exemple 5.1
Nous reprenons le terme $t = (X_1 \blacktriangleright X_2) \blacktriangleright (((X_3 \blacktriangleright X_4) \blacktriangleright \mathit{Nil}) \blacktriangleright (X_5 \blacktriangleright X_6))$

(vu sous forme d'arbre dans la figure 5.1). Il s'écrit :

$$X_1 \blacktriangleright^2 (X_2, (X_3 \blacktriangleright^3 (X_4, \mathit{Nil}, (X_5 \blacktriangleright X_6))))$$

lorsqu'il est vu comme un peigne gauche. Cette écriture est expliquée par la figure 5.2.

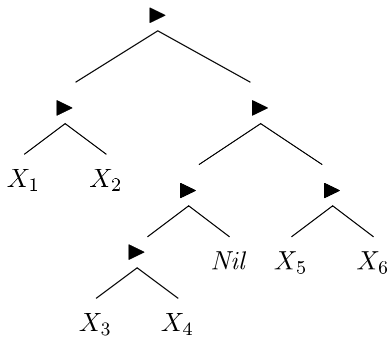


FIG. 5.1. – Le terme t représenté sous forme d'arbre.

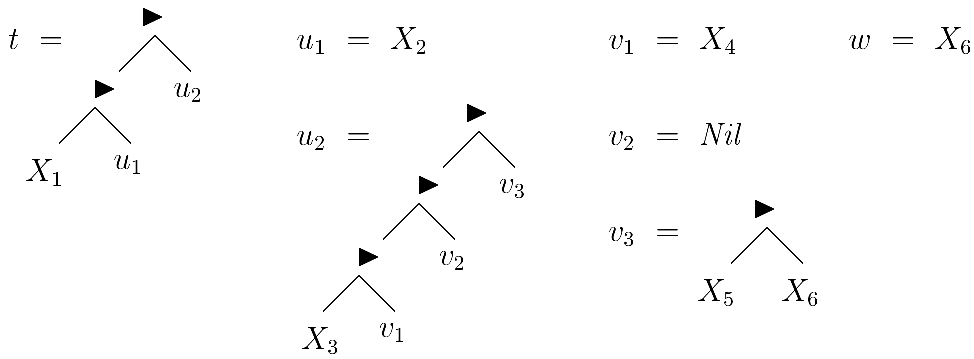


FIG. 5.2. – Le terme t vu comme un peigne.

Le terme u est écrit sous forme d'un peigne gauche quand il est écrit de la façon suivante :

$$(\ldots((C \blacktriangleright u_1) \blacktriangleright u_2) \ldots \blacktriangleright u_n)$$

où C est un élément de *Atomes* et $u_1, \ldots, u_n$ sont des termes RPA. Nous utilisons alors l'abréviation $C \blacktriangleright^n (u_1, \ldots, u_n)$ définie inductivement par :

$$C \blacktriangleright^0 () = C,$$
$$C \blacktriangleright^n (u_1, \ldots, u_n) = \big(C \blacktriangleright^{n-1} (u_1, \ldots, u_{n-1})\big) \blacktriangleright u_n.$$

Cette écriture permet de faire ressortir notre hiérarchie : C est alors le père et $u_1, \ldots, u_n$ ses fils.

Remarque 5.2 Nous utiliserons surtout l'abréviation « $\blacktriangleright^n$ » avec un élément de *Atomes* à gauche du symbole $\blacktriangleright$. En effet, cette abréviation met en évidence un père et ses fils ; elle n'a plus beaucoup de sens quand il y a un terme composé à gauche du symbole $\blacktriangleright$.

5.2. Codage d'un schéma RPPS par une algèbre RPA

Dans cette section, nous montrons comment coder un schéma RPPS par une algèbre RPA dans le cas général. Nous illustrons ensuite cette construction sur l'exemple du fast-food (chapitre 3).

5.2.1. Le codage

Nous considérons un schéma de programme récursif-parallèle donné $G = (Q, q_{init}, \Lambda)$ (défini au chapitre 3). Nous lui associons une algèbre RPA en définissant les éléments suivants :

1. **Les actions.**
   Les actions de l'algèbre RPA associée à G sont les mêmes que celles de G, donc $\Sigma_G = \tilde{A}$.
2. **Les constantes de processus.**
   Nous associons à G un ensemble $\mathcal{X}_G$ de constantes de processus de la façon suivante : pour tout $q \in Q$, $X_q \in \mathcal{X}_G$. De plus, si $\Lambda(q) = (\texttt{wait}, q')$, $X_q \in \mathcal{X}^?_G$ sinon $X_q \in \mathcal{X}^!_G$.
3. **La déclaration RPA.**
   Nous associons à G la déclaration Δ_G, de la façon suivante : pour tout $q \in Q$,

$$\begin{aligned}
\textbf{action :}\ & \text{si } \Lambda(q) = (a, q'),\ \Lambda(q) = (a, q', q'') \text{ ou } \Lambda(q) = (a, q'', q'), \\
& \text{alors } X_q \xrightarrow{a} X_{q'}, \mathit{Nil} \text{ est une règle de } \Delta_G, && \text{(Da)} \\
\textbf{end :}\ & \text{si } \Lambda(q) = \texttt{end}, \\
& \text{alors } X_q \xrightarrow{\texttt{end}} \mathit{Nil}, \mathit{Nil} \text{ appartient à } \Delta_G, && \text{(De)} \\
\textbf{call :}\ & \text{si } \Lambda(q) = (\texttt{call}, q', q''), \\
& \text{alors } X_q \xrightarrow{\texttt{call}} X_{q'}, X_{q''} \text{ appartient à } \Delta_G, && \text{(Dc)} \\
\textbf{wait :}\ & \text{si } \Lambda(q) = (\texttt{wait}, q'), \\
& \text{alors } X_q \xrightarrow{\texttt{wait}} X_{q'}, \mathit{Nil} \text{ appartient à } \Delta_G. && \text{(Dw)}
\end{aligned}$$

Lorsqu'il n'y a pas d'ambiguïté, nous omettons les indices G pour plus de lisibilité.

Remarque 5.3 Contrairement à Λ qui est une application, Δ est une relation de $\mathcal{X}$ dans $\tilde{A} \times Const \times Const$. En particulier, quand $\Lambda(q) = (a, q', q'')$, ce qui correspond au codage d'un choix, cette relation est non déterministe (c'est le cas pour q_{12} qui code l'interrogation sur les sandwichs dans l'exemple du fast-food du chapitre 3).

Cependant, comme Λ est une application, une constante de processus est toujours associée à la même action. Cela a donc un sens de partionner l'ensemble des constantes de processus en fonction de l'action qui leur est associée.

La fonction $\mathcal{T}$ permet d'associer à un état emboîté s de $M(G)$ un unique terme $\mathcal{T}(s)$.

Définition 5.4 *La fonction $\mathcal{T}$ est définie inductivement par :*

$$\mathcal{T}(\{(q_1, s_1), \ldots, (q_n, s_n)\}) \stackrel{def}{=} Nil \blacktriangleright^n (X_{q_1} \blacktriangleright \mathcal{T}(s_1), \ldots, X_{q_n} \blacktriangleright \mathcal{T}(s_n)) \qquad \text{(T)}$$

Cette fonction explique la notation sous forme de peigne gauche et montre que C est le père dans le terme $C \blacktriangleright^n (u_1, \ldots, u_n)$ et $u_1, \ldots, u_n$ les fils.

Nous comprenons alors mieux le rôle des étiquettes ! et ?. Une étiquette ? signale que l'action faite est un `wait` et, donc, que nous ne pouvons pas ajouter n'importe quel fils. Or nous rajoutons un fils v à un terme u en considérant le terme $u \blacktriangleright v$; ceci explique donc la règle R6 et, par suite, la règle R5 qui définissent la sémantique de RPA dans le paragraphe 4.1.2 du chapitre 4.

Lorsque $v \blacktriangleright u$ est un sous-terme d'un terme t, les seuls fils de u dans t sont les fils de u dans u, ceci explique les règles R3 et R4. En effet, si u peut faire une action, t le peut aussi.

5.2.2. Illustration sur l'exemple du fast-food

Dans le codage du schéma de la figure 3.5, nous utilisons l'abréviation X_i pour X_{q_i}. Alors, $\mathcal{X}^?$ contient les constantes de processus X_3 et X_8 et $\mathcal{X}^!$ toutes les autres. La déclaration RPA associée est la suivante :

$$\Delta \stackrel{\text{def}}{=} \left\{ \begin{array}{lllllll}
X_0 \xrightarrow{a} X_1, Nil & ; & X_1 \xrightarrow{\texttt{call}} X_2, X_7 & ; & X_2 \xrightarrow{b} X_3, Nil & ; \\
X_3 \xrightarrow{c} X_4, Nil & ; & X_4 \xrightarrow{\texttt{wait}} X_5, Nil & ; & X_5 \xrightarrow{\texttt{call}} X_6, X_0 & ; \\
X_6 \xrightarrow{\textsf{end}} Nil, Nil & ; & X_7 \xrightarrow{\texttt{call}} X_8, X_{16} & ; & X_8 \xrightarrow{d} X_9, Nil & ; \\
X_9 \xrightarrow{\texttt{wait}} X_{10}, Nil & ; & X_{10} \xrightarrow{e} X_{11}, Nil & ; & X_{11} \xrightarrow{f} X_{12}, Nil & ; \\
X_{12} \xrightarrow{g} X_{13}, Nil & ; & X_{12} \xrightarrow{g} X_{18}, Nil & ; & X_{13} \xrightarrow{h} X_{14}, Nil & ; \\
X_{14} \xrightarrow{i} X_{15}, Nil & ; & X_{14} \xrightarrow{i} X_{19}, Nil & ; & X_{15} \xrightarrow{\textsf{end}} Nil, Nil & ; \\
X_{16} \xrightarrow{j} X_{17}, Nil & ; & X_{17} \xrightarrow{\textsf{end}} Nil, Nil & ; & X_{19} \xrightarrow{\texttt{call}} X_{15}, X_{22} & ; \\
X_{20} \xrightarrow{k} X_{21}, Nil & ; & X_{21} \xrightarrow{\textsf{end}} Nil, Nil & ; & X_{22} \xrightarrow{l} X_{23}, Nil & ; \\
X_{23} \xrightarrow{\textsf{end}} Nil, Nil & . & & & &
\end{array} \right\}$$

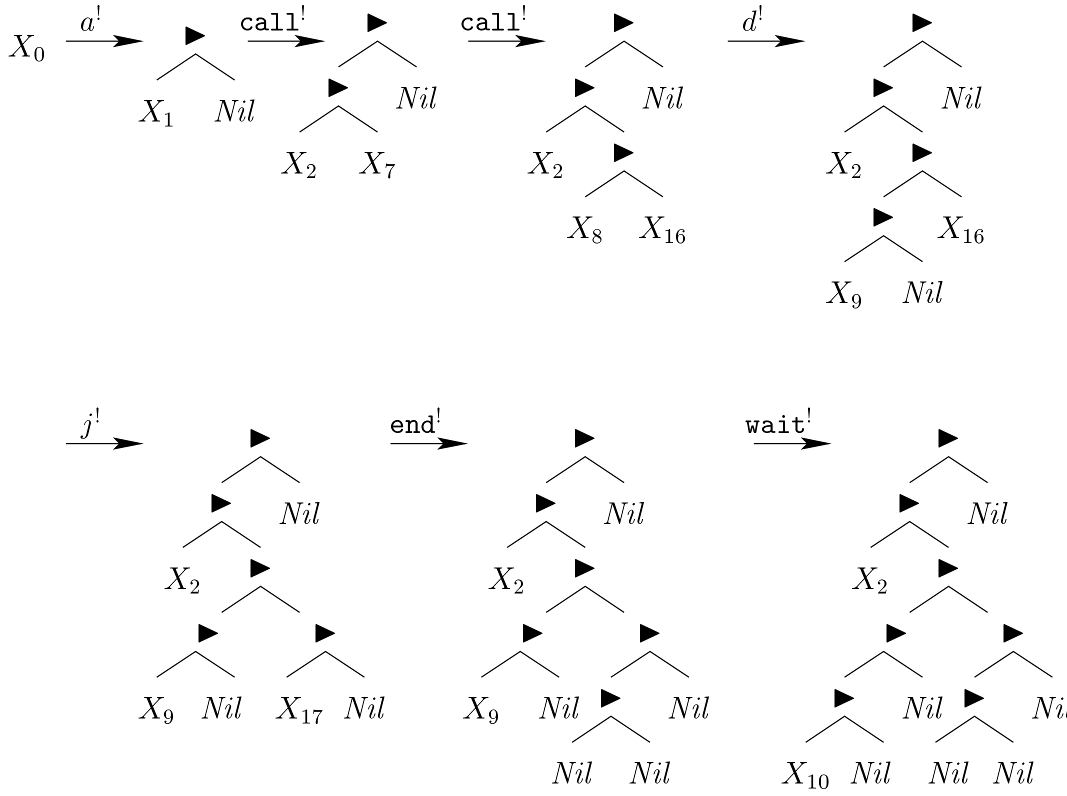


FIG. 5.3. – L'exécution associée à celle de la figure 3.7.

La figure 5.3 présente l'exécution de l'algèbre définie par Δ correspondant à l'exécution proposée par la figure 3.7.

Nous remarquons que la taille du terme grossit à chaque transition de l'exécution de la figure 5.3 contrairement à la taille des états emboîtés. Ceci n'est qu'une illustration du lemme 4.9. Cependant, le nombre d'éléments de Q contenus dans un terme augmente peu (il n'augmente qu'avec les `call`), ce sont surtout des *Nil* qui sont ajoutés.

Quand nous regardons les termes comme des peignes gauches, nous retrouvons tout à gauche l'état qui est au sommet de l'état emboîté correspondant, puis tout à gauche des peignes fils, ses fils et ainsi de suite. Ainsi le cinquième terme proposé $X_2 \blacktriangleright^2 (X_9 \blacktriangleright^2 (Nil, X_{16}), Nil)$ est formé comme $(X_2, \{(X_9, \{X_{16}\})\})$ d'un père X_2 qui a un fils X_9 qui a lui-même un fils X_{16}.

Nous remarquons enfin que lorsque X_9 fait une action `wait`, sa descendance n'est composée que de *Nil*.

5.3. Le couple abstraction-concrétisation

Jusqu'à présent, nous avons vu comment coder un schéma RPPS en une algèbre RPA. La fonction $\mathcal{T}$ permet d'exprimer un état emboîté en un terme RPA. Nous allons voir comment décoder un terme RPA.

Définition 5.5 *Soit G un schéma RPPS et (Σ, Δ) l'algèbre RPA associée. La fonction de concrétisation, notée $\mathcal{S}$, associe à un terme RPA t un état emboîté $\mathcal{S}(t)$. Elle est définie inductivement par :*

$$\mathcal{S}\left(Nil \blacktriangleright^n (u_1, \ldots, u_n)\right) \stackrel{def}{=} \mathcal{S}(u_1) + \cdots + \mathcal{S}(u_n) \tag{S1}$$

$$\mathcal{S}\left(X_q \blacktriangleright^n (u_1, \ldots, u_n)\right) \stackrel{def}{=} (X_q, \mathcal{S}(u_1) + \cdots + \mathcal{S}(u_n)) \tag{S2}$$

La fonction $\mathcal{S}$ est bien une concrétisation de $\mathcal{T}$, en effet elle vérifie la propriété suivante :

Lemme 5.6 *Pour tout $s \in M(G)$, $\mathcal{S}(\mathcal{T}(s)) = s$.*

Démonstration. Nous allons montrer la propriété par induction sur s.

- Si $s = \emptyset$, alors $\mathcal{T}(s) = Nil$ et $\mathcal{S}(\mathcal{T}(s)) = \emptyset = s$.
- Si $s = \{(q_1, s_1), \ldots, (q_n, s_n)\}$, alors $\mathcal{T}(s) = Nil \blacktriangleright^n (X_{q_1} \blacktriangleright \mathcal{T}(s_1), \ldots, X_{q_n} \blacktriangleright \mathcal{T}(s_n))$ d'après l'équation (T), donc $\mathcal{S}(\mathcal{T}(s)) = \mathcal{S}(X_{q_1} \blacktriangleright \mathcal{T}(s_1)) + \cdots + \mathcal{S}(X_{q_n} \blacktriangleright \mathcal{T}(s_n))$ d'après l'équation (S1), d'où $\mathcal{S}(\mathcal{T}(s)) = (q_1, \mathcal{T}(s_1)) + \cdots + (q_n, \mathcal{T}(s_n))$ d'après l'équation (S2). Les multi-ensembles s_1,...,s_n sont plus petits que s donc par hypothèse d'induction, $\mathcal{S}(\mathcal{T}(s_1)) = s_1$, ..., $\mathcal{S}(\mathcal{T}(s_n)) = s_n$. Il s'ensuit que $\mathcal{S}(\mathcal{T}(s)) = (q_1, s_1) + \cdots + (q_n, s_n) = \{(q_1, s_1), \ldots, (q_n, s_n)\}$.

$\lrcorner$

Par contre, nous n'avons pas la propriété $\mathcal{T}(\mathcal{S}(t)) = t$ pour tout t. Par exemple, $\mathcal{T}(\mathcal{S}(X_q)) = Nil \blacktriangleright X_q$.

De plus, une infinité de termes ont le même état emboîté comme image par $\mathcal{S}$, par exemple, X_q, $Nil \blacktriangleright X_q$, $X_q \blacktriangleright Nil$, $Nil \blacktriangleright^2 (X_q, Nil)$ sont associés à q. Il est donc naturel d'introduire l'équivalence $\equiv_{\mathcal{S}}$ qui regroupe dans une même classe tous les termes qui représentent le même schéma RPPS :

Définition 5.7 *Soient t et u deux termes RPA, $t \equiv_{\mathcal{S}} u$ si $\mathcal{S}(t) = \mathcal{S}(u)$. Nous notons $[u]$ la classe d'équivalence de u et $T_{\equiv_{\mathcal{S}}}$ l'ensemble des classes d'équivalence de T.*

Remarque 5.8 Nous remarquons que $\equiv_{\mathcal{S}}$ n'est pas une congruence ; en effet, $(Nil \blacktriangleright u) \equiv_{\mathcal{S}} u$ tandis que $(Nil \blacktriangleright u) \blacktriangleright v \not\equiv_{\mathcal{S}} u \blacktriangleright v$. Par contre, nous avons vu que $(Nil \blacktriangleright u) \blacktriangleright v$ a pour abbréviation $Nil \blacktriangleright^2 (u, v)$ donc $(Nil \blacktriangleright u) \blacktriangleright v \equiv_{\mathcal{S}} Nil \blacktriangleright^2 (u, v)$.

Lemme 5.9 $[\![Nil]\!] = \{t \mid \mathit{Constante}(t) = \emptyset\}$.

Démonstration. $\mathcal{S}(\mathit{Nil}) = \emptyset$. Nous avons par induction sur t que $\mathit{Constante}(t) = \emptyset$ si et seulement si $\mathcal{S}(t) = \emptyset$. ⌟

5.4. Correction du codage

Il nous est maintenant possible de lier les transitions entre les états emboîtés de RPPS et les transitions entre les termes RPA associés. Il suffit, pour cela, d'abstraire les étiquettes ! et ? des transitions RPA qui ne servent qu'à définir ces transitions par des règles de SOS. Pour ce faire nous notons $u \xrightarrow{\alpha} t$ quand $u \xrightarrow{\alpha^\varepsilon} t$ pour un certain $\varepsilon \in \{!, ?\}$. Nous avons alors la proposition suivante :

Proposition 5.10

1. *Pour tous u, v de T et α de $\tilde{A}$, si $u \xrightarrow{\alpha} v$, alors $\mathcal{S}(u) \xrightarrow{\alpha} \mathcal{S}(v)$.*
2. *Pour tous s, s' de $M(G)$ et α de $\tilde{A}$, si $s \xrightarrow{\alpha} s'$, alors $\mathcal{T}(s) \xrightarrow{\alpha} u$ pour un certain $u \in T$ tel que $\mathcal{S}(u) = s'$.*

Cette proposition montre que, modulo l'abstraction faite entre Act et $\tilde{A}$ qui associe α^ε à α, $\mathcal{S}$ est une bisimulation entre le système de transitions RPA généré par Δ et le système de transitions $\mathcal{M}$ que nous voulons analyser.

La preuve de cette proposition est longue. Nous commençons par énoncer quelques lemmes techniques.

5.4.1. Les propriétés des transitions des peignes

Les lemmes qui suivent présentent les propriétés des transitions RPA quand les termes sont écrits sous forme de peignes.

Lemme 5.11 *Si $X \xrightarrow{\alpha^!} C \blacktriangleright w$, alors $X \blacktriangleright^n (t_1, \ldots, t_n) \xrightarrow{\alpha^!} C \blacktriangleright^{n+1} (w, t_1, \ldots, t_n)$.*

Démonstration. La règle R5 entraîne $X \blacktriangleright t_1 \xrightarrow{\alpha^!} C \blacktriangleright^2 (w, t_1)$ et, ensuite, pour tout $i \leq n$, $X \blacktriangleright^i (t_1, \ldots, t_i) = (X \blacktriangleright^{i-1} (t_1, \ldots, t_{i-1})) \blacktriangleright t_i \xrightarrow{\alpha^!} (C \blacktriangleright^i (w, t_1, \ldots, t_{i-1})) \blacktriangleright t_i) = C \blacktriangleright^{i+1} (w, t_1, \ldots, t_i)$. ⌟

Lemme 5.12 *Si $X \xrightarrow{\alpha^?} C \blacktriangleright w$ et si pour tout i, $\mathit{Constante}(t_i) = \emptyset$, alors $X \blacktriangleright^n (t_1, \ldots, t_n) \xrightarrow{\alpha^?} C \blacktriangleright^{n+1} (w, t_1, \ldots, t_n)$.*

Démonstration. La règle R6 entraîne $X \blacktriangleright t_1 \xrightarrow{\alpha^?} C \blacktriangleright^2 (w, t_1)$ et, ensuite, pour tout $i \leq n$, $X \blacktriangleright^i (t_1, \ldots, t_i) = (X \blacktriangleright^{i-1} (t_1, \ldots, t_{i-1})) \blacktriangleright t_i \xrightarrow{\alpha^?} (C \blacktriangleright^i (w, t_1, \ldots, t_{i-1})) \blacktriangleright t_i) = C \blacktriangleright^{i+1} (w, t_1, \ldots, t_i)$. ⌟

Lemme 5.13 *Si* $t_j \xrightarrow{\alpha^\varepsilon} t'_j$ *avec* $\varepsilon \in \{?, !\}$*, alors* $C \blacktriangleright^n (t_1, \ldots, t_n) \xrightarrow{\alpha^!} C \blacktriangleright^n (t_1, \ldots, t_{j-1}, t'_j, t_{j+1}, \ldots, t_n)$.

Démonstration. Si $t_j \xrightarrow{\alpha^\varepsilon} t'_j$ avec $\varepsilon \in \{?, !\}$, alors $C \blacktriangleright^j (t_1, \ldots, t_j) = (C \blacktriangleright^{j-1} (t_1, \ldots, t_{j-1})) \blacktriangleright t_j \xrightarrow{\alpha^!} (C \blacktriangleright^{j-1} (t_1, \ldots, t_{j-1})) \blacktriangleright t'_j = C \blacktriangleright^j (t_1, \ldots, t'_j)$ en raison de la règle R3 si $\varepsilon =!$ et de la règle R4 si $\varepsilon =?$.

Ensuite, pour tout $j < i \leq n$, la règle R5 entraîne $C \blacktriangleright^i (t_1, \ldots, t_i) = (C \blacktriangleright^{i-1} (t_1, \ldots, t_{i-1})) \blacktriangleright t_i \xrightarrow{\alpha^!} (C \blacktriangleright^{i-1} (t_1, \ldots, t_{j-1}, t'_j, t_{j+1}, \ldots, t_{i-1})) \blacktriangleright t_i) = C \blacktriangleright^i (t_1, \ldots, t_{j-1}, t'_j, t_{j+1}, \ldots, t_i)$. ⌟

La réciproque des trois précédents lemmes peut s'énoncer ainsi :

Lemme 5.14 *Si* $C \blacktriangleright^n (t_1, \ldots, t_n) \xrightarrow{\alpha^\varepsilon} v$ *avec* $\varepsilon \in \{?, !\}$*, alors*

Soit (1) $C \in \mathcal{X}^!$, $C \xrightarrow{\alpha^!} C' \blacktriangleright w$ *et* $C \blacktriangleright^n (t_1, \ldots, t_n) \xrightarrow{\alpha^!} C' \blacktriangleright^{n+1} (w, t_1, \ldots, t_n)$*,*

Soit (2) $C \in \mathcal{X}^?$, $C \xrightarrow{\alpha^?} C' \blacktriangleright w$*,* $State(t_i) = \emptyset$ *pour tout* i *et* $C \blacktriangleright^n (t_1, \ldots, t_n) \xrightarrow{\alpha^?} C' \blacktriangleright^{n+1} (w, t_1, \ldots, t_n)$*,*

Soit (3) $t_j \xrightarrow{\alpha^{\varepsilon'}} t'_j$ *avec* $\varepsilon' \in \{?, !\}$ *et* $C \blacktriangleright^n (t_1, \ldots, t_n) \xrightarrow{\alpha^!} C \blacktriangleright^n (t_1, \ldots, t_{j-1}, t'_j, t_{j+1}, \ldots, t_n)$*.*

Démonstration. Supposons que $C \blacktriangleright^n (t_1, \ldots, t_n) \xrightarrow{\alpha^\varepsilon} v$ avec $\varepsilon \in \{?, !\}$, nous allons montrer par récursivité sur n que alors (1), (2) ou (3) est vérifiée.

- $\mathbf{n = 0}$:

  Si $C \in \mathcal{X}^!$, alors d'après la règle R1, il existe $C \xrightarrow{\alpha}_\Delta C', w \in \Delta$, $\varepsilon =!$ et $v = C' \blacktriangleright w$. Nous avons donc la propriété (1).

  Si $C \in \mathcal{X}^?$, alors d'après la règle R2, il existe $C \xrightarrow{\alpha}_\Delta C', w \in \Delta$, $\varepsilon =?$ et $v = C' \blacktriangleright w$. Nous avons donc la propriété (2).

- $\mathbf{n > 0}$:

  Nous supposons que la propriété est vraie pour $n - 1$. La transition $C \blacktriangleright^n (t_1, \ldots, t_n) \xrightarrow{\alpha^\varepsilon} v$ a été obtenue en appliquant comme dernière règle l'une des règles R3, R4, R5 ou R6.

  - Règles R3-R4 :

    Alors $\varepsilon =!$, $v = C \blacktriangleright^n (t_1, \ldots, t'_n)$ et $t_n \xrightarrow{\alpha^{\varepsilon'}} t'_n$ avec $\varepsilon' \in \{?, !\}$. Nous avons donc la propriété (3).

  - Règle R5 :

    Alors $\varepsilon =!$, $v = v' \blacktriangleright t_n$ et $C \blacktriangleright^{n-1} (t_1, \ldots, t_{n-1}) \xrightarrow{\alpha^!} v'$. Nous pouvons appliquer l'hypothèse de récurrence, nous avons alors soit $C \in \mathcal{X}^!$ et

$C \xrightarrow{\alpha^!} C' \blacktriangleright w$, soit $t_j \xrightarrow{\alpha^{\varepsilon'}} t'_j$ avec $\varepsilon' \in \{?, !\}$. Le cas (2) est éliminé car l'étiquette de la transition $C \blacktriangleright^{n-1} (t_1, \dots, t_{n-1}) \xrightarrow{\alpha^!} v'$ est !.

Si $C \in \mathcal{X}^!$ et $C \xrightarrow{\alpha^!} C' \blacktriangleright w$, alors le lemme 5.11 entraîne que $C \blacktriangleright^n (t_1, \dots, t_n) \xrightarrow{\alpha^!} C' \blacktriangleright^{n+1} (w, t_1, \dots, t_n)$ donc nous avons la propriété (1).

Si $t_j \xrightarrow{\alpha^{\varepsilon'}} t'_j$ avec $\varepsilon' \in \{?, !\}$, alors le lemme 5.13 entraîne que $C \blacktriangleright^n (t_1, \dots, t_n) \xrightarrow{\alpha^!} C \blacktriangleright^n (t_1, \dots, t_{j-1}, t'_j, t_{j+1}, \dots, t_n)$ donc nous avons la propriété (3).

- Règle R6 :

  Alors $\varepsilon =?$, $v = v' \blacktriangleright t_n$, $State(t_n) = \emptyset$ et $C \blacktriangleright^{n-1} (t_1, \dots, t_{n-1}) \xrightarrow{\alpha^?} v'$. Nous pouvons appliquer l'hypothèse de récurrence, nous avons $C \in \mathcal{X}^?$ et $C \xrightarrow{\alpha^?} C' \blacktriangleright w$ et $State(t_i) = \emptyset$ pour tout $i \leq n$. Les autres cas sont éliminés car l'étiquette de la transition $C \blacktriangleright^{n-1} (t_1, \dots, t_{n-1}) \xrightarrow{\alpha^?} v'$ est ?.

  Le lemme 5.12 entraîne que $C \blacktriangleright^n (t_1, \dots, t_n) \xrightarrow{\alpha^?} C' \blacktriangleright^{n+1} (w, t_1, \dots, t_n)$ donc nous avons la propriété (2).

$\lrcorner$

Ce lemme montre que le seul cas où un peigne $C \blacktriangleright^n (t_1, \dots, t_n)$ fait une action étiquetée ? est celui où le père C appartient à $\mathcal{X}^?$ et où il est à l'origine de cette transition.

5.4.2. Première partie de la proposition 5.10

Soient u et v appartenant à T et α appartenant à $\tilde{A}$, nous supposons que $u \xrightarrow{\alpha} v$. Nous allons montrer par induction sur u que $\mathcal{S}(u) \xrightarrow{a} \mathcal{S}(v)$.

u peut s'écrire $C \blacktriangleright^n (t_1, \dots, t_n)$ avec $C \in Atomes$ et $t_i \in T$. D'après le lemme 5.14,

Soit (1) $C \in \mathcal{X}^!$, $C \xrightarrow{\alpha^!} C' \blacktriangleright w$ et $v = C' \blacktriangleright^{n+1} (w, t_1, \dots, t_n)$,

Soit (2) $C \in \mathcal{X}^?$, $C \xrightarrow{\alpha^?} C' \blacktriangleright w$, $State(t_i) = \emptyset$ pour tout i et $v = C' \blacktriangleright^{n+1} (w, t_1, \dots, t_n)$,

Soit (3) $t_j \xrightarrow{\alpha} t'_j$ et $v = C \blacktriangleright^n (t_1, \dots, t_{j-1}, t'_j, t_{j+1}, \dots, t_n)$.

- Cas (1) :

  $C \in \mathcal{X}^!$ donc $C = X_q$ avec $q \in Q$. De plus, $C \xrightarrow{\alpha^!} C' \blacktriangleright w$ par conséquent, d'après la règle R1, $C \xrightarrow{\alpha^!} C', w \in \Delta$. Il y a alors trois cas : $\alpha = a$, $\alpha = \texttt{end}$ et $\alpha = \texttt{call}$. Nous n'avons pas $\alpha = \texttt{wait}$ car $C \in \mathcal{X}^!$.

- Si $\alpha = a$:

  D'après la règle Da qui définit Δ, $C' \in \mathcal{X}$ donc $C' = X_{q'}$, $w = \mathit{Nil}$ et $L(q) = (a, q')$ ou $L(q) = (a, q', q'')$ ou $L(q) = (a, q'', q')$. Donc d'après la règle Ga qui définit les transitions RPPS,

  $$(q, \mathcal{S}(t_1) + \cdots + \mathcal{S}(t_n)) \xrightarrow{a} (q', \mathcal{S}(t_1) + \cdots + \mathcal{S}(t_n)).$$

  Or $\mathcal{S}(u) = (q, \mathcal{S}(t_1) + \cdots + \mathcal{S}(t_n))$ et $\mathcal{S}(v) = (q', \mathcal{S}(t_1) + \cdots + \mathcal{S}(t_n))$ d'après l'équation S2. Il s'ensuit que $\mathcal{S}(u) \xrightarrow{a} \mathcal{S}(v)$.

- Si $\alpha = \texttt{end}$:

  D'après la règle De, $C' = \mathit{Nil}$, $w = \mathit{Nil}$ et $L(q) = \texttt{end}$. Donc d'après la règle Ge, $(q, \mathcal{S}(t_1) + \cdots + \mathcal{S}(t_n)) \xrightarrow{\texttt{end}} \mathcal{S}(t_1) + \cdots + \mathcal{S}(t_n)$. Or $\mathcal{S}(v) = \mathcal{S}(t_1) + \cdots + \mathcal{S}(t_n)$. Il s'ensuit que $\mathcal{S}(u) \xrightarrow{\texttt{end}} \mathcal{S}(v)$.

- Si $\alpha = \texttt{call}$:

  D'après la règle Dc, $C' \in \mathcal{X}$ donc $C' = X_{q'}$, $w \in \mathcal{X}$ donc $w = X_{q''}$, et $L(q) = (\texttt{call}, q', q'')$. Donc d'après la règle Gc, $(q, \mathcal{S}(t_1) + \cdots + \mathcal{S}(t_n)) \xrightarrow{\texttt{call}} (q', \mathcal{S}(t_1) + \cdots + \mathcal{S}(t_n) + (q'', \emptyset))$. Or $\mathcal{S}(v) = (q', (q'', \emptyset) + \mathcal{S}(t_1) + \cdots + \mathcal{S}(t_n)) = (q', \mathcal{S}(t_1) + \cdots + \mathcal{S}(t_n) + (q'', \emptyset))$. Il s'ensuit que $\mathcal{S}(u) \xrightarrow{\texttt{call}} \mathcal{S}(v)$.

- Cas (2) :

  $C \in \mathcal{X}^?$ donc $C = X_q$ avec $q \in Q$. De plus, $C \xrightarrow{\alpha^?} C' \blacktriangleright w$ par conséquent, d'après la règle R1, $C \xrightarrow{\alpha^?} C', w \in \Delta$. $C \in \mathcal{X}^?$, par conséquent, $\alpha = \texttt{wait}$.
  D'après la règle Dw qui définit Δ, $C' \in \mathcal{X}$ donc $C' = X_{q'}$, $w = \mathit{Nil}$ et $L(q) = (\texttt{wait}, q')$. Donc d'après la règle Gw, $(q, \emptyset) \xrightarrow{\texttt{wait}} (q', \emptyset)$.
  $\mathcal{S}(u) = (q, \mathcal{S}(t_1) + \cdots + \mathcal{S}(t_n))$ et $\mathcal{S}(v) = (q', \mathcal{S}(t_1) + \cdots + \mathcal{S}(t_n))$. Or $\mathit{State}(t_i) = \emptyset$ pour tout i, donc d'après le lemme 5.9, $\mathcal{S}(t_i) = \emptyset$. Il s'ensuit que $\mathcal{S}(u) \xrightarrow{\texttt{wait}} \mathcal{S}(v)$.

- Cas (3) :

  $t_j \xrightarrow{\alpha} t'_j$, par hypothèse d'induction, $\mathcal{S}(t_j) \xrightarrow{\alpha} \mathcal{S}(t'_j)$.

  - Si $C = \mathit{Nil}$:

    Alors d'après la règle Gp1, $\mathcal{S}(t_1) + \cdots + \mathcal{S}(t_n) \xrightarrow{\alpha} \mathcal{S}(t_1) + \cdots + \mathcal{S}(t_{j-1}) + \mathcal{S}(t'_j) + \mathcal{S}(t_{j+1}) + \cdots + \mathcal{S}(t_n)$. Or $\mathcal{S}(u) = \mathcal{S}(t_1) + \cdots + \mathcal{S}(t_n)$ et $\mathcal{S}(v) = \mathcal{S}(t_1) + \cdots + \mathcal{S}(t_{j-1}) + \mathcal{S}(t'_j) + \mathcal{S}(t_{j+1}) + \cdots + \mathcal{S}(t_n)$. Donc $\mathcal{S}(u) \xrightarrow{\alpha} \mathcal{S}(v)$.

  - Si $C \in \mathcal{X}$:

    Alors $C = X_q$ avec $q \in Q$ et d'après la règle Gp2, $(q, \mathcal{S}(t_1) + \cdots + \mathcal{S}(t_n)) \xrightarrow{\alpha} (q, \mathcal{S}(t_1) + \cdots + \mathcal{S}(t_{j-1}) + \mathcal{S}(t'_j) + \mathcal{S}(t_{j+1}) + \cdots + \mathcal{S}(t_n))$. Or $\mathcal{S}(u) = (q, \mathcal{S}(t_1) + \cdots + \mathcal{S}(t_n))$ et $\mathcal{S}(v) = (q, \mathcal{S}(t_1) + \cdots + \mathcal{S}(t_{j-1}) + \mathcal{S}(t'_j) + \mathcal{S}(t_{j+1}) + \cdots + \mathcal{S}(t_n))$. Donc $\mathcal{S}(u) \xrightarrow{\alpha} \mathcal{S}(v)$.

5.4.3. Deuxième partie de la proposition 5.10

Soient s et s' appartenant à $M(G)$ et α appartenant à $\tilde{A}$, on suppose que $s \xrightarrow{\alpha} s'$. Nous allons montrer par induction sur s qu'il existe $u \in T$ tel que $\mathcal{T}(s) \xrightarrow{\alpha} u$ et $\mathcal{S}(u) = s'$.

Si $s = \emptyset$, alors la propriété est vérifiée car il n'existe aucun s' vérifiant les hypothèses. Nous supposons donc maintenant que $s = (q_1, s_1) + \cdots + (q_n, s_n)$. Alors $\mathcal{T}(s) = \mathit{Nil} \blacktriangleright^n (X_{q_1} \blacktriangleright \mathcal{T}(s_1), \ldots, X_{q_n} \blacktriangleright \mathcal{T}(s_n))$.

La règle **paral1** entraîne qu'il existe i tel que $(q_i, s_i) \xrightarrow{\alpha} s'_i$ et $s' = (q_1, s_1) + \cdots + s'_i + \cdots + (q_n, s_n)$.

Il y a alors cinq cas suivant que la dernière règle appliquée pour obtenir la transition $(q_i, s_i) \xrightarrow{\alpha} s'_i$ est la règle **action** (Ga), **end**(Ge), **call**(Gc), **wait**(Gw) ou **paral2**(Gp2).

- **Règle action :**

  Alors $L(q_i) = (\alpha, q'_i)$ ou $L(q_i) = (\alpha, q'_i, q''_i)$ ou $L(q_i) = (\alpha, q''_i, q'_i)$, $X_{q_i} \in \mathcal{X}^!$ et $s'_i = (q'_i, s_i)$. Par conséquent, d'après la règle Da qui définit Δ, $X_{q_i} \xrightarrow{\alpha}_\Delta X_{q'_i}, \mathit{Nil} \in \Delta$.

  La règle R1 entraîne $X_{q_i} \xrightarrow{\alpha^!}_\Delta X_{q'_i} \blacktriangleright \mathit{Nil}$, puis la règle R5, $X_{q_i} \blacktriangleright \mathcal{T}(s_i) \xrightarrow{\alpha^!}_\Delta X_{q'_i} \blacktriangleright^2 (\mathit{Nil}, \mathcal{T}(s_i))$. Il s'ensuit, d'après le lemme 5.13, $\mathcal{T}(s) = \mathit{Nil} \blacktriangleright^n (X_{q_1} \blacktriangleright \mathcal{T}(s_1), \ldots, X_{q_i} \blacktriangleright \mathcal{T}(s_i), \ldots, X_{q_n} \blacktriangleright \mathcal{T}(s_n)) \xrightarrow{\alpha^!}_\Delta u = \mathit{Nil} \blacktriangleright^n (X_{q_1} \blacktriangleright \mathcal{T}(s_1), \ldots, X_{q'_i} \blacktriangleright^2 (\mathit{Nil}, \mathcal{T}(s_i)), \ldots, X_{q_n} \blacktriangleright \mathcal{T}(s_n))$.
  Or, $\mathcal{S}(u) = \mathcal{S}(X_{q_1} \blacktriangleright \mathcal{T}(s_1)) + \cdots + \mathcal{S}(X_{q'_i} \blacktriangleright^2 (\mathit{Nil}, \mathcal{T}(s_i))) + \cdots + \mathcal{S}(X_{q_n} \blacktriangleright \mathcal{T}(s_n)) = (q_1, \mathcal{S}(\mathcal{T}(s_1))) + \cdots + (q'_i, \mathcal{S}(\mathcal{T}(s_i))) + \cdots + (q_n, \mathcal{S}(\mathcal{T}(s_n))) = (q_1, s_1) + \cdots + (q'_i, s_i) + \cdots + (q_n, s_n) = (q_1, s_1) + \cdots + s'_i + \cdots + (q_n, s_n) = s'$.
  Donc la propriété est satisfaite.

- **Règle end :**

  Alors $L(q_i) = (\texttt{end}, \mathit{Nil})$, $q_i \in \mathcal{X}^!$ et $s'_i = s_i$. Par conséquent, d'après la règle De, $X_{q_i} \xrightarrow{\texttt{end}}_\Delta \mathit{Nil}, \mathit{Nil} \in \Delta$.
  Comme dans le cas précédent, la règle R1 puis la règle R5 et le lemme 5.13 entraînent $\mathcal{T}(s) \xrightarrow{\texttt{end}^!}_\Delta u = \mathit{Nil} \blacktriangleright^n (X_{q_1} \blacktriangleright \mathcal{T}(s_1), \ldots, \mathit{Nil} \blacktriangleright^2 (\mathit{Nil}, \mathcal{T}(s_i)), \ldots, X_{q_n} \blacktriangleright \mathcal{T}(s_n))$.
  Or, $\mathcal{S}(u) = \mathcal{S}(X_{q_1} \blacktriangleright \mathcal{T}(s_1)) + \cdots + \mathcal{S}(\mathit{Nil} \blacktriangleright^2 (\mathit{Nil}, \mathcal{T}(s_i))) + \cdots + \mathcal{S}(X_{q_n} \blacktriangleright \mathcal{T}(s_n)) = (q_1, \mathcal{S}(\mathcal{T}(s_1))) + \cdots + \mathcal{S}(\mathcal{T}(s_i))) + \cdots + (q_n, \mathcal{S}(\mathcal{T}(s_n))) = s'$.
  Donc la propriété est satisfaite.

- **Règle call :**

  Alors $L(q_i) = (\texttt{call}, q'_i, q''_i)$, $q_i \in \mathcal{X}^!$ et $s'_i = (q'_i, s_i + (q''_i, \mathit{Nil})) = (q'_i, (q''_i, \mathit{Nil}) + s_i$. D'après la règle Dc, $X_{q_i} \xrightarrow{\texttt{call}}_\Delta X_{q'_i}, X_{q''_i} \in \Delta$.

Comme dans les cas précédents, la règle R1 puis la règle R5 et le lemme 5.13 entraînent $\mathcal{T}(s) \xrightarrow{\texttt{call}!}_\Delta u = Nil \blacktriangleright^n (X_{q_1} \blacktriangleright \mathcal{T}(s_1), \ldots, X_{q'_i} \blacktriangleright^2 (X_{q''_i}, \mathcal{T}(s_i)), \ldots, X_{q_n} \blacktriangleright \mathcal{T}(s_n))$.

Or, $\mathcal{S}(u) = \mathcal{S}(X_{q_1} \blacktriangleright \mathcal{T}(s_1)) + \cdots + \mathcal{S}(X_{q'_i} \blacktriangleright^2 (X_{q''_i}, \mathcal{T}(s_i))) + \cdots + \mathcal{S}(X_{q_n} \blacktriangleright \mathcal{T}(s_n)) = (q_1, \mathcal{S}(\mathcal{T}(s_1))) + \cdots + (q'_i, \mathcal{S}(X_{q''_i}) + \mathcal{S}(\mathcal{T}(s_i))) + \cdots + (q_n, \mathcal{S}(\mathcal{T}(s_n))) = (q_1, s_1) + \cdots + (q'_i, (q''_i, \emptyset) + s_i) + \cdots + (q_n, s_n) = (q_1, s_1) + \cdots + s'_i + \cdots + (q_n, s_n) = s'$.

Donc la propriété est satisfaite.

. **Règle wait :**

Alors $L(q_i) = (\texttt{wait}, q'_i)$, $q_i \in \mathcal{X}^?$, $s_i = \emptyset$ et $s'_i = (q'_i, \emptyset)$. D'après la règle Dw, $X_{q_i} \xrightarrow{\texttt{wait}}_\Delta X_{q'_i}, Nil$.

La règle R2 entraîne $X_{q_i} \blacktriangleright Nil \xrightarrow{\texttt{wait}?}_\Delta X_{q'_i} \blacktriangleright^2 (Nil, Nil)$, puis la règle R6, $X_{q_i} \blacktriangleright Nil \xrightarrow{\texttt{wait}?}_\Delta X_{q'_i} \blacktriangleright^2 (Nil, Nil)$. Il s'ensuit, d'après le lemme 5.13, $\mathcal{T}(s) \xrightarrow{\texttt{wait}!}_\Delta u = Nil \blacktriangleright^n (X_{q_1} \blacktriangleright \mathcal{T}(s_1), \ldots, X_{q'_i} \blacktriangleright^2 (Nil, Nil), \ldots, X_{q_n} \blacktriangleright \mathcal{T}(s_n))$.

Or, $\mathcal{S}(u) = \mathcal{S}(X_{q_1} \blacktriangleright \mathcal{T}(s_1)) + \cdots + \mathcal{S}(X_{q'_i} \blacktriangleright^2 (Nil, Nil)) + \cdots + \mathcal{S}(X_{q_n} \blacktriangleright \mathcal{T}(s_n)) = (q_1, \mathcal{S}(\mathcal{T}(s_1))) + \cdots + (q'_i, \emptyset) + \cdots + (q_n, \mathcal{S}(\mathcal{T}(s_n))) = s'$.

. **Règle paral2 :**

Alors $s_i \xrightarrow{\alpha} s''_i$ et $s'_i = (q_i, s''_i)$. Par hypothèse d'induction, il existe $t \in T$ tel que $\mathcal{T}(s_i) \xrightarrow{\alpha^\varepsilon}_\Delta t$ avec $\varepsilon \in \{!, ?\}$ et $\mathcal{S}(t) = s''_i$.

D'après le lemme 5.13, $X_{q_i} \blacktriangleright \mathcal{T}(s_i) \xrightarrow{\alpha!}_\Delta X_{q_i} \blacktriangleright t$, puis $\mathcal{T}(s) \xrightarrow{\alpha!}_\Delta u = Nil \blacktriangleright^n (X_{q_1} \blacktriangleright \mathcal{T}(s_1), \ldots, X_{q_i} \blacktriangleright t, \ldots, X_{q_n} \blacktriangleright \mathcal{T}(s_n))$.

De plus, $\mathcal{S}(u) = \mathcal{S}(X_{q_1} \blacktriangleright \mathcal{T}(s_1)) + \cdots + \mathcal{S}(X_{q_i} \blacktriangleright t) + \cdots + \mathcal{S}(X_{q_n} \blacktriangleright \mathcal{T}(s_n)) = (q_1, \mathcal{S}(\mathcal{T}(s_1))) + \cdots + (q_i, \mathcal{S}(t)) + \cdots + (q_n, \mathcal{S}(\mathcal{T}(s_n))) = (q_1, s_1) + \cdots + (q_i, s''_i) + \cdots + (q_n, s_n) = s'$.

Nous pouvons maintenant associer à tout schéma RPPS une algèbre RPA.

6. L'accessibilité entre des états emboîtés

Sommaire

Le but de ce chapitre est de montrer que le problème de l'accessibilité entre deux états emboîtés est NP-complet. Pour cela, nous montrons le lien très fort qu'il y a entre ce problème et celui de l'accessibilité entre deux termes RPA. Cette démarche a été calquée sur celle de D. Lugiez et Ph. Schnoebelen [LS01] qui réduit le problème de l'accessibilité entre deux termes PA modulo congruence structurelle à l'accessibilité entre deux termes PA sans congruence structurelle.

La première étape de cette démarche consiste à décomposer $\equiv_{\mathcal{S}}$ en des relations de simplification et de permutation. Dans la preuve proposée par D. Lugiez et Ph. Schnoebelen, l'équivalence étudiée est la congruence structurelle sur les termes PA et les relations sont bien connues. Au contraire, le passage des états emboîtés aux termes RPA est un des apports originaux de notre travail. Les relations correspondantes sont définies dans la section 6.1 et leur liens avec $\equiv_{\mathcal{S}}$ sont présentés dans la section 6.2.

Pour la seconde étape, nous montrons que certaines de ces relations sont reconnaissables par un automate d'arbres (voir section 6.3) et enfin nous donnons un algorithme qui décide l'accessibilité entre deux états emboîtés dans la section 6.4.

6.1. Relations SPC sur les termes

Nous commençons donc par définir les relations de simplification, de permutation et de conversion (SPC) entre les termes RPA qui nous permettront dans la section suivante de caractériser l'équivalence $\equiv_{\mathcal{S}}$.

6.1.1. Simplification

La relation de simplification lie des termes équivalents pour $\equiv_{\mathcal{S}}$, qui ne diffèrent que par le nombre et l'emplacement de leur *Nil*.

Une première règle exprime le fait qu'un processus dont le père a fini ses tâches se comporte comme s'il était seul.

Définition 6.1 *La relation $\curvearrowright$ est définie par :*

$$\forall u \in T,\ \mathit{Nil} \blacktriangleright u \curvearrowright u. \tag{R1}$$

Dans la définition qui suit, la première règle peut s'interprété par si "un processus a fini ses tâches, son père adopte ses fils". La seconde propage cette propriété.

Définition 6.2 *La relation $\searrow$ est définie inductivement par les règles suivantes :*

$$\begin{aligned} C \blacktriangleright^n (t_1, \ldots, t_{i-1}, \mathit{Nil} \blacktriangleright^m (u_1, \ldots, u_m), t_{i+1}, \ldots, t_n) \searrow \\ C \blacktriangleright^{n+m-1} (t_1, \ldots, t_{i-1}, u_1, \ldots, u_m, t_{i+1}, \ldots, t_n). \end{aligned} \tag{R2}$$

$$\text{Si } t_i \searrow u, \text{ alors } C \blacktriangleright^n (t_1, \ldots, t_n) \searrow C \blacktriangleright^n (t_1, \ldots, t_{i-1}, u, t_{i+1}, \ldots, t_n). \tag{R3}$$

Remarque 6.3 Si $m = 0$, la règle R2 se lit $C \blacktriangleright^n (t_1, \ldots, t_{i-1}, \mathit{Nil}, t_{i+1}, \ldots, t_n) \searrow C \blacktriangleright^{n-1} (t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n)$. Cette règle peut s'interpréter par "un fils qui a fini ses tâches n'influence pas le comportement de son père, ni de ses frères".

Définition 6.4
*Nous notons $\Searrow$ l'union de $\curvearrowright$ et de $\searrow$ et $\Searrow^*$ sa clôture réflexive et transitive. Quand $t \Searrow^* u$, nous disons que u est une **simplification** de t.*

Nous notons $\Swarrow$ et ${}^*\!\Swarrow$ les relations inverses $(\Searrow)^{-1}$ et $(\Searrow^*)^{-1}$.

Exemple 6.5 Nous avons :

$$\mathit{Nil} \blacktriangleright (X \blacktriangleright^2 (X, \mathit{Nil} \blacktriangleright^2 (Y, Z))) \Searrow^* X \blacktriangleright^3 (X, Y, Z).$$

En effet, $\mathit{Nil} \blacktriangleright (X \blacktriangleright^2 (X, \mathit{Nil} \blacktriangleright^2 (Y, Z))) \curvearrowright X \blacktriangleright^2 (X, \mathit{Nil} \blacktriangleright^2 (Y, Z))$ et $X \blacktriangleright^2 (X, \mathit{Nil} \blacktriangleright^2 (Y, Z)) \searrow X \blacktriangleright^3 (X, Y, Z)$.

Lemme 6.6
– *Si $t \curvearrowright \searrow u$, alors $t \searrow \curvearrowright u$.*

– *Si* $t \curvearrowright\curvearrowright u$, *alors* $t \searrow \curvearrowright u$.

Démonstration.

– Nous supposons que $t \curvearrowright\searrow u$, alors $t = \mathit{Nil} \blacktriangleright t'$ et $\mathit{Nil} \blacktriangleright t' \curvearrowright t' \searrow u$. D'après la règle R3, $\mathit{Nil} \blacktriangleright t' \searrow \mathit{Nil} \blacktriangleright u$ et $\mathit{Nil} \blacktriangleright u \curvearrowright u$ en utilisant la règle R1. Donc $t \searrow\curvearrowright u$.

– Nous supposons que $t \curvearrowright\curvearrowright u$, alors $t = \mathit{Nil} \blacktriangleright (\mathit{Nil} \blacktriangleright u)$. Donc $t \searrow \mathit{Nil} \blacktriangleright u$ en utilisant la règle R2 et $\mathit{Nil} \blacktriangleright u \curvearrowright u$ en utilisant la règle R1. D'où $t \searrow\curvearrowright u$.

$\lrcorner$

Par conséquent, $\Searrow^*$ coïncide avec $\searrow^* \cup \searrow^* \curvearrowright$.

Comme $u \Searrow t$ entraîne $|u| > |t|$, nous avons :

Lemme 6.7 *La relation* $\Searrow$ *est noethérienne et* $\Searrow^*$ *est un ordre partiel bien-fondé.*

Le lemme suivant implique que la relation $\Searrow$ est localement confluente :

Lemme 6.8 *Si* $u \Swarrow w \Searrow v$, *alors soit* $u = v$, *soit il existe un terme* w' *tel que* $u \Searrow w' \Swarrow v$.

Démonstration. Supposons que $u \Swarrow w \Searrow v$, nous allons montrer par induction sur u que $u = v$ ou il existe un terme w' tel que $u \Searrow w' \Swarrow v$.

Si u appartient à *Atomes*, alors $w = \mathit{Nil} \blacktriangleright u$ et $v = u$. Si u est de la forme $C \blacktriangleright^n (u_1, \ldots, u_n)$ avec $n > 0$, nous considérons tous les cas d'extension de u suivie d'une simplification.

- Si l'extension est une instance de la règle R1 :
  Alors $w = \mathit{Nil} \blacktriangleright u$. $\mathit{Nil} \blacktriangleright u \Searrow v$, donc $v = u$ ou il existe un terme u' tel que $v = \mathit{Nil} \blacktriangleright u'$ et $u \Searrow u'$. Par conséquent, $u \Searrow u' \Swarrow v$ d'après la règle R1.
- Si l'extension est une instance de la règle R2 :
  Alors $w = C \blacktriangleright^{n-m+1} (u_1, \ldots, u_{i-1}, \mathit{Nil} \blacktriangleright^m (u_i, \ldots, u_{i+m-1}), u_{i+m}, \ldots, u_n)$.
  - Si la simplification est une instance de la règle R1 :
    Alors $w = \mathit{Nil} \blacktriangleright (\mathit{Nil} \blacktriangleright^m (u_i, \ldots, u_{i+m-1}))$ car alors $C = \mathit{Nil}$ et $n - m + 1 = 1$ ou $w = \mathit{Nil} \blacktriangleright \mathit{Nil}$ car alors $C = \mathit{Nil}$ et $n - 1 = 1$. Nous avons alors soit $u = v = \mathit{Nil} \blacktriangleright^m (u_i, \ldots, u_{i+m-1})$, soit $u = v = \mathit{Nil}$.
  - Si la simplification est une instance de la règle R2 :
    Alors $u = v$ ou nous pouvons faire cette simplification sur u et ensuite faire la même extension.
  - Si la simplification est une instance de la règle R3 :
    Alors nous pouvons faire la même simplification sur u, puis la même extension.

- Si l'extension est une instance de la règle R3 :

  Alors il existe un indice i et un terme u'_i tels que $u_i \swarrow u'_i$ et $u \swarrow C \blacktriangleright^n (u_1, \ldots, u_{i-1}, u'_i, u_{i+1}, \ldots, u_n)$.

  - Si la simplification est une instance de la règle R1 :

    Alors $w = \mathit{Nil} \blacktriangleright u'_i$, $v = u'_i$ et $u = \mathit{Nil} \blacktriangleright u_i$. Donc d'après la règle R2, $u \Searrow u_i$. Il s'ensuit que $u \Searrow \swarrow v$.
  - Si la simplification est une instance de la règle R2 ou une instance de la règle R3 qui ne porte pas sur u'_i :

    Alors la même simplification peut être appliquée à v et ensuite nous utilisons la règle R3.
  - Si la simplification est une instance de la règle R3 appliquée au terme u'_i :

    Alors il existe un terme u''_i tel que $u_i \swarrow u'_i \Searrow u''_i$ et $v = C \blacktriangleright^n (u_1, \ldots, u_{i-1}, u''_i, u_{i+1}, \ldots, u_n)$. Par hypothèse d'induction, $u_i = u''_i$ et la propriété est vérifiée ou il existe un terme w_i tel que $u_i \Searrow w_i \Swarrow u''_i$, la règle R3 entraine alors que $u \Searrow C \blacktriangleright^n (u_1, \ldots, u_{i-1}, w_i, u_{i+1}, \ldots, u_n) \Swarrow v$.

$\lrcorner$

Par conséquent, le lemme de Newman entraîne que $\Searrow$ est convergente donc que pour tout terme t, il existe un terme u tel que $t \Searrow^* u$ et u ne peut pas se simplifier.

Définition 6.9 *La* **forme normale simplifiée** *de t, notée $t\downarrow$ est l'unique u obtenu en simplifiant t au maximum.*

Exemple 6.10 La forme normale simplifiée de $\mathit{Nil} \blacktriangleright^2 (\mathit{Nil}, X \blacktriangleright^3 (\mathit{Nil}, X, \mathit{Nil}))$ est $X \blacktriangleright X$.

6.1.2. La permutation

Définition 6.11 *La relation $\rightleftarrows$ est définie inductivement par les règles suivantes :*

$$C \blacktriangleright^n (t_1, \ldots, t_n) \rightleftarrows C \blacktriangleright^n (t_1, \ldots, t_{i-1}, t_{i+1}, t_i, t_{i+2}, \ldots, t_n). \tag{P1}$$

$$\begin{array}{l} \text{Si } t_i \rightleftarrows u, \\ \quad \text{alors } C \blacktriangleright^n (t_1, \ldots, t_n) \rightleftarrows C \blacktriangleright^n (t_1, \ldots, t_{i-1}, u, t_{i+1}, \ldots, t_n). \end{array} \tag{P2}$$

La relation $\rightleftarrows$ est symétrique. Nous notons $\overset{*}{\rightleftarrows}$ la fermeture réflexive, symétrique et transitive de $\rightleftarrows$. Si $t \overset{*}{\rightleftarrows} u$, nous dirons que t et u sont **équivalents par permutation**.

Le lemme suivant permet de commuter simplification et permutation :

Lemme 6.12 *(commutation) Si* $u \Swarrow w \rightleftarrows v$, *alors il existe* w' *tel que* $u \stackrel{*}{\rightleftarrows} w' \Swarrow v$.

Par symétrie, ce lemme entraine que si $u \rightleftarrows w \Searrow v$, alors il existe w' tel que $u \Searrow w' \stackrel{*}{\rightleftarrows} v$

Démonstration. Supposons que $u \Swarrow w \rightleftarrows v$, nous allons montrer par induction sur u qu'il existe w' tel que $u \stackrel{*}{\rightleftarrows} w' \Swarrow v$.

Si u appartient à *Atomes*, alors $v = w = Nil \blacktriangleright u$ et la propriété est vraie. Si u est de la forme $C \blacktriangleright^n (u_1, \ldots, u_n)$ avec $n > 0$, nous allons considérer tous les cas d'extension de u suivie par une permutation.

- Si l'extension est une instance de la règle R1 :

  Alors $w = Nil \blacktriangleright u$. Comme $w \rightleftarrows v$, il existe u' tel que $v = Nil \blacktriangleright u'$ et $u \rightleftarrows u'$. Donc $u \rightleftarrows u' \Swarrow v$ d'après la règle R1.

- Si l'extension est une instance de la règle R2 :

  Alors $w = C \blacktriangleright^{n-m+1} (u_1, \ldots, u_{i-1}, Nil \blacktriangleright^m (u_i, \ldots, u_{i+m-1}), u_{i+m}, \ldots, u_n)$ si $m \geq Nil$ ou $w = C \blacktriangleright^{n+1} (u_1, \ldots, u_{i-1}, Nil, u_i, \ldots, u_n)$.

  - Si la permutation est une instance de la règle P1 et ne concerne pas $Nil \blacktriangleright^m (u_i, \ldots, u_{i+m-1})$:

    Alors nous pouvons permuter le même couple et ensuite faire la même extension.

  - Si la permutation est une instance de la règle P1 et permute $Nil \blacktriangleright^m (u_i, \ldots, u_{i+m-1})$ avec u_j :

    Alors nous faisons plusieurs permutations pour que u_j soit à la place de u_i et que $u_i, \ldots, u_{i+m-1}$ apparaissent entre u_{j-1} et u_{j+1}. Ensuite, nous appliquons la règle R2 aux termes $u_i, \ldots, u_{i+m-1}$.

  - Si la permutation est une instance de la règle P2 :

    Nous commençons par appliquer la même permutation à v si la permutation porte sur un u_j et une instance de la règle P1 si la permutation porte sur deux termes de l'ensemble $\{u_i, \ldots, u_{i+m-1}\}$. Ensuite, nous appliquons une instance de la règle R2 sur les termes $u_i, \ldots, u_{i+m-1}$.

- Si l'extension est une instance de la règle R3 :

  Alors il existe i et u'_i tel que $u_i \swarrow u'_i$ et $u \swarrow C \blacktriangleright^n (u_1, \ldots, u_{i-1}, u'_i, u_{i+1}, \ldots, u_n)$.

  - Si la permutation est une instance de la règle P1 ou une instance de la règle P2 qui ne concerne pas u'_i :

    Alors la même permutation peut être appliquée à v et ensuite nous appliquons la règle R3.

- Si la permutation est une instance de la règle P2 sur le terme u'_i :
  Alors il existe u''_i tel que $u_i \swarrow u'_i \rightleftarrows u''_i$ et $v = C \blacktriangleright^n (u_1, \ldots, u_{i-1}, u''_i, u_{i+1}, \ldots, u_n)$. Par hypothèse d'induction, il existe un terme w_i tel que $u_i \overset{*}{\rightleftarrows} w_i \Swarrow u''_i$. Les règles R3 et P2 entrainent que $u \overset{*}{\rightleftarrows} C \blacktriangleright^n (u_1, \ldots, u_{i-1}, w_i, u_{i+1}, \ldots, u_n) \Swarrow v$.

⌟

6.1.3. La convertibilité

Nous finissons en définissant la relation $\leftrightsquigarrow$, appelée **convertibilité**, comme la relation $(\Searrow \cup \rightleftarrows \cup \Swarrow)^*$. La conversion d'un terme en un autre est donc une séquence de simplifications directes ou inverses et de permutations.

Lemme 6.13 *Les assertions suivantes sont équivalentes :*
(a) $u \leftrightsquigarrow v$,
(b) il existe deux termes u' *et* v' *tels que* $u \Searrow^* u' \overset{*}{\rightleftarrows} v' \,{}^*\!\!\Swarrow v$,
(c) $u\downarrow \overset{*}{\rightleftarrows} v\downarrow$.

Démonstration. Il est immédiat que $(c) \Rightarrow (b)$ et $(b) \Rightarrow (a)$. Il nous suffit donc de montrer que $(a) \Rightarrow (c)$ pour avoir l'équivalence entre les trois assertions.

Nous commençons par montrer que $(a) \Rightarrow (b)$ car $(a) \Rightarrow (c)$ s'en déduit facilement. En effet, supposons que $u \leftrightsquigarrow v$, alors $(u\downarrow) \,{}^*\!\!\Swarrow u \leftrightsquigarrow v \Searrow^* (v\downarrow)$, ce que nous pouvons aussi écrire $(u\downarrow) \leftrightsquigarrow (v\downarrow)$. Si $(a) \Rightarrow (b)$, nous avons alors $(u\downarrow) \Searrow^* \leftrightsquigarrow {}^*\!\!\Swarrow (v\downarrow)$. Or comme $u\downarrow$ et $v\downarrow$ ne peuvent pas être simplifiés plus, nous obtenons $u\downarrow \leftrightsquigarrow v\downarrow$.

Montrons que $(a) \Rightarrow (b)$.

Supposons que $u \leftrightsquigarrow v$, alors il existe une suite de conversion $u = u_0\ \mathcal{R}_1\ u_1\ \mathcal{R}_2 \cdots \mathcal{R}_n\ u_n = v$ avec $\mathcal{R}_i \in \{\Searrow, \Swarrow, \rightleftarrows\}$. Dans une suite, nous disons qu'il y a un **pic** à la position i si $\mathcal{R}_{i-1}$ et $\mathcal{R}_i$ suivent l'un des motifs suivants :

$$\begin{array}{ccc} & u_i & \\ \Swarrow & & \Searrow \\ u_{i-1} & & u_{i+1} \end{array} \qquad \begin{array}{ccc} & u_i \rightleftarrows u_{i+1} \\ \Swarrow & \\ u_{i-1} & \end{array} \qquad \begin{array}{ccc} u_{i-1} \rightleftarrows u_i & \\ & \Searrow \\ & u_{i+1} \end{array}$$

Les lemmes 6.8 et 6.12 permettent de faire disparaître n'importe quel pic d'une suite de conversion. En contrepartie, la longueur de la suite peut augmenter et de nouveaux pics apparaître.

Cependant, nous ne pouvons pas enlever des pics indéfiniment : à chaque fois que nous utilisons le lemme 6.8 ou le lemme 6.12, nous enlevons un terme u_i de la suite et les termes v_j que nous rajoutons sont strictement plus petit que u_i. En effet, ces termes vérifient $u_i \Searrow^* \overset{*}{\rightleftarrows} \Searrow^* v_j$ ce qui entraîne que $|u_i| > |v_j|$.

Quand la suite ne contient plus de pic, nous obtenons $u \Searrow^* \overset{*}{\rightleftarrows} {}^*\!\!\Swarrow v$. ⌟

6.2. Les liens entre $\equiv_{\mathcal{S}}$ et les relations SPC

6.2.1. Caractérisation de $\equiv_{\mathcal{S}}$ par $\leftrightsquigarrow$

Nous avons défini la relation $\leftrightsquigarrow$ pour donner une caractérisation de $\equiv_{\mathcal{S}}$ plus facile à utiliser que sa définition. Nous allons donc vérifier que la relation $\leftrightsquigarrow$ coïncide avec $\equiv_{\mathcal{S}}$.

Proposition 6.14 *$u \equiv_{\mathcal{S}} v$ si et seulement si $u \leftrightsquigarrow v$.*

Démonstration. La direction ($\Leftarrow$) est évidente. En observant les règles de R1 jusqu'à P2, nous voyons que $u \searrow v$ ou $u \rightleftarrows v$ entraîne $\mathcal{S}(u) = \mathcal{S}(v)$.

Nous utilisons les deux lemmes qui suivent pour montrer la direction ($\Rightarrow$). ⌟

Lemme 6.15 *Si $Constante(u) = \emptyset$, alors $u \searrow {}^*Nil$.*

Démonstration. Par une induction facile sur u. ⌟

Lemme 6.16
Si $t_i \leftrightsquigarrow u$, alors $C \blacktriangleright^n (t_1, \ldots, t_n) \leftrightsquigarrow C \blacktriangleright^n (t_1, \ldots, t_{i-1}, u, t_{i+1}, \ldots, t_n)$.

Démonstration. Nous faisons une démonstration par induction sur la longueur de la dérivation $t_i \leftrightsquigarrow u$.

Si $t_i \searrow u$ ou $t_i \rightleftarrows u$, alors nous concluons avec les règles R3 et P2.

Si $t_i \curvearrowright u$, alors $t_i = Nil \blacktriangleright u$. La règle R2 entraîne que $C \blacktriangleright^n (t_1, \ldots, t_{i-1}, Nil \blacktriangleright u, t_{i+1}, \ldots, t_n) \searrow C \blacktriangleright^n (t_1, \ldots, t_{i-1}, u, t_{i+1}, \ldots, t_n)$. ⌟

Démonstration de la fin de la proposition 6.14.

Nous pouvons maintenant montrer que si $u \equiv_{\mathcal{S}} v$, alors $u \leftrightsquigarrow v$ par induction sur $|u| + |v|$. Nous supposons que $u \equiv_{\mathcal{S}} v$ (et par suite $Constante(u) = Constante(v)$). Nous considérons les différents cas :

- Si $Constante(u) = Constante(v) = \emptyset$, alors $u \searrow {}^*Nil \swarrow {}^*v$ d'après le lemme 6.15.
- Sinon u est de la forme $C \blacktriangleright^n (u_1, \ldots, u_n)$ et v de la forme $C' \blacktriangleright^m (v_1, \ldots, v_m)$.
  - Si $C \in \mathcal{X}$ et $C' = Nil$:

    Alors $\mathcal{S}(u) = (C, \mathcal{S}(u_1) + \ldots + \mathcal{S}(u_n))$ et $\mathcal{S}(v) = \mathcal{S}(v_1) + \ldots + \mathcal{S}(v_m)$. Donc il existe un indice $i \in \{1, 2, \ldots, m\}$ tel que $\mathcal{S}(v_i) = \mathcal{S}(u)$ et pour tout $j \neq i$, $\mathcal{S}(v_j) = \emptyset$.

    Comme $\mathcal{S}(v_j) = \emptyset$, $v_j \leftrightsquigarrow Nil$ pour tout $j \neq i$ par hypothèse d'induction. De plus, $\mathcal{S}(v_i) = \mathcal{S}(u)$ et v_i est plus petit que v, donc par hypothèse d'induction $u \leftrightsquigarrow v_i$. Le lemme 6.16 entraîne que $v \leftrightsquigarrow Nil \blacktriangleright^m (Nil, \ldots, v_i, \ldots, Nil) \leftrightsquigarrow Nil \blacktriangleright^m (Nil, \ldots, u, \ldots, Nil)$. En appliquant la règle R2, nous avons $v \leftrightsquigarrow Nil \blacktriangleright u$ et nous concluons avec la règle R1 que $v \leftrightsquigarrow u$.

- Si $C = Nil$ et $C' \in \mathcal{X}$:
  Nous sommes dans le cas symétrique du cas précédent donc $v \leftrightsquigarrow u$.
- Si $C = Nil$ et $C' = Nil$ ou si $C \in \mathcal{X}$ et $C' \in \mathcal{X}$:
  Si $C = Nil$ et $C' = Nil$, alors $\mathcal{S}(u) = \mathcal{S}(u_1) + \ldots + \mathcal{S}(u_n)$ et $\mathcal{S}(v) = \mathcal{S}(v_1) + \ldots + \mathcal{S}(v_m)$. Si $C \in \mathcal{X}$ et $C' \in \mathcal{X}$, alors $\mathcal{S}(u) = (C, \mathcal{S}(u_1) + \ldots + \mathcal{S}(u_n))$ et $\mathcal{S}(v) = (C', \mathcal{S}(v_1) + \ldots + \mathcal{S}(v_m))$. Dans les deux cas, $C = C'$ et $\mathcal{S}(u_1) + \ldots + \mathcal{S}(u_n) = \mathcal{S}(v_1) + \ldots + \mathcal{S}(v_m)$.
  - Si tous les u_i et tous les v_j sont de la forme $X \blacktriangleright^\alpha (w_1, \ldots, w_\alpha)$:
    Alors $n = m$ et il existe une bijection h entre les indices i et les indices j telle que $\mathcal{S}(u_i) = \mathcal{S}(v_{h(i)})$. Par hypothèse d'induction, $u_i \leftrightsquigarrow v_{h(i)}$. Le lemme 6.16 entraîne que $u \leftrightsquigarrow C \blacktriangleright^n (v_{h(1)}, \ldots, v_{h(n)})$. En appliquant la règle P1, nous avons $u \leftrightsquigarrow C \blacktriangleright^n (v_1, \ldots, v_n)$ et donc $u \leftrightsquigarrow v$.
  - S'il existe un terme u_i ou un terme v_j de la forme $Nil \blacktriangleright^k (w_1, \ldots, w_k)$:
    Nous utilisons la règle R2 pour aplatir ce terme et nous utilisons l'hypothèse d'induction.

⌟

6.2.2. Transitions modulo $\equiv_\mathcal{S}$

Dans ce qui suit, si u et v sont deux termes RPA et α un élément de $\tilde{A}$, nous notons par abus $u \xrightarrow{\alpha} v$ lorsqu'il existe $\varepsilon \in \{!, ?\}$ tel que $u \xrightarrow{\alpha^\varepsilon}$.

Nous définissons une notion de transitions compatibles avec $\equiv_\mathcal{S}$.

Définition 6.17 *Soient u et v deux termes RPA, $\alpha \in \tilde{A}$, $u \overset{\alpha}{\Rightarrow} v$ s'il existe deux termes u' et v', tels que $u \equiv_\mathcal{S} u' \xrightarrow{\alpha} v' \equiv_\mathcal{S} v$.*

Nous utilisons les extensions usuelles $u \overset{\sigma}{\Rightarrow} v$ (pour $\sigma \in \tilde{A}^*$) et $u \overset{*}{\Rightarrow} v$.

La propriété 5.10 réduit l'accessibilité entre deux états emboîtés à la $\overset{*}{\Rightarrow}$-accessibilité entre des termes RPA.

Proposition 6.18 *Soient s et s' deux états emboîtés, $s \xrightarrow{*} s'$ dans $\mathcal{M}_G$ si et seulement si $\mathcal{T}(s) \overset{*}{\Rightarrow} \mathcal{T}(t)$ dans T_G.*

Une propriété fondamentale de $\equiv_\mathcal{S}$ est le lemme suivant qui montre que $\equiv_\mathcal{S}$ respecte les comportements dans un sens très fort.

Proposition 6.19 *La relation $\equiv_\mathcal{S}$ est une bisimulation modulo l'abstraction des étiquettes $\{!, ?\}$, c'est-à-dire que $u \equiv_\mathcal{S} v$ et $u \xrightarrow{\alpha} u'$ entraîne qu'il existe un terme v' tel que $v \xrightarrow{\alpha} v'$ et $v \equiv_\mathcal{S} v'$.*

Démonstration. Nous utilisons la propriété 6.14 et nous montrons que $\leftrightsquigarrow$ est une relation de bisimulation. Cette preuve est standard mais fastidieuse. Nous montrons que $\curvearrowright$, $\searrow$ et $\rightleftarrows$ sont des relations de bisimulation.

- La relation $\curvearrowright$ est une bisimulation.

  Soit $(Nil \blacktriangleright u, u) \in \curvearrowright$.

  – Si $Nil \blacktriangleright u \xrightarrow{\alpha} v$, alors d'après le lemme 5.14, il existe un terme u' tel que $u \xrightarrow{\alpha} u'$ et $v = Nil \blacktriangleright u'$. De plus, $Nil \blacktriangleright u' \curvearrowright u'$.

  – Si $u \xrightarrow{\alpha} v$, le lemme 5.13 entraîne que $Nil \blacktriangleright u \xrightarrow{\alpha} Nil \blacktriangleright v$. De plus, $Nil \blacktriangleright v \curvearrowright v$.

  Donc, $\curvearrowright$ est une relation de bisimulation.

- $\searrow$ est une relation de bisimulation.

  Nous allons montrer que $\mathcal{S}$ et $\mathcal{S}'$ sont des relations de bisimulation fortes où $\mathcal{S} = \{(C \blacktriangleright^n (t_1, \ldots, t_{i-1}, Nil \blacktriangleright^m (u_1, \ldots, u_m), t_{i+1}, \ldots, t_n), C \blacktriangleright^{n+m-1} (t_1, \ldots, t_{i-1}, u_1, \ldots, u_m, t_{i+1}, \ldots, t_n) \mid C \in Atomes,\ n, m \in \mathbb{N},\ t_i, u_j \in T_G\}$ et $\mathcal{S}' = \{(C \blacktriangleright^n (t_1, \ldots, t_n), C \blacktriangleright^n (t_1, \ldots, t_{i-1}, u, t_{i+1}, \ldots, t_n) \mid t_i \searrow u,\ C \in Atomes,\ n \in \mathbb{N},\ t_i, u \in T_G\}$.

  - $\mathcal{S}$ est une relation de bisimulation.

    Soit $(C \blacktriangleright^n (t_1, \ldots, t_{i-1}, Nil \blacktriangleright^m (u_1, \ldots, u_m), t_{i+1}, \ldots, t_n), C \blacktriangleright^{n+m-1} (t_1, \ldots, t_{i-1}, u_1, \ldots, u_m, t_{i+1}, \ldots, t_n) \in \mathcal{S}$.

    – Supposons que $C \blacktriangleright^n (t_1, \ldots, t_{i-1}, Nil \blacktriangleright^m (u_1, \ldots, u_m), t_{i+1}, \ldots, t_n) \xrightarrow{\alpha} v$. Le lemme 5.14 montre qu'il y a quatre cas :

    1. $C \xrightarrow{\alpha!} q' \blacktriangleright w$ et $v = q' \blacktriangleright^{n+1} (w, t_1, \ldots, t_{i-1}, Nil \blacktriangleright^m (u_1, \ldots, u_m), t_{i+1}, \ldots, t_n)$.
       Alors le lemme 5.11 entraîne que $C \blacktriangleright^{n+m-1} (t_1, \ldots, t_{i-1}, u_1, \ldots, u_m, t_{i+1}, \ldots, t_n) \xrightarrow{\alpha!} v' = C \blacktriangleright^{n+m} (w, t_1, \ldots, t_{i-1}, u_1, \ldots, u_m, t_{i+1}, \ldots, t_n)$ et $(v, v') \in \mathcal{S}$.
    2. $C \xrightarrow{\alpha?} q' \blacktriangleright w$, $v = q' \blacktriangleright^{n+1} (w, t_1, \ldots, t_{i-1}, Nil \blacktriangleright^m (u_1, \ldots, u_m), t_{i+1}, \ldots, t_n)$ et $Constante(u_j) = \emptyset$ et $Constante(t_j) = \emptyset$ pour tout i et tout j.
       Alors le lemme 5.12 entraîne que $C \blacktriangleright^{n+m-1} (t_1, \ldots, t_{i-1}, u_1, \ldots, u_m, t_{i+1}, \ldots, t_n) \xrightarrow{\alpha?} v' = C \blacktriangleright^{n+m} (w, t_1, \ldots, t_{i-1}, u_1, \ldots, u_m, t_{i+1}, \ldots, t_n)$ et $(v, v') \in \mathcal{S}$.
    3. $t_j \xrightarrow{\alpha} t'_j$ et $v = C \blacktriangleright^n (t_1, \ldots, t_{j-1}, t'_j, t_{j+1}, \ldots, t_{i-1}, Nil \blacktriangleright^m (u_1, \ldots, u_m), t_{i+1}, \ldots, t_n)$.
       Alors le lemme 5.13 entraîne que $C \blacktriangleright^{n+m-1} (t_1, \ldots, t_{j-1}, t_j, t_{j+1}, \ldots, t_{i-1}, u_1, \ldots, u_m, t_{i+1}, \ldots, t_n) \xrightarrow{\alpha!} v' = C \blacktriangleright^{n+m-1} (t_1, \ldots, t_{j-1}, t'_j, t_{j+1}, \ldots, t_{i-1}, u_1, \ldots, u_m, t_{i+1}, \ldots, t_n)$ et $(v, v') \in \mathcal{S}$.
    4. $u_j \xrightarrow{\alpha} u'_j$ est un cas qui se traite comme le précédent.

    – Supposons que $C \blacktriangleright^{n+m-1} (t_1, \ldots, t_{i-1}, u_1, \ldots, u_m, t_{i+1}, \ldots, t_n) \xrightarrow{\alpha} v$. Il y a alors quatre cas qui se traitent de façon analogue aux quatre cas précédents.

Il s'ensuit que $\mathcal{S}$ est une relation de bisimulation forte.

- $\mathcal{S}'$ est une relation de bisimulation.

  Soit $(C \blacktriangleright^n (t_1, \dots, t_n), C \blacktriangleright^n (t_1, \dots, t_{i-1}, u, t_{i+1}, \dots, t_n) \in \mathcal{S}'$. Nous allons montrer par induction sur la longueur de $C \blacktriangleright^n (t_1, \dots, t_n)$ que, d'une part, si $C \blacktriangleright^n (t_1, \dots, t_n) \xrightarrow{\alpha} v$, alors il existe un terme v' tel que $C \blacktriangleright^n (t_1, \dots, t_{j-1}, t'_j, t_{j+1}, \dots, t_{i-1}, u, t_{i+1}, \dots, t_n) \xrightarrow{\alpha!} v'$ et $(v, v') \in \mathcal{S}'$ et, d'autre part, que si $C \blacktriangleright^n (t_1, \dots, t_{j-1}, t'_j, t_{j+1}, \dots, t_{i-1}, u, t_{i+1}, \dots, t_n) \xrightarrow{\alpha!} v'$, alors il existe un terme v tel que $C \blacktriangleright^n (t_1, \dots, t_n) \xrightarrow{\alpha} v$ et $(v, v') \in \mathcal{S}'$.

  - Si la longueur de $C \blacktriangleright^n (t_1, \dots, t_n)$ est minimale.

    Alors $n = 1$, $t_i = t = C' \blacktriangleright (Nil \blacktriangleright C'')$ et $u = C' \blacktriangleright C''$ avec $C', C'' \in Atomes$.

    – Supposons que $C \blacktriangleright t \xrightarrow{\alpha} v$. Il y a cinq cas :

      1. $C \xrightarrow{\alpha!} q' \blacktriangleright w$ et $v = q' \blacktriangleright^2 (w, t)$.

         Alors le lemme 5.11 entraîne que $C \blacktriangleright u \xrightarrow{\alpha!} v' = C \blacktriangleright^2 (w, u)$ et $(v, v') \in \mathcal{S}$.

      2. $C \xrightarrow{\alpha?} q' \blacktriangleright w$, $v = q' \blacktriangleright^2 (w, t)$ et $Constante(t) = \emptyset$.

         Si $Constante(t) = \emptyset$, alors $Constante(u) = \emptyset$. Donc d'après le lemme 5.12, $C \blacktriangleright u \xrightarrow{\alpha?} v' = C \blacktriangleright^2 (w, u)$ et $(v, v') \in \mathcal{S}$.

      3. $C' \xrightarrow{\alpha!} q' \blacktriangleright w$ et $v = C \blacktriangleright (q' \blacktriangleright^2 (w, Nil \blacktriangleright C''))$.

         Alors le lemme 5.11 entraîne que $C' \blacktriangleright C'' \xrightarrow{\alpha!} q' \blacktriangleright^2 (w, C'')$.

         Nous avons donc d'après le lemme 5.13, $C \blacktriangleright (C' \blacktriangleright C'') \xrightarrow{\alpha!} v' = C \blacktriangleright (q' \blacktriangleright^2 (w, C''))$ et $(v, v') \in \mathcal{S}$.

      4. $C' \xrightarrow{\alpha?} q' \blacktriangleright w$, $Constante(Nil \blacktriangleright C'') = \emptyset$ et $v = C \blacktriangleright (q' \blacktriangleright^2 (w, Nil \blacktriangleright C''))$.

         Alors $Constante(C'') = \emptyset$ et le lemme 5.11 entraîne que $C' \blacktriangleright C'' \xrightarrow{\alpha?} q' \blacktriangleright^2 (w, C'')$. Nous avons donc d'après le lemme 5.13, $C \blacktriangleright (C' \blacktriangleright C'') \xrightarrow{\alpha!} v' = C \blacktriangleright (q' \blacktriangleright^2 (w, C''))$ et $(v, v') \in \mathcal{S}$.

      5. $C'' \xrightarrow{\alpha} q' \blacktriangleright w$ et $v = C \blacktriangleright (C' \blacktriangleright (Nil \blacktriangleright (q' \blacktriangleright w)))$.

         Alors le lemme 5.13 entraîne que $C' \blacktriangleright C'' \xrightarrow{\alpha} C' \blacktriangleright (q' \blacktriangleright w)$ et $C \blacktriangleright (C' \blacktriangleright C'') \xrightarrow{\alpha} v' = C \blacktriangleright (C' \blacktriangleright (q' \blacktriangleright w))$ et $(v, v') \in \mathcal{S}$.

    – Supposons que $C \blacktriangleright u \xrightarrow{\alpha} v$. Il y a cinq cas qui se traitent de façon analogue aux cinq cas précédents.

  - Nous supposons que la propriété est vraie pour les $C \blacktriangleright^n (t_1, \dots, t_n)$ de taille strictement inférieur à m

    Soit $C \blacktriangleright^n (t_1, \dots, t_n)$ de taille m.

    – Supposons que $C \blacktriangleright^n (t_1, \dots, t_n) \xrightarrow{\alpha} v$. Il y a quatre cas :

1. $C \xrightarrow{\alpha!} q' \blacktriangleright w$ et $v = q' \blacktriangleright^{n+1} (w, t_1, \ldots, t_n)$.
   Alors le lemme 5.11 entraîne que $C \blacktriangleright^n (t_1, \ldots, t_{i-1}, u, t_{i+1}, \ldots, t_n) \xrightarrow{\alpha!} v' = C \blacktriangleright^n (w, t_1, \ldots, t_{i-1}, u, t_{i+1}, \ldots, t_n)$ et $(v, v') \in \mathcal{S}$.
2. $C \xrightarrow{\alpha?} q' \blacktriangleright w$, $v = q' \blacktriangleright^{n+1} (w, t_1, \ldots, t_n)$ et $Constante(t_j) = \emptyset$ pour tout j.
   Alors $Constante(u) = \emptyset$ et le lemme 5.12 entraîne que $C \blacktriangleright^n (t_1, \ldots, t_{i-1}, u, t_{i+1}, \ldots, t_n) \xrightarrow{\alpha?} v' = C \blacktriangleright^n (w, t_1, \ldots, t_{i-1}, u, t_{i+1}, \ldots, t_n)$ et $(v, v') \in \mathcal{S}$.
3. $t_j \xrightarrow{\alpha} t'_j$, $i \neq j$ et $v = C \blacktriangleright^n (t_1, \ldots, t_{j-1}, t'_j, t_{j+1}, \ldots, t_n)$. Alors le lemme 5.13 entraîne que $C \blacktriangleright^n (t_1, \ldots, t_{j-1}, t'_j, t_{j+1}, \ldots, t_{i-1}, u, t_{i+1}, \ldots, t_n) \xrightarrow{\alpha!} v' = C \blacktriangleright^n (t_1, \ldots, t_{j-1}, t'_j, t_{j+1}, \ldots, t_{i-1}, u, t_{i+1}, \ldots, t_n)$ et $(v, v') \in \mathcal{S}'$.
4. $t_i \xrightarrow{\alpha} t'_i$ et $v = C \blacktriangleright^n (t_1, \ldots, t_{i-1}, t'_i, t_{i+1}, \ldots, t_n)$.
   – Si $t_i \searrow u$ avec la règle R2, alors $u \xrightarrow{\alpha} u'$ car $\mathcal{S}$ est une relation de bisimulation. Nous concluons que $C \blacktriangleright^n (t_1, \ldots, t_{i-1}, u, t_{i+1}, \ldots, t_n) \xrightarrow{\alpha} v' = C \blacktriangleright^n (t_1, \ldots, t_{i-1}, u', t_{i+1}, \ldots, t_n)$ grâce au lemme 5.13 et $(v, v') \in \mathcal{S}'$.
   – Si $t_i \searrow u$ avec la règle R3, t_i est plus petit que $C \blacktriangleright^n (t_1, \ldots, t_n)$, donc par hypothèse d'induction, $u \xrightarrow{\alpha} u'$ et $t'_i \searrow u'$. Il s'ensuit que $C \blacktriangleright^n (t_1, \ldots, t_{i-1}, u, t_{i+1}, \ldots, t_n) \xrightarrow{\alpha} v' = C \blacktriangleright^n (t_1, \ldots, t_{i-1}, u', t_{i+1}, \ldots, t_n)$ et $(v, v') \in \mathcal{S}'$.

Dans tous les cas, nous avons exhibé un terme v' tel que $C \blacktriangleright^n (t_1, \ldots, t_{i-1}, u, t_{i+1}, \ldots, t_n) \xrightarrow{\alpha} v'$ et $(v, v') \in \mathcal{S}'$.

– Le cas $C \blacktriangleright^n (t_1, \ldots, t_{i-1}, u, t_{i+1}, \ldots, t_n) \xrightarrow{\alpha} v$ se traite de la même façon que le cas précédent.

Donc $\mathcal{S}'$ est une relation de bisimulation forte.

Les deux relations $\mathcal{S}$ et $\mathcal{S}'$ sont des bisimulations fortes. Il s'ensuit que $\searrow$ est une relation de bisimulation comme union de deux bisimulations (proposition 1(4), page 91 de [Mil89]).

. La relation $\rightleftarrows$ est une bisimulation forte.

Nous montrons de façon semblable à $\searrow$ que les deux relations $\mathcal{S}$ et $\mathcal{S}'$ sont des bisimulations fortes où $\mathcal{S} = \{(C \blacktriangleright^n (t_1, \ldots, t_n), C \blacktriangleright^n (t_1, \ldots, t_{i-1}, t_j, t_{i+1} \ldots, t_{j-1}, t_i, t_{i+1}, \ldots, t_n) \mid C \in Atomes,\ n \in \mathbb{N},\ t_i \in T_G\}$ et $\mathcal{S}' = \{(C \blacktriangleright^n (t_1, \ldots, t_n), C \blacktriangleright^n (t_1, \ldots, t_{i-1}, u, t_{i+1}, \ldots, t_n) \mid t_i \rightleftarrows u,\ C \in Atomes,\ n \in \mathbb{N},\ t_i, u \in T_G\}$.

┘

Proposition 6.20 *Pour tout $\sigma \in Act^*$, $t \overset{\sigma}{\Rightarrow} u$ si et seulement s'il existe un terme u' tel que $t \xrightarrow{\sigma} u'$ et $u' \equiv_{\mathcal{S}} u$.*

Démonstration. Le sens ($\Leftarrow$) découle de la définition de $\Rightarrow$.

Nous allons montrer par récurrence sur la longueur de σ le sens ($\Rightarrow$). Nous supposons que $t \overset{\sigma}{\Rightarrow} u$.

- Si la longueur de σ est nulle $u' = u = t$ convient.
- Supposons qu'il existe un terme u' tel que $t \overset{\sigma}{\rightarrow} u'$ et $u' \equiv_{\mathcal{S}} u$ si la longueur de σ est $n-1$.

  Soit σ de longueur n, $\sigma = \sigma'\alpha$. Nous avons $t \overset{\sigma'}{\Rightarrow} v \overset{\alpha}{\Rightarrow} u$.

  La longueur de σ' est $n-1$, donc par hypothèse de récurrence, il existe un terme v' tel que $t \overset{\sigma'}{\rightarrow} v'$ et $v' \equiv_{\mathcal{S}} v$.

  D'après la définition de $\Rightarrow$, comme $v \overset{\alpha}{\Rightarrow} u$, il existe v_1 et u_1 tels que $v \equiv_{\mathcal{S}} v_1$, $u \equiv_{\mathcal{S}} u_1$ et $v_1 \overset{\alpha}{\rightarrow} u_1$. Nous avons donc $v_1 \overset{\alpha}{\rightarrow} u_1$ et $v' \equiv_{\mathcal{S}} v_1$. Donc d'après le propriété 6.19, il existe un terme v_1' tel que $v' \overset{\alpha}{\rightarrow} v_1'$ et $v_1' \equiv_{\mathcal{S}} u_1$.

  Donc $t \xrightarrow{\sigma'\alpha} v_1'$ et $v_1' \equiv_{\mathcal{S}} u$. D'où il existe un terme w tel que $t \overset{\sigma}{\rightarrow} w$ et $w \equiv_{\mathcal{S}} u$.

$\lrcorner$

Le lemme 6.13, la propriété 6.14 et la propriété précédente permettent d'obtenir une caractérisation précise de l'accessibilité pour $\overset{*}{\Rightarrow}$:

Lemme 6.21 *$u \overset{*}{\Rightarrow} v$ si et seulement s'il existe un terme t tel que $u \overset{*}{\rightarrow} t$ et $t\downarrow \overset{*}{\rightleftarrows} v\downarrow$.*

6.3. Reconnaissabilité des relations SPC

Nous continuons à suivre la démarche de D. Lugiez et Ph. Schnoebelen [LS01]. Nous examinons donc quelles règles de simplification et de permutation sont reconnaissables et laissent reconnaissable un langage reconnaissable.

Définition 6.22 *Soit L un langage d'arbres. Nous définissons les langages suivants :*

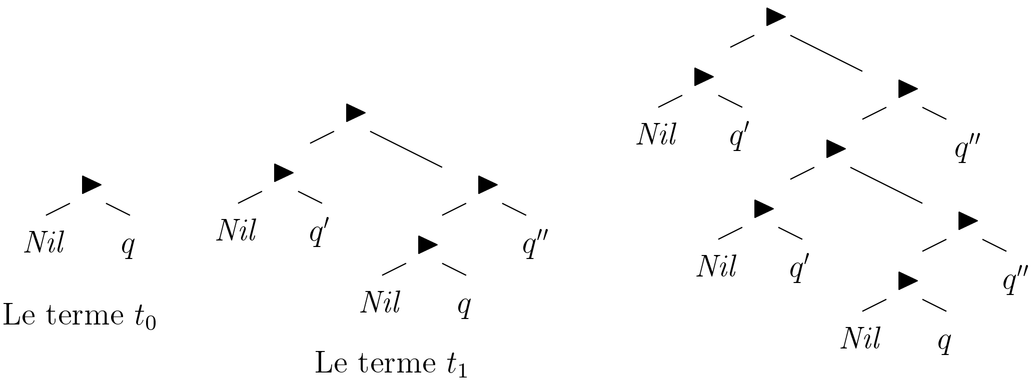
$$
\begin{aligned}
[L]_{\Searrow} &\overset{def}{=} \{u \mid \exists t \in L, t \Searrow^{*} u\}, & [L]_{\downarrow} &\overset{def}{=} \{t\downarrow \mid t \in L\},\\
[L]_{\swarrow} &\overset{def}{=} \{u \mid \exists t \in L, u \searrow^{*} t\}, & [L]_{\Swarrow} &\overset{def}{=} \{u \mid \exists t \in L, u \Searrow^{*} t\},\\
[L]_{\rightleftarrows} &\overset{def}{=} \{u \mid \exists t \in L, u \overset{*}{\rightleftarrows} t\}, & [L]_{\leftrightsquigarrow} &\overset{def}{=} \{u \mid \exists t \in L, u \leftrightsquigarrow t\}.
\end{aligned}
$$

En général, même si L est reconnaissable, $[L]_{\Searrow}$, $[L]_{\downarrow}$, $[L]_{\rightleftarrows}$ et $[L]_{\leftrightsquigarrow}$ ne sont pas reconnaissables comme les exemples suivants l'illustrent.

Exemple 6.23 $[L]_{\downarrow}$ n'est pas toujours reconnaissable.

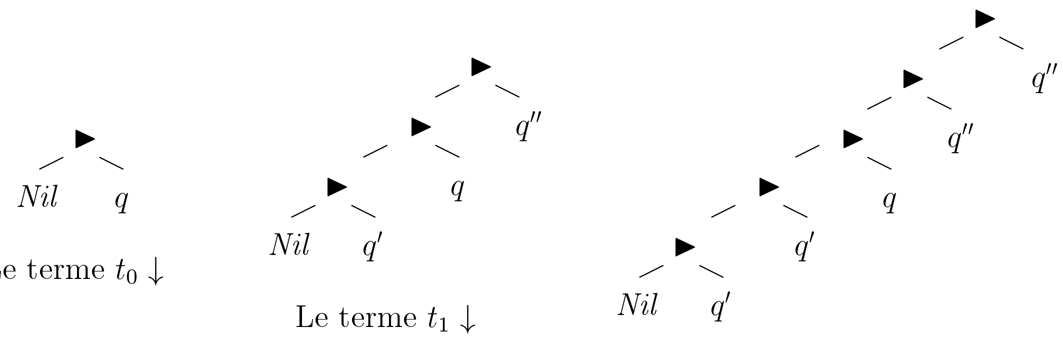
Soient $t_0 = \mathit{Nil} \blacktriangleright q$ et $t_n = \mathit{Nil} \blacktriangleright^2 (q', t_{n-1} \blacktriangleright q'')$ (t_0, t_1 et t_2 sont représentés dans la figure 6.1). Soit $L_0 = \{t_n\}_{n\in\mathbb{N}}$. L_0 est reconnaissable. $[L_0]_{\downarrow} = \{\mathit{Nil} \blacktriangleright^{2n+1}$

$(q', \ldots, q', q, q'', \ldots, q'')$ avec exactement n q' et n $q''\}_{n \in \mathbb{N}}$. Donc, $[L_0]_\downarrow$ n'est pas reconnaissable ($t_0 \downarrow$, $t_1 \downarrow$ et $t_2 \downarrow$ sont représentés dans la figure 6.2).



Le terme t_0 Le terme t_1 Le terme t_2

FIG. 6.1. – Le langage L_0.



Le terme $t_0 \downarrow$ Le terme $t_1 \downarrow$ Le terme $t_2 \downarrow$

FIG. 6.2. – Le langage $[L_0]_\downarrow$.

Exemple 6.24 $[L]_{\Searrow}$ n'est pas toujours reconnaissable.

Nous considérons le langage L_0 définit précédemment. $[L_0]_\downarrow = [L_0]_{\Searrow} \cap [T]_\downarrow$. Or $[T]_\downarrow = \{C \blacktriangleright^n (u_1, \ldots, u_n) \mid$ aucun des u_i ne contient de *Nil* et si $n = 1$, alors $C \neq Nil\}$. Donc $[T]_\downarrow$ est reconnaissable. Il s'ensuit que $[L_0]_{\Searrow}$ n'est pas reconnaissable sinon $[L_0]_\downarrow$ serait reconnaissable.

Exemple 6.25 $[L]_{\rightleftarrows}$ n'est pas toujours reconnaissable.

Soient $L_1 = \{Nil \blacktriangleright^{2n} (q, q', q, q', \ldots, q, q')\}_{n \in \mathbb{N}}$ et $L_2 = \{Nil \blacktriangleright^n (q, q, \ldots, q, q', q', \ldots, q')\}_{n \in \mathbb{N}}$. L_1 et L_2 sont reconnaissables. Or $[L_1]_{\rightleftarrows} \cap L_2 = \{Nil \blacktriangleright^{2n} (q, \ldots, q, q', \ldots, q')$ avec exactement n q et n $q'\}_{n \in \mathbb{N}}$ n'est pas reconnaissable. Donc, $[L_1]_{\rightleftarrows}$ n'est pas reconnaissable.

Voici maintenant des résultats positifs :

Lemme 6.26 *Pour tout terme t, l'ensemble $[t]_{\rightleftarrows}$ est un langage reconnaissable d'arbres et il existe un automate de $O(|t| \times (|t|/2)!)$ états qui reconnait $[t]_{\rightleftarrows}$.*

Remarque 6.27 La complexité exponentielle ne peut pas être évitée. Pour $t = \mathit{Nil} \blacktriangleright^n (X_1, X_2, \ldots, X_n)$, $[t]_{\rightleftarrows} = \{\mathit{Nil} \blacktriangleright^n (X_{\sigma(1)}, X_{\sigma(2)}, \ldots, X_{\sigma(n)}) \mid \sigma$ est une bijection sur $\{1, \ldots, n\}\}$.

Proposition 6.28 *Si L est un langage régulier, alors $[L]_{\swarrow}$ est régulier. De plus, étant donné un automate d'arbres $\mathcal{A}$ qui reconnait L, nous pouvons construire en temps polynomial un automate d'arbres $\mathcal{A}'$ pour $[L]_{\swarrow}$ tel que $|\mathcal{A}'| = O(|\mathcal{A}|^2)$.*

Soit L un langage reconnaissable et $\mathcal{A}$ un automate d'arbres qui reconnait L. Nous considérons l'automate $\mathcal{A}'$ obtenu de la façon suivante :

- Nous ajoutons à $\mathcal{A}$ $O(|\mathcal{A}|^2)$ états que nous notons r_p^s où p et s sont deux états de $\mathcal{A}$.
- Nous ajoutons un état p_{Nil}.
- Nous ajoutons les neuf types de règles suivants aux règles de $\mathcal{A}$:
  1. $\mathit{Nil} \mapsto_{\mathcal{A}'} p_{\mathit{Nil}}$,
  2. $p_{\mathit{Nil}} \blacktriangleright p_{\mathit{Nil}} \mapsto_{\mathcal{A}'} p_{\mathit{Nil}}$,
  3. $p \blacktriangleright p_{\mathit{Nil}} \mapsto_{\mathcal{A}'} p$, pour tout état p de $\mathcal{A}$,
  4. $r_p^s \blacktriangleright p_{\mathit{Nil}} \mapsto_{\mathcal{A}'} r_p^s$, pour tous les états p, s de $\mathcal{A}$,
  5. $p_{\mathit{Nil}} \blacktriangleright s \mapsto_{\mathcal{A}'} r_p^{s'}$, pour tous les états p, s, s' de $\mathcal{A}$ tels que $p \blacktriangleright s \mapsto_{\mathcal{A}} s'$,
  6. $p_{\mathit{Nil}} \blacktriangleright r_p^s \mapsto_{\mathcal{A}'} r_p^s$, pour tous les états p, s de $\mathcal{A}$,
  7. $p \blacktriangleright r_p^s \mapsto_{\mathcal{A}'} s$, pour tous les états p, s de $\mathcal{A}$,
  8. $r_p^s \blacktriangleright r_s^{s'} \mapsto_{\mathcal{A}'} r_p^{s'}$, pour tous les états p, s, s' de $\mathcal{A}$,
  9. $r_p^s \blacktriangleright p' \mapsto_{\mathcal{A}'} r_p^{s'}$, pour tous les états p, s, p', s' de $\mathcal{A}$ tels que si $s \blacktriangleright p' \mapsto_{\mathcal{A}} s'$.

Nous voulons montrer que $L(\mathcal{A}') = [L]_{\swarrow}$. Nous allons établir le résultat suivant qui est plus fort.

Proposition 6.29 *Soient t un terme et p un état de l'automate $\mathcal{A}$, $t \mapsto^*_{\mathcal{A}'} p$ si et seulement s'il existe un terme u tel que $t \searrow^* u$ et $u \mapsto^*_{\mathcal{A}} p$.*

Pour montrer cette proposition, nous utilisons le lemme qui suit.

Lemme 6.30 *$u \searrow^* t$ si et seulement s'il existe des entiers $\alpha_1, \ldots, \alpha_n$ et des termes $t_{i,j}$ tels que $u = \blacktriangleright^n (u_1, \ldots, u_n)$, $t = \blacktriangleright^{\alpha_1+\cdots+\alpha_n} (t_{1,1}, \ldots, t_{1,\alpha_1}, \ldots, t_{n,1}, \ldots, t_{n,\alpha_n})$ et pour tout i,*

- *Si $\alpha_i = 0$, alors $u_i \searrow^* Nil$.*
- *Si $\alpha_i = 1$, alors $u_i \searrow^* t_{i,1}$ ou $u_i \searrow^* Nil \blacktriangleright t_{i,1}$.*
- *Si $\alpha_i > 1$, alors $u_i \searrow^* Nil \blacktriangleright^{\alpha_i} (t_{i,1}, \ldots, t_{i,\alpha_i})$.*

Remarque 6.31 Le lemme 6.30 peut se reformuler en :
$u \searrow^* t$ si et seulement si $u = c \blacktriangleright^n (u_1, \ldots, u_n)$, $t = c \blacktriangleright^{\alpha_1+\cdots+\alpha_n} (t_{1,1}, \ldots, t_{1,\alpha_1}, \ldots, t_{n,1}, \ldots, t_{n,\alpha_n})$ et pour tout i, $u_i \searrow^* Nil \blacktriangleright^{\alpha_i} (t_{i,1}, \ldots, t_{i,\alpha_i})$ ou si $\alpha_i = 1$, $u_i \searrow^* t_{i,1}$.

Démonstration.

($\Rightarrow$) : Nous montrons que $u \searrow^* t$ entraîne que $u = c \blacktriangleright^n (u_1, \ldots, u_n)$, $t = c \blacktriangleright^{\alpha_1+\cdots+\alpha_n} (t_{1,1}, \ldots, t_{1,\alpha_1}, \ldots, t_{n,1}, \ldots, t_{n,\alpha_n})$ et pour tout i, $u_i \searrow^* Nil \blacktriangleright^{\alpha_i} (t_{i,1}, \ldots, t_{i,\alpha_i})$ ou si $\alpha_i = 1$, $u_i \searrow^* t_{i,1}$ par récurrence sur la longueur de la simplification.

Supposons que $u \searrow^* t$. Nous avons $u = c \blacktriangleright^n (u_1, \ldots, u_n)$.

- Si $u = v$, alors il y deux cas.
  - Si $u \in Atomes$, nous avons la propriété avec $n = 0$.
  - Si $u \notin Atomes$, nous avons la propriété avec $\alpha_i = 1$ et $t_{i,1} = u_i$ pour tout i.

- Si $u \searrow v \searrow^* t$, alors la réduction entre u et v a été obtenue en appliquant la règle R2 ou la règle R3.
  - Si la règle R2 a été appliquée :
    $u_i = Nil \blacktriangleright^m (w_1, \ldots, w_m)$ et $v = c \blacktriangleright^{n+m-1} (u_1, \ldots, u_{i-1}, w_1, \ldots, w_m, u_{i+1}, \ldots, u_n)$ avec $m \geq 0$.
    Nous avons $v \searrow^* t$ et la longueur de cette simplification est plus petite que celle entre u et t. Donc, par hypothèse de récurrence, $t = c \blacktriangleright^{\alpha_1+\cdots+\alpha_{i-1}+\beta_1+\cdots+\beta_m+\alpha_{i+1}+\cdots+\alpha_n} (t_{1,1}, \ldots, t_{1,\alpha_1}, \ldots, t_{i-1,1}, \ldots, t_{i-1,\alpha_{i-1}}, t'_{1,1}, \ldots, t'_{1,\beta_1}, \ldots, t'_{m,1}, \ldots, t_{m,\beta_m}, \ldots, t_{i+1,1}, \ldots, t_{i+1,\alpha_{i+1}}, t_{n,1}, \ldots, t_{n,\alpha_n})$ et pour tout $j \neq i$, $u_j \searrow^* Nil \blacktriangleright^{\alpha_j} (t_{j,1}, \ldots, t_{j,\alpha_j})$ ou si $\alpha_j = 1$, $u_j \searrow^* t_{j,1}$ et pour tout j, $w_j \searrow^* Nil \blacktriangleright^{\beta_j} (t'_{j,1}, \ldots, t'_{j,\beta_j})$ ou si $\beta_j = 1$, $w_j \searrow^* t'_{j,1}$.
    $u_i = Nil \blacktriangleright^m (w_1, \ldots, w_m)$ et pour tout j, $w_j \searrow^* Nil \blacktriangleright^{\beta_j} (t'_{j,1}, \ldots, t'_{j,\beta_j})$ ou si $\beta_j = 1$, $w_j \searrow^* t'_{j,1}$. La règle R3 appliquée maintes fois entraîne que $u_i \searrow^* Nil \blacktriangleright^m (w'_1, \ldots, w'_m)$ où $w'_j = Nil \blacktriangleright^{\beta_j} (t'_{j,1}, \ldots, t'_{j,\beta_j})$ ou si $\beta_j = 1$, $w'_j = Nil \blacktriangleright t'_{j,1}$.
    La règle R2 appliquée successivement à $w'_1, \ldots, w'_m$ si nécessaire entraîne que $u_i \searrow^* Nil \blacktriangleright^{\beta_1+\cdots+\beta_m} (t'_{1,1}, \ldots, t'_{1,\beta_1}, \ldots, t'_{m,1}, \ldots, t'_{m,\beta_m})$.
    Donc la propriété est vérifiée.

- Si la règle R3 a été appliquée :
  $u_i \searrow w$ et $v = c \blacktriangleright^n (u_1, \ldots, u_{i-1}, w, u_{i+1}, \ldots, u_n)$. La longueur de cette simplification entre v et t est plus petite que celle entre u et t, donc par hypothèse de récurrence, $t = c \blacktriangleright^{\alpha_1+\cdots+\alpha_n} (t_{1,1}, \ldots, t_{1,\alpha_1}, \ldots, t_{n,1}, \ldots, t_{n,\alpha_n})$ et pour tout $j \neq i$, $u_j \searrow^* \mathit{Nil} \blacktriangleright^{\alpha_j} (t_{j,1}, \ldots, t_{j,\alpha_j})$ ou si $\alpha_j = 1$, $u_j \searrow^* t_{j,1}$ et $w \searrow^* \mathit{Nil} \blacktriangleright^{\alpha_i} (t_{i,1}, \ldots, t_{i,\alpha_i})$ ou si $\alpha_i = 1$, $w \searrow^* t_{i,1}$.
  Donc $u_i \searrow^* \mathit{Nil} \blacktriangleright^{\alpha_i} (t_{i,1}, \ldots, t_{i,\alpha_i})$ ou si $\alpha_i = 1$, $u_i \searrow^* t_{i,1}$. Dans tous les cas, la propriété est vérifiée pour u et t.

($\Leftarrow$) : Si $u = c \blacktriangleright^n (u_1, \ldots, u_n)$ et $t = c \blacktriangleright^{\alpha_1+\cdots+\alpha_n} (t_{1,1}, \ldots, t_{1,\alpha_1}, \ldots, t_{n,1}, \ldots, t_{n,\alpha_n})$ avec pour tout i, $u_i \searrow^* \mathit{Nil} \blacktriangleright^{\alpha_i} (t_{i,1}, \ldots, t_{i,\alpha_i})$ ou si $\alpha_i = 1$, $u_i \searrow^* t_{i,1}$.
La règle R3 entraîne que $u \searrow^* c \blacktriangleright^n (w_1, \ldots, w_n)$ avec tout i, $u_i = \mathit{Nil} \blacktriangleright^{\alpha_i} (t_{i,1}, \ldots, t_{i,\alpha_i})$ ou si $\alpha_i = 1$, $w_i = t_{i,1}$. La règle R2 entraîne $u \searrow^* c \blacktriangleright^{\alpha_1+\cdots+\alpha_n} (t_{1,1}, \ldots, t_{1,\alpha_1}, \ldots, t_{n,1}, \ldots, t_{n,\alpha_n})$.
Nous en concluons que $u \searrow^* t$.

$\lrcorner$

Nous allons aussi utiliser le lemme 6.30 sous la forme suivante :

Corollaire 6.32 *$t \blacktriangleright t' \searrow^* u$ si et seulement si $u = v \blacktriangleright^\alpha (w_1, \ldots, w_\alpha)$ avec $t \searrow^* v$ et $t' \searrow^* \mathit{Nil} \blacktriangleright^\alpha (w_1, \ldots, w_\alpha)$ ou si $\alpha = 1$, $t' \searrow^* w_1$.*

Le lemme suivant entraîne le premier sens de la proposition 6.29 ($t \mapsto^*_{\mathcal{A}'} p$ s'il existe un terme u tel que $t \searrow^* u$ et $u \mapsto^*_{\mathcal{A}} p$).

Lemme 6.33 *Si $t \mapsto^*_{\mathcal{A}'}$ alors*
- *Soit $t \mapsto^*_{\mathcal{A}'} p_{\mathit{Nil}}$ et $t \searrow^* \mathit{Nil}$.*
- *Soit $t \mapsto^*_{\mathcal{A}'} p$ et il existe un terme u tel que $t \searrow^* u$ et $u \mapsto^*_{\mathcal{A}} p$.*
- *Soit $t \mapsto^*_{\mathcal{A}'} r^s_p$, $t = \mathit{Nil} \blacktriangleright^n (t_1, \ldots, t_n)$ avec $n > 0$ et pour tout terme v tel que $v \mapsto^*_{\mathcal{A}} p$, il existe un terme u tel que $v \blacktriangleright^n (t_1, \ldots, t_n) \searrow^* u$ et $u \mapsto^*_{\mathcal{A}} s$.*

Démonstration. Nous supposons que $t \overset{k}{\mapsto}_{\mathcal{A}'}$ et nous allons montrer la propriété par récurrence sur k qui est la longueur du calcul.

- Si $k = 1$, alors $t \in \mathit{Atomes}$.
  La règle 1 est la seule règle propre à $\mathcal{A}'$ qui s'applique à Nil ou une constante de processus. Donc, soit $t = \mathit{Nil}$ et la propriété est vérifiée, soit le calcul a été fait en utilisant une règle de $\mathcal{A}$, d'où $t \mapsto_{\mathcal{A}} p$ et nous avons de nouveau la propriété.
- Si $k > 1$ et la dernière étape du calcul est une règle de $\mathcal{A}$:

Alors $t \mapsto^*_{\mathcal{A}'} p_1 \blacktriangleright p_2 \mapsto_{\mathcal{A}} p$. Par conséquent, $t = t_1 \blacktriangleright t_2$ avec $t_1 \mapsto^*_{\mathcal{A}'} p_1$ et $t_2 \mapsto^*_{\mathcal{A}'} p_2$. Par hypothèse de récurrence comme les calculs pour t_1 et t_2 sont moins longs que celui pour t, il existe deux termes u_1 et u_2 tels que $t_1 \searrow^* u_1$, $t_2 \searrow^* u_2$, $u_1 \mapsto^*_{\mathcal{A}} p_1$ et $u_2 \mapsto^*_{\mathcal{A}} p_2$.

D'après le corollaire 6.32, $t \searrow^* u_1 \blacktriangleright u_2$. De plus, $u_1 \blacktriangleright u_2 \mapsto^*_{\mathcal{A}} p_1 \blacktriangleright p_2 \mapsto_{\mathcal{A}} p$. Donc la propriété est vérifiée avec $u = u_1 \blacktriangleright u_2$.

- Si $k > 1$ et la dernière étape du calcul est une règle du type 2 de $\mathcal{A}'$:

  Alors $t \mapsto^*_{\mathcal{A}'} p_{Nil} \blacktriangleright p_{Nil} \mapsto_{\mathcal{A}'} p_{Nil}$. Par conséquent, $t = t_1 \blacktriangleright t_2$ avec $t_1 \mapsto^*_{\mathcal{A}'} p_{Nil}$ et $t_2 \mapsto^*_{\mathcal{A}'} p_{Nil}$. Par hypothèse de récurrence, comme les calculs pour t_1 et t_2 sont moins longs que celui pour t, nous avons $t_1 \searrow^* Nil$ et $t_2 \searrow^* Nil$.

  D'après le corollaire 6.32, $Nil \blacktriangleright Nil \searrow Nil$. Par conséquent, $t \searrow^* Nil$ et la propriété est vérifiée.

- Si $k > 1$ et la dernière étape du calcul est une règle du type 3 de $\mathcal{A}'$:

  Alors $t \mapsto^*_{\mathcal{A}'} p \blacktriangleright p_{Nil} \mapsto_{\mathcal{A}'} p$. Par conséquent, $t = t_1 \blacktriangleright t_2$ avec $t_1 \mapsto^*_{\mathcal{A}'} p$ et $t_2 \mapsto^*_{\mathcal{A}'} p_{Nil}$. Par hypothèse de récurrence, il existe un terme u_1 tel que $t_1 \searrow^* u_1$ et $u_1 \mapsto^*_{\mathcal{A}} p$, d'une part et $t_2 \searrow^* Nil$, d'autre part.

  D'après le corollaire 6.32, $t \searrow^* u_1$. Donc la propriété est vérifiée avec $u = u_1$.

- Si $k > 1$ et la dernière étape du calcul est une règle du type 4 de $\mathcal{A}'$:

  Alors $t \mapsto^*_{\mathcal{A}'} r^s_p \blacktriangleright p_{Nil} \mapsto_{\mathcal{A}'} p$. Par conséquent, $t = t_1 \blacktriangleright t_2$ avec $t_1 \mapsto^*_{\mathcal{A}'} r^s_p$ et $t_2 \mapsto^*_{\mathcal{A}'} p_{Nil}$.

  Soit t' un terme tel que $t' \mapsto^*_{\mathcal{A}} p$. Par hypothèse de récurrence, $t_1 = Nil \blacktriangleright^m (v_1, \ldots, v_m)$ avec $m > 0$ et il existe un terme u_1 tel que $t' \blacktriangleright^m (v_1, \ldots, v_m) \searrow^* u_1$ et $u_1 \mapsto^*_{\mathcal{A}} s$.

  Par ailleurs, $t_2 \mapsto^*_{\mathcal{A}'} p_{Nil}$, donc par hypothèse de récurrence, $t_2 \searrow^* Nil$.

  Nous avons donc $t = (Nil \blacktriangleright^m (v_1, \ldots, v_m)) \blacktriangleright t_2$ ce qui se réécrit $t = Nil \blacktriangleright^{m+1} (v_1, \ldots, v_m, t_2)$. Nous considérons alors $t' \blacktriangleright^{m+1} (v_1, \ldots, v_m, t_2)$.

  D'après le corollaire 6.32, $t' \blacktriangleright^{m+1} (v_1, \ldots, v_m, t_2) \searrow^* u_1$. Nous en déduisons que la propriété est vérifiée avec $u = u_1$.

- Si $k > 1$ et la dernière étape du calcul est une règle du type 5 de $\mathcal{A}'$:

  Alors $t \mapsto^*_{\mathcal{A}'} p_{Nil} \blacktriangleright s \mapsto_{\mathcal{A}'} r^{s'}_p$ et $p \blacktriangleright s \mapsto_{\mathcal{A}} s'$. Par conséquent, $t = t_1 \blacktriangleright t_2$ avec $t_1 \mapsto^*_{\mathcal{A}'} p_{Nil}$ et $t_2 \mapsto^*_{\mathcal{A}'} s$. Par hypothèse de récurrence comme les calculs pour pour t_1 et t_2 sont moins longs que celui pour t, $t_1 \searrow^* Nil$ et il existe un terme u_2 tel que $t_2 \searrow^* u_2$ et $u_2 \mapsto^*_{\mathcal{A}} s$.

  Soit v un terme tel que $v \mapsto^*_{\mathcal{A}} p$. Comme $t_1 \searrow^* Nil$, d'après le corollaire 6.32, soit $t_1 = Nil$, soit $t_1 = Nil \blacktriangleright^m (v_1, \ldots, v_m)$ avec $w_i \searrow^* Nil$ pour tout i.

  - Si $t_1 = Nil$:

    Alors $t = Nil \blacktriangleright t_2$, nous considérons alors $v \blacktriangleright u_2$. Nous avons $v \blacktriangleright u_2 \mapsto^*_{\mathcal{A}} p \blacktriangleright s \mapsto_{\mathcal{A}} s'$. De plus, $v \blacktriangleright t_2 \searrow v \blacktriangleright u_2$ d'après le corollaire 6.32. Donc la propriété est vérifiée avec $u = v \blacktriangleright u_2$.

- Si $t_1 = Nil \blacktriangleright^m (v_1, \ldots, v_m)$ avec $w_i \searrow^* Nil$ pour tout i :

  Nous avons donc $t = (Nil \blacktriangleright^m (v_1, \ldots, v_m)) \blacktriangleright t_2$ ce qui se réécrit $t = Nil \blacktriangleright^{m+1} (v_1, \ldots, v_m, t_2)$. Nous considérons alors $v \blacktriangleright^{m+1} (v_1, \ldots, v_m, t_2)$ qui se réécrit $(v \blacktriangleright^m (v_1, \ldots, v_m)) \blacktriangleright t_2$.

  Nous avons $(v \blacktriangleright^m (v_1, \ldots, v_m)) \blacktriangleright t_2 \searrow^* v \blacktriangleright t_2$ d'après le corollaire 6.32. De plus, $v \blacktriangleright t_2 \mapsto^*_{\mathcal{A}'} p \blacktriangleright s \mapsto_{\mathcal{A}} s'$. Donc la propriété est vérifiée avec $u = v \blacktriangleright t_2$.

- Si $k > 1$ et la dernière étape du calcul est une règle du type 6 de $\mathcal{A}'$:

  Alors $t \mapsto^*_{\mathcal{A}'} p_{Nil} \blacktriangleright r^s_p \mapsto_{\mathcal{A}'} p$. Par conséquent, $t = t_1 \blacktriangleright t_2$ avec $t_1 \mapsto^*_{\mathcal{A}'} p_{Nil}$ et $t_2 \mapsto^*_{\mathcal{A}'} r^s_p$.

  Soit t' un terme tel que $t' \mapsto^*_{\mathcal{A}} p$. Par hypothèse de récurrence, $t_2 = Nil \blacktriangleright^\alpha (w_1, \ldots, w_\alpha)$ avec $\alpha > 0$ et il existe un terme u_2 tel que $t' \blacktriangleright^\alpha (w_1, \ldots, w_\alpha) \searrow^* u_2$ et $u_2 \mapsto^*_{\mathcal{A}} s$.

  Comme $t_1 \searrow^* Nil$, d'après le corollaire 6.32, soit $t_1 = Nil$, soit $t_1 = Nil \blacktriangleright^m (v_1, \ldots, v_m)$ avec $w_i \searrow^* Nil$ pour tout i.

  - Si $t_1 = Nil$:

    Alors $t = Nil \blacktriangleright (Nil \blacktriangleright^\alpha (w_1, \ldots, w_\alpha))$. D'après le corollaire 6.32, $t' \blacktriangleright (Nil \blacktriangleright^\alpha (w_1, \ldots, w_\alpha, Nil)) \searrow t' \blacktriangleright^\alpha (w_1, \ldots, w_\alpha) \searrow^* u_1$. Nous en déduisons que la propriété est vérifiée avec $u = u_1$.

  - Si $t_1 = Nil \blacktriangleright^m (v_1, \ldots, v_m)$ avec $w_i \searrow^* Nil$ pour tout i :

    Alors $t = (Nil \blacktriangleright^m (v_1, \ldots, v_m)) \blacktriangleright t_2$ ce qui se réécrit $t = Nil \blacktriangleright^{m+1} (v_1, \ldots, v_m, t_2)$. Nous considérons alors $t' \blacktriangleright^{m+1} (v_1, \ldots, v_m, t_2)$ qui se réécrit $(t' \blacktriangleright^{m+1} (v_1, \ldots, v_m, Nil \blacktriangleright^\alpha (w_1, \ldots, w_\alpha))$.

    Nous avons $t' \blacktriangleright^{m+1} (v_1, \ldots, v_m, Nil \blacktriangleright^\alpha (w_1, \ldots, w_\alpha)) \searrow^* t' \blacktriangleright^\alpha (w_1, \ldots, w_\alpha)$ d'après le corollaire 6.32. Par suite, $t' \blacktriangleright^{m+1} (v_1, \ldots, v_m, Nil \blacktriangleright^\alpha (w_1, \ldots, w_\alpha)) \searrow^* u_2$. Donc la propriété est vérifiée avec $u = u_2$.

- Si $k > 1$ et la dernière étape du calcul est une règle du type 7 de $\mathcal{A}'$:

  Alors $t \mapsto^*_{\mathcal{A}'} p \blacktriangleright r^s_p \mapsto_{\mathcal{A}'} s$. Par conséquent, $t = t_1 \blacktriangleright t_2$ avec $t_1 \mapsto^*_{\mathcal{A}'} p$ et $t_2 \mapsto^*_{\mathcal{A}'} r^s_p$. Par hypothèse de récurrence, comme le calcul pour t_1 est moins long que celui pour t, il existe un terme u_1 tel que $t_1 \searrow^* u_1$ et $u_1 \mapsto^*_{\mathcal{A}} s$.

  Par hypothèse de récurrence comme le calcul pour t_2 est moins long que celui pour t, $t_2 = Nil \blacktriangleright^n (w_1, \ldots, w_n)$ avec $n > 0$ et il existe un terme u_2 tel que $u_1 \blacktriangleright^n (w_1, \ldots, w_n) \searrow^* u_2$ et $u_2 \mapsto^*_{\mathcal{A}} s$.

  Nous avons $t = t_1 \blacktriangleright (Nil \blacktriangleright^n (w_1, \ldots, w_n))$. Par conséquent, d'après le corollaire 6.32, $t \searrow^* u_1 \blacktriangleright^n (w_1, \ldots, w_n)$. D'où $t \searrow^* u_2$.

  Par ailleurs, nous venons de voir que $u_2 \mapsto^*_{\mathcal{A}} s$, nous en déduisons que la propriété est vérifiée avec $u = u_2$.

- Si $k > 1$ et la dernière étape du calcul est une règle du type 8 de $\mathcal{A}'$:

Alors $t \mapsto^*_{\mathcal{A}'} r^s_p \blacktriangleright r^{s'}_s \mapsto_{\mathcal{A}'} r^{s'}_p$. Par conséquent, $t = t_1 \blacktriangleright t_2$ avec $t_1 \mapsto^*_{\mathcal{A}'} r^s_p$ et $t_2 \mapsto^*_{\mathcal{A}'} r^{s'}_s$. Par hypothèse de récurrence comme les calculs pour t_1 et t_2 sont moins long que celui pour t, $t_1 = Nil \blacktriangleright^m (v_1, \dots, v_m)$ et $t_2 = Nil \blacktriangleright^n (w_1, \dots, w_n)$ avec $m > 0$ et $n > 0$.

Soit t' un terme tel que $t' \mapsto^*_{\mathcal{A}} p$. Par hypothèse de récurrence, il existe un terme u_1 tel que $t' \blacktriangleright^m (v_1, \dots, v_m) \searrow^* u_1$ et $u_1 \mapsto^*_{\mathcal{A}} s$. Il existe un terme u_2 tel que $u_1 \blacktriangleright^n (w_1, \dots, w_n) \searrow^* u_2$ et $u_2 \mapsto^*_{\mathcal{A}} s'$.

Nous avons donc $t = (Nil \blacktriangleright^m (v_1, \dots, v_m)) \blacktriangleright (Nil \blacktriangleright^n (w_1, \dots, w_n))$ ce qui se réécrit $t = Nil \blacktriangleright^{m+1} (v_1, \dots, v_m, Nil \blacktriangleright^n (w_1, \dots, w_n))$. Nous considérons alors $u = t' \blacktriangleright^{m+1} (v_1, \dots, v_m, Nil \blacktriangleright^n (w_1, \dots, w_n))$, ce qui se réécrit $u = (t' \blacktriangleright^m (v_1, \dots, v_m)) \blacktriangleright (Nil \blacktriangleright^n (w_1, \dots, w_n))$.

D'après le corollaire 6.32, $u \searrow^* u_1 \blacktriangleright^n (w_1, \dots, w_n)$. Donc $u \searrow^* u_2$. Nous en déduisons que la propriété est vérifiée puisque $u_2 \mapsto^*_{\mathcal{A}} s'$.

. Si $k > 1$ et la dernière étape du calcul est une règle du type 9 de $\mathcal{A}'$:

Alors $t \mapsto^*_{\mathcal{A}'} r^s_p \blacktriangleright p' \mapsto_{\mathcal{A}'} r^{s'}_p$ et $s \blacktriangleright p' \mapsto_{\mathcal{A}} s'$. Par conséquent, $t = t_1 \blacktriangleright t_2$ avec $t_1 \mapsto^*_{\mathcal{A}'} r^s_p$ et $t_2 \mapsto^*_{\mathcal{A}'} p'$.

Soit t' un terme tel que $t' \mapsto^*_{\mathcal{A}} p$. Par hypothèse de récurrence, $t_1 = Nil \blacktriangleright^m (v_1, \dots, v_m)$ avec $m > 0$ et il existe un terme u_1 tel que $t' \blacktriangleright^m (v_1, \dots, v_m) \searrow^* u_1$ et $u_1 \mapsto^*_{\mathcal{A}} s$. Par hypothèse de récurrence, il existe un terme u_2 tel que $t_2 \searrow^* u_2$ et $u_2 \mapsto^*_{\mathcal{A}} p'$.

Nous avons donc $t = (Nil \blacktriangleright^m (v_1, \dots, v_m)) \blacktriangleright t_2$ ce qui se réécrit $t = Nil \blacktriangleright^{m+1} (v_1, \dots, v_m, t_2)$. Nous considérons alors $t' \blacktriangleright^{m+1} (v_1, \dots, v_m, t_2)$, ce qui se réécrit $(t' \blacktriangleright^m (v_1, \dots, v_m)) \blacktriangleright t_2$.

D'après le corollaire 6.32, $t' \blacktriangleright^{m+1} (v_1, \dots, v_m, t_2) \searrow^* u_1 \blacktriangleright u_2$. Par ailleurs, $u_1 \blacktriangleright u_2 \mapsto^*_{\mathcal{A}} s \blacktriangleright p' \mapsto_{\mathcal{A}} s'$. Nous en déduisons que la propriété est vérifiée avec $u = u_1 \blacktriangleright u_2$.

$\lrcorner$

Le lemme suivant entraîne le deuxième sens de la proposition 6.29 (s'il existe un terme u tel que $t \searrow^* u$ et $u \mapsto^*_{\mathcal{A}} p$, alors $t \mapsto^*_{\mathcal{A}'} p$).

Lemme 6.34 *Soit t un terme, s'il existe un terme u tel que* $t \searrow^* u$ *et si* $u \mapsto^*_{\mathcal{A}'} P$ *avec* $P = p_{Nil}$ *ou* $P = p$ *ou* $P = r^s_p$*, alors* $t \mapsto^*_{\mathcal{A}'} P$.

Démonstration. Soit t un terme tel qu'il existe un terme u tel que $t \searrow^* u$ et $u \mapsto^*_{\mathcal{A}'} P$. Nous allons montrer par induction sur t que $t \mapsto^*_{\mathcal{A}'} P$.

- Si $t \in Atomes$, nous avons immédiatement $t \mapsto^*_{\mathcal{A}'} P$.

- Nous supposons maintenant que $t = t_1 \blacktriangleright t_2$, alors d'après le corollaire 6.32, $u = u_1 \blacktriangleright^\alpha (w_1, \dots, w_\alpha)$, $t_1 \searrow^* u_1$ et
  - Soit $\alpha = 1$ et $t_2 \searrow^* w_1$,

– Soit $t_2 \searrow^* Nil \blacktriangleright^\alpha (w_1, \ldots, w_\alpha)$.

. Si $t_1 \neq u_1$:

Alors $t_1 \searrow t'_1 \searrow^* u_1$. Le corollaire 6.32 entraîne que $t'_1 \blacktriangleright t_2 \searrow^* u_1 \blacktriangleright^\alpha (w_1, \ldots, w_\alpha)$. Or t'_1 est plus petit que t_1, par conséquent, $t'_1 \blacktriangleright t_2$ est plus petit que t. De plus, $u \mapsto^*_{\mathcal{A}'} P$, donc par hypothèse d'induction, $t'_1 \blacktriangleright t_2 \mapsto^*_{\mathcal{A}'} P$.

Il s'ensuit que $t'_1 \mapsto^*_{\mathcal{A}'} P_1$ et $t_2 \mapsto^*_{\mathcal{A}'} P_2$ avec $P_1 \blacktriangleright P_2 \mapsto^*_{\mathcal{A}'} P$. Or $t_1 \searrow t'_1$ et t'_1 est plus petit que t_1, donc par hypothèse d'induction, $t_1 \mapsto^*_{\mathcal{A}'} P_1$.

D'où $t_1 \blacktriangleright t_2 \mapsto^*_{\mathcal{A}'} P_1 \blacktriangleright P_2$ et par suite, $t \mapsto^*_{\mathcal{A}'} P$.

. Si $t_2 \searrow^* w_1$ et $t_2 \neq w_1$:

Alors $t_2 \searrow t'_2 \searrow^* w_1$. Le corollaire 6.32 entraîne que $t_1 \blacktriangleright t'_2 \searrow^* u_1 \blacktriangleright w_1$. Donc par hypothèse d'induction, $t_1 \blacktriangleright t'_2 \mapsto^*_{\mathcal{A}'} P$.

Il s'ensuit que $t_1 \mapsto^*_{\mathcal{A}'} P_1$ et $t'_2 \mapsto^*_{\mathcal{A}'} P_2$ avec $P_1 \blacktriangleright P_2 \mapsto^*_{\mathcal{A}'} P$. Or $t_2 \searrow t'_2$ donc par hypothèse d'induction, $t_2 \mapsto^*_{\mathcal{A}'} P_2$.

D'où $t_1 \blacktriangleright t_2 \mapsto^*_{\mathcal{A}'} P_1 \blacktriangleright P_2$ et par suite, $t \mapsto^*_{\mathcal{A}'} P$.

. Si $t_2 \searrow^* Nil \blacktriangleright^\alpha (w_1, \ldots, w_\alpha)$ et $t_2 \neq Nil \blacktriangleright^\alpha (w_1, \ldots, w_\alpha)$:

Nous montrons que $t \mapsto^*_{\mathcal{A}'} P$ de façon analogue au cas précédent.

. Si $t_1 = u_1$ et $t_2 = w_1$:

Alors $t = u$ et nous avons immédiatement $t \mapsto^*_{\mathcal{A}'} P$.

. Si $t_1 = u_1$ et $t_2 = Nil \blacktriangleright^\alpha (w_1, \ldots, w_\alpha)$:

Alors $u = t_1 \blacktriangleright^\alpha (w_1, \ldots, w_\alpha)$.

Nous allons montrer le lemme suivant :

Lemme 6.35 *Si $t_1 \blacktriangleright^\alpha (w_1, \ldots, w_\alpha) \mapsto^*_{\mathcal{A}'} P$, alors il existe deux états de $\mathcal{A}'$ P' et P'' tels que :*
– $t_1 \mapsto^*_{\mathcal{A}'} P'$,
– $Nil \blacktriangleright^\alpha (w_1, \ldots, w_\alpha) \mapsto^*_{\mathcal{A}'} P''$,
– $P' \blacktriangleright P'' \mapsto^*_{\mathcal{A}'} P$,
– $P'' = p_{Nil}$ *ou* $P'' = r^s_p$ *où s et p sont deux états de $\mathcal{A}$.*

En appliquant ce lemme, nous avons $t_1 \blacktriangleright t_2 \mapsto^*_{\mathcal{A}'} P' \blacktriangleright P''$ et donc $t \mapsto^*_{\mathcal{A}'} P$.

Démonstration. Nous allons montrer le lemme par récurrence sur α.

• Si $\alpha = 0$, alors $P' = P$ et $P'' = p_{Nil}$ conviennent. Nous avons $P' \blacktriangleright P'' \mapsto^*_{\mathcal{A}'} P$ en raison des types 2, 3 et 4.

• Nous supposons que le lemme est vrai pour $\alpha - 1$.

Si $t_1 \blacktriangleright^\alpha (w_1, \ldots, w_\alpha) \mapsto^*_{\mathcal{A}'} P$. Alors $(t_1 \blacktriangleright^{\alpha-1} (w_1, \ldots, w_{\alpha-1})) \blacktriangleright w_\alpha \mapsto^*_{\mathcal{A}'} P$. Par conséquent, il existe deux états de $\mathcal{A}'$ Q' et Q'' tels que $t_1 \blacktriangleright^{\alpha-1} (w_1, \ldots, w_{\alpha-1})) \mapsto^*_{\mathcal{A}'} Q'$, $w_\alpha \mapsto^*_{\mathcal{A}'} Q''$ et $Q' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} P$.

D'après l'hypothèse de récurrence, il existe deux états de $\mathcal{A}'$ P' et P'' tels que :
- $t_1 \mapsto^*_{\mathcal{A}'} P'$,
- $Nil \blacktriangleright^{\alpha-1} (w_1, \ldots, w_{\alpha-1}) \mapsto^*_{\mathcal{A}'} P''$,
- $P' \blacktriangleright P'' \mapsto_{\mathcal{A}'} Q'$,
- $P'' = p_{Nil}$ ou $P'' = r^s_p$ où s et p sont deux états de $\mathcal{A}$.

Nous avons alors $Nil \blacktriangleright^{\alpha} (w_1, \ldots, w_\alpha) \mapsto^*_{\mathcal{A}'} P'' \blacktriangleright Q''$ et $t \mapsto^*_{\mathcal{A}'} P' \blacktriangleright (P'' \blacktriangleright Q'')$. Nous allons montrer en fonction des valeurs de Q' et Q'' que :

(a) $P'' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} p_{Nil}$ ou il existe s' et p' tels que $P'' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} r^{s'}_{p'}$, **(b)** $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto^*_{\mathcal{A}'} P$.

Nous aurons alors vérifier le lemme pour α.

Nous allons étudier un par un tout les cas pour $P' \blacktriangleright P'' \mapsto_{\mathcal{A}'} Q'$ où $P'' = p_{Nil}$ ou $P'' = r^s_p$ où s et p sont deux états de $\mathcal{A}$. Il y en a six.

- Si $P' \blacktriangleright P'' \mapsto_{\mathcal{A}'} Q'$ provient du type 2 :

  Alors $P' = P'' = Q' = p_{Nil}$. Il s'ensuit que $Q' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} P$ ne peut avoir été obtenu qu'à partir des types 2, 5 ou 6.

  - Si $Q' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} P$ provient du type 2 :

    Alors $Q'' = P = p_{Nil}$. Nous pouvons appliquer le type 2 et nous obtenons que $P'' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} p_{Nil}$, donc (a) est vérifié et que $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto^*_{\mathcal{A}'} p_{Nil}$, donc (b) est vérifié.
  - Si $Q' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} P$ provient du type 5 :

    Alors il existe trois états p, s et s' de $\mathcal{A}$ tels que $p \blacktriangleright s \mapsto_{\mathcal{A}} s'$, $Q'' = s$ et $P = r^{s'}_p$.

    Nous avons $P'' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} r^{s'}_p$ en utilisant une règle du type 5, donc (a) est vérifié. De plus, $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto^*_{\mathcal{A}'} p_{Nil} \blacktriangleright r^{s'}_p \mapsto_{\mathcal{A}'} r^{s'}_p$ en utilisant une règle du type 6. Donc $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto^*_{\mathcal{A}'} P$ et (b) est vérifié.
  - Si $Q' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} P$ provient du type 6 :

    Alors il existe deux états p et s de $\mathcal{A}$ tels que $Q'' = r^s_p$ et $P = r^s_p$. Nous avons $P'' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} r^s_p$ en utilisant une règle du type 6, donc (a) est vérifié. De plus, $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto^*_{\mathcal{A}'} p_{Nil} \blacktriangleright r^s_p \mapsto_{\mathcal{A}'} r^s_p$ en utilisant une règle du type 6. Donc $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto^*_{\mathcal{A}'} P$ et (b) est vérifié.
- Si $P' \blacktriangleright P'' \mapsto_{\mathcal{A}'} Q'$ provient du type 3 :

  Alors il existe un état p de $\mathcal{A}$ tel que $P' = p$, $P'' = p_{Nil}$ et $Q' = p$. Il s'ensuit que $Q' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} P$ est une règle de $\mathcal{A}$ ou a été obtenu à partir des types 3 ou 7.

- Si $Q' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} P$ est une règle de $\mathcal{A}$:

  Alors il existe deux états q'' et s de $\mathcal{A}$ tels que $p \blacktriangleright q'' \mapsto_{\mathcal{A}} s$, $Q'' = q''$ et $P = s$.

  Nous avons $P'' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} r_p^s$ en utilisant une règle du type 5, donc (a) est vérifié. De plus, $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto^*_{\mathcal{A}'} p \blacktriangleright r_p^s \mapsto_{\mathcal{A}'} s$ en utilisant une règle du type 7. Donc $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto^*_{\mathcal{A}'} P$ et (b) est vérifié.

- Si $Q' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} P$ provient du type 3 :

  Alors $Q'' = p_{Nil}$ et $P = p$.

  Nous avons $P'' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} p_{Nil}$ en utilisant la règle du type 2, donc (a) est vérifié. De plus, $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto^*_{\mathcal{A}'} p \blacktriangleright p_{Nil} \mapsto_{\mathcal{A}'} p$ en utilisant une règle du type 3. Donc $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto^*_{\mathcal{A}'} P$ et (b) est vérifié.

- Si $Q' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} P$ provient du type 7 :

  Alors il existe un état s de $\mathcal{A}$ tel que $Q'' = r_p^s$ et $P = s$.

  Nous avons $P'' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} r_p^s$ en utilisant une règle du type 6, donc (a) est vérifié. De plus, $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto^*_{\mathcal{A}'} p \blacktriangleright r_p^s \mapsto_{\mathcal{A}'} s$ en utilisant une règle du type 7. Donc $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto^*_{\mathcal{A}'} P$ et (b) est vérifié.

- Si $P' \blacktriangleright P'' \mapsto_{\mathcal{A}'} Q'$ provient du type 4 :

  Alors il existe deux états p et s de $\mathcal{A}$ tels que $P' = r_p^s$, $P'' = p_{Nil}$ et $Q' = r_p^s$. Il s'ensuit que $Q' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} P$ a été obtenu à partir des types 4, 8 ou 9.

  - Si $Q' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} P$ provient du type 4 :

    Alors $Q'' = p_{Nil}$ et $P = r_p^s$.

    Nous avons $P'' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} p_{Nil}$ en utilisant la règle du type 2, donc (a) est vérifié. De plus, $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto^*_{\mathcal{A}'} r_p^s \blacktriangleright p_{Nil} \mapsto_{\mathcal{A}'} r_p^s$ en utilisant une règle du type 4. Donc $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto^*_{\mathcal{A}'} P$ et (b) est vérifié.

  - Si $Q' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} P$ provient du type 8 :

    Alors il existe un état s' de $\mathcal{A}$ tel que $Q'' = r_s^{s'}$ et $P = r_p^{s'}$.

    Nous avons $P'' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} r_s^{s'}$ en utilisant une règle du type 6, donc (a) est vérifié. De plus, $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto^*_{\mathcal{A}'} r_p^s \blacktriangleright r_s^{s'} \mapsto_{\mathcal{A}'} r_p^{s'}$ en utilisant une règle du type 8. Donc $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto^*_{\mathcal{A}'} P$ et (b) est vérifié.

  - Si $Q' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} P$ provient du type 9 :

    Alors il existe deux états p' et s' de $\mathcal{A}$ tels que $s \blacktriangleright p' \mapsto_{\mathcal{A}} s'$, $Q'' = p'$ et $P = r_p^{s'}$.

Nous avons $P'' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} r_s^{s'}$ en utilisant une règle du type 5, donc (a) est vérifié. De plus, $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto^*_{\mathcal{A}'} r_p^s \blacktriangleright r_s^{s'} \mapsto_{\mathcal{A}'} r_p^{s'}$ en utilisant une règle du type 8. Donc $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto^*_{\mathcal{A}'} P$ et (b) est vérifié.

- Si $P' \blacktriangleright P'' \mapsto_{\mathcal{A}'} Q'$ provient du type 6 :

  Alors il existe deux états p et s de $\mathcal{A}$ tels que $P' = p_{Nil}$, $P'' = r_p^s$ et $Q' = r_p^s$. Il s'ensuit que $Q' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} P$ a été obtenu à partir des types 4, 8 ou 9.

  - Si $Q' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} P$ provient du type 4 :

    Alors $Q'' = p_{Nil}$ et $P = r_p^s$.

    Nous avons $P'' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} r_p^s$ en utilisant une règle du type 4, donc (a) est vérifié. De plus, $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto^*_{\mathcal{A}'} p_{Nil} \blacktriangleright r_p^s \mapsto_{\mathcal{A}'} r_p^s$ en utilisant une règle du type 6. Donc $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto^*_{\mathcal{A}'} P$ et (b) est vérifié.

  - Si $Q' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} P$ provient du type 8 :

    Alors il existe un état s' de $\mathcal{A}$ tel que $Q'' = r_s^{s'}$ et $P = r_p^{s'}$.

    Nous avons $P'' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} r_p^{s'}$ en utilisant une règle du type 8, donc (a) est vérifié. De plus, $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto^*_{\mathcal{A}'} p_{Nil} \blacktriangleright r_p^{s'} \mapsto_{\mathcal{A}'} r_p^{s'}$ en utilisant une règle du type 6. Donc $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto^*_{\mathcal{A}'} P$ et (b) est vérifié.

  - Si $Q' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} P$ provient du type 9 :

    Alors il existe deux états p' et s' de $\mathcal{A}$ tels que $s \blacktriangleright p' \mapsto_{\mathcal{A}} s'$, $Q'' = p'$ et $P = r_p^{s'}$.

    Nous avons $P'' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} r_p^{s'}$ en utilisant une règle du type 8, donc (a) est vérifié. De plus, $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto^*_{\mathcal{A}'} p_{Nil} \blacktriangleright r_p^{s'} \mapsto_{\mathcal{A}'} r_p^{s'}$ en utilisant une règle du type 6. Donc $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto^*_{\mathcal{A}'} P$ et (b) est vérifié.

- Si $P' \blacktriangleright P'' \mapsto_{\mathcal{A}'} Q'$ provient du type 7 :

  Alors il existe deux états p et s de $\mathcal{A}$ tels que $P' = p$, $P'' = r_p^s$ et $Q' = s$. Il s'ensuit que $Q' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} P$ est une règle de $\mathcal{A}$ ou a été obtenu à partir des types 3 ou 7.

  - Si $Q' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} P$ est une règle de $\mathcal{A}$:

    Alors il existe deux états q'' et s' de $\mathcal{A}$ tels que $s \blacktriangleright q'' \mapsto_{\mathcal{A}} s'$, $Q'' = q''$ et $P = s'$.

    Nous avons $P'' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} r_p^{s'}$ en utilisant une règle du type 9, donc (a) est vérifié. De plus, $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto^*_{\mathcal{A}'} p \blacktriangleright r_p^{s'} \mapsto_{\mathcal{A}'} s'$ en utilisant une règle du type 7. Donc $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto^*_{\mathcal{A}'} P$ et (b) est vérifié.

- Si $Q' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} P$ provient du type 3 :

  Alors $Q'' = p_{Nil}$ et $P = s$.

  Nous avons $P'' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} r_p^s$ en utilisant une règle du type 4, donc (a) est vérifié. De plus, $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto_{\mathcal{A}'}^* p \blacktriangleright r_p^s \mapsto_{\mathcal{A}'} s$ en utilisant une règle du type 7. Donc $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto_{\mathcal{A}'}^* P$ et (b) est vérifié.

- Si $Q' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} P$ provient du type 7 :

  Alors il existe un état s' de $\mathcal{A}$ tel que $Q'' = r_s^{s'}$ et $P = s'$.

  Nous avons $P'' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} r_p^{s'}$ en utilisant une règle du type 8, donc (a) est vérifié. De plus, $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto_{\mathcal{A}'}^* p \blacktriangleright r_p^{s'} \mapsto_{\mathcal{A}'} s'$ en utilisant une règle du type 7. Donc $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto_{\mathcal{A}'}^* P$ et (b) est vérifié.

- Si $P' \blacktriangleright P'' \mapsto_{\mathcal{A}'} Q'$ provient du type 8 :

  Alors il existe trois états p, s et s' de $\mathcal{A}$ tels que $P' = r_p^s$, $P'' = r_s^{s'}$ et $Q' = r_p^{s'}$. Il s'ensuit que $Q' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} P$ a été obtenu à partir des types 4, 8 ou 9.

  - Si $Q' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} P$ provient du type 4 :

    Alors $Q'' = p_{Nil}$ et $P = r_p^{s'}$.

    Nous avons $P'' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} r_s^{s'}$ en utilisant une règle du type 4, donc (a) est vérifié. De plus, $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto_{\mathcal{A}'}^* r_p^s \blacktriangleright r_s^{s'} \mapsto_{\mathcal{A}'} r_p^{s'}$ en utilisant une règle du type 8. Donc $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto_{\mathcal{A}'}^* P$ et (b) est vérifié.

  - Si $Q' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} P$ provient du type 8 :

    Alors il existe un état p' de $\mathcal{A}$ tel que $Q'' = r_{s'}^{p'}$ et $P = r_p^{p'}$.

    Nous avons $P'' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} r_s^{p''}$ en utilisant une règle du type 8, donc (a) est vérifié. De plus, $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto_{\mathcal{A}'}^* r_p^s \blacktriangleright r_s^{p'} \mapsto_{\mathcal{A}'} r_p^{p'}$ en utilisant une règle du type 8. Donc $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto_{\mathcal{A}'}^* P$ et (b) est vérifié.

  - Si $Q' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} P$ provient du type 9 :

    Alors il existe deux états p' et s'' de $\mathcal{A}$ tels que $s' \blacktriangleright p' \mapsto_{\mathcal{A}} s''$, $Q'' = p'$ et $P = r_p^{s''}$.

    Nous avons $P'' \blacktriangleright Q'' \mapsto_{\mathcal{A}'} r_s^{s''}$ en utilisant une règle du type 9, donc (a) est vérifié. De plus, $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto_{\mathcal{A}'}^* r_p^s \blacktriangleright r_s^{s''} \mapsto_{\mathcal{A}'} r_p^{s''}$ en utilisant une règle du type 8. Donc $P' \blacktriangleright (P'' \blacktriangleright Q'') \mapsto_{\mathcal{A}'}^* P$ et (b) est vérifié.

    ⌟

Le lemme 6.35 étant vérifié, nous pouvons conclure que $t \mapsto_{\mathcal{A}'}^* P$. ⌟

Une conséquence de la proposition 6.28 est la proposition suivante :

Proposition 6.36 *Si L est un langage régulier, alors $[L]_{\Swarrow}$ est régulier. De plus, étant donné un automate d'arbres $\mathcal{A}$ qui reconnait L, nous pouvons construire en temps polynomial un automate d'arbres $\mathcal{A}'$ pour $[L]_{\Swarrow}$ tel que $|\mathcal{A}'| = O(|\mathcal{A}|^2)$.*

Démonstration. Comme nous l'avons déja vu, une conséquence du lemme 6.6 est que $\Searrow^*$ coïncide avec $\searrow^* \cup \searrow^* \curvearrowright$. Donc si $u \in [t]_{\Swarrow}$, $u \searrow^* t$ ou $u \searrow^* \curvearrowright t$. Il s'ensuit que si $u \in [t]_{\Swarrow}$, $u \searrow^* t$ ou $u \searrow^* \mathit{Nil} \blacktriangleright t$. Donc $[t]_{\Swarrow} = [t]_{\swarrow} \cup [\mathit{Nil} \blacktriangleright t]_{\swarrow}$.

D'après la proposition 6.28, nous pouvons construire un automate qui reconnaît $[t]_{\swarrow}$ et qui a $O(|t|^2)$ états, ainsi qu'un automate qui reconnaît $[\mathit{Nil} \blacktriangleright t]_{\swarrow}$ et qui a $O(|t|^2)$ états. Donc nous pouvons construire un automate d'arbres pour $[t]_{\Swarrow}$ ayant $O(|t|^2)$ états en considérant l'automate union de ces deux automates. ⌟

6.4. Un algorithme NP pour l'accessibilité de $\overset{*}{\Rightarrow}$

Nous pouvons maintenant montrer :

Théorème 6.37 *L'accessibilité de $\overset{*}{\Rightarrow}$ entre les termes RPA est un problème NP-complet.*

Démonstration.

- Nous allons montrer que ce problème est un problème NP.

  Soient u et v deux termes. Nous calculons en temps polynomial $v_1 = v\downarrow$, puis nous devinons un terme v_2 tel que $v_1 \overset{*}{\rightleftarrows} v_2$ (v_1 et v_2 ont la même taille). Nous construisons alors un automate d'arbres qui reconnait $L = [v_2]_{\Swarrow}$ en utilisant la proposition 6.36. Ensuite le théorème 4.43, nous permet de construire un automate d'arbres qui reconnait le langage $L' = \mathit{Pre}^*(L) = \{t \mid t \xrightarrow{*} t' \in L\}$. Ces deux automates se construisent en temps polynomial. Nous répondons alors que $u \overset{*}{\Rightarrow} v$ si et seulement si $u \in L'$. Le lemme 6.21 prouve que cet algorithme est correct.

- Nous allons montrer que ce problème est un problème NP-dur.

  Nous reprenons la méthode utlisée dans [Esp97b] pour montrer que le problème de l'accessibilité pour BPP est NP-dur : nous allons coder le problème 3-SAT en un problème d'accessibilité de $\overset{*}{\Rightarrow}$ entre deux termes RPA.

  Étant donnée une conjonction $C_1 \wedge C_2 \wedge \cdots \wedge C_n$ de n clauses de trois litérals dont les variables sont $X_1, \ldots, X_m$. Une instance du problème 3-SAT est : existe-t-il une assignation de $X_1, \ldots, X_m$ telle que $C_1 \wedge C_2 \wedge \cdots \wedge C_n$ soit satisfaite ?

  Nous allons associer une déclaration RPA et une fonction de durée à ce problème. Pour chaque $1 \leq i \leq n$, $C_i = \varepsilon_{i,1}X_{i,1} \vee \varepsilon_{i,2}X_{i,2} \vee \varepsilon_{i,3}X_{i,3}$ où $\varepsilon_{i,j} \in \{0, 1\}$, $0X_{i,1} = \overline{X_{i,1}}$ et $1X_{i,1} = X_{i,1}$.

Nous considérons la déclaration RPA :

$$\begin{aligned}\Delta \stackrel{\text{def}}{=} & \{Y_i \xrightarrow{a} Nil, X_i\}_{1\le i\le m}\\ & \cup \{Y_i \xrightarrow{a} Nil, \overline{X_i}\}_{1\le i\le m}\\ & \cup \{X_i \xrightarrow{a} Nil, Nil\}_{1\le i\le m}\\ & \cup \{\overline{X_i} \xrightarrow{a} Nil, Nil\}_{1\le i\le m}\\ & \cup \{X_i \xrightarrow{a} Nil, Nil \blacktriangleright^2 (C_j, X_i),\ \text{si } X_i \in \{\varepsilon_{j,1}X_{j,1}, \varepsilon_{j,2}X_{j,2}, \varepsilon_{j,3}X_{j,3}\}\}_{1\le i\le m}\\ & \cup \{\overline{X_i} \xrightarrow{a} Nil, Nil \blacktriangleright^2 (C_j, \overline{X_i}),\ \text{si } \overline{X_i} \in \{\varepsilon_{j,1}X_{j,1}, \varepsilon_{j,2}X_{j,2}, \varepsilon_{j,3}X_{j,3}\}\}_{1\le i\le m}\end{aligned}$$

où toutes les constantes de processus appartienent à $\mathcal{X}^!$.

Nous allons montrer qu'il existe une assignation de $X_1, \dots, X_m$ telle que $C_1 \wedge C_2 \wedge \cdots \wedge C_n$ est satisfaite si et seulement si $Nil \blacktriangleright^m (Y_1, \dots, Y_m) \overset{*}{\Rightarrow} Nil \blacktriangleright^n (C_1, \dots, C_n)$.

- S'il existe une assignation de $X_1, \dots, X_m$ telle que $C_1 \wedge C_2 \wedge \cdots \wedge C_n$ est satisfaite :

  alors pour tout i, $\varepsilon_i = 0$ si X_i est assigné à faux et $\varepsilon_i = 1$ si X_i est assigné à vrai. Nous avons $Nil \blacktriangleright^m (Y_1, \dots, Y_m) \xrightarrow{m} Nil \blacktriangleright^m (Nil \blacktriangleright \varepsilon_1 X_1, \dots, Nil \blacktriangleright \varepsilon_m X_m)$, donc $Nil \blacktriangleright^m (Y_1, \dots, Y_m) \overset{m}{\Rightarrow} Nil \blacktriangleright^m (\varepsilon_1 X_1, \dots, \varepsilon_m X_m)$.

  Nous notons $C_{1,1}, \dots, C_{1,n_1}$ l'ensemble des clauses qui sont satisfaites par $\varepsilon_1 X_1$, puis $C_{2,1}, \dots, C_{2,n_2}$ l'ensemble des clauses qui sont satisfaites par $\varepsilon_2 X_2$ et qui ne le sont pas par $\varepsilon_1 X_1$. Nous continuons ainsi nous notons $C_{i,1}, \dots, C_{i,n_i}$ l'ensemble des clauses qui sont satisfaites par $\varepsilon_i X_i$ et qui ne le sont pas par $\varepsilon_1 X_1$, ni par $\varepsilon_2 X_2$, $\dots$, ni par $\varepsilon_{i-1} X_{i-1}$. L'ensemble $\{C_{1,1}, \dots, C_{1,n_1}, \dots, C_{m,1}, \dots, C_{m,n_m}\}$ est égal à l'ensemble des clauses car $C_1 \wedge C_2 \wedge \cdots \wedge C_n$ est satisfaite.

  Nous avons $Nil \blacktriangleright^m (\varepsilon_1 X_1, \dots, \varepsilon_m X_m) \to Nil \blacktriangleright^m (Nil \blacktriangleright^2 (C_{1,1}, \varepsilon_1 X_1), \dots, \varepsilon_m X_m)$, donc $Nil \blacktriangleright^m (\varepsilon_1 X_1, \dots, \varepsilon_m X_m) \Rightarrow Nil \blacktriangleright^{m+1} (C_{1,1}, \varepsilon_1 X_1, \dots, \varepsilon_m X_m)$. Il s'ensuit que $Nil \blacktriangleright^m (\varepsilon_1 X_1, \dots, \varepsilon_m X_m) \overset{*}{\Rightarrow} Nil \blacktriangleright^{m+n_1} (C_{1,1}, \dots, C_{1,n_1}, \varepsilon_1 X_1, \dots, \varepsilon_m X_m)$. Puis $Nil \blacktriangleright^m (\varepsilon_1 X_1, \dots, \varepsilon_m X_m) \overset{*}{\Rightarrow} Nil \blacktriangleright^{m+n_1-1} (C_{1,1}, \dots, C_{1,n_1}, \varepsilon_2 X_2, \dots, \varepsilon_m X_m)$. En continuant nous arrivons à $Nil \blacktriangleright^m (\varepsilon_1 X_1, \dots, \varepsilon_m X_m) \overset{*}{\Rightarrow} Nil \blacktriangleright^{n_1+\cdots+n_m} (C_{1,1}, \dots, C_{1,n_1}, C_{m,1}, \dots, C_{m,n_m})$. D'où $Nil \blacktriangleright^m (Y_1, \dots, Y_m) \overset{*}{\Rightarrow} Nil \blacktriangleright^n (C_1, \dots, C_n)$.

- Si $Nil \blacktriangleright^m (Y_1, \dots, Y_m) \overset{*}{\Rightarrow} Nil \blacktriangleright^n (C_1, \dots, C_n)$:

  alors il existe des termes $u_0, u_1, \dots, u_k$ tels que $u_0 = Nil \blacktriangleright^m (Y_1, \dots, Y_m) \Rightarrow u_1 \Rightarrow \dots \Rightarrow u_{k-1} \Rightarrow u_k = Nil \blacktriangleright^n (C_1, \dots, C_n)$. Nous nous intéressons aux constantes de processus contenues dans $u_0, \dots, u_k$. Nous allons montrer que :

  1. Pour tout i, $Atomes(u_0) \cup Atomes(u_1) \cup \cdots \cup Atomes(u_k)$ contient ou X_i ou $\overline{X_i}$ et pas les deux.

2. Si pour tout i, nous assignons X_i à vrai s'il apparait et à faux sinon, nous obtenons une assignation qui satisfait $C_1 \wedge C_2 \wedge \cdots \wedge C_n$.

Soit $i \in [\![1, m]\!]$, la constante de processus Y_i n'apparait dans aucun des membres droits des règles de Δ. Donc au cours de l'exécution, le nombre d'occurences de Y_i contenues dans u_j ne peut pas augmenter. Nous en déduisons qu'il existe un indice l_i tel que, pour tout $j < l_i$, le terme u_j contient exactement une occurrence de Y_i et pour tout $j \geq l_i$, u_j ne contient aucune occurrence de Y_i.

Il existe exactement deux règles de Δ dont Y_i est le membre gauche $Y_i \xrightarrow{a} Nil, X_i$ et $Y_i \xrightarrow{a} Nil, \overline{X_i}$, donc u_{l_i} contient une occurrence de $\varepsilon_i X_i$ avec $\varepsilon_i \in \{0, 1\}$.

Initialement, $Nil \blacktriangleright^m (Y_1, \ldots, Y_m)$ ne contient aucune occurence de X_i ou $\overline{X_i}$. Or pour tout $\varepsilon \in \{0, 1\}$, la constante de processus εX_i n'apparait dans le membre droit d'une règle de Δ que si Y_i ou εX_i en est le membre gauche. Nous en déduisons que pour tout $j < l_i$, u_j ne contient aucune occurrence de X_i ni de $\overline{X_i}$. Pour les mêmes raisons, pour tout $j \geq l_i$, u_j ne contienne que des occurrences de $\varepsilon_i X_i$. Nous venons de montrer que pour tout i, $Atomes(u_0) \cup Atomes(u_1) \cup \cdots \cup Atomes(u_k)$ contient ou X_i ou $\overline{X_i}$ et pas les deux.

Pour tout i, nous assignons X_i à vrai si $\varepsilon_i = 1$ et à faux sinon. Soit $j \in [\![1, n]\!]$, le terme $Nil \blacktriangleright^n (C_1, \ldots, C_n)$ contient une occurrence de C_j et pas $Nil \blacktriangleright^m (Y_1, \ldots, Y_m)$ donc il exite un indice l_i tel que u_{l_i} n'est aucune occurence de C_j et u_{l_i+1} en contienne au moins une. Comme C_j n'est le membre droit des règles $\varepsilon X \xrightarrow{a} Nil, Nil \blacktriangleright^2 (C_j, \varepsilon X)$ telles que $\varepsilon X \in \{\varepsilon_{j,1} X_{j,1}, \varepsilon_{j,2} X_{j,2}, \varepsilon_{j,3} X_{j,3}$. Il existe $l' \in \{1, 2, 3\}$ tel que $\varepsilon_{j,l'} X_{j,l'}$ apparaisse dans le terme u_{l_i+1}. Alors $X_{j,l'}$ a été assigné à vrai si $\varepsilon_j, l' = 1$ et à faux sinon. Donc C_j est satisfaite et par suite $C_1 \wedge C_2 \wedge \cdots \wedge C_n$ est satisfaite.

⌟

Ce théorème a pour conséquence :

Corollaire 6.38 *L'accessibilité de $\xrightarrow{*}$ entre les états emboîtés est un problème NP-complet.*

Démonstration. C'est une conséquence du théorème 6.37, du lemme 6.18 et du fait que le codage d'un schéma RPPS par une algèbre RPA est linéaire. ⌟

6.5. Le model-checking de la logique EF pour les schémas RPPS

La logique EF est présentée dans le paragraphe 4.4.1. Nous allons nous servir des résultats de ce paragraphe pour montrer que le problème du model-checking de la logique EF est décidable pour les schémas RPPS. Ceci explique que nous ne considérions que des propositions atomiques P telles que l'ensemble, noté $P_{\mathcal{S}}$ des termes t tels que $\mathcal{S}(t)$ appartient à $Sol(P)$ soit reconnaissable par un automate d'arbres. C'est le cas de la proposition P définie par l'état emboîté : s appartient à $Sol(P)$ si s contient un nombre pair d'occurrences de q.

Remarque 6.39 Par définition, $P_{\mathcal{S}}$ est compatible avec $\equiv_{\mathcal{S}}$, c'est-à-dire que si $t \vDash P$, alors pour tout terme u tel que $t \equiv_{\mathcal{S}} u$, $u \vDash P$.

Nous définissons alors la sémantique de la logique EF pour les schémas RPPS.

Définition 6.40 *Si φ est une formule de la logique EF, alors*

$$\begin{array}{rcl} Sol(\neg\varphi) & \stackrel{def}{=} & M(G) \setminus Sol(\varphi), \\ Sol(\varphi \wedge \varphi') & \stackrel{def}{=} & Sol(\varphi) \cap Sol(\varphi'), \\ Sol(\mathsf{EX}\varphi) & \stackrel{def}{=} & \{s \mid \exists s',\ s' \vDash \varphi_{\mathcal{S}} \textit{ et } s \Rightarrow s'\}, \\ Sol(\mathsf{EF}\varphi) & \stackrel{def}{=} & \{s \mid \exists s',\ s' \vDash \varphi_{\mathcal{S}} \textit{ et } s \Rightarrow^* s'\}. \end{array}$$

Pour relier le model-checking de la logique EF pour les termes RPA à celui de la logique EF pour les schémas RPPS, nous avons besoin de définir la sémantique de la logique EF modulo $\equiv_{\mathcal{S}}$.

Définition 6.41 *Si $\varphi_{\mathcal{S}}$ est une formule de la logique EF obtenue en remplaçant toutes les propositions atomiques P portant sur des états emboîtés par les propositions atomiques $P_{\mathcal{S}}$, alors*

$$\begin{array}{rcl} Sol(\neg\varphi_{\mathcal{S}})_{\equiv_{\mathcal{S}}} & \stackrel{def}{=} & T \setminus Sol(\varphi_{\mathcal{S}}), \\ Sol(\varphi_{\mathcal{S}} \wedge \varphi'_{\mathcal{S}})_{\equiv_{\mathcal{S}}} & \stackrel{def}{=} & Sol(\varphi_{\mathcal{S}}) \cap Sol(\varphi'_{\mathcal{S}}), \\ Sol(\mathsf{EX}\varphi_{\mathcal{S}})_{\equiv_{\mathcal{S}}} & \stackrel{def}{=} & \{t \mid \exists t',\ t' \in Sol(\varphi_{\mathcal{S}})_{\equiv_{\mathcal{S}}} \textit{ et } t \Rightarrow t'\}, \\ Sol(\mathsf{EF}\varphi_{\mathcal{S}})_{\equiv_{\mathcal{S}}} & \stackrel{def}{=} & \{t \mid \exists t',\ t' \in Sol(\varphi_{\mathcal{S}})_{\equiv_{\mathcal{S}}} \textit{ et } t \Rightarrow^* t'\}. \end{array}$$

La logique EF modulo $\equiv_{\mathcal{S}}$ a été construite pour vérifier :

Lemme 6.42 *Si s est un état emboîté et φ une formule de la logique EF,*

$$s \in Sol(\varphi) \textit{ si et seulement si } \mathcal{T}(s) \in Sol(\varphi_{\mathcal{S}})_{\equiv_{\mathcal{S}}}.$$

Le lemme suivant nous permet de faire le lien avec le model-checking de la logique EF pour les termes RPA.

Lemme 6.43 *Pour tout formule $\varphi_{\mathcal{S}}$ de la logique EF,*

$$Sol(\varphi_{\mathcal{S}})_{\equiv_{\mathcal{S}}} = Sol(\varphi_{\mathcal{S}}) = [Sol(\varphi_{\mathcal{S}})].$$

Démonstration. Pour montrer que $Sol(\varphi_{\mathcal{S}}) = [Sol(\varphi_{\mathcal{S}})]$, il suffit de démontrer que $[Sol(\varphi_{\mathcal{S}})] \subset Sol(\varphi_{\mathcal{S}})$ car $Sol(\varphi_{\mathcal{S}}) \subset [Sol(\varphi_{\mathcal{S}})]$. Soit t un terme RPA, nous montrons par une induction structurelle sur $\varphi_{\mathcal{S}}$ que si $t \in [Sol(\varphi_{\mathcal{S}})]$, alors $t \in [Sol(\varphi_{\mathcal{S}})]$.

Si $t \in [Sol(\varphi_{\mathcal{S}})]$, alors il existe un terme u tel que $t \equiv_{\mathcal{S}} u$ et $u \vDash \varphi_{\mathcal{S}}$.

- Si $\varphi_{\mathcal{S}} = P$ d'après la remarque 6.39, $t \vDash \varphi_{\mathcal{S}}$.
- Si $\varphi_{\mathcal{S}} = \neg\varphi'_{\mathcal{S}}$, alors $u \nvDash \varphi'_{\mathcal{S}}$. Donc par hypothèse d'induction, $t \nvDash \varphi'_{\mathcal{S}}$. Il s'ensuit que $t \vDash \neg\varphi_{\mathcal{S}}$.
- Si $\varphi_{\mathcal{S}} = \varphi'_{\mathcal{S}} \wedge \varphi''_{\mathcal{S}}$, alors $u \vDash \varphi'_{\mathcal{S}}$ et $u \vDash \varphi''_{\mathcal{S}}$. Donc par hypothèse d'induction, $t \vDash \varphi'_{\mathcal{S}}$ et $t \vDash \varphi'_{\mathcal{S}}$. Il s'ensuit que $t \vDash \varphi_{\mathcal{S}}$.
- Si $\varphi_{\mathcal{S}} = \mathsf{EX}\varphi'_{\mathcal{S}}$, alors il existe un terme u' tel que $u' \vDash \varphi'_{\mathcal{S}}$ et $u \xrightarrow{u}{}'$. Comme $\equiv_{\mathcal{S}}$ est une bisimulation (proposition 6.19), il existe t' tel que $t \xrightarrow{t}{}'$ et $t' \equiv_{\mathcal{S}} u'$. Par hypothèse d'induction, $t' \vDash \varphi'_{\mathcal{S}}$. Il s'ensuit que $t \vDash \varphi_{\mathcal{S}}$.
- Si $\varphi_{\mathcal{S}} = \mathsf{EF}\varphi'_{\mathcal{S}}$, alors il existe un terme u' tel que $u' \vDash \varphi'_{\mathcal{S}}$ et $u \xrightarrow{*} u'$. Comme $\equiv_{\mathcal{S}}$ est une bisimulation (proposition 6.19), il existe t' tel que $t \xrightarrow{*} t'$ et $t' \equiv_{\mathcal{S}} u'$. Par hypothèse d'induction, $t' \vDash \varphi'_{\mathcal{S}}$. Il s'ensuit que $t \vDash \varphi_{\mathcal{S}}$.

Nous avons immédiatement en faisant une démonstration par induction structurelle sur $\varphi_{\mathcal{S}}$ que si $t \in Sol(\varphi_{\mathcal{S}})$, alors $t \in Sol(\varphi_{\mathcal{S}})_{\equiv_{\mathcal{S}}}$. Donc nous avons $Sol(\varphi_{\mathcal{S}}) \subset Sol(\varphi_{\mathcal{S}})_{\equiv_{\mathcal{S}}}$. Il nous reste à montrer que si $t \in Sol(\varphi_{\mathcal{S}})_{\equiv_{\mathcal{S}}}$, alors $t \in Sol(\varphi_{\mathcal{S}})$.

Soit t un terme RPA tel que $t \in Sol(\varphi_{\mathcal{S}})_{\equiv_{\mathcal{S}}}$. Nous montrons par induction structurelle sur $\varphi_{\mathcal{S}}$ que $t \in Sol(\varphi_{\mathcal{S}})$.

- Si $\varphi_{\mathcal{S}} = P$, alors $t \vDash \varphi_{\mathcal{S}}$, donc $t \in Sol(\varphi_{\mathcal{S}})$.
- Si $\varphi_{\mathcal{S}} = \neg\varphi'_{\mathcal{S}}$, alors $t \notin Sol(\varphi'_{\mathcal{S}})_{\equiv_{\mathcal{S}}}$. Donc par hypothèse d'induction, $t \notin Sol(\varphi'_{\mathcal{S}})$, d'où $t \in Sol(\varphi_{\mathcal{S}})$.
- Si $\varphi_{\mathcal{S}} = \varphi'_{\mathcal{S}} \wedge \varphi''_{\mathcal{S}}$, alors $t \in Sol(\varphi'_{\mathcal{S}})_{\equiv_{\mathcal{S}}}$ et $t \in Sol(\varphi''_{\mathcal{S}})_{\equiv_{\mathcal{S}}}$. Donc par hypothèse d'induction, $t \in Sol(\varphi'_{\mathcal{S}})$ et $t \in Sol(\varphi''_{\mathcal{S}})$, d'où $t \in Sol(\varphi_{\mathcal{S}})$.
- Si $\varphi_{\mathcal{S}} = \mathsf{EX}\varphi'_{\mathcal{S}}$, alors il existe un terme t' tel que $t' \in Sol(\varphi'_{\mathcal{S}})_{\equiv_{\mathcal{S}}}$ et $t \xRightarrow{t}{}'$. Par hypothèse d'induction, $t' \in Sol(\varphi'_{\mathcal{S}})$.
  De plus, $t \xRightarrow{t}{}'$, donc d'après le lemme 6.20, il existe u' tel que $t \xrightarrow{u}{}'$ et $t' \equiv_{\mathcal{S}} u'$. Donc $u' \in [Sol(\varphi'_{\mathcal{S}})]$. Or nous venons de montrer que $Sol(\varphi'_{\mathcal{S}}) = [Sol(\varphi'_{\mathcal{S}})]$. Donc $u' \in Sol(\varphi'_{\mathcal{S}})$. Comme $t \xrightarrow{u}{}'$, $t \in Sol(\varphi_{\mathcal{S}})$.
- Si $\varphi_{\mathcal{S}} = \mathsf{EF}\varphi'_{\mathcal{S}}$, alors il existe un terme t' tel que $t' \in Sol(\varphi'_{\mathcal{S}})_{\equiv_{\mathcal{S}}}$ et $t \xRightarrow{*} t'$. Par hypothèse d'induction, $t' \in Sol(\varphi'_{\mathcal{S}})$.
  De plus, $t \xRightarrow{*} t'$, donc d'après le lemme 6.20, il existe u' tel que $t \xrightarrow{*} u'$ et $t' \equiv_{\mathcal{S}} u'$. Donc $u' \in [Sol(\varphi'_{\mathcal{S}})]$. Or nous venons de montrer que $Sol(\varphi'_{\mathcal{S}}) = [Sol(\varphi'_{\mathcal{S}})]$. Donc $u' \in Sol(\varphi'_{\mathcal{S}})$. Comme $t \xrightarrow{*} u'$, nous obtenons finalement $t \in Sol(\varphi_{\mathcal{S}})$.

vspace-1.5em ⌟

Nous déduisons des lemmes 6.42 et 6.43 qu'un état emboîté s satisfait une formule φ de la logique EF si et seulement si $\mathcal{T}(s) \models \varphi_{\mathcal{S}}$. Il s'ensuit en utilisant le théorème 4.50 que :

Proposition 6.44 *Le problème du model-checking de la logique EF est décidable pour les schémas RPPS.*

7. Conclusion

Nous avons codé les systèmes RPPS par RPA qui est un système de réécriture de processus combinant les caractéristiques suivantes :

– Il possède une relation d'accessibilité qui est effectivement reconnaissable ;
– Une méthode uniforme utilisant des automates d'arbres permet de calculer les modèles de toutes les formules de la logique de transitions TL ;
– Tout ceci peut être utilisé pour l'analyse de l'atteignabilité des systèmes RPPS.

La difficulté dans ce travail était de trouver une présentation des états emboîtés en termes d'algèbre de processus dans laquelle les transitions soient assez locales pour que la relation d'accessibilité soit reconnaissable. C'est pourquoi le lien entre les états emboîtés et les termes RPA n'est pas immédiat : la relation $\equiv_{\mathcal{S}}$ n'est pas une congruence et nous avons besoin d'utiliser deux notions « $u \xrightarrow{\alpha!} v$ » et « $u \xrightarrow{\alpha?} v$ ».

Ce travail renforce l'idée que les systèmes de réécriture de processus permettent d'analyser une variété très importante de systèmes d'états infinis. Dans le même temps, ce travail montre aussi que les automates d'arbres sont un outil très puissant pour l'analyse de ce type de systèmes de réécriture de processus.

Deuxième partie .

Vérification de l'équivalence de performance pour BPP temporisé

8. Introduction

La vérification de systèmes temporisés infinis est un domaine très vaste qui va de l'analyse statique [EK99] à la modélisation et la vérification de protocoles de communication [CFP95].

Comme nous l'avons vu dans l'introduction de cette thèse, la notion de **performance** permet de comparer l'efficacité de systèmes qui ont les mêmes comportements avec des vitesses différentes : ils exécutent les mêmes séquences d'actions qui n'ont pas les mêmes durées. Cette notion a donné naissance à plusieurs préordres [MT91, AH92, GRS95, CGR97] et équivalences [FM95, AM96, Cor98, CD01], notamment l'équivalence de performance [GRS95, CGR97] qui n'est autre que la bisimulation pour un système de transitions où les étiquettes ont deux composantes : une action et une date.

Dans cette partie, nous nous intéressons à la décidabilité de l'équivalence de performance quand il n'est fait aucune restriction sur le nombre d'états. Dans ce cadre, comme aucune synchronisation n'est autorisée, les systèmes considérés semblent être obtenus à partir de l'algèbre BPP (Basic Parallel Processes [Chr93]) en ajoutant du temps par l'intermédiaire d'horloges locales. C'est pourquoi nous appelons cette algèbre TBPP.

La bisimulation est un problème co-NP dur [May00] pour BPP (non temporisée). Les liens entre TBPP et BPP ont motivé notre étude : nous voulions cerner l'apport des horloges locales.

Notre résultat principal est que l'équivalence de performance est décidable pour TBPP et qu'elle peut être décidée en temps polynomial (le problème est en fait PTIME-complet). De façon surprenante, l'ajout d'horloges locales ne rend pas le problème plus compliqué : ces horloges permettent de décomposer le système d'une manière proche du cas normé [HJM96].

Dans le premier chapitre de cette partie, nous proposons une présentation de TBPP qui est orientée par notre résultat principal. Dans le deuxième, nous étudions l'équivalence de performance, démontrons qu'elle est décidable pour TBPP et montrons les conséquences de ce résultat pour la congruence de performance étudiée par F. Corradini, D. Di Cola, M. Pistore [Cor98, CD01, CP01].

9. Une algèbre de processus temporisée : TBPP

Sommaire

Dans ce chapitre, nous présentons une algèbre de processus temporisée. Nous l'avons appelée TBPP pour Timed Basic Parallel Processes car nous la voyons comme une extension temporisée de BPP (Basic Parallel Processes) [Chr93]. Cette algèbre a d'abord été proposée par F. Corradini, R. Gorrieri et M. Roccetti dans [CGR97] et nous n'avons apporté que peu de changements à sa définition.

9.1. Une temporisation de BPP

La particularité de TBPP est sa temporisation : le temps n'est pas introduit pour obliger différentes composantes du système à se synchroniser. Elle a été empruntée à R. Gorrieri, M. Roccetti et E. Stancampiano qui dans [GRS95] proposent une algèbre temporisée comprenant des opérateurs de CCS [Mil89] et de TCSP [BHR84].

Les caractéristiques principales de cette temporisation sont les suivantes :
- Le temps est représenté par un ensemble **discret**. Le domaine du temps est l'ensemble des entiers naturels $\mathbb{N}$.
- Les actions sont **urgentes** : elles se produisent dès que possible.

– Les actions **possèdent une durée** : une fonction associe à chaque action sa durée d'exécution. Cette fonction est définie indépendamment de la syntaxe de l'algèbre.
– Chaque composante parallèle possède sa propre horloge. Les exécutions sont **asynchrones** et différentes composantes parallèles peuvent se décaler temporellement les unes par rapport aux autres, tout en préservant les propriétés de causalité. Cet aspect a été étudié par L. Aceto et D. Murphy [AM96] et fait l'objet du paragraphe 9.3.3.

9.2. La syntaxe de TBPP

La syntaxe de TBPP peut être présentée de deux façons :

- Les règles de réécriture :

  La méthode que nous utilisons définit les processus par l'intermédiaire de règles de réécriture suivant l'approche de F. Moller [Mol96]. Cela simplifie les preuves sans nécessiter de changement sémantique.

- Les équations récursives :

  La méthode habituelle pour définir les processus dans les algèbres de processus consiste à utiliser des équations récursives. C'est la méthode retenue dans [GRS95, CGR97]. Nous ne l'emploierons que pour présenter les exemples. En effet, ils sont ainsi plus concis et plus lisibles.

9.2.1. Les règles de réécriture

Nous considérons un ensemble de **noms d'actions** Act dont les éléments sont notés a, b, etc. et un ensemble de **constantes de processus** $\mathcal{X}$ dont les éléments sont notés X, Y, etc.

Définition 9.1 *L'ensemble $\mathcal{T}$ des **termes TBPP** est donné par la syntaxe suivante :*

$$t, u ::= Nil \ \mid \ X \ \mid \ t \parallel u \ \mid \ 1 \triangleright t.$$

Nil et $\parallel$ sont les deux opérateurs présents dans l'algèbre BPP [Chr93]. Nil représente le **processus vide**, celui qui ne peut plus faire d'action. La combinaison $t \parallel u$ est la **composition parallèle** de t et u. Elle ne permet aucune synchronisation. L'ensemble des actions que peut faire le terme $t \parallel u$ est la réunion des ensembles d'actions correspondant à t et u.

L'**opérateur de délai** $\triangleright$ est destiné à introduire le temps. Le processus $1 \triangleright t$ agit comme t mais avec un retard d'une unité de temps.

Nous utilisons les abréviations suivantes :

– $n \triangleright t$ pour le terme $\overbrace{1 \triangleright (1 \triangleright (\cdots (1 \triangleright}^{n} t) \cdots))$,

– X^n pour le terme $\overbrace{X \parallel X \parallel \cdots \parallel X}^{n}$,
et nous remplaçons $0 \triangleright t$ par t et X^0 par Nil.

Soit t un terme, nous désignons par $Constante(t)$ l'ensemble de tous les éléments de $\mathcal{X}$ qui ont une occurrence dans t.

Exemple 9.2 $Constante(X \parallel (1 \triangleright (X \parallel Y))) = \{X, Y\}$.

Définition 9.3 *Une **déclaration TBPP** est un ensemble fini* $\Delta \subseteq \mathcal{X} \times Act \times \mathcal{T}$ *de règles de réécriture, noté* $\{X_i \xrightarrow{a}_{\Delta} t_i \mid i = 1, \ldots, n\}$.

Nous demandons que $Constante(t_i) \subseteq \{X_1, \ldots, X_n\}$ pour tout i. Les X_i ne sont pas nécessairement distincts ce qui permet au système de transitions associé de n'être pas déterministe (cf. le paragraphe 9.3). Pour des raisons techniques, nous imposons que tous les éléments X de $\mathcal{X}$ apparaissent dans le membre gauche d'au moins une règle. Dans le cas contraire, nous remplaçons toutes les occurences du X concerné par Nil.

Le sens informel d'une règle $X \xrightarrow{a}_{\Delta} Y$ est que le processus X peut faire une action a et agir ensuite comme le processus Y.

9.2.2. Les équations récursives de processus

Dans les exemples, nous définissons souvent les processus avec des équations récursives ce qui est la méthode traditionnellement utilisée dans les algèbres de processus [Mil89].

L'ensemble des termes TBPP est alors donné par la syntaxe :

$$t, u ::= Nil \mid X \mid a.t \mid t \parallel u \mid t + u \mid 1 \triangleright t.$$

Les nouveaux opérateurs sont les opérateurs CCS notés $a.$ et $+$. Le premier correspond au **préfixage par une action** : le processus $a.t$ peut faire l'action a et se comporte ensuite comme t. Le second est le **choix non déterministe**. L'ensemble des actions que peut faire $u + t$ est soit l'ensemble des actions que peut faire u, soit l'ensemble des actions que peut faire t. Lorsque $u + t$ fait sa première action, il choisit un des deux processus.

Une déclaration TBPP est alors un ensemble fini d'équations récursives de processus $\Delta = \{X_i \stackrel{\text{def}}{=} t_i \mid i = 1, \ldots, n\}$ dans lequel les X_i sont distincts et tels que $Constante(t_i)$ est inclus dans l'ensemble $\{X_1, \ldots, X_n\}$ pour tout i. De plus, nous supposons que chaque occurrence d'une constante d'un terme t_i est **gardée**. Cela signifie que toute variable doit être dans le champ d'une action préfixe.

Exemple 9.4
Les deux déclarations suivantes définissent les mêmes processus.

La déclaration :

$$\begin{array}{rcl} X & \stackrel{\text{def}}{=} & a.(1 \triangleright (X \parallel X)) + b.(1 \triangleright (a.Y)), \\ Y & \stackrel{\text{def}}{=} & c.((1 \triangleright (a.X)) \parallel (1 \triangleright a)) \end{array}$$

est plus concise que la déclaration :

$$\Delta \stackrel{\text{def}}{=} \left\{ \begin{array}{l} X \xrightarrow{a} 1 \triangleright (X \parallel X)\ ,\quad X \xrightarrow{b} 1 \triangleright Z\ , \\ Z \xrightarrow{a} Y\ ,\quad Y \xrightarrow{c} (1 \triangleright T) \parallel (1 \triangleright U)\ , \\ T \xrightarrow{a} X\ ,\quad U \xrightarrow{a} \mathit{Nil}\ . \end{array} \right\}$$

c'est pourquoi nous la préférons.

9.3. La sémantique opérationnelle

L'évolution d'un processus TBPP est représentée par un système de transitions dont les étiquettes sont de la forme (a, n), où $a \in \mathit{Act}$ et $n \in \mathbb{N}$. Dans cette sémantique, n est la date à laquelle commence l'action. Pour savoir à quel moment elle finit, il suffit de connaître sa durée. La sémantique proposée dans [GRS95, CGR97] est différente : n y représente la date où l'action a finit.

9.3.1. La fonction de durée

Les durées de chacune des actions de Act sont données par une fonction.

Définition 9.5 *Une **fonction de durée** f est une fonction de Act dans $\mathbb{N} \setminus \{0\}$.*

Par exemple, $f(a) = 3$ signifie que a dure trois unités de temps.
Nous noterons **1** la fonction associant 1 comme durée à toute action a.

Ce cadre permet de séparer la définition fonctionnelle Δ de la notion de performance f et d'associer à cette dernière une machine particulière. Nous pouvons ainsi définir d'une part, un ensemble de processus et d'autre part, les performances de différentes machines, ce qui permet de comparer les comportements de ces processus en fonction de la machine considérée.

9.3.2. Le système de transitions

Une paire (Δ, f) où Δ est une déclaration TBPP et f est une fonction de durée définit une relation de transitions étiquetée $\rightarrow_f \subseteq \mathcal{T} \times (\mathit{Act} \times \mathbb{N}) \times \mathcal{T}$, donnée inductivement par les règles de sémantique opérationnelle structurelle (SOS) suivantes :

$$\text{Ra}\quad \frac{}{X \xrightarrow{a,0}_f f(a) \triangleright t}\ (X \xrightarrow{a}_\Delta t) \in \Delta \qquad\qquad \text{Rc}\quad \frac{t \xrightarrow{a,n}_f t'}{t \parallel u \xrightarrow{a,n}_f t' \parallel u}$$

$$\text{Rb}\quad \frac{t \xrightarrow{a,n}_f t'}{1 \triangleright t \xrightarrow{a,n+1}_f 1 \triangleright t'} \qquad\qquad \text{Rd}\quad \frac{t \xrightarrow{a,n}_f t'}{u \parallel t \xrightarrow{a,n}_f u \parallel t'}$$

La règle Ra introduit le temps : si $X \xrightarrow{a}_\Delta t$ est une règle de Δ, alors le processus X peut faire l'action a à la date 0 et se comporter ensuite comme le processus t retardé de $f(a)$ unités de temps.

La règle Rb indique comment se propagent les délais : le processus $1 \triangleright t$ fait les mêmes actions que t mais une unité de temps plus tard et ce retard se propage.

Les règles Rc et Rd montrent que le processus $u \parallel t$ peut faire les actions de u ou celles de t. Dans chaque cas, seul le processus u ou t qui fait une action est modifié et cette modification est la même que s'il était seul. L'autre processus reste inchangé.

Définition 9.6
Une ***exécution*** *de t est une suite finie ou infinie* $(t =)\ t_0 \xrightarrow{a_1,n_1}_f t_1 \xrightarrow{a_2,n_2}_f t_2 \xrightarrow{a_3,n_3}_f \cdots \xrightarrow{a_k,n_k}_f t_k \cdots$
La ***trace*** *d'une exécution est la suite formée des étiquettes des transitions.*

Exemple 9.7 Soient $f = \mathbf{1}$ et le terme X donné par $X \stackrel{\text{def}}{=} a.(b.b.Nil \parallel c.Nil) + a.c.(b.Nil \parallel b.Nil)$.

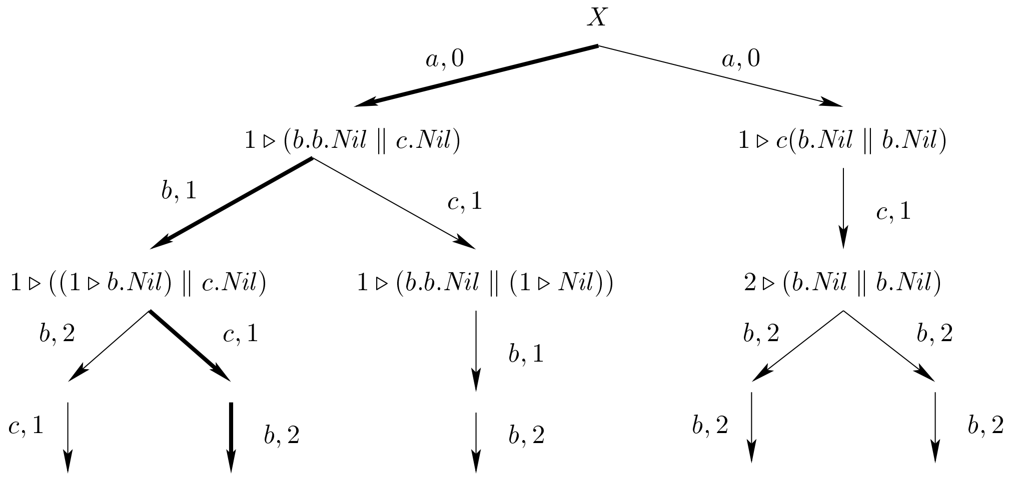


FIG. 9.1. – Le système de transitions associé à X.

Une exécution de X est la suivante :

$$\begin{aligned} X &\xrightarrow{a,0}_f 1 \triangleright (b.b.Nil \parallel c.Nil) \xrightarrow{b,1}_f 1 \triangleright ((1 \triangleright b.Nil) \parallel c.Nil) \\ &\xrightarrow{c,1}_f 1 \triangleright ((1 \triangleright b.Nil) \parallel (1 \triangleright Nil)) \xrightarrow{b,2}_f 1 \triangleright ((2 \triangleright Nil) \parallel (1 \triangleright Nil)). \end{aligned}$$

Elle est mise en relief dans la figure 9.1 qui présente le système de transitions associé à X. La trace de cette exécution est $(a,0)(b,1)(c,1)(b,2)$.

Par la suite, nous utiliserons l'abréviation suivante : $t \xrightarrow{w}_f t'$ quand $t \xrightarrow{a_1,n_1}_f t_1 \xrightarrow{a_2,n_2}_f t_2 \xrightarrow{a_3,n_3}_f \cdots \xrightarrow{a_{l-1},n_{l-1}}_f t_{l-1} \xrightarrow{a_l,n_l}_f t'$ où $w = (a_1,n_1)(a_2,n_2)\cdots(a_l,n_l) \in (Act \times \mathbb{N})^*$. Nous noterons $\xrightarrow{*}_f$ la clôture transitive et réflexive de $\rightarrow_f$. De plus, $u \rightarrow_f$ et $u \not\rightarrow_f$ signifient respectivement qu'il existe (resp. qu'il n'existe pas) un v tel que $u \rightarrow_f v$. Enfin, nous omettons l'indice f quand il n'y a pas d'ambiguïté.

Définition 9.8
Une exécution finie $t_0 \xrightarrow{a_1,n_1} \cdots \xrightarrow{a_k,n_k} t_k$ *est* ***maximale*** *si* $t_k \not\rightarrow$.
La trace d'une exécution maximale est ***maximale****.*

Exemple 9.9 La trace $(a,0)(b,1)(c,1)(b,2)$ de l'exemple 9.7 est maximale.

Lemme 9.10 *Étant données* Δ *une déclaration TBPP et* f *une fonction de durée, un terme* u *peut faire une action si et seulement si* $Constante(u)$ *est non vide.*

Démonstration.

- Si u peut faire une action.

  Nous allons montrer par induction sur u que $Constante(u)$ est non vide. Le terme u est différent de Nil car aucune des règles Ra, Rb, Rc et Rd définissant la sémantique de TBPP ne s'applique à Nil.
  - Si $u \in X$, alors la propriété est vérifiée.
  - Si $u = 1 \triangleright u'$, alors d'après la règle Rb, u' peut faire une action. Donc par hypothèse d'induction, $Constante(u')$ est non vide il s'ensuit que $Constante(u)$ est non vide.
  - Si $u = u_1 \triangleright u_2$, alors d'après les règles Rc et Rd, soit u_1 peut faire une action et par hypothèse d'induction, $Constante(u_1)$ est non vide, soit u_2 peut faire une action et par hypothèse d'induction, $Constante(u_2)$ est non vide. Comme $Constante(u) = Constante(u_1) \cup Constante(u_2)$, $Constante(u)$ est non vide.
- Si $Constante(u)$ est non vide.

  Nous allons montrer par induction sur u que u peut faire une action. Le terme u est différent de Nil car $Constante(Nil)$ est vide.
  - Si $u \in X$, alors comme tous les éléments de $\mathcal{X}$ apparaissent dans le membre gauche d'au moins une règle de Δ, il existe une règle $u \xrightarrow{a}_\Delta t$ dans Δ. La règle Ra produit la transition $u \xrightarrow{a,0} f(a) \triangleright t$, donc u peut faire une action.
  - Si $u = 1 \triangleright u'$, alors comme $Constante(u) = Constante(u')$, par hypothèse d'induction, u' peut faire une action. La règle Rb entraîne que u peut faire une action.

– Si $u = u_1 \rhd u_2$, alors comme $Constante(u) = Constante(u_1) \cup Constante(u_2)$, $Constante(u_1)$ ou $Constante(u_2)$ est non vide. Donc par hypothèse d'induction, u_1 ou u_2 peut faire une action. Les règles Rc et Rd entraînent qu'alors u peut faire une action. ⌟

9.3.3. Branchement et temporisation

La notion d'exécution **mal temporisée** a été introduite dans [BB90, MT90]. Elle désigne une exécution dont les actions n'ont pas des dates croissantes. Elles semblent donc faire des retours dans le temps d'où leur nom.

L. Aceto et D. Murphy ont montré dans [AM96] que ces exécutions sont acceptables à condition que la causalité soit respectée, c'est-à-dire, que les actions produites par une même composante aient des dates croissantes. Ils ont mis en avant les nombreux avantages techniques à ne pas se restreindre à des exécutions « bien temporisées ».

Dans ce paragraphe, nous définissons la notion d'exécution mal temporiséee ainsi que celle de d'exécution causalement correcte.

Définition 9.11 *Une exécution* $t_0 \xrightarrow{a_1,n_1} t_1 \xrightarrow{a_2,n_2} t_2 \cdots \xrightarrow{a_k,n_k} t_k \cdots$ *est* ***mal temporisée*** *s'il existe deux indices* $i > j$ *tels que* $n_i < n_j$*.*

L'exemple qui suit montre que l'algèbre TBPP autorise les exécutions mal temporisées.

Exemple 9.12 Les traces maximales du processus X défini dans l'exemple 9.7 sont :
- $(a,0)(b,1)(b,2)(c,1)$,
- $(a,0)(b,1)(c,1)(b,2)$,
- $(a,0)(c,1)(b,1)(b,2)$,
- $(a,0)(c,1)(b,2)(b,2)$.

La première est mal temporisée.

Afin de définir la notion de causalité, nous commençons par présenter la notion de déparallélisation.

Un terme (resp. une déclaration, resp. une exécution déparallélisé) est un terme (resp. une déclaration, resp. une exécution) qui ne contient pas d'opérateur $\|$. Il est obtenu en ne tenant compte, à chaque fois que nous rencontrons l'opérateur $\|$, que d'une seule des deux composantes présentes.

Définition 9.13 *Le* ***déparallélisé*** *d'un terme t est l'ensemble des termes, noté* $< t >$*, défini inductivement par :*
- $< Nil >= \{Nil\}$,
- $< X >= \{X\}$,
- $< u \parallel t >=< u > \cup < t >$,

– $< 1 \triangleright t >= \{1 \triangleright t' \mid t' \in < t >\}$.

Exemple 9.14 $< 1 \triangleright ((1 \triangleright X) \parallel (1 \triangleright Y)) >= \{2 \triangleright X, 2 \triangleright Y\}$.

Définition 9.15 *La déclaration TBPP* ***déparallélisée*** $< \Delta >$ *associée à une déclaration* Δ *est l'ensemble* $\{X \xrightarrow{a}_{<\Delta>} t' \mid X \xrightarrow{a}_{\Delta} t \in \Delta$ *et* $t' \in < t >\}$.

Exemple 9.16 Une déclaration avec des règles de réécriture pour l'exemple 9.7 est la suivante :

$$\Delta \stackrel{\text{def}}{=} \left\{ \begin{array}{l} X \xrightarrow{a} Y \parallel Z \ , \quad X \xrightarrow{a} X' \ , \\ Y \xrightarrow{b} Y' \ , \quad Y' \xrightarrow{b} Nil \ , \\ Z \xrightarrow{c} Nil \ , \quad X' \xrightarrow{c} Y' \parallel Y' \ . \end{array} \right\}$$

La déclaration déparallélisée associée est :

$$< \Delta >= \left\{ \begin{array}{l} X \xrightarrow{a} Y \ , \quad X \xrightarrow{a} Z \ , \quad X \xrightarrow{a} X' \ , \\ Y \xrightarrow{b} Y' \ , \quad Y' \xrightarrow{b} Nil \ , \\ Z \xrightarrow{c} Nil \ , \quad X' \xrightarrow{c} Y' \ . \end{array} \right\}$$

Définition 9.17 *Une exécution* ***déparallélisée*** *d'un processus t pour une déclaration* Δ *est une exécution de t pour la déclaration déparallélisée* $< \Delta >$.

Exemple 9.18 Les exécutions maximales déparallélisées de X sont :
- $X \xrightarrow{a,0} 1 \triangleright Y \xrightarrow{b,1} 2 \triangleright Y' \xrightarrow{b,2} 3 \triangleright Nil$,
- $X \xrightarrow{a,0} 1 \triangleright Z \xrightarrow{c,1} 2 \triangleright Nil$,
- $X \xrightarrow{a,0} 1 \triangleright X' \xrightarrow{c,1} 1 \triangleright Y' \xrightarrow{b,2} 3 \triangleright Nil$.

Définition 9.19 *Une exécution* e' *est dite* ***extraite*** *d'une exécution* e *si* e' *est une exécution déparallélisée et si elle peut être obtenue à partir de* e *en supprimant des transitions et en remplaçant chaque processus par un élément de son déparallélisé.*

Une exécution extraite est donc obtenue en suivant toujours la même composante parallèle.

Définition 9.20 *Une exécution est* ***causalement correcte*** *si aucune de ses exécutions extraites n'est mal temporisée.*

Une interprétation possible de la notion d'exécutions mal temporisées mais causalement correctes est que les exécutions sont observées par quelqu'un qui est à l'extérieur du système. Il fait régulièrement le tour des différentes composantes et note à chaque fois les actions qui se sont produites. Il lui arrive donc de prendre connaissance d'une action effectuée par la composante 1 à la date d_1 après avoir enregistré une action effectuée par la composante 2 à la date d_2 même si $d_1 < d_2$.

Théorème 9.21 *Les exécutions de TBPP sont causalement correctes.*

Démonstration. Soient Δ une déclaration TBPP et f une fonction de durée, nous allons montrer par récurrence sur k que toutes exécutions de longueur k du système de transitions associé à Δ et f sont causalement correctes. Nous en déduirons que toutes exécutions du système de transitions associé à Δ et f sont causalement correctes.

- Soit $e = t \xrightarrow{a,n} t'$ une exécution de longueur 1.

  Une exécution extraite est de longueur 0 ou 1, elle est donc bien temporisée.
- Soit $e = t_0 \xrightarrow{a_1,n_1} \cdots \xrightarrow{a_k,n_k} t_k$ une exécution de longueur k.

  Soit e' une exécution extraite de e. Alors, soit e' a été obtenue en supprimant la première transition de e, soit e' a été obtenue en gardant la première transition de e.

  - Si e' a été obtenue en supprimant la première transition de e.

    Alors e' est une exécution extraite de $t_1 \xrightarrow{a_2,n_2} \cdots \xrightarrow{a_k,n_k} t_k$ qui est de longueur k et donc par hypothèse de récurrence e' est bien temporisée.
  - Si e' a été obtenue en gardant la première transition de e.

    Alors $e' = u_{i_0} \xrightarrow{a_{i_0},n_{i_0}} \cdots \xrightarrow{a_{i_l},n_{i_l}} u_{i_l}$ où u_{i_j} est un élément du déparallélié de t_{i_j} et $i_0 = 1$.

    Si e' est de longueur 0 ou 1, elle est bien temporisée.

    Sinon nous considérons l'exécution $u_{i_0} \xrightarrow{a_{i_0},n_{i_0}} u_{i_1} \xrightarrow{a_{i_1},n_{i_1}} u_{i_2}$. C'est une exécution de $< \Delta >$, donc aucun des termes u_{i_j} ne contient plusieurs composantes parallèles. D'où $u_{i_j} = m_j \triangleright X_j$. D'après, les règles Ra et Rb, $n_{i_0} = m_0$, $n_{i_1} = m_1$ et $< \Delta >$ contient les règles $X_0 \xrightarrow{a_{i_0}} (m_1 - f(a_{i_0}) - m_0) \triangleright X_{i_1}$, $X_{i_1} \xrightarrow{a_{i_1}} (m_2 - f(a_{i_1}) - m_1) \triangleright X_2$. Nous avons donc $m_1 - f(a_{i_0}) - m_0 \geq 0$, il s'ensuit que $m_1 > m_0$ car $f(a_{i_0}) > 0$. D'où $n_{i_1} > n_{i_0}$.

    Par ailleurs, l'exécution $u_{i_1} \xrightarrow{a_{i_1},n_{i_1}} \cdots \xrightarrow{a_{i_l},n_{i_l}} u_{i_l}$ est une exécution extraite de $t_1 \xrightarrow{a_2,n_2} \cdots \xrightarrow{a_k,n_k} t_k$ qui est de longueur $k-1$ et donc par hypothèse de récurrence $u_{i_1} \xrightarrow{a_{i_1},n_{i_1}} \cdots \xrightarrow{a_{i_l},n_{i_l}} u_{i_l}$ est bien temporisée. D'où $n_{i_1} < \ldots < n_{i_l}$.

    Nous concluons que $n_{i_0} < \ldots < n_{i_l}$ et par suite que e' est bien temporisée.

$\lrcorner$

Exemple 9.22 L'exécution qui correspond à la trace mal temporisée de l'exemple 9.12 est

$$X \xrightarrow{a,0} 1 \triangleright (Y \parallel Z) \xrightarrow{b,1} 1 \triangleright ((1 \triangleright Y') \parallel Z) \xrightarrow{b,2} 1 \triangleright ((1 \triangleright (1 \triangleright Nil)) \parallel Z)$$
$$\xrightarrow{c,1} 1 \triangleright ((1 \triangleright (1 \triangleright Nil)) \parallel (1 \triangleright Nil)).$$

Cette exécution est causalement correcte. En effet, ses deux exécutions extraites sont : $X \xrightarrow{a,0} 1 \triangleright Y \xrightarrow{b,1} 2 \triangleright Y' \xrightarrow{b,2} 3 \triangleright Nil$ et $X \xrightarrow{a,0} 1 \triangleright Z \xrightarrow{c,1} 2 \triangleright Nil$, et aucune des deux n'est mal temporisée.

9.4. Les mesures d'horloges

Dans cette section, nous introduisons deux mesures structurelles pour les termes :

- $\mathtt{H}_{\mathtt{min}}(u) \in \mathbb{N} \cup \{+\infty\}$ est la date la plus précoce à laquelle u peut commencer une action.
- $\mathtt{H}_{\mathtt{max}}(u) \in \mathbb{N} \cup \{-\infty\}$ est la date la plus tardive.

Ces deux mesures nous permettent d'obtenir des informations sur les dates auxquelles un proccessus pourra faire des actions au cours d'une exécution. Elles servent de critère pour distinguer les différentes composantes parallèles d'un processus.

Définition 9.23 *Nous supposons que l'ordre et l'addition des entiers naturels sont étendus de façon naturelle à $+\infty$ et à $-\infty$. Nous définissons l'**horloge minimale**, notée $\mathtt{H}_{\mathtt{min}}$, et l'**horloge maximale**, notée $\mathtt{H}_{\mathtt{max}}$, d'un terme par induction structurelle :*

$$
\begin{aligned}
\mathtt{H}_{\mathtt{min}}(Nil) &\stackrel{def}{=} +\infty, \\
\mathtt{H}_{\mathtt{min}}(X) &\stackrel{def}{=} 0, \\
\mathtt{H}_{\mathtt{min}}(1 \triangleright u) &\stackrel{def}{=} 1 + \mathtt{H}_{\mathtt{min}}(u), \\
\mathtt{H}_{\mathtt{min}}(u \parallel v) &\stackrel{def}{=} \min(\mathtt{H}_{\mathtt{min}}(u), \mathtt{H}_{\mathtt{min}}(v)).
\end{aligned}
$$

$$
\begin{aligned}
\mathtt{H}_{\mathtt{max}}(X) &\stackrel{def}{=} 0, \\
\mathtt{H}_{\mathtt{max}}(1 \triangleright u) &\stackrel{def}{=} 1 + \mathtt{H}_{\mathtt{max}}(u), \\
\mathtt{H}_{\mathtt{max}}(Nil) &\stackrel{def}{=} -\infty, \\
\mathtt{H}_{\mathtt{max}}(u \parallel v) &\stackrel{def}{=} \max(\mathtt{H}_{\mathtt{max}}(u), \mathtt{H}_{\mathtt{max}}(v)).
\end{aligned}
$$

Exemple 9.24 Soit $u = 1 \triangleright (X \parallel (2 \triangleright X))$, alors $\mathtt{H}_{\mathtt{min}}(u) = 1$ et $\mathtt{H}_{\mathtt{max}}(u) = 3$. Nous remarquons que si $X \xrightarrow{a} t$ est une règle de Δ, alors $u \xrightarrow{a,1} 1 \triangleright ((f(a) \triangleright t) \parallel (2 \triangleright X))$ et $u \xrightarrow{a,3} 1 \triangleright (X \parallel ((2 + f(a)) \triangleright t))$.

Remarque 9.25 Les valeurs de $\mathtt{H}_{\mathtt{max}}$ (resp. $\mathtt{H}_{\mathtt{min}}$) pour *Nil* ont été choisies afin que la formule définissant $\mathtt{H}_{\mathtt{max}}(u \parallel v)$ (resp. $\mathtt{H}_{\mathtt{min}}(u \parallel v)$) fournisse la date la plus tardive (respectivement la plus précoce) à laquelle le processus peut commencer une action même si u ou v est égal à *Nil*.

Lemme 9.26 *Les propositions suivantes sont équivalentes :*

– $Constante(u) = \emptyset$,
– $\mathtt{H}_{\mathtt{max}}(u) < \mathtt{H}_{\mathtt{min}}(u)$,
– $\mathtt{H}_{\mathtt{min}}(u) = +\infty$,
– $\mathtt{H}_{\mathtt{max}}(u) = -\infty$.

Démonstration. Nous allons d'abord montrer que si $Constante(u) = \emptyset$, alors $\mathtt{H}_{\mathtt{min}}(u) = +\infty$ et $\mathtt{H}_{\mathtt{max}}(u) = -\infty$ par induction sur u. Une conséquence immédiate est que si $Constante(u) = \emptyset$, alors $\mathtt{H}_{\mathtt{max}}(u) < \mathtt{H}_{\mathtt{min}}(u)$.

- Si $u = Nil$.
  Alors $\mathtt{H}_{\mathtt{min}}(u) = +\infty$, $\mathtt{H}_{\mathtt{max}}(u) = -\infty$.
- Si $u = 1 \triangleright u'$.
  Alors $Constante(u') = \emptyset$ et par hypothèse d'induction, $\mathtt{H}_{\mathtt{min}}(u') = +\infty$ et $\mathtt{H}_{\mathtt{max}}(u') = -\infty$. Il s'en suit que $\mathtt{H}_{\mathtt{min}}(u) = +\infty$ et $\mathtt{H}_{\mathtt{max}}(u) = -\infty$.
- Si $u = u' \parallel u''$.
  Alors $Constante(u') = \emptyset$ et $Constante(u'') = \emptyset$. Donc par hypothèse d'induction, $\mathtt{H}_{\mathtt{min}}(u') = \mathtt{H}_{\mathtt{min}}(u'') = +\infty$ et $\mathtt{H}_{\mathtt{max}}(u') = \mathtt{H}_{\mathtt{max}}(u'') = -\infty$. Il s'en suit que $\mathtt{H}_{\mathtt{min}}(u) = +\infty$ et $\mathtt{H}_{\mathtt{max}}(u) = -\infty$.

Nous allons maintenant montrer que si $Constante(u) \neq \emptyset$, alors $\mathtt{H}_{\mathtt{min}}(u) \neq +\infty$, $\mathtt{H}_{\mathtt{max}}(u) \neq -\infty$ et $\mathtt{H}_{\mathtt{max}}(u) \geq \mathtt{H}_{\mathtt{min}}(u)$ par induction sur u.

- Si $u \in \mathcal{X}$.
  Alors $\mathtt{H}_{\mathtt{max}}(u) = \mathtt{H}_{\mathtt{min}}(u) = 0$, donc $\mathtt{H}_{\mathtt{max}}(u) \geq \mathtt{H}_{\mathtt{min}}(u)$ et la propriété est vérifiée.
- Si $u = 1 \triangleright u'$.
  Alors $Constante(u') \neq \emptyset$ et donc par hypothèse d'induction, $\mathtt{H}_{\mathtt{min}}(u') \neq +\infty$, $\mathtt{H}_{\mathtt{max}}(u') \neq -\infty$ et $\mathtt{H}_{\mathtt{max}}(u') \geq \mathtt{H}_{\mathtt{min}}(u')$. Il s'en suit que $\mathtt{H}_{\mathtt{min}}(u) \neq +\infty$, $\mathtt{H}_{\mathtt{max}}(u) \neq -\infty$ et $\mathtt{H}_{\mathtt{max}}(u) \geq \mathtt{H}_{\mathtt{min}}(u)$.
- Si $u = u' \parallel u''$.
  Alors $Constante(u') \neq \emptyset$ ou $Constante(u'') \neq \emptyset$. Ces deux cas se traitent de manière analogue, nous supposons que $Constante(u') \neq \emptyset$.
  D'après l'hypothèse d'induction, $\mathtt{H}_{\mathtt{min}}(u') \neq +\infty$, $\mathtt{H}_{\mathtt{max}}(u') \neq -\infty$ et $\mathtt{H}_{\mathtt{max}}(u') \geq \mathtt{H}_{\mathtt{min}}(u')$. Or, $\mathtt{H}_{\mathtt{max}}(u) \geq \mathtt{H}_{\mathtt{max}}(u')$ et $\mathtt{H}_{\mathtt{min}}(u) \leq \mathtt{H}_{\mathtt{min}}(u')$, donc $\mathtt{H}_{\mathtt{min}}(u) \neq +\infty$, $\mathtt{H}_{\mathtt{max}}(u) \neq -\infty$ et $\mathtt{H}_{\mathtt{max}}(u) \leq \mathtt{H}_{\mathtt{min}}(u)$.

⌟

Nous déduisons des lemmes 9.10 et 9.26 :

Conséquence 9.27 *Un terme TBPP u peut faire une action si et seulement si* $\mathtt{H}_{\mathtt{min}}(u) \neq +\infty$.

Lemme 9.28 *Étant donnés Δ, une déclaration TBPP et f une fonction de durée,*

- *Pour tout u, si $u \xrightarrow{a,n} v$, alors* $\mathtt{H_{min}}(u) \leq n \leq \mathtt{H_{max}}(u)$ *et* $\mathtt{H_{min}}(u) \leq \mathtt{H_{min}}(v)$*.*
- *Dans l'autre sens, si u peut faire une action, alors il existe une transition $u \xrightarrow{a,n} v$ avec $n = \mathtt{H_{min}}(u)$ et une transition $u \xrightarrow{a',n'} v'$ avec $n' = \mathtt{H_{max}}(u)$.*

Démonstration. Nous démontrons d'abord la première partie du lemme 9.28 par induction sur u. Nous supposons que $u \xrightarrow{a,n} v$.

. Si $u = \mathit{Nil}$.

Alors la propriété est vérifiée car $u \not\rightarrow$.

. Si $u = X \in \mathcal{X}$.

Alors $n = 0$ et il existe $X \xrightarrow{a} t \in \Delta$ tel que $v = f(a) \triangleright t$. $\mathtt{H_{min}}(u) = \mathtt{H_{max}}(u) = 0$ et $n = 0$, donc $\mathtt{H_{min}}(u) \leq n \leq \mathtt{H_{max}}(u)$ et $\mathtt{H_{min}}(v) = f(a) > 0$, donc $\mathtt{H_{min}}(u) \leq \mathtt{H_{min}}(v)$.

. Si $u = u_1 \parallel u_2$, alors :

– Soit $v = v_1 \parallel u_2$ et $u_1 \xrightarrow{a,n} v_1$;
– Soit $v = u_1 \parallel v_2$ et $u_2 \xrightarrow{a,n} v_2$.

Ces deux cas se traitent de manière analogue, nous supposons que $v = v_1 \parallel u_2$ et $u_1 \xrightarrow{a,n} v_1$. Par hypothèse d'induction, $\mathtt{H_{min}}(u_1) \leq n \leq \mathtt{H_{max}}(u_1)$ et $\mathtt{H_{min}}(u_1) \leq \mathtt{H_{min}}(v_1)$.

Or d'une part, $\mathtt{H_{min}}(u_1) \geq \mathtt{H_{min}}(u)$ et $\mathtt{H_{max}}(u_1) \leq \mathtt{H_{max}}(u)$ et d'autre part, $\mathtt{H_{min}}(v_1) \geq \mathtt{H_{min}}(v)$. Il s'ensuit que $\mathtt{H_{min}}(u) \leq n \leq \mathtt{H_{max}}(u)$ et $\mathtt{H_{min}}(u) \leq \mathtt{H_{min}}(v)$.

. Si $u = 1 \triangleright u'$.

Alors $u' \xrightarrow{a,n-1} v'$ et $v = 1 \triangleright v'$. Par hypothèse d'induction, $\mathtt{H_{min}}(u') \leq n-1 \leq \mathtt{H_{max}}(u')$ et $\mathtt{H_{min}}(u') \leq \mathtt{H_{min}}(v')$. Il s'ensuit que $\mathtt{H_{min}}(u')+1 \leq n \leq \mathtt{H_{max}}(u')+1$ et $\mathtt{H_{min}}(u') + 1 \leq \mathtt{H_{min}}(v') + 1$. D'où $\mathtt{H_{min}}(u) \leq n \leq \mathtt{H_{max}}(u)$ et $\mathtt{H_{min}}(u) \leq \mathtt{H_{min}}(v)$.

Nous montrons maintenant la seconde partie du lemme 9.28 par induction sur u. Nous supposons que u peut faire une action donc $\mathit{Constante}(u) \neq \emptyset$.

. Si $u = X \in \mathcal{X}$.

Alors il existe $X \xrightarrow{a} t \in \Delta$ par définition d'une déclaration. Donc $u \xrightarrow{a,0} f(a) \triangleright t$. $0 = \mathtt{H_{min}}(u) = \mathtt{H_{max}}(u)$, donc la propriété est vérifiée.

. Si $u = u_1 \parallel u_2$.

Alors soit $\mathtt{H_{min}}(u) = \mathtt{H_{min}}(u_1)$, soit $\mathtt{H_{min}}(u) = \mathtt{H_{min}}(u_2)$.

Nous supposons que $\mathtt{H_{min}}(u) = \mathtt{H_{min}}(u_1)$, le cas $\mathtt{H_{min}}(u) = \mathtt{H_{min}}(u_2)$ se traite de manière analogue. u peut faire une action donc, d'après la conséquence 9.27, $\mathtt{H_{min}}(u) \neq \infty$, par conséquent $\mathtt{H_{min}}(u_1) \neq \infty$ et d'après la conséquence 9.27, u_1 peut faire une action. Par hypothèse d'induction, il existe une transition $u_1 \xrightarrow{a,n} v_1$ avec $n = \mathtt{H_{min}}(u_1)$. Il s'ensuit que $u \xrightarrow{a,n} v_1 \parallel u_2$ et $n = \mathtt{H_{min}}(u)$.

$\mathtt{H_{max}}(u)$ se traite de manière analogue.

. Si $u = 1 \triangleright u'$.

Alors comme u peut faire une action, u' aussi. Or, par hypothèse d'induction, il existe une transition $u' \xrightarrow{a,n} v'$ avec $n = \mathtt{H}_{\mathtt{min}}(u')$ et une transition $u' \xrightarrow{a',n'} v''$ avec $n' = \mathtt{H}_{\mathtt{max}}(u')$. Par conséquent, $u \xrightarrow{a,n+1} 1 \triangleright v'$ et $u \xrightarrow{a',n'+1} 1 \triangleright v''$. De plus, $\mathtt{H}_{\mathtt{min}}(u) = \mathtt{H}_{\mathtt{min}}(u')+1 = n+1$ et $\mathtt{H}_{\mathtt{max}}(u) = \mathtt{H}_{\mathtt{max}}(u')+1 = n'+1$. Donc, la propriété est vérifiée.

⌟

Le lemme qui suit est fondamental, il exprime le fait que nous pouvons toujours rendre $\mathtt{H}_{\mathtt{min}}$ plus grand qu'une valeur arbitraire.

Lemme 9.29 *Étant donnés Δ, une déclaration TBPP et f une fonction de durée, pour tout u et tout $n \in \mathbb{N}$, il existe une exécution $u \xrightarrow{*} v$ telle que $\mathtt{H}_{\mathtt{min}}(v) > n$.*

Démonstration. Nous montrons par induction sur $(n, |u|)$ ordonné lexico-graphiquement qu'il existe une exécution $u \xrightarrow{*} v$ telle que $\mathtt{H}_{\mathtt{min}}(v) > n$.

- Si $u = \mathit{Nil}$ et n est quelconque.

  Alors $\mathtt{H}_{\mathtt{min}}(u) = \infty$, donc $v = u$ convient.
- Si $u = X \in \mathcal{X}$ et $n = 0$

  Alors il existe $X \xrightarrow{a}_{\Delta} t \in \Delta$, donc $u \xrightarrow{a,0} f(a) \triangleright t$. Nous avons $\mathtt{H}_{\mathtt{min}}(f(a) \triangleright t) \geq f(a)$ et $f(a) \neq 0$. Il s'ensuit que $v = f(a) \triangleright t$ convient.
- Si $u = X \in \mathcal{X}$ et $n > 0$.

  Alors il existe $u \xrightarrow{a}_{\Delta} t \in \Delta$, donc $u \xrightarrow{a,0} f(a) \triangleright t$. Nous avons $\mathtt{H}_{\mathtt{min}}(v') > n-1$, donc par hypothèse d'induction, il existe v' tel que $t \xrightarrow{*} v'$ et $\mathtt{H}_{\mathtt{min}}(v') > n-1$. D'où $f(a) \triangleright t \xrightarrow{*} f(a) \triangleright v'$ et $\mathtt{H}_{\mathtt{min}}(f(a) \triangleright v') = f(a) + \mathtt{H}_{\mathtt{min}}(v')$. Donc, $v = f(a) \triangleright v'$ convient.
- Si $u = u_1 \parallel u_2$ et n est quelconque.

  Alors u_1 et u_2 sont plus petits que u, donc par hypothèse d'induction, il existe deux termes v_1 et v_2 tels que $u_1 \xrightarrow{*} v_1$, $\mathtt{H}_{\mathtt{min}}(v_1) > 0$, $u_2 \xrightarrow{*} v_2$ et $\mathtt{H}_{\mathtt{min}}(v_2) > 0$. Le terme $v = v_1 \parallel v_2$ convient.
- Si $u = 1 \triangleright u'$ et n est quelconque.

  Alors u' est plus petit que u, donc par hypothèse d'induction, il existe v' tel que $\mathtt{H}_{\mathtt{min}}(v') > 0$ et $u' \xrightarrow{*} v'$. Le terme $v = 1 \triangleright v'$ convient.

⌟

10. Une équivalence temporisée décidable pour TBPP

Sommaire

Dans le chapitre précédent, nous avons présenté une algèbre de processus dont la sémantique permet d'attribuer à chaque action la date à laquelle elle commence. Nous allons maintenant chercher à comparer différents processus, avec un critère de performance.

Plus précisément, nous voulons ordonner des processus qui font les mêmes actions en fonction du temps qu'ils mettent à accomplir ces actions. Nous allons donc présenter différentes équivalences et un préordre qui distinguent les processus en fonction de leur rapidité. Puis, nous allons nous concentrer sur l'équivalence de performance et nous allons montrer qu'elle est décidable pour TBPP.

10.1. Des équivalences et un préordre d'efficacité

Dans cette section, nous présentons plusieurs équivalences et un préordre qui classent des processus accomplissant les mêmes actions en fonction de leur vitesse. Nous allons en particulier distinguer l'équivalence de performance de l'équivalence de compétition.

Une relation classique est la bisimulation [Mil89]. Cette équivalence peut être décrite comme un jeu entre deux processus qui cherchent à s'imiter l'un l'autre lors de *parties*. Une *partie* est une succession d'actions effectuées par les deux processus. À chaque tour, un processus est désigné comme *imitant* et l'autre comme *imité*. Le processus *imité* fait une action et le processus *imitant* doit pouvoir faire une action de même nom. Les rôles d'*imitant* et d'*imité* sont redistribués à chaque tour.

Les équivalences que nous étudions ont été définies par R. Gorrieri, M. Roccetti et E. Stancampiano [GRS95]. Outre les propriétés de bisimulation, elles doivent remplir les conditions suivantes :

- Pour l'équivalence de performance, l'action imitant a lieu à la même date que l'action imitée ;
- Pour l'équivalence de compétition, au début de la *partie*, l'un des deux processus doit être *choisi* et, durant toute la partie, quel que soit son rôle (*imitant* ou *imité*), ses actions doivent avoir lieu à la même date ou avant les actions de l'autre processus. À chaque partie, le processus *choisi* peut changer, mais il ne change pas au cours d'une partie.

10.1.1. L'équivalence de performance

L'équivalence de f-performance correspond à la bisimulation forte de R. Milner [Mil89] sur les transitions de TBPP où les informations sur les dates sont prises en compte.

Définition 10.1 *Une relation $\mathcal{R} \subseteq \mathcal{T} \times \mathcal{T}$ est appelée une **relation de f-performance** si $u \,\mathcal{R}\, v$ entraîne*

1. *Pour tout $u \xrightarrow{a,n}_f u'$, il existe $v \xrightarrow{a,n}_f v'$ tel que $u' \,\mathcal{R}\, v'$;*
2. *Et vice versa : pour tout $v \xrightarrow{a,n}_f v'$, il existe $u \xrightarrow{a,n}_f u'$ avec $u' \,\mathcal{R}\, v'$.*

Cette définition est illustrée par la figure 10.1.

Définition 10.2 *Deux termes TBPP u et v sont **f-p équivalents** (notés $u \sim_f v$) s'il existe une relation de f-performance $\mathcal{R}$ telle que $u \,\mathcal{R}\, v$.*

Exemple 10.3 Nous supposons que $f(a) = 1$ et nous considérons les définitions de processus $X \stackrel{\text{def}}{=} a.(X \parallel X)$ et $Y \stackrel{\text{def}}{=} a.Y$. Alors $X \not\sim_f Y$, en effet l'exécution

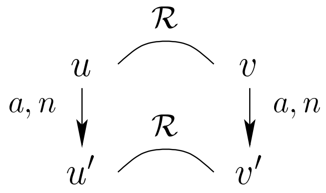


FIG. 10.1. – Une relation de **f**-performance.

$X \xrightarrow{a,0} 1 \triangleright (X \parallel X) \xrightarrow{a,1} 1 \triangleright (X \parallel 1 \triangleright (X \parallel X)) \xrightarrow{a,1} 1 \triangleright (1 \triangleright (X \parallel X) \parallel 1 \triangleright (X \parallel X))$ ne peut pas être imitée par Y. La figure 10.2 montre l'exécution de X dans le système de transitions qui lui est associé.

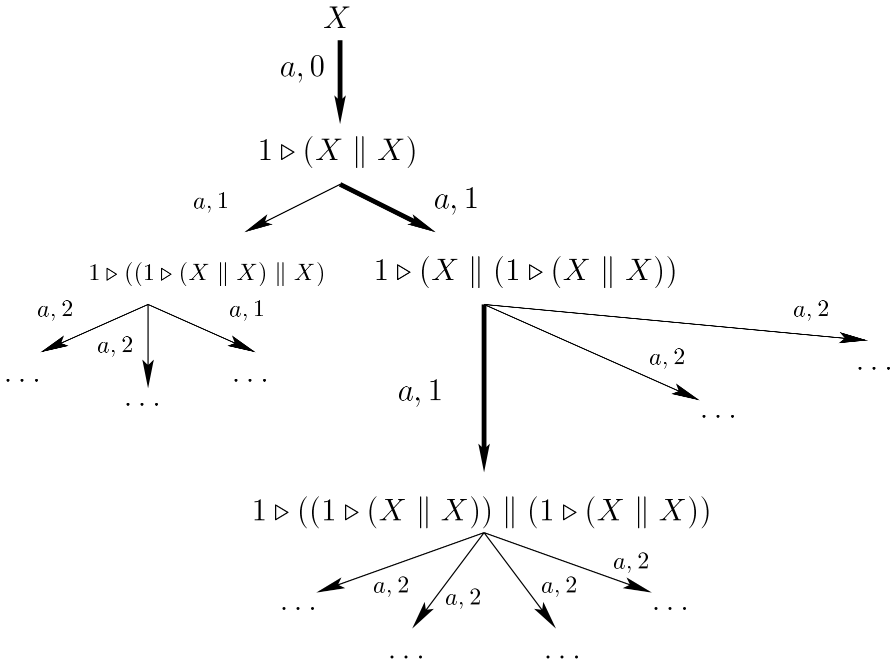


FIG. 10.2. – Le système de transitions associé à X.

Pourtant X et Y sont bisimilaires quand le temps n'est pas pris en compte, ils sont tous les deux bisimilaires à a^ω, c'est-à-dire au processus qui ne fait que des a. Cependant, X «va plus vite» que Y : les dates de ses actions sont inférieures ou égales à celles des actions de Y.

Nous allons maintenant donner un certain nombre de propriétés de la relation de **f**-performance qui, pour la plupart, sont des traductions ou des adaptations des mêmes propriétés concernant la bisimulation.

Remarque 10.4 Étant données Δ une déclaration TBPP et f une fonction de durée, $u \sim_f Nil$ si et seulement si u ne peut pas faire d'action.

Nous avons donc la propriété suivante qui est un résumé du lemme 9.10, du lemme 9.26 et de la remarque 10.4.

Lemme 10.5 *Étant données* Δ *une déclaration TBPP et* f *une fonction de durée, les propositions suivantes sont équivalentes :*
- $u \sim_f \mathit{Nil}$,
- *u ne peut pas faire d'action,*
- $\mathit{Constante}(u) = \emptyset$,
- $\mathtt{H}_{\mathtt{max}}(u) < \mathtt{H}_{\mathtt{min}}(u)$,
- $\mathtt{H}_{\mathtt{min}}(u) = +\infty$,
- $\mathtt{H}_{\mathtt{max}}(u) = -\infty$.

Proposition 10.6 *Étant données* Δ *une déclaration TBPP et* f *une fonction de durée,*
- $\sim_f$ *est la plus grande relation de* f*-performance,*
- $\sim_f$ *est une équivalence.*

Démonstration. Ces résultats sont les traductions du fait que la bisimulation est la plus grande relation de bisimulation et du fait que la bisimulation est une équivalence. Une preuve est proposée à la page 91 de [Mil89]. ⌟

Proposition 10.7 *La relation* $\sim_f$ *est une congruence pour les opérateurs* $\|$ *et* $1\triangleright$*, si* $u \sim_f v$ *et* $u' \sim_f v'$*, alors*

$$\begin{array}{rcl} u \parallel u' & \sim_f & v \parallel v', \\ 1 \triangleright u & \sim_f & 1 \triangleright v. \end{array}$$

Démonstration. Nous utilisons le fait que les règles SOS de $\rightarrow_f$ sont dans le format *tyft/tyxt* [GV92] ou encore dans le format de De Simone [De 85] pour appliquer le théorème 2.2.31 de [BV01] qui donne le résultat. ⌟

Remarque 10.8 Nous retrouvons le résultat de la proposition 4.3 de [CGR97], ce qui n'est pas surprenant étant donné que notre algèbre diffère peu de celle de F. Corradini, R. Gorrieri et M. Roccetti.

Proposition 10.9 *Étant données* Δ *une déclaration TBPP et* f *une fonction de durée, si* $u \sim_f v$*, alors* $\mathtt{H}_{\mathtt{min}}(u) = \mathtt{H}_{\mathtt{min}}(v)$ *et* $\mathtt{H}_{\mathtt{max}}(u) = \mathtt{H}_{\mathtt{max}}(v)$.

Démonstration. Si $u \sim_f v \sim_f \mathit{Nil}$, nous avons le résultat (lemme 10.5).

Nous supposons que $u \sim_f v$ et que $u \not\sim_f \mathit{Nil}$. Nous allons montrer que $\mathtt{H}_{\mathtt{min}}(v) \leq \mathtt{H}_{\mathtt{min}}(u)$.

Nous avons $u \not\sim_f \mathit{Nil}$, donc u peut faire une action. D'après la seconde partie du lemme 9.28, il existe une transition $u \xrightarrow{a,\mathtt{H}_{\mathtt{min}}(u)} u'$. Comme $u \sim_f v$, il existe une transition $v \xrightarrow{a,\mathtt{H}_{\mathtt{min}}(u)} v'$. Donc d'après la première partie lemme 9.28, $\mathtt{H}_{\mathtt{min}}(v) \leq \mathtt{H}_{\mathtt{min}}(u)$.

Les rôles de u et v étant équivalents, $\mathtt{H_{min}}(v) = \mathtt{H_{min}}(u)$.
$\mathtt{H_{max}}(v) = \mathtt{H_{max}}(u)$ se démontre de manière analogue. ⌟

L'équivalence de f-performance obéit aux lois d'associativité, de commutativité et de distributivité suivantes :

Proposition 10.10 *Pour tous les termes* t, u, v *:*

$$\begin{aligned} u \parallel t &\sim_f t \parallel u, && \text{(Eq1)}\\ (u \parallel t) \parallel v &\sim_f u \parallel (t \parallel v), && \text{(Eq2)}\\ t \parallel Nil &\sim_f t, && \text{(Eq3)}\\ 1 \triangleright (u \parallel v) &\sim_f (1 \triangleright u) \parallel (1 \triangleright v), && \text{(Eq4)}\\ 1 \triangleright Nil &\sim_f Nil. && \text{(Eq5)} \end{aligned}$$

La loi distributive, (Eq4), est appelée l'**équation de distribution des horloges** dans [CGR97].

Remarque 10.11 Les équations (Eq1), (Eq2) et (Eq3) sont des adaptations à l'algèbre TBPP et à l'équivalence de f-performance des équations (1), (2) et (3) de la propriété 8 de la page 94 de [Mil89] qui concernent CCS et la bisimulation.

Démonstration. La démonstration d'une équation du type de celles de la proposition 10.10 est standard. C'est pourquoi nous ne donnerons que la preuve de l'équation (Eq4) qui est la plus complexe.

Pour montrer l'équation (Eq4), nous allons montrer que la relation $\mathcal{R} = \{(1 \triangleright (u \parallel v), (1 \triangleright u) \parallel (1 \triangleright v)) \mid u, v \in \mathcal{T}\}$ est une relation de f-performance.

- Si $1 \triangleright (u \parallel v) \xrightarrow{a,n} t$.
  Alors d'après la règle de transition Rb, il existe un terme t' tel que $u \parallel v \xrightarrow{a,n-1} t'$ et $t = 1 \triangleright t'$. Il s'ensuit d'après les règles de transition Rc et Rd que :
  – Soit $u \xrightarrow{a,n-1} u'$ et $t' = u' \parallel v$;
  – Soit $v \xrightarrow{a,n-1} v'$ et $t' = u \parallel v'$.
  - Si $u \xrightarrow{a,n-1} u'$.
    Alors d'après la règle de transition Rb, $1 \triangleright u \xrightarrow{a,n} 1 \triangleright u'$. Puis d'après la règle de transition Rc, $(1 \triangleright u) \parallel (1 \triangleright v) \xrightarrow{a,n} (1 \triangleright u') \parallel (1 \triangleright v)$.
    De plus, $(t, (1 \triangleright u') \parallel (1 \triangleright v)) \in \mathcal{R}$.
  - Si $v \xrightarrow{a,n-1} v'$.
    Alors d'après la règle de transition Rb, $1 \triangleright v \xrightarrow{a,n} 1 \triangleright v'$. Puis d'après la règle de transition Rd, $(1 \triangleright u) \parallel (1 \triangleright v) \xrightarrow{a,n} (1 \triangleright u) \parallel (1 \triangleright v')$.
    De plus, $(t, (1 \triangleright u) \parallel (1 \triangleright v')) \in \mathcal{R}$.

Dans les deux cas, il existe t'' tel que $(1 \triangleright u) \parallel (1 \triangleright v) \xrightarrow{a,n} t''$ et $(t, t'') \in \mathcal{R}$.

- Si $(1 \triangleright u) \parallel (1 \triangleright v) \xrightarrow{a,n} t$.

  Alors d'après les règles de transition Rc et Rd :
  – Soit $1 \triangleright u \xrightarrow{a,n} t'$ et $t = t' \parallel (1 \triangleright v)$;
  – Soit $1 \triangleright v \xrightarrow{a,n} t'$ et $t = (1 \triangleright u) \parallel t'$.

  - Si $1 \triangleright u \xrightarrow{a,n} t'$.

    Alors d'après la règle de transition Rb, il existe un terme u' tel que $u \xrightarrow{a,n-1} u'$ et $t' = 1 \triangleright u'$.
    Donc d'après la règle de transition Rc, $u \parallel v \xrightarrow{a,n-1} u' \parallel v$. Il s'ensuit d'après la règle Rb que $1 \triangleright (u \parallel v) \xrightarrow{a,n} 1 \triangleright (u' \parallel v)$.
    De plus, $(1 \triangleright (u' \parallel v), t) \in \mathcal{R}$.

  - Si $1 \triangleright v \xrightarrow{a,n} t'$.

    Alors d'après la règle de transition Rb, il existe un terme v' tel que $v \xrightarrow{a,n-1} v'$ et $t' = 1 \triangleright v'$.
    Donc d'après la règle de transition Rd, $u \parallel v \xrightarrow{a,n-1} u \parallel v'$. Il s'ensuit d'après la règle Rb que $1 \triangleright (u \parallel v) \xrightarrow{a,n} 1 \triangleright (u \parallel v')$.
    De plus, $(1 \triangleright (u \parallel v'), t) \in \mathcal{R}$.

  Dans les deux cas, il existe t'' tel que $(1 \triangleright (u \parallel v)) \xrightarrow{a,n} t''$ et $(t'', t) \in \mathcal{R}$.

Nous concluons que $\mathcal{R}$ est une relation de f-performance et donc que

$$1 \triangleright (u \parallel v) \sim_f (1 \triangleright u) \parallel (1 \triangleright v).$$

┘

L'équivalence de f-performance a donc de nombreuses propriétés qui la font apparaître comme une équivalence particulièrement intéressante pour comparer deux processus. La plupart de ces propriétés sont dues au fait que l'équivalence de f-performance est la bisimulation dans un cadre temporisé.

10.1.2. L'équivalence de performance indépendante

Nos définitions suivent celles de F. Corradini, R. Gorrieri et M. Roccetti dans la mesure où nous ne mélangeons pas les définitions fonctionnelles (les règles de Δ qui correspondent au programme) et les définitions liées au temps (la fonction de durée f qui correspond à une machine) [CGR97]. Cela nous permet de définir une notion d'équivalence de performance qui ne dépend pas de f :

Définition 10.12 *Deux termes u et v sont **i-équivalents** (notés $u \sim_{\mathrm{i}} v$) si $u \sim_f v$ pour toute fonction de durée f.*

La relation $\sim_{\mathrm{i}}$ est une congruence comme intersection de congruences.

Remarque 10.13 Un résultat annexe de notre étude est la preuve que $\sim_f$ et $\sim_{\mathrm{i}}$ coïncident pour tout f (corollaire 10.48). Cela nous paraît expliquer pourquoi F. Corradini, R. Gorrieri et M. Roccetti introduisent [CGR97] à la fois le préordre de f-performance et celui de performance indépendante (ces deux préordres ne coïncident pas) mais une seule équivalence de performance et ne font aucun commentaire sur ce point.

Cependant, comme nous ne pouvons pas prouver le corollaire 10.48 sans les développements techniques des sections suivantes, nous continuerons d'écrire $\sim_f$ aussi longtemps que nécessaire.

10.1.3. Le préordre de performance

Le préordre de performance introduit par F. Corradini, R. Gorrieri et M. Roccetti introduisent [CGR97] relie les vitesses de deux processus en comparant la date du début de leurs actions. Un processus A est plus rapide qu'un processus B s'il peut l'imiter ou en être imité en faisant les mêmes actions et si les dates des actions de A sont toujours antérieures ou égales à celles de B.

Définition 10.14 *Une relation $\mathcal{R} \in \mathcal{T} \times \mathcal{T}$ est appelée un **préordre de f-performance** si $u \mathcal{R} v$ entraîne pour tout $a \in Act$ et $n \in \mathbb{N}$*

1. *Si $u \xrightarrow{a,n}_f u'$, alors $v \xrightarrow{a,m}_f v'$ pour un v' et un $m \in \mathbb{N}$ avec $n \leq m$ et $u' \mathcal{R} v'$;*
2. *Si $v \xrightarrow{a,n}_f v'$, alors $u \xrightarrow{a,m}_f u'$ pour un u' et un $m \in \mathbb{N}$ avec $n \geq m$ et $u' \mathcal{R} v'$.*

Définition 10.15 *Nous écrivons $u \sqsubseteq_f v$ s'il existe un préordre de f-performance $\mathcal{R}$ tel que $u \mathcal{R} v$.*

Lemme 10.16 *Si $u \sim_f v$, alors $u \sqsubseteq_f v$.*

Démonstration. C'est une conséquence du fait qu'une relation de f-performance est un préordre de f-performance. ⌟

Proposition 10.17 *La relation $\sqsubseteq_f$ est une précongruence pour les opérateurs $\|$ et $1\triangleright$: si $u \sqsubseteq_f v$, alors*

$$\begin{array}{rcl} u \parallel t & \sqsubseteq_f & v \parallel t, \\ t \parallel u & \sqsubseteq_f & t \parallel v, \\ 1 \triangleright u & \sqsubseteq_f & 1 \triangleright v. \end{array}$$

Démonstration. Les preuves ont la même structure que celles pour l'équivalence de f-performance (propriété 10.7). ⌟

Définition 10.18 *L'**équivalence de f-compétition** est définie par*

$$\simeq_f \stackrel{def}{=} \sqsubseteq_f \cap \sqsupseteq_f .$$

Lemme 10.19 *Si $u \sim_f v$, alors $u \simeq_f v$.*

Démonstration. C'est une conséquence du lemme 10.16. ⌟

Remarque 10.20 Comme l'ont montré F. Corradini, R. Gorrieri et M. Roccetti [CGR97], $u \simeq_f v$ n'a pas pour conséquence que $u \sim_f v$. En effet, considérons l'exemple suivant :

$$\begin{aligned} X_1 &\stackrel{\text{def}}{=} a.(1 \triangleright (a.Nil \parallel a.Nil \parallel a.Nil)) + a.(1 \triangleright (a.a.a.Nil)), \\ X_2 &\stackrel{\text{def}}{=} a.(1 \triangleright (a.Nil \parallel a.Nil \parallel a.Nil)) + a.(1 \triangleright (a.a.Nil) \parallel 1 \triangleright a.Nil) \\ &\quad + a.(1 \triangleright (a.a.a.Nil)) \end{aligned}$$

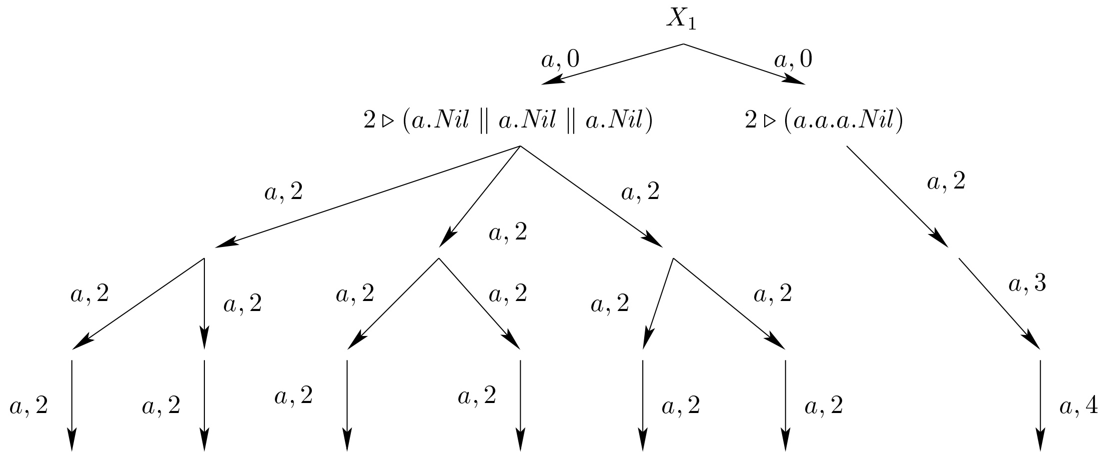


FIG. 10.3. – Le système de transitions associé à X_1.

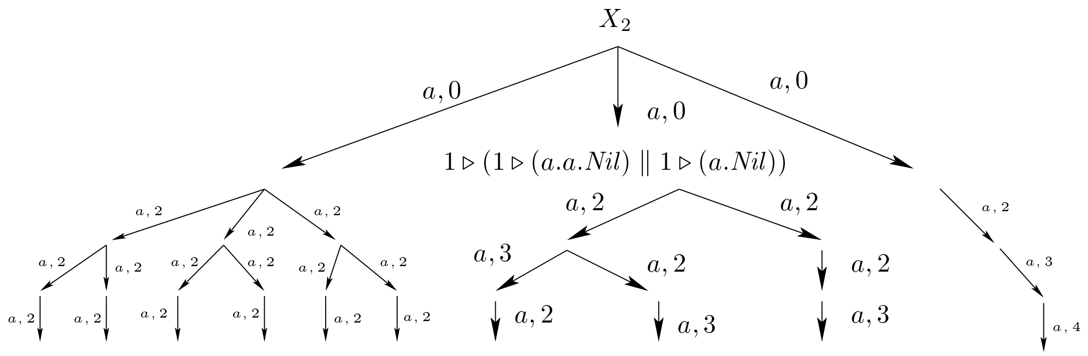


FIG. 10.4. – Le système de transitions associé à X_2.

Nous avons $X_1 \simeq_f X_2$ car les exécutions de gauche des figures 10.3 et 10.4 ont des dates inférieures ou égales à toutes celles des autres exécutions. Par contre, $X_1 \not\sim_f X_2$ car X_1 ne peut pas imiter les exécutions maximales du milieu de la figure 10.4 (les exécutions les plus visibles sur la figure).

10.2. La congruence structurelle

Dans cette section, nous introduisons la congruence structurelle pour les termes de TBPP. Cela nous permet d'exhiber une forme normale pour les termes qui généralise la forme normale usuelle pour BPP [CHM93]. Cette forme normale va jouer un rôle très important pour montrer que l'équivalence de performance est décidable pour TBPP.

Définition 10.21 *Nous notons $\equiv$ la plus petite congruence induite par les cinq équations de la proposition 10.10 :*

$$\begin{aligned} u \parallel t &\equiv t \parallel u, && (\mathrm{Eq1}') \\ (u \parallel t) \parallel v &\equiv u \parallel (t \parallel v), && (\mathrm{Eq2}') \\ t \parallel Nil &\equiv t, && (\mathrm{Eq3}') \\ 1 \triangleright (u \parallel v) &\equiv (1 \triangleright u) \parallel (1 \triangleright v), && (\mathrm{Eq4}') \\ 1 \triangleright Nil &\equiv Nil. && (\mathrm{Eq5}') \end{aligned}$$

Proposition 10.22 *Si $u \equiv v$, alors $u \sim_f v$ et $u \sim_{\mathrm{i}} v$.*

Démonstration. Clairement $u \equiv v$ entraîne $u \sim_f v$ car $\sim_f$ est une congruence (propriété 10.7) et elle satisfait les cinq équations. De plus, comme $\equiv$ ne dépend pas de f, $u \equiv v$ entraîne $u \sim_{\mathrm{i}} v$. ⌟

Définition 10.23 *Un terme $u \in \mathcal{T}$ est en* ***forme normale*** *s'il est de la forme $n_1 \triangleright X_1 \parallel \ldots \parallel n_k \triangleright X_k$, où les $X_i \in \mathcal{X}$ ne sont pas nécessairement distincts et où nous autorisons $n_i = 0$ ou $k = 0$ (auquel cas la forme normale est Nil).*

Proposition 10.24 *Tout terme peut être réécrit en un terme structurellement équivalent qui est en forme normale. De plus, cette forme normale est unique (modulo l'associativité et la commutativité de l'opérateur $\parallel$).*

Démonstration. L'équation Eq4′ et l'équation Eq2′ permettent de réécrire un terme en un terme structurellement équivalent qui est de la forme $n_1 \triangleright u_1 \parallel \ldots \parallel n_k \triangleright u_k$ où $u_i \in \mathcal{X} \cup \{Nil\}$. L'équation Eq5′ et l'équation Eq3′ permettent de supprimer tous les *Nil* sauf dans le cas où le terme est structurellement équivalent à *Nil*. Donc tout terme peut être réécrit en un terme structurellement équivalent qui est en forme normale.

Les équations Eq1′, Eq2′, Eq3′, Eq4′ et Eq5′ conservent le nom et le nombre de constantes de processus d'un terme et conservent pour toute variable le nombre d'opérateurs $\triangleright$ dans le champ desquels elle est. Par conséquent, la forme normale est unique (modulo l'associativité et la commutativité de l'opérateur $\parallel$). ⌟

Parfois, nous ne serons intéressés que par les sous-termes «$0 \rhd X_i$» de la forme normale d'un terme. Nous écrirons alors ce terme sous la forme

$$X_1 \parallel \ldots \parallel X_n \parallel 1 \rhd u.$$

Lorsqu'un terme t est en forme normale : $t = n_1 \rhd X_1 \parallel \ldots \parallel n_k \rhd X_k$, l'ensemble des n_i est exactement l'ensemble de toutes les dates auxquelles t peut faire une transition. Ceci a une conséquence très utile exprimée par le lemme suivant :

Lemme 10.25 *Soit u un terme, $u \equiv \mathit{Nil}$ si et seulement si $\mathtt{H_{min}}(u) = +\infty$.*

Démonstration. Si $u \equiv \mathit{Nil}$, alors d'après la propriété 10.22, $u \sim_f \mathit{Nil}$ donc $\mathtt{H_{min}}(u) = +\infty$ (lemme 10.5).

Si $\mathtt{H_{min}}(u) = +\infty$, alors la forme normale de u est Nil. En effet sinon, $\mathtt{H_{min}}(u)$ est le plus petit des n_i de la forme normale de u. Par conséquent, $u \equiv \mathit{Nil}$. ⌟

Le lemme 10.5 peut donc s'étendre de la façon suivante :

Conséquence 10.26 *Les propositions suivantes sont équivalentes :*
- $u \equiv \mathit{Nil}$,
- $u \sim_f \mathit{Nil}$,
- *u ne peut pas faire d'action,*
- $\mathit{Constante}(u) = \emptyset$,
- $\mathtt{H_{max}}(u) < \mathtt{H_{min}}(u)$,
- $\mathtt{H_{min}}(u) = +\infty$,
- $\mathtt{H_{max}}(u) = -\infty$.

10.3. Des résultats de simplification pour l'équivalence de performance

Dans cette section, nous démontrons un résultat très surprenant : l'équivalence de performance peut se réduire à l'égalité de formes normales. Pour cela, nous utilisons une approche qui consiste à décomposer les termes ligne après ligne. Cette approche a été introduite par R. Milner et F. Moller [MM93] et convient bien au cadre temporisé et au cadre normé (voir la proposition 30 de [AM96] et la proposition 2.2.8 de [Hen88]).

10.3.1. Préliminaires

Le lemme suivant est la réciproque de la proposition 10.7. Il met en évidence le lien entre les comportements des termes u et $1 \rhd u$.

Lemme 10.27 *$1 \rhd u \sim_f 1 \rhd v$ entraîne $u \sim_f v$.*

Démonstration. Nous faisons une preuve standard qui consiste à montrer que la relation $\mathcal{R} \stackrel{\text{def}}{=} \{(u_1, u_2) \mid 1 \triangleright u_1 \sim_f 1 \triangleright u_2\}$ est une équivalence de f-performance.

Soit $(u_1, u_2) \in \mathcal{R}$, $1 \triangleright u_1 \sim_f 1 \triangleright u_2$.

- Si $u_1 \xrightarrow{a,n} u_1'$.

  Alors d'après la règle de transition Rb, $1 \triangleright u_1 \xrightarrow{a,n+1} 1 \triangleright u_1'$. Comme $1 \triangleright u_1 \sim_f 1 \triangleright u_2$, il existe un terme t tel que $1 \triangleright u_2 \xrightarrow{a,n+1} t$ et $1 \triangleright u_1' \sim_f t$.

  Donc d'après la règle de transition Rb, il existe un terme u_2' tel que $u_2 \xrightarrow{a,n} u_2'$ et $t = 1 \triangleright u_2'$.

  Nous concluons qu'il existe un terme u_2' tel que $u_2 \xrightarrow{a,n} u_2'$ et $1 \triangleright u_1' \sim_f 1 \triangleright u_2'$.

- Si $u_2 \xrightarrow{a,n} u_2'$.

  Nous pouvons montrer de manière analogue au cas précédent qu'il existe un terme u_1' tel que $u_1 \xrightarrow{a,n} u_1'$ et $1 \triangleright u_1' \sim_f 1 \triangleright u_2'$.

Par conséquent, $\mathcal{R}$ est une équivalence de f-performance. ⌟

10.3.2. Produit séparé

Comme nous l'avons dit plus haut, nous avons besoin de décomposer les termes pour décider l'équivalence de performance. En particulier, nous allons décomposer en u et v le terme $u \parallel v$ lorsque celui-ci forme un produit séparé, c'est-à-dire lorsque les actions que peut faire u ont lieu avant chacune des actions que peut faire v.

Définition 10.28 *Soient u et v deux termes TBPP. Nous disons que u **précède** v si $\mathtt{H_{max}}(u) < \mathtt{H_{min}}(v)$, $u \not\sim_f Nil$ et $v \not\sim_f Nil$.*

Définition 10.29 *Un **produit séparé** est un terme $u \parallel v$ tel que u précède v.*

Cette notion syntaxique est utile car quand $u \parallel v$ exécute une action à l'instant n, il est possible d'attribuer cette action à u ou à v en se basant uniquement sur la valeur de n.

Remarque 10.30 Il est aisé de séparer un terme t écrit en forme normale : $t = n_1 \triangleright X_1 \parallel \dots \parallel n_k \triangleright X_k$, il suffit de choisir un entier n_i et de regrouper tous les termes $n_j \triangleright X_j$ où $n_j \leq n_i$ dans un terme u et tous les termes $n_k \triangleright X_k$ où $n_k > n_i$ dans un terme v. Alors t est congru à $u \parallel v$ qui est séparé.

Lemme 10.31 *Supposons que $u_1 \parallel u_2$ et $v_1 \parallel v_2$ soient deux produits séparés tels que u_1 et v_1 aient la même horloge maximale. Alors, $u_1 \parallel u_2 \sim_f v_1 \parallel v_2$ entraîne*

$$u_2 \sim_f v_2 \text{ et } u_1 \sim_f v_1.$$

Nous allons démontrer ce lemme en deux temps, nous commençons par montrer que $u_2 \sim_f v_2$, puis nous utilisons ce résultat pour montrer que $u_1 \sim_f v_1$.

Démonstration de $\mathbf{u_2 \sim_f v_2}$.

Cette propriété est facile à mettre en lumière avec les hypothèses de séparation.

Soit $\mathcal{R}$ l'ensemble de toutes les paires (u, v) telles que $u_1 \parallel u \sim_f v_1 \parallel v$ et telles que $u_1 \parallel u$ et $v_1 \parallel v$ soient tous deux séparés. Nous allons montrer que $\mathcal{R} \cup \sim_f$ est une équivalence de f-performance.

- Si $u \xrightarrow{a,n} u'$.

  Alors d'après la règle de transition Rd, $u_1 \parallel u \xrightarrow{a,n} u_1 \parallel u'$. De plus, d'après le lemme 9.28, $\mathtt{H_{min}}(u) \leq \mathtt{H_{min}}(u')$, donc $u_1 \parallel u'$ est également séparé ou $u' \sim_f Nil$.

  $u_1 \parallel u \sim_f v_1 \parallel v$, donc il existe un terme t tel que $v_1 \parallel v \xrightarrow{a,n} t$ et $u_1 \parallel u' \sim_f t$. D'après les règles de transition Rc et Rd, soit $v_1 \xrightarrow{a,n} v_1'$ et $t = v_1' \parallel v$, soit $v \xrightarrow{a,n} v'$ et $t = v_1 \parallel v'$.

  Comme $u \xrightarrow{a,n}$, $\mathtt{H_{min}}(u) \leq n$ (lemme 9.28), il s'ensuit comme $u_1 \parallel u$ est séparé que $\mathtt{H_{max}}(u_1) \leq n$. Or u_1 et v_1 ont la même horloge maximale, donc $\mathtt{H_{max}}(v_1) \leq n$. Par conséquent (lemme 9.28), $v \xrightarrow{a,n} v'$ et $t = v_1 \parallel v'$.

  – Si $u' \not\sim_f Nil$ et $v' \not\sim_f Nil$, alors $v_1 \parallel v'$ est un produit séparé car $\mathtt{H_{min}}(v) \leq \mathtt{H_{min}}(v')$ (lemme 9.28). Donc $v \xrightarrow{a,n} v'$, $u_1 \parallel u' \sim_f v_1 \parallel v'$ et $u_1 \parallel u'$ et $v_1 \parallel v'$ sont séparés. Nous concluons que $(u', v') \in \mathcal{R} \cup \sim_f$.

  – Si $u' \sim_f Nil$, alors $\mathtt{H_{max}}(u_1 \parallel u') = \mathtt{H_{max}}(u_1)$. Donc, d'après la proposition 10.9 comme $u_1 \parallel u' \sim_f v_1 \parallel v'$, $\mathtt{H_{max}}(v_1 \parallel v') = \mathtt{H_{max}}(u_1)$. Or u_1 et v_1 ont la même horloge maximale, par conséquent, $\mathtt{H_{max}}(v_1 \parallel v') = \mathtt{H_{max}}(v_1)$, d'où $\mathtt{H_{max}}(v') \leq \mathtt{H_{max}}(v_1)$.
    Par ailleurs, $\mathtt{H_{min}}(v) \leq \mathtt{H_{min}}(v')$ (lemme 9.28) et $\mathtt{H_{max}}(v_1) \leq \mathtt{H_{min}}(v)$ car $v_1 \parallel v$ est séparé. D'où, $\mathtt{H_{max}}(v') \leq \mathtt{H_{min}}(v')$. Le lemme 10.5 entraîne que $v' \sim_f Nil$ et donc $(u', v') \in \mathcal{R} \cup \sim_f$.

  – Si $v' \sim_f Nil$, nous montrons de manière analogue au cas précédent que $(u', v') \in \mathcal{R} \cup \sim_f$.

  Donc dans les deux cas, il existe un terme v' tel que $v \xrightarrow{a,n} v'$ et $(u', v') \in \mathcal{R} \cup \sim_f$.

- Si $v \xrightarrow{a,n} v'$.

  u et v ayant des rôles équivalents, nous montrons de façon analogue au cas précédent qu'il existe un terme u' tel que $u \xrightarrow{a,n} u'$ et $(u', v') \in \mathcal{R} \cup \sim_f$.

Nous en déduisons que $\mathcal{R} \cup \sim_f$ est une équivalence de f-performance et donc que $u_2 \sim_f v_2$. ⌟

Démonstration de $\mathbf{u_1 \sim_f v_1}$.

Soit $\mathcal{R}$ l'ensemble des paires (u, v) telles que u et v aient la même horloge maximale et telles qu'il existe un terme t tel que $u \parallel t \sim_f v \parallel t$ avec $u \parallel t$ et $v \parallel t$ séparés. Nous allons montrer que $\mathcal{R} \cup \sim_f$ est une équivalence de f-performance.

Soit (u, v) une paire de $\mathcal{R}$ associée à un certain t. Si ni u, ni v ne peuvent faire d'action, alors d'après le lemme 10.5, $u \sim_f v \sim_f Nil$. Sinon, nous commençons par montrer le résultat intermédiaire suivant :

Lemme 10.32 *Soit K la plus grande des horloges maximales de tous les successeurs immédiats de u et v. Il existe deux termes t' et t'' tels que $\mathtt{H_{min}}(t') > K$, $\mathtt{H_{min}}(t'') > K$ et $u \parallel t' \sim_f v \parallel t''$.*

Remarque 10.33 K est fini car TBPP est à branchement fini, c'est-à-dire qu'un terme TBPP ne peut faire qu'un nombre fini de transitions (il y a un nombre fini d'occurences de constantes de processus dans ce terme et chacune est le membre gauche d'un nombre fini de règles de Δ).

Démonstration. Grâce au lemme 9.29, il existe un mot w tel que $t \xrightarrow{w} t'$ et $\mathtt{H_{min}}(t') > K$.

D'après la règle de transition Rd, $u \parallel t \xrightarrow{w} u \parallel t'$. Comme $u \parallel t \sim_f v \parallel t$, il existe r tel que $v \parallel t \xrightarrow{w} r$ et $u \parallel t' \sim_f r$. Or w est une trace produite à partir de t, donc toutes ses dates sont supérieures ou égales à $\mathtt{H_{min}}(t)$. Comme $v \parallel t$ est séparé, $\mathtt{H_{max}}(v) < \mathtt{H_{min}}(t)$. Il s'ensuit d'après le lemme 9.28 que v ne peut faire aucune des actions de w. Par conséquent, d'après les règles de transition Rc et Rd, il existe t'' tel que $t \xrightarrow{w} t''$ et $r = v \parallel t''$. Nous avons donc un terme t'' tel que $u \parallel t' \sim_f v \parallel t''$.

Nous allons montrer que t'' peut faire une action si et seulement si t' peut faire la même action. Nous supposons qu'il existe une transition $t'' \xrightarrow{a,n}$, alors d'après lemme 9.28, $\mathtt{H_{min}}(t'') \leq n$. Comme $t \xrightarrow{w} t''$, d'après lemme 9.28, $\mathtt{H_{min}}(t) \leq \mathtt{H_{min}}(t'')$. De plus, comme $v \parallel t$ est séparé, $\mathtt{H_{max}}(v) < \mathtt{H_{min}}(t)$. Par conséquent, $\mathtt{H_{max}}(v) < n$, d'où $\mathtt{H_{max}}(u) < n$.

$t'' \xrightarrow{a,n}$, donc d'après la règle de transition Rd, $v \parallel t'' \xrightarrow{a,n}$. Comme $u \parallel t' \sim_f v \parallel t''$, $u \parallel t' \xrightarrow{a,n}$, d'après les règles de transition Rc et Rd, soit $u \xrightarrow{a,n}$, soit $t' \xrightarrow{a,n}$. Or nous venons de voir que $\mathtt{H_{max}}(u) < n$, donc d'après le lemme 9.28 $t' \xrightarrow{a,n}$.

Une démonstration analogue permet de voir que si t' peut faire une action, alors t'' peut faire la même action.

Par conséquent, si t' ne peut pas faire d'action, alors t'' ne peut pas faire d'action. Il s'ensuit, d'après la remarque 10.4, que $t'' \sim_f Nil$ et $\mathtt{H_{min}}(t'') > K$. Il nous reste donc à montrer que, dans le cas où $t' \not\sim_f Nil$, $\mathtt{H_{min}}(t'') > K$. Nous allons montrer la contraposée. Nous supposons que $\mathtt{H_{min}}(t'') \leq K$, alors d'après le lemme 9.28, comme t'' peut faire une action, t'' peut faire une action datée de $\mathtt{H_{min}}(t'')$. Or t' ne peut pas faire une action dont la date est plus petite que K car

$\mathtt{H_{min}}(t') > K$ (lemme 9.28). Ceci est en contradiction avec le fait que t' peut faire les mêmes actions que t''. ⌟

Il existe donc deux termes t et t' tels que $\mathtt{H_{min}}(t') > K$, $\mathtt{H_{min}}(t'') > K$ et $u \parallel t' \sim_f v \parallel t''$.

- Si $u \xrightarrow{a,n} u'$.

  Alors $u \parallel t' \xrightarrow{a,n} u' \parallel t'$. Comme $u \parallel t' \sim_f v \parallel t''$, il existe un terme r tel que $v \parallel t'' \xrightarrow{a,n}$ et $u' \parallel t' \sim_f r$. D'après les règles de transition Rc et Rd, soit $v \xrightarrow{a,n} v'$ et $r = v' \parallel t''$, soit $t'' \xrightarrow{a,n} t_1$ et $r = v \parallel t_1$.

  Or $\mathtt{H_{min}}(t'') > K$ et $K \geq n$ par définition de K. Donc d'après le lemme 9.28, t'' ne peut pas faire une action datée par n. Il s'ensuit que $v \xrightarrow{a,n} v'$.

  $u' \parallel t' \sim_f v' \parallel t''$, u' et v' sont des successeurs immédiats de u et v, donc, par définition de K, $\mathtt{H_{max}}(u') \leq K$ et $\mathtt{H_{max}}(v') \leq K$. Il s'ensuit que $u' \parallel t'$ et $v' \parallel t''$ sont des produits séparés.

  De plus, nous allons montrer que $\mathtt{H_{max}}(u') = \mathtt{H_{max}}(v')$. En effet, u' peut faire une action datée par $\mathtt{H_{max}}(u')$, donc $u' \parallel t'$, d'où $v' \parallel t''$ aussi. Or $\mathtt{H_{max}}(u') \leq K < \mathtt{H_{min}}(t'')$, il s'ensuit que v' est à l'origine de l'action. Nous en déduisons que $\mathtt{H_{max}}(v') \geq \mathtt{H_{max}}(u')$. Comme les rôles de u' et v' sont similaires dans ce cas, $\mathtt{H_{max}}(v') \leq \mathtt{H_{max}}(u')$ et donc $\mathtt{H_{max}}(v') = \mathtt{H_{max}}(u')$.

  Nous avons donc : $u' \parallel t' \sim_f v' \parallel t''$, $u' \parallel t'$ et $v' \parallel t''$ sont des produits séparés et $\mathtt{H_{max}}(v') = \mathtt{H_{max}}(u')$. D'après la première partie de notre lemme, $t' \sim_f t''$. Donc $u' \parallel t' \sim_f u' \parallel t''$, il s'ensuit que $u' \parallel t'' \sim_f v' \parallel t''$. Enfin, comme $\mathtt{H_{max}}(v') = \mathtt{H_{max}}(u')$ et $u' \parallel t'$ et $v' \parallel t''$ sont des produits séparés, $u' \parallel t''$ et $v' \parallel t''$ sont des produits séparés.

  Il existe donc un terme v' tel que $v \xrightarrow{a,n} v'$, $\mathtt{H_{max}}(v') = \mathtt{H_{max}}(u')$ et il existe un terme t'' tel que $u' \parallel t'' \sim_f v' \parallel t''$ et $u' \parallel t''$ et $v' \parallel t''$ sont des produits séparés. Par conséquent, il existe un terme v' tel que $v \xrightarrow{a,n} v'$ et $(u', v') \in \mathcal{R} \cup \sim_f$.

- Si $v \xrightarrow{a,n} v'$.

  Nous montrons de façon analogue qu'il existe u' tel que $u \xrightarrow{a,n} u'$ et $(u', v') \in \mathcal{R} \cup \sim_f$.

Nous en déduisons que $\mathcal{R} \cup \sim_f$ est une équivalence de f-performance et donc que $u_1 \sim_f v_1$. ⌟

10.3.3. Théorème de décomposition

Nous allons maintenant pouvoir donner un théorème de décomposition pour TBPP.

Théorème 10.34 *Supposons* $u \equiv n_1 \triangleright X_1 \parallel \ldots \parallel n_m \triangleright X_m$ *et* $v \equiv n'_1 \triangleright X'_1 \parallel \ldots \parallel n'_{m'} \triangleright X'_{m'}$*. Alors* $u \sim_f v$ *si et seulement s'il existe une bijection* $h : [1..m] \to [1..m']$ *telle que* $n_i = n'_{h(i)}$ *et* $X_i \sim_f X'_{h(i)}$ *pour tout* i*.*

Ce théorème exprime que l'équivalence de f-performance de u et de v peut se réduire à la combinaison de l'équivalence de f-performance de constantes.

La démonstration repose sur les deux lemmes suivants.

Lemme 10.35 *Supposons* $X_1 \parallel \ldots \parallel X_m \sim_f X'_1 \parallel \ldots \parallel X'_{m'}$*. Alors* $m = m'$ *et à chaque* X_i*, nous pouvons associer un* X'_j *tel que* $X_i \sim_f X'_j$*.*

Démonstration. $m = m'$ car toutes les exécutions maximales de $X_1 \parallel \ldots \parallel X_m$ ont exactement m transitions datées par 0.

Maintenant, si nous prenons des actions a_i telles que $X_i \xrightarrow{a_i,0} u_i$ pour tout $i \geq 1$. Nous avons $X_1 \parallel \ldots \parallel X_m \xrightarrow{a_2,0}\xrightarrow{a_3,0} \cdots \xrightarrow{a_m,0} X_1 \parallel 1 \triangleright u$.

Comme $X_1 \parallel \ldots \parallel X_m \sim_f X'_1 \parallel \ldots \parallel X'_{m'}$, il existe une exécution $X'_1 \parallel \ldots \parallel X'_m \xrightarrow{a_2,0}\xrightarrow{a_3,0} \cdots \xrightarrow{a_m,0} v$ avec $X_1 \parallel 1 \triangleright u \sim_f v$. Mais v est atteint en $m-1$ transitions datées par 0 à partir de $X'_1 \parallel \ldots \parallel X'_m$. Par conséquent, v est structurellement équivalent à un terme de la forme $X'_j \parallel 1 \triangleright u'$.

Le lemme 10.31 entraîne que $X_1 \sim_f X'_j$ (et $u \sim_f u'$), ce qui conclut la démonstration. ⌟

Lemme 10.36 *Supposons* $X_1 \parallel \ldots \parallel X_m \sim_f X'_1 \parallel \ldots \parallel X'_m$*. Alors, il existe une bijection* $h : [1..m] \to [1..m]$ *telle que* $X_i \sim_f X'_{h(i)}$ *pour tout* i*.*

Démonstration. Nous considérons les différentes classes d'équivalence du multi-ensemble $\{X_1, \ldots, X_m, X'_1, \ldots, X'_m\}$ induites par $\sim_f$. Si chaque classe contient le même nombre de X_i que de X'_j, alors h est facile à construire. Nous allons montrer par l'absurde que c'est toujours le cas.

Supposons qu'une classe ne contienne pas exactement le même nombre de X_i que de X'_j, par exemple supposons que cette classe soit $\{X_1, X_2, \ldots, X_p, X'_1, X'_2, \ldots, X'_q\}$ avec $p < q$ (les autres cas se traitent de manière analogue). Pour chaque $i \leq p$, nous prenons des actions a_i telles que $X_i \xrightarrow{a_i,0}$ et nous considérons $w = (a_1, 0) \ldots (a_p, 0)$. Nous avons l'exécution $X_1 \parallel \ldots \parallel X_m \xrightarrow{w} 1 \triangleright u \parallel X_{p+1} \parallel \ldots \parallel X_m$. Cette exécution est imitée par $X'_1 \parallel \ldots \parallel X'_m \xrightarrow{w} 1 \triangleright u' \parallel X'_{i_{p+1}} \parallel \ldots \parallel X'_{i_m}$.

Le lemme 10.31 entraîne que $X_{p+1} \parallel \ldots \parallel X_m \sim_f X'_{i_{p+1}} \parallel \ldots \parallel X'_{i_m}$. Maintenant, l'un des indices de l'ensemble $\{i_{p+1}, \ldots, i_m\}$, par exemple, j, doit appartenir à l'ensemble $\{1, \ldots, q\}$. Ceci est en contradiction avec le lemme 10.35 car nous avons supposé que X'_j n'était dans la classe d'équivalence d'aucune des constantes $X_{p+1}, \ldots, X_m$. ⌟

Remarque 10.37 Le théorème 10.34 n'est plus correct si nous autorisons les fonctions de durée à prendre la valeur 0 (ce qui a d'ailleurs peu de sens dans le cadre de notre travail). Par exemple, les termes de l'exemple 10.3 sont f-p équivalents si $f(a) = 0$.

10.4. L'équivalence de performance ne dépend pas de f

Dans cette section, nous montrons que l'équivalence de performance ne dépend pas de f. Nous montrons que $\sim_f$ restreinte aux paires de constantes de processus est une base de bisimulation, cette nouvelle notion ne dépend pas de f. Ce travail est à rapprocher de la recherche d'une base finie engendrant la bisimulation pour certaines algèbres de processus comme BPA [CHS95].

10.4.1. L'extension aux termes d'une relation sur les constantes

Lemme 10.38 *Soit $\mathcal{R}$ une relation d'équivalence entre des constantes de $\mathcal{X}$. Soit $\mathcal{R}'$ la relation sur les termes définie par $u\ \mathcal{R}'\ v$ si les formes normales $n_1 \triangleright X_1 \parallel \ldots$ et $n'_1 \triangleright X'_1 \parallel \ldots$ de u et v peuvent être reliées par une bijection h telle que $n_i = n'_{h(i)}$ et $X_i\ \mathcal{R}\ X'_{h(i)}$.*

$\mathcal{R}'$ est une relation d'équivalence sur les termes qui prolonge $\mathcal{R}$.

Démonstration. $\mathcal{R}'$ prolonge $\mathcal{R}$ de façon évidente. De plus, $\mathcal{R}'$ est réflexive, symétrique (il suffit de prendre la bijection réciproque) et transitive (nous composons les bijections). ⌟

Par abus, nous noterons encore $\mathcal{R}$ cette extension.

Le théorème 10.34 entraîne que $\sim_f$ est l'extension de $\sim_f \cap (\mathcal{X} \times \mathcal{X})$.

10.4.2. Base de bisimulation

Définition 10.39 *Une relation d'équivalence $\mathcal{R}$ entre les constantes de $\mathcal{X}$ est une* ***base de bisimulation*** *si pour tout couple $X\ \mathcal{R}\ Y$ et toute règle $X \xrightarrow{a}_\Delta u$, il existe $Y \xrightarrow{a}_\Delta v$ telle que $u\mathcal{R}v$.*

Définition 10.40 *Deux termes TBPP u et v sont* ***b-équivalents*** *(notés $u \approx v$) s'il existe une base de bisimulation $\mathcal{R}$ telle que $u\mathcal{R}v$.*

Proposition 10.41 *$\approx \cap (\mathcal{X} \times \mathcal{X})$ est la plus grande des bases de bisimulation.*

Démonstration. Ce résultat est une adaptation du fait que la bisimulation est la plus grande relation de bisimulation. La preuve se trouve à la page 90 de [Mil89]. ⌟

10.4.3. L'extension de $\approx$ et l'équivalence de performance coïncident

Le but de ce paragraphe est de montrer la propriété suivante :

Proposition 10.42 *La relation $\sim_f$ coïncide avec $\approx$.*

Lemme 10.43 *L'extension d'une base de bisimulation est une équivalence de f-performance.*

Démonstration. Soit une base de bisimulation $\mathcal{R}$. Nous allons montrer que $\mathcal{R}$ est une relation de f-performance «up to» $\sim_f$ c'est-à-dire que si $(u, v) \in \mathcal{R}$,

1. Pour tout $u \xrightarrow{a,n}_f u'$, il existe $v \xrightarrow{a,n}_f v'$ tel que $u' \sim_f \mathcal{R} \sim_f v'$;
2. Et vice versa : pour tout $v \xrightarrow{a,n}_f v'$, il existe $u \xrightarrow{a,n}_f u'$ avec $u' \sim_f \mathcal{R} \sim_f v'$.

Une adaptation de la proposition 6 page 93 de [Mil89] permet de déduire alors que $\mathcal{R}$ est une relation de f-performance.

Soit $(u, v) \in \mathcal{R}$, nous supposons que $u \xrightarrow{a,n} u'$.

Soit $n_1 \triangleright X_1 \parallel \ldots$ la forme normale de u, d'après la propriété 10.22, $u \sim_f X_1 \parallel \ldots$ Donc, il existe un terme t tel que $n_1 \triangleright X_1 \parallel \ldots \xrightarrow{a,n} t$ et $u' \sim_f t$. D'après les règles de transition, il existe i tel que $X_i \xrightarrow{a,n-n_i} f(a) \triangleright t_i$ et $t = n_1 \triangleright X_1 \parallel \ldots \parallel n_{i-1} \triangleright X_{i-1} \parallel (n_i + f(a)) \triangleright t_i \parallel n_{i+1} \triangleright X_{i+1} \parallel \ldots$. D'où $n = n_i$ et il existe une règle $X_i \xrightarrow{a}_\Delta t_i$ dans Δ.

Comme $(u, v) \in \mathcal{R}$, il existe une bijection h telle que la forme normale de v soit $n_1 \triangleright X'_1 \parallel \ldots$ et telle que $X_j \ \mathcal{R} \ X'_j$ pour tout j.

Comme $\mathcal{R}$ est une base de bisimulation, il exite t'_i tel que $X'_i \xrightarrow{a}_\Delta t'_i$ dans Δ avec $t_i \ \mathcal{R} \ t'_i$. Il s'ensuit que $n_1 \triangleright X'_1 \parallel \ldots \xrightarrow{a,n} t'$ avec $t' = n_1 \triangleright X'_1 \parallel \ldots \parallel n_{i-1} \triangleright X'_{i-1} \parallel (n_i + f(a)) \triangleright t'_i \parallel n_{i+1} \triangleright X'_{i+1} \parallel \ldots$ Nous avons donc $n_1 \triangleright X'_1 \parallel \ldots \xrightarrow{a,n} t'$ avec $t \ \mathcal{R} \ t'$.

Or $n_1 \triangleright X'_1 \parallel \ldots$ est la forme normale de v donc d'après la propriété 10.22, $v \sim_f X_1 \parallel \ldots$ Donc, il existe un terme v' tel que $v \xrightarrow{a,n} v'$ et $v' \sim_f t'$. Nous avons donc $v \xrightarrow{a,n} v'$ et $u' \sim_f \mathcal{R} \sim_f v'$.

De même, nous avons que si $v \xrightarrow{a,n} v'$, alors il existe un terme u' tel que $u \xrightarrow{a,n} u'$ et $u' \sim_f \mathcal{R} \sim_f v'$. Nous en concluons que $\mathcal{R}$ est une relation de f-performance «up to» $\sim_f$ et par conséquent, que $\mathcal{R}$ est une équivalence de f-performance.

⌟

Comme $\sim_f$ est la plus grande des équivalences de f-performance, nous avons :

Conséquence 10.44 $\approx \subset \sim_f$.

Lemme 10.45 *$\sim_f \cap(\mathcal{X} \times \mathcal{X})$ est une base de bisimulation.*

Démonstration. Nous supposons que $X \sim_f Y$.

Si $X \xrightarrow{a}_\Delta u$, alors $X \xrightarrow{a,0} f(a) \triangleright u$. Or $X \sim_f Y$, donc il existe v tel que $Y \xrightarrow{a,0} f(a) \triangleright v$ et $f(a) \triangleright u \sim_f f(a) \triangleright v$. D'après le lemme 10.27, $u \sim_f v$. Donc il existe $Y \xrightarrow{a}_\Delta v$ telle que $u\mathcal{R}v$ (le théorème 10.34 entraîne que $\sim_f$ est bien l'extension de $\sim_f \cap(\mathcal{X} \times \mathcal{X})$).

Comme X et Y ont des rôles équivalents, si $Y \xrightarrow{a}_\Delta v$, il existe $X \xrightarrow{a}_\Delta u$ telle que $u\mathcal{R}v$.

Donc $\sim_f \cap(\mathcal{X} \times \mathcal{X})$ est une base de bisimulation. ⌟

Conséquence 10.46 *$\sim_f \subset \approx$.*

Démonstration. Comme $\approx$ est la plus grande des bases de bisimulation, $\sim_f \cap(\mathcal{X} \times \mathcal{X}) \subset \approx$. Il s'ensuit que $\sim_f \subset \approx$. ⌟

10.4.4. Indépendance de l'équivalence de performance

Comme $\approx$ ne dépend pas de f, une conséquence de la proposition 10.42 est :

Théorème 10.47 *L'équivalence $\sim_f$ ne dépend pas de f.*

Corollaire 10.48 *$u \sim_{\mathrm{i}} v$ si et seulement s'il existe une fonction de durée f telle que $u \sim_f v$ si et seulement si $u \sim_{\mathbf{1}} v$.*

Remarque 10.49 Le corollaire 10.48 n'est pas un paradoxe et il peut être comparé avec la proposition 13 de [AM96]. Encore une fois, nous voyons qu'il n'est pas facile de prouver l'indépendance de l'équivalence de performance par rapport à f sans faire l'analyse nécessaire au théorème 10.34.

Remarque 10.50 Le corollaire 10.48 n'est plus correct si nous associons plusieurs durées à une même action. Par exemple, si nous considérons la déclaration TBPP suivante :

$$\Delta = \{X \xrightarrow{a} X, X \xrightarrow{a} 2 \triangleright X, \quad Y \xrightarrow{a} X, Y \xrightarrow{a} 1 \triangleright X, Y \xrightarrow{a} 2 \triangleright X\}$$

nous avons $X \not\sim_{\mathrm{i}} Y$, or $X \sim_f Y$ si $f(a) = \{1, 2\}$.

Ces résultats ont pour conséquence que nous pouvons écrire indifféremment $\sim$ pour tout $\sim_f$ et pour $\sim_{\mathrm{i}}$. C'est ce que nous faisons dans la suite de ce chapitre où nous supposons que f est la fonction de durée constante $\mathbf{1}$.

10.5. Décidabilité de l'équivalence de performance

Avec les résultats de la section 10.3, décider l'équivalence de performance est simple puisque cela revient à calculer la plus grande base de bisimulation.

Proposition 10.51 *Calculer $\sim \cap(\mathcal{X} \times \mathcal{X})$ peut être fait en temps polynomial en $|\Delta|$ (où $|\Delta|$ est le nombre de règles de Δ plus la somme des tailles des membres gauches des règles).*

Démonstration.

Un algorithme pour calculer $\sim \cap(\mathcal{X} \times \mathcal{X})$

Nous utilisons une méthode de partition comme cela a déjà été fait dans dans [KS90] pour montrer que la bisimulation est décidable en temps polynômial pour les systèmes d'états finis.

Étant donné Δ, nous partitionnons l'ensemble $\mathcal{X}$ des constantes en classes d'équivalence. Pour commencer, nous considérons $\mathcal{R}_0 = \mathcal{X} \times \mathcal{X}$, c'est-à-dire nous considérons qu'il n'y a qu'une seule classe d'équivalence $\mathcal{X}$ tout entier et nous raffinons la relation d'équivalence $\mathcal{R}_i$ en la relation d'équivalence $\mathcal{R}_{i+1}$ jusqu'à stabilisation.

Une étape de raffinement consiste à retirer toutes les paires (X, Y) et (Y, X) de $\mathcal{R}_i$ telles qu'il existe une règle $X \overset{a}{\rightarrow} u$ dans Δ telle qu'il n'existe aucune règle $Y \overset{a}{\rightarrow} v$ avec $u\mathcal{R}_i v$. Ceci peut être testé facilement par un algorithme de tri quand u et v sont en forme normale.

Nous obtenons alors une relation $\mathcal{R}_{i+1}$ qui est réflexive et symétrique par construction. Il nous reste à voir qu'elle est transitive.

Nous supposons que $X\mathcal{R}_{i+1}Y$ et $Y\mathcal{R}_{i+1}Z$.

- S'il existe une règle $X \overset{a}{\rightarrow} u$ dans Δ, alors comme $X\mathcal{R}_{i+1}Y$, il existe une règle $Y \overset{a}{\rightarrow} v$ avec $u\mathcal{R}_i v$. Comme $Y\mathcal{R}_{i+1}Z$, il existe une règle $Z \overset{a}{\rightarrow} w$ avec $v\mathcal{R}_i w$. Comme $\mathcal{R}_i$ est une relation d'équivalence, $u\mathcal{R}_i w$. Il existe donc une règle $Z \overset{a}{\rightarrow} w$ avec $u\mathcal{R}_i w$.
- S'il existe une règle $Z \overset{a}{\rightarrow} w$ dans Δ, nous montrons de façon analogue qu'il existe une règle $X \overset{a}{\rightarrow} u$ dans Δ avec $w\mathcal{R}_i u$.

Par conséquent, $\mathcal{R}_{i+1}$ est transitive, c'est donc une relation d'équivalence.

La stabilisation s'obtient après au plus $|\mathcal{X}|$ étapes de raffinement car, à chaque étape, le nombre de classes d'équivalence augmente et il y en a au plus $|\mathcal{X}|$.

La correction de l'algorithme

Si k est la dernière étape de l'algorithme, nous avons besoin de montrer que $\mathcal{R}_k$ est égale à $\approx$. En effet, la propriété 10.42 nous permet de conclure que $\mathcal{R}$ est alors égale à $\sim \cap(\mathcal{X} \times \mathcal{X})$.

Par construction, $\mathcal{R}_k$ est une relation d'équivalence. De plus, comme elle est égale à $\mathcal{R}_{k-1}$, il n'existe aucune paire (X, Y) ou (Y, X) appartenant à $\mathcal{R}_k$ et telle qu'il existe une règle $X \xrightarrow{a} u$ dans Δ, mais aucune règle $Y \xrightarrow{a} v$ avec $u\mathcal{R}_k v$. Par conséquent, R_k est une base de bisimulation. Nous en déduisons que $R_k \subset\approx$.

Nous allons maintenant montrer par récurrence sur i que, pour tout i, $\approx\subset R_i$.

- $\approx\subset R_0$ car $_0$ contient toutes les paires de variables.
- Supposons que $\approx\subset R_i$. S'il existe une règle $X \xrightarrow{a} u$ dans Δ telle qu'il n'existe aucune règle $Y \xrightarrow{a} v$ avec $u\mathcal{R}_i v$, alors il n'existe aucune règle $Y \xrightarrow{a} v$ avec $u \approx v$. Donc $X \not\approx Y$. Par conséquent, $\approx\subset R_{i+1}$.

Nous en déduisons que $\approx\subset R_k$. Il s'ensuit que $\approx= R_k$. ⌟

Puisque décider si $u \sim v$ peut être fait en temps polynomial en $|u| + |v| + |\Delta|$. Nous avons :

Théorème 10.52 *Le problème de la décision de l'équivalence de performance sur TBPP est P-complet.*

Démonstration. Quand aucune composition parallèle n'est autorisée, les termes TBPP se comportent comme des processus d'états finis où l'unique horloge locale permet simplement de mesurer la longueur de l'exécution. C'est pourquoi l'équivalence de performance sur ces termes séquentiels se réduit à la bisimulation forte non temporisée des arbres dépliés sous-jacents, qui n'est autre que la bisimulation forte des processus d'états finis non temporisés, ce qui entraîne la P-dureté [BGS92]. ⌟

10.6. Application

Dans [CD01], F. Corradini et D. Di Cola considèrent, entre autres, une algèbre qui a une syntaxe proche de celle de TBPP et une sémantique légèrement différente obtenue en supprimant l'hypothèse d'actions urgentes. Nous appelons TBPP$^+$ cette nouvelle algèbre. Ils montrent que pour cette algèbre deux équivalences : la congruence de performance et l'équivalence de performance sans urgence coïncident avec l'équivalence de performance. Ils en déduisent que ces deux équivalences sont décidables en temps polynomial. Dans cette section, nous définissons TBPP$^+$, la congruence de performance et l'équivalence de performance sans urgence, puis nous énonçons les résultats de F. Corradini et D. Di Cola.

L'ensemble $\mathcal{T}^+$ des termes TBPP$^+$ coïncide avec $\mathcal{T}$, une déclaration TBPP$^+$ vérifie les mêmes conditions qu'une déclaration TBPP et la définition d'une fonction de durée reste la même.

Par contre, la sémantique de TBPP$^+$ est différente. Une paire (Δ, f) où Δ est une déclaration TBPP$^+$ et f est une fonction de durée définit une relation de transitions étiquetée $\rightarrow_f \subseteq \mathcal{T}^+ \times (Act \times \mathbb{N} \times \mathbb{N}) \times \mathcal{T}^+$, donnée inductivement par les règles de SOS (sémantique opérationnelle strucurelle) suivantes :

$$\text{Ra'} \quad \frac{}{X \xrightarrow{a,r,r}_f r + f(a) \triangleright t} \; (X \xrightarrow{a}_\Delta t) \in \Delta, r \geq 0$$

$$\text{Rb'} \quad \frac{t \xrightarrow{a,n,r}_f t'}{1 \triangleright t \xrightarrow{a,n+1,r}_f 1 \triangleright t'}$$

$$\text{Rc'} \quad \frac{t \xrightarrow{a,n,r}_f t'}{t \parallel u \xrightarrow{a,n,r}_f t' \parallel u}$$

$$\text{Rd'} \quad \frac{t \xrightarrow{a,n,r}_f t'}{u \parallel t \xrightarrow{a,n,r}_f u \parallel t'}$$

Dans cette algèbre, un processus peut attendre avant de faire une action. La nouvelle étiquette r, appelée le **délai d'exécution**, visualise cette attente. En effet, elle indique que l'action a a commencé r unités de temps après la fin de la précédente action faite par la composante parallèle qui exécute a.

Nous retrouvons l'algèbre BPP si nous imposons que tous les délais d'exécution soient égaux à 0, c'est à dire en faisant l'intersection entre $\rightarrow_f$ et $\mathcal{T}^+ \times (Act \times \mathbb{N} \times \{0\}) \times \mathcal{T}^+$.

La définition de l'équivalence de performance adaptée à cette sémantique est :

Définition 10.53 *Une relation $\mathcal{R} \subseteq \mathcal{T}^+ \times \mathcal{T}^+$ est appelée une* ***relation de* f*-performance*** *si $u \mathcal{R} v$ entraîne que*

1. *Pour tout $u \xrightarrow{a,n,0}_f u'$, il existe $v \xrightarrow{a,n,0}_f v'$ tel que $u' \mathcal{R} v'$;*
2. *Et vice versa : pour tout $v \xrightarrow{a,n,0}_f v'$, il existe $u \xrightarrow{a,n,0}_f u'$ avec $u' \mathcal{R} v'$.*

Définition 10.54 *Deux termes TBPP$^+$ u et v sont* **f*-p équivalents*** *(notés $u \sim_p v$) s'il existe une relation de f-performance $\mathcal{R}$ telle que $u \mathcal{R} v$.*

La congruence de performance et l'équivalence de performance sans urgence introduites dans [Cor98] sont définies comme suit :

Définition 10.55 *Une relation $\mathcal{R} \subseteq \mathcal{T}^+ \times \mathcal{T}^+$ est appelée une* ***relation de* f*-performance sans urgence*** *si $u \mathcal{R} v$ entraîne que*

1. *Pour tout $u \xrightarrow{a,n,r}_f u'$, il existe $v \xrightarrow{a,n,r'}_f v'$ tel que $u' \mathcal{R} v'$;*
2. *Et vice versa : pour tout $v \xrightarrow{a,n,r}_f v'$, il existe $u \xrightarrow{a,n,r'}_f u'$ avec $u' \mathcal{R} v'$.*

Définition 10.56 *Deux termes TBPP$^+$ u et v sont* **f-*s* *équivalents*** *(notés $u \sim_s v$) s'il existe une relation de f-performance sans urgence $\mathcal{R}$ telle que $u\,\mathcal{R}\,v$.*

Définition 10.57 *Une relation $\mathcal{R} \subseteq \mathcal{T}^+ \times \mathcal{T}^+$ est appelée une* ***relation de* f*-congruence*** *si $u\,\mathcal{R}\,v$ entraîne que*

1. *Pour tout $u \xrightarrow{a,n,r}_f u'$, il existe $v \xrightarrow{a,n,r'}_f v'$ tel que $u'\,\mathcal{R}\,v'$ et $r' = 0$ si $r = 0$;*
2. *Et vice versa : pour tout $v \xrightarrow{a,n,r}_f v'$, il existe $u \xrightarrow{a,n,r'}_f u'$ avec $u'\,\mathcal{R}\,v'$ et $r' = 0$ si $r = 0$.*

Définition 10.58 *Deux termes TBPP$^+$ u et v sont* **f-*c* *équivalents*** *(notés $u \sim_c v$) s'il existe une relation de f-congruence $\mathcal{R}$ telle que $u\,\mathcal{R}\,v$.*

Le théorème 3.2 de [CD01] s'énonce alors :

Théorème 10.59 *L'équivalence de performance, la congruence de performance et l'équivalence de performance sans urgence coïncident pour TBPP$^+$.*

La conséquence de notre théorème 10.52 est donc :

Théorème 10.60 *L'équivalence de performance sans urgence et la congruence de performance sont décidables en temps polynomial pour TBPP$^+$.*

11. Conclusion

Nous nous sommes intéressés dans cette partie à l'algèbre de processus TBPP qui est une extension temporisée de BPP. Cette algèbre est à peu de choses près, l'algèbre de F. Corradini, R. Gorrieri et M. Roccetti [CGR97], elle-même obtenue en interdisant la synchronisation dans une algèbre de processus plus large, avec des actions urgentes et munies de durée [GRS95].

Dans ce cadre, F. Corradini, R. Gorrieri et M. Roccetti [CGR97] étudie l'équivalence de performance qui est un moyen de regrouper les processus qui ont le même comportement et la même efficacité.

Notre résultat principal, présenté dans [BLS00], est une méthode en temps polynomial pour décider l'équivalence de performance sur une classe de systèmes qui peuvent avoir un nombre infini d'états distincts (même sans tenir compte du temps). Il semble alors que BPP temporisé soit plus simple que BPP (la bisimulation pour BPP est co-NP dur [May00]), ce qui est un résultat surprenant. Ceci amène à penser que les extensions temporisées d'algèbre de processus ayant un nombre infini d'états mais proches de BPP peuvent être étudiées et pourraient se révéler plus simples que leurs parties non temporisées qui sont bien-connues. Nous voyons plusieurs directions qu'il nous semble judicieux d'étudier :

1. La bisimulation pour les processus PA normés est décidable [HJ99] mais paraît assez complexe. Qu'en est-il de l'équivalence de performance pour PA temporisé ?
2. La décidabilité de l'équivalence observationnelle (appelée aussi τ-bisimulation) entre les processus BPP est un problème ouvert important [Esp97b]. Qu'en est-il de l'équivalence observationnelle de performance ? (ajouter des actions τ à l'algèbre TBPP peut se faire de différentes façons : par exemple, elles peuvent modéliser les actions internes qui ont une durée nulle [CD01]).
3. La plupart des équivalences comportementales (autre que la bisimulation) sur les processus BPP sont indécidables [Hüt94]. En est-il de même avec leur versions temporisées ?

Bibliographie

[ACD93] R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1) :2–34, 1993.

[AČJY96] P. A. Abdulla, K. Čerāns, B. Jonsson, and T. Yih-Kuen. General decidability theorems for infinite-state systems. In *Proc. 11th IEEE Symp. Logic in Computer Science (LICS'96), New Brunswick, NJ, USA, July 1996*, pages 313–321. IEEE Comp. Soc. Press, 1996.

[AD94] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2) :183–235, 1994.

[ADS86] B. Alpern, A. J. Demers, and F. B. Schneider. Safety without stuttering. *Information Processing Letters*, 23(4) :177–180, 1986.

[AH92] S. Arun-Kumar and M. Hennessy. An efficiency preorder for processes. *Acta Informatica*, 29(8) :737–760, 1992.

[AJ96a] P. A. Abdulla and B. Jonsson. Undecidable verification problems for programs with unreliable channels. *Information and Computation*, 130(1) :71–90, 1996.

[AJ96b] P. A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2) :91–101, 1996.

[AJ98] P. A. Abdulla and B. Jonsson. Verifying networks of timed processes. In *Proc. 4th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98), Lisbon, Portugal, March 1998*, volume 1384 of *Lecture Notes in Computer Science*, pages 298–312. Springer, 1998.

[AM96] L. Aceto and D. Murphy. timing and causality in process algebra. *Acta Informatica*, 33(4) :317–350, 1996.

[AS85] B. Alpern and F. B. Schneider. Defining liveness. *Information Processing Letters*, 21(4) :181–185, 1985.

[AS87] B. Alpern and F. B. Schneider. Recognizing safety and liveness. *Distributed Computing*, 2 :117–126, 1987.

[BB90] J. C. M. Baeten and J. A. Bergstra. Real time process algebra. Research Report CS-R9053, CWI, October 1990.

[BB91] J. C. M. Baeten and J. A. Bergstra. Real time process algebra. *Formal Aspects of Computing*, 3(2) :142–188, 1991.

[BB96] J. C. M. Baeten and J. A. Bergstra. Discrete time process algebra. *Formal Aspects of Computing*, 8(2) :188–208, 1996.

[BBF$^+$01] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and Ph. Schnoebelen. *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer, 2001.

[BCMS01] O. Bukart, D. Caucal, F. Moller, and B. Steffen. Verification on infinite structures. In J. A. Bergstra, A. Ponse, and S. A. Smolka, editors, *Handbook of Process Algebra*, chapter 9, pages 545–623. Elsevier Science, 2001.

[BCS97a] O. Bukart, D. Caucal, and B. Steffen. Bisimulation collapse and process taxonomy. In *Proc. 7th Int. Conf. Concurrency Theory (CONCUR'96), Pisa, Italy, Aug. 1996*, volume 1119 of *Lecture Notes in Computer Science*, pages 247–262, 1997.

[BCS97b] O. Bukart, D. Caucal, and B. Steffen. An elementary bisimulation decision procedure for arbitrary context-free processes. In *Proc. 20th Int. Symp. Math. Found. Comp. Sci. (MFCS'95), Prague, Czech Republic, Aug.-Sep. 1995*, volume 969 of *Lecture Notes in Computer Science*, pages 423–433, 1997.

[BDFP00] P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Are timed automata updatable ? In *Proc. 12th Int. Conf. Computer Aided Verification (CAV'2000), Chicago, IL, USA, July 2000*, volume 1855 of *Lecture Notes in Computer Science*, pages 464–479. Springer, 2000.

[BEM97] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata : application to model-checking. In *Proc. 8th Int. Conf. Concurrency Theory (CONCUR'97), Warsaw, Poland, Jul. 1997*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 1997.

[BGS92] J. Balcázar, J. Gabarró, and M. Sántha. Deciding bisimilarity is P-complete. *Formal Aspects of Computing*, 4(6A) :638–648, 1992.

[BH96] A. Bouajjani and P. Habermehl. Constrained properties, semilinear systems, and petri nets. In *Proc. 7th Int. Conf. Concurrency Theory (CONCUR'96), Pisa, Italy, Aug. 1996*, volume 1119 of *Lecture Notes in Computer Science*. Springer, 1996.

[BHR84] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31(3) :560–599, 1984.

[BJNT00] A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In *Proc. 12th Int. Conf. Computer Aided Verification*

(CAV'2000), Chicago, IL, USA, July 2000, volume 1855 of *Lecture Notes in Computer Science*, pages 403–418. Springer, 2000.

[BK85] J. A. Bergstra and J. W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37(1) :77–121, 1985.

[BLS00] B. Bérard, A. Labroue, and Ph. Schnoebelen. Verifying performance equivalence for Timed Basic Parallel Processes. In *Proc. 3rd Int. Conf. Foundations of Software Science and Computation Structures (FOSSACS'2000), Berlin, Germany, Mar.-Apr. 2000*, volume 1784 of *Lecture Notes in Computer Science*, pages 35–47. Springer, 2000.

[BM01] J. C. M. Baeten and C. A. Middelburg. Process algebra with timing : real time and discrete time. In J. A. Bergstra, A. Ponse, and S. A. Smolka, editors, *Handbook of Process Algebra*, chapter 10, pages 627–684. Elsevier Science, 2001.

[Boc78] G. von Bochmann. Finite state description of communication protocols. *Computer Networks and ISDN Systems*, 2 :361–372, 1978.

[BS90] J. Bradfield and C. Stirling. Verifying temporal properties of processes. In *Proc. Theories of Concurrency (CONCUR'90), Amsterdam, NL, Aug. 1990*, volume 458 of *Lecture Notes in Computer Science*, pages 115–125. Springer, 1990.

[BSW69] K. Bartlett, R. Scantlebury, and P. Wilkinson. A note on reliable full-duplex transmission over half-duplex links. *Communications of the ACM*, 12(5) :260–261, 1969.

[BV01] J. C. M. Baeten and C. Verhoef. Concrete process algebra. In S. Abramsky, Dov M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 4, chapter 2, pages 149–268. Clarendon Press Oxford, 2001.

[BW90] J. C. M. Baeten and W. P. Weijland. *Process algebra*, volume 18 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge Univ. Press, 1990.

[Cau92] D. Caucal. On the regular structure of prefix rewriting. *Theoretical Computer Science*, 106 :61–86, 1992.

[Cau00] D. Caucal. On word rewriting systems having a rational derivation. In *Proc. 3rd Int. Conf. Foundations of Software Science and Computation Structures (FOSSACS'2000), Berlin, Germany, Mar.-Apr. 2000*, volume 1784 of *Lecture Notes in Computer Science*, pages 48–62, 2000.

[CC97] Xiao Jun Chen and F. Corradini. On the specification and verification of performance properties for a timed process algebra. In *Proc. 6th Int. Conf. Algebraic Methodology and Software Technology (AMAST'97), Sydney, Australia, Dec. 1997*, volume 1349 of *Lecture Notes in Computer Science*, pages 123–137, 1997.

[CD01] F. Corradini and D. Di Cola. The expressive power of urgent, lazy and busy-waiting actions in timed processes. *Math. Struct. in Comp. Science*, 2001. To appear.

[CDG+99] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree Automata Techniques and Applications, 1997–99.

[CE81] E. M. Clarke and E. A. Emerson. Design and synthesis of sunchronization skeletons using branching time temporal logic. In *Proc. Logics of Programs Workshop, Yorktown Heights, New York, May 1981*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981.

[CFP95] G. Cécé, A. Finkel, and S. Purushothaman Iyer. Unreliable channels are easier to verify than perfect channels. *Information and Computation*, 124(1) :20–31, 1995.

[CGL94] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5) :1512–1542, 1994.

[CGR97] F. Corradini, R. Gorrieri, and M. Roccetti. Performance preorder and competitive equivalence. *Acta Informatica*, 34(11) :805–835, 1997.

[CHM93] S. Christensen, Y. Hirshfeld, and F. Moller. Bisimulation equivalence is decidable for basic parallel processes. In *Proc. 4th Int. Conf. Concurrency Theory (CONCUR'93), Hildesheim, Germany, Aug. 1993*, volume 715 of *Lecture Notes in Computer Science*, pages 143–157. Springer, 1993.

[Chr93] S. Christensen. Decidability and decomposition in process algebras. PhD thesis CST-105-93, Dept. of Computer Science, University of Edinburgh, UK, 1993.

[CHS95] S. Christensen, H. Hüttel, and C. Stirling. Bisimulation equivalence is decidable for all context-free processes. *Information and Computation*, 121(2) :143–148, 1995.

[CJ99] H. Comon and Y. Jurski. Timed automata and the theory of real numbers. In *Proc. 10th Int. Conf. Concurrency Theory (CONCUR'99), Eindhoven, The Netherlands, Aug. 1999*, volume 1664 of *Lecture Notes in Computer Science*, pages 242–257. Springer, 1999.

[CKS81] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1) :114–133, 1981.

[Cor98] F. Corradini. On performance congruences for process algebras. *Information and Computation*, 145(2) :191–230, 1998.

[CP96] F. Corradini and M. Pistore. Specification and verification of timed lazy systems. In *Proc. 21st Int. Symp. Math. Found. Comp. Sci. (MFCS'96), Cracow, Poland, Sep. 1996*, volume 1113 of *Lecture Notes in Computer Science*, pages 279–290, 1996.

[CP01] F. Corradini and M. Pistore. 'Closed interval process algebra' versus 'interval process algebra'. *Acta Informatica*, 37(7) :467–509, 2001.

[CW96] E. M. Clarke and J. M. Wing. Formal methods : state of the art and future directions. *ACM Computing Surveys*, 28(4) :626–643, 1996.

[De 85] R. De Simone. Higher-level synchronising devices in MEIJE-SCCS. *Theoretical Computer Science*, 37 :245–267, 1985.

[EH86] E. A. Emerson and J. Y. Halpern. "Sometimes" and "Not Never" revisited : On branching versus linear time temporal logic. *Journal of the ACM*, 33(1) :151–178, 1986.

[EJM99] J. Esparza, P. Jančar, and F. Moller. Petri nets and regular behaviours. *Journal of Computer and System Sciences*, 59(3) :476–503, 1999.

[EK95] J. Esparza and A. Kiehn. On the model checking problem for branching time logics and basic parallel processes. In *Proc. 7th Int. Conf. Computer Aided Verification (CAV'95), Liège, Belgium, July 1995*, volume 939 of *Lecture Notes in Computer Science*, pages 353–366. Springer, 1995.

[EK99] J. Esparza and J. Knoop. An automata-theoretic approach to interprocedural data-flow analysis. In *Proc. 2nd Int. Conf. Foundations of Software Science and Computation Structures (FOSSACS'99), Amsterdam, The Netherlands, Mar. 1999*, volume 1578 of *Lecture Notes in Computer Science*, pages 14–30. Springer, 1999.

[EN98] E. A. Emerson and K. S. Namjoshi. On model checking for non-deterministic infinite-state systems. In *Proc. 13th IEEE Symp. Logic in Computer Science (LICS'98), Indianapolis, IN, USA, June 1998*, pages 70–80. IEEE Comp. Soc. Press, 1998.

[EP00] J. Esparza and A. Podelski. Efficient algorithms for $pre*$ and $post*$ on interprocedural parallel flow graphs. In *Proc. 27th ACM Symp. Principles of Programming Languages (POPL'2000), Boston, MA, USA, Jan. 2000*, pages 1–11, 2000.

[Esp94] J. Esparza. On the decidability of model checking for several μ-calculi and Petri nets. In *Proc. 19th Int. Coll. on Trees in Algebra and Programming (CAAP'94), Edinburgh, UK, Apr. 1994*, volume 787 of *Lecture Notes in Computer Science*, pages 115–129. Springer, 1994.

[Esp97a] J. Esparza. Decidability of model checking for infinite-sate concurrent systems. *Acta Informatica*, 34(2) :85–107, 1997.

[Esp97b] J. Esparza. Petri nets, commutative context-free grammars, and basic parallel processes. *Fundamenta Informaticae*, 31(1) :13–25, 1997.

[Fin87] A. Finkel. Well structured transition systems. Research Report 365, Lab. de Recherche en Informatique (LRI), Univ. Paris-Sud, Orsay, August 1987.

[FM95] G. Ferrari and U. Montanari. Dynamic matrices and the cost analysis of concurrent programs. In *Proc. 4th Int. Conf. Algebraic Methodology and Software Technology (AMAST'95), Montreal, Canada, July 1995*, volume 936 of *Lecture Notes in Computer Science*, pages 307–321, 1995.

[FS01] A. Finkel and Ph. Schnoebelen. Well structured transition systems everywhere! *Theoretical Computer Science*, 256(1–2) :63–92, 2001.

[GH94] J. F. Groote and H. Hüttel. Undecidable equivalences for Basic Process Algebra. *Information and Computation*, 115(2) :354–371, 1994.

[Gla90] R. J. van Glabbeek. The linear time – branching time spectrum. In *Proc. Theories of Concurrency (CONCUR'90), Amsterdam, NL, Aug. 1990*, volume 458 of *Lecture Notes in Computer Science*, pages 278–297. Springer, 1990.

[GRS95] R. Gorrieri, M. Roccetti, and E. Stancampiano. A theory of processes with durational actions. *Theoretical Computer Science*, 140(1) :73–94, 1995.

[GV92] J. F. Groote and F. W. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100(2) :202–260, 1992.

[Hen88] M. Hennessy. Axiomatising finite concurrent processes. *SIAM J. Comput.*, 17(5) :997–1017, 1988.

[HHW97] T. A. Henzinger, Pei-Hsin Ho, and H. Wong-Toi. HyTech : A model-checker for hybrid systems. In *Proc. 9th Int. Conf. Computer Aided Verification (CAV'97), Haifa, Israel, June 1997*, volume 1254 of *Lecture Notes in Computer Science*, pages 460–463. Springer, 1997.

[HJ99] Y. Hirshfeld and M. Jerrum. Bisimulation equivalence is decidable for normed process algebra. In *Proc. 26th Int. Coll. Automata, Languages, and Programming (ICALP'99), Prague, Czech Republic, July 1999*, volume 1644 of *Lecture Notes in Computer Science*, pages 412–421. Springer, 1999.

[HJM94] Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial algorithm for deciding bisimilarity of normed context-free processes. Research Report ECS-LFCS-94-286, Lab. for Foundations of Computer Science, Edinburgh, 1994.

[HJM96] Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial-time algorithm for deciding bisimulation equivalence of normed Basic Parallel Processes. *Math. Struct. in Comp. Science*, 6(3) :251–259, 1996.

[HKPV95] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata ? In *Proc. 27th ACM Symp. Theory of Computing (STOC'95), Las Vegas, NV, May-June 1995*, pages 373–382, 1995.

[HM00] T. A. Henzinger and R. Majumdar. A classification of symbolic transition systems. In *Proc. 17th Ann. Symp. Theoretical Aspects of Computer Science (STACS'2000), Lille, France, Feb. 2000*, volume 1770 of *Lecture Notes in Computer Science*, pages 13–34. Springer, 2000.

[HT95] D. T. Huynh and Lu Tian. On deciding readiness and failure equivalences for processes. *Information and Computation*, 117(2) :193–205, 1995.

[Hüt94] H. Hüttel. Undecidable equivalences for Basic Parallel Processes. In *Proc. 2nd Int. Symp. Theoretical Aspects of Computer Software (TACS'94), Sendai, Japan, Apr. 1994*, volume 789 of *Lecture Notes in Computer Science*, pages 454–464. Springer, 1994.

[I. 96] I. Walukiewicz. Pushdown processes : Games and model checking. In Rajeev Alur and Thomas A. Henzinger, editors, *Proc. 8th Int. Conf. Computer Aided Verification (CAV'96), New Brunswick, NJ, USA, July-Aug. 1996*, volume 1102 of *Lecture Notes in Computer Science*, pages 62–74. Springer, 1996.

[Jan95] P. Jančar. Undecidability of bisimilarity for Petri nets and some related problems. *Theoretical Computer Science*, 148(2) :281–301, 1995.

[Jan97] P. Jančar. Bisimulation equivalence is decidable for one-counter process. In *Proc. 24th Int. Coll. Automata, Languages, and Programming (ICALP'97), Bologna, Italy, July 1997*, volume 1256 of *Lecture Notes in Computer Science*, pages 549–559, 1997.

[JKM00] P. Jančar, A. Kučera, and R. Mayr. Deciding bisimulation-like equivalences with finite-state processes. *Theoretical Computer Science*, 258(1–2) :409–433, 2000.

[Kin94] E. Kindler. Safety and liveness properties : A survey. *EATCS Bull.*, 53 :268–272, 1994.

[KM99] A. Kučera and R. Mayr. Weak bisimilarity with finite state systems can be decided in polynomial time. In *Proc. 10th Int. Conf. Concurrency Theory (CONCUR'99), Eindhoven, The Netherlands, Aug. 1999*, volume 1664 of *Lecture Notes in Computer Science*, pages 368–374. Springer, 1999.

[Kou97] O. Kouchnarenko. *Sémantique des programmes récursifs-parallèles et méthodes pour leur analyse*. Thèse de Doctorat, Univ. Joseph Fourier-Grenoble I, France, February 1997.

[KS90] P. C. Kanellakis and S. A. Smolka. CCS expressions, finite state processes and three problems of equivalence. *Information and Computation*, 86(1) :43–68, 1990.

[KS97a] O. Kouchnarenko and Ph. Schnoebelen. A formal framework for the analysis of recursive-parallel programs. In *Proc. 4th Int. Conf. Parallel Computing Technologies (PaCT'97), Yaroslavl, Russia, Sep. 1997*, volume 1277 of *Lecture Notes in Computer Science*, pages 45–59. Springer, 1997.

[KS97b] O. Kouchnarenko and Ph. Schnoebelen. A model for recursive-parallel programs. In *Proc. 1st Int. Workshop on Verification of Infinite State Systems (INFINITY'96), Pisa, Italy, Aug. 1996*, volume 5 of *Electronic Notes in Theor. Comp. Sci.* Elsevier Science, 1997.

[Lam73] L. Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, 3(2) :125–143, 1973.

[LPY97] K. G. Larsen, P. Pettersson, and Wang Yi. UPPAAL in a nutshell. *Journal of Software Tools for Technology Transfer*, 1(1–2) :134–152, 1997.

[LS85] L. Lamport and F. B. Schneider. Formal foundation for specification and verification. In *Distributed systems : methods and tools for specification : an advanced course*, volume 190 of *Lecture Notes in Computer Science*, pages 203–285. Springer, 1985.

[LS00] D. Lugiez and Ph. Schnoebelen. Decidable first-order transition logics for PA-processes. In *Proc. 27th Int. Coll. Automata, Languages, and Programming (ICALP'2000), Geneva, Switzerland, July 2000*, volume 1853 of *Lecture Notes in Computer Science*, pages 342–353. Springer, 2000.

[LS01] D. Lugiez and Ph. Schnoebelen. The regular viewpoint on PA-processes. *Theoretical Computer Science*, 2001. To appear.

[Mai01] M. Maidl. A unifying model checking approach for safety properties of parameterized systems. In *Proc. 13th Int. Conf. Computer Aided Verification (CAV'2001), Paris, France, July 2001*, volume 2102 of *Lecture Notes in Computer Science*, pages 311–323. Springer, 2001.

[May84] E. W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM J. Comput.*, 13(3) :441–460, 1984.

[May97] R. Mayr. Model checking PA-processes. In *Proc. 8th Int. Conf. Concurrency Theory (CONCUR'97), Warsaw, Poland, Jul. 1997*, volume 1243 of *Lecture Notes in Computer Science*, pages 332–346. Springer, 1997.

[May98] R. Mayr. Decidability of complexity of model checking problems for infinite-state systems. PhD thesis, TU-Müchen, 1998.

[May00] R. Mayr. On the complexity of bisimulation problems for basic parallel processes. In *Proc. 27th Int. Coll. Automata, Languages, and Programming (ICALP'2000), Geneva, Switzerland, July 2000*, volume 1853 of *Lecture Notes in Computer Science*, pages 329–341. Springer, 2000.

[May01] R. Mayr. On the complexity of bisimulation problems for pushdown automata. 2001. submitted for publication.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice Hall Int., 1989.

[Min67] M. Minski. *Computation : Finite and Infinite Machines.* Prentice Hall Int., 1967.

[MM93] R. Milner and F. Moller. Unique decomposition of processes. *Theoretical Computer Science*, 107(2) :357–363, 1993.

[Mol96] F. Moller. Infinite results. In *Proc. 7th Int. Conf. Concurrency Theory (CONCUR'96), Pisa, Italy, Aug. 1996*, volume 1119 of *Lecture Notes in Computer Science*, pages 195–216. Springer, 1996.

[MT90] F. Moller and C. Tofts. A temporal calculus of communicating systems. In *Proc. Theories of Concurrency (CONCUR'90), Amsterdam, NL, Aug. 1990*, volume 458 of *Lecture Notes in Computer Science*, pages 401–415. Springer, 1990.

[MT91] F. Moller and C. Tofts. Relating processes with respect to speed. In *Proc. 2nd Int. Conf. Theory of Concurrency (CONCUR'91), Amsterdam, NL, Aug. 1991*, volume 527 of *Lecture Notes in Computer Science*, pages 424–438. Springer, 1991.

[Par81] D.M.R. Park. Concurrency and automata on infinite sequences. In *Proc. 5th GI Conf. on Theor. Comp. Sci., Karlsruhe, FRG, Mar. 1981*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer, 1981.

[Pet62] C. A. Petri. *Kommunikation mit Automaten.* PhD thesis, Univ. Bonn, FRG, 1962. Schriften des Instituts für Instrumentelle Mathematik.

[Plo81] G. D. Plotkin. A structural approach to operational semantics. Lect. Notes, Aarhus University, Aarhus, DK, 1981.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. 18th IEEE Symp. Foundations of Computer Science (FOCS'77), Providence, RI, USA, Oct.-Nov. 1977*, pages 46–57, 1977.

[SBB+99] Ph. Schnoebelen, B. Bérard, M. Bidoit, F. Laroussinie, and A. Petit. *Vérification de logiciels : Techniques et outils du model-checking.* Vuibert, 1999.

[SC85] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3) :733–749, 1985.

[Sch01] Ph. Schnoebelen. Bisimulation and other undecidable equivalences for lossy channel systems. In *Proc. 4th Int. Symp. Theoretical Aspects of Computer Software (TACS'2001), Sendai, Japan, Oct. 2001*, volume 2215 of *Lecture Notes in Computer Science*, pages 385–399. Springer, 2001.

[Sha87] E. Shapiro. *Concurrent Prolog : Collected Papers*. MIT Press, Cambridge, MA, 1987.

[SK95] V. Sokolov and O. Kouchnarenko. Analysis of semantic properties of a class of recursive-parallel programs. In *IV Conf. Application Logic, Irkutsk, Russia*, pages 72–73, 1995.

[Sén98] G. Sénizergues. Decidability of bisimulation equivalence for equational graphs of finite out-degree. In *Proc. 39th IEEE Symp. Foundations of Computer Science (FOCS'98), Palo Alto, 1998*, pages 120–129. IEEE Comp. Soc. Press, 1998.

[Str00] Jitka Stribrna. Hardness results for weak bisimilarity of simple process algebras. In Petr Jancar and Mojmir Kretinsky, editors, *Electronic Notes in Theoretical Computer Science*, volume 18. Elsevier Science Publishers, 2000.

[Tur93] K. J. Turner. *Using Formal Description Techniques (An Introduction to Estelle, LOTOS and SDL).* Wiley series in communication and distributed systems. John Wiley & Sons, 1993.

[TW68] J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1) :57–81, 1968.

[Var97] M. Vardi. Alternating automata : Checking truth and validity for temporal logics. In *Proc. 14th Int. Conf. Automated Deduction (CADE'97), Townsville, North Queensland, Australia, July 1997*, volume 1249 of *Lecture Notes in Computer Science*, pages 191–206. Springer, 1997.

[VEKM94] V. Vasilchikov, V. Emielyn, V. Kurchidis, and Y. Mamatov. *Recursive-parallel programming and work in RPMSHELL.* IPVT RAN, Iaroslavl, Russia, 1994.

[Yov97] S. Yovine. Kronos : A verification tool for real-time systems. *Journal of Software Tools for Technology Transfer*, 1(1–2) :123–133, 1997.