République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mentouri – Constantine
Faculté des Sciences de l'Ingénieur
Département d'Informatique

Année : 2005-2006
N ° d'ordre : 100/Mag/2006
Série : 006/Inf/2006

Mémoire de Magistère

En vue de l'obtention du diplôme de Magistère en Informatique
Option : Information & Computation

Présenté par

Adel BENAMIRA

Thème

Vérification des équivalences de comportements des systèmes concurrents

Soutenu le 12/06/2006 devant le jury composé de :

Pr Zizette BOUFAIDA	**Président**	**Prof**	**U. M. Constantine**
Dr Djamel-Eddine SAIDOUNI	**Rapporteur**	**M. C**	**U. M. Constantine**
Dr Faïza BELALA	**Examinateur**	**M. C**	**U. M. Constantine**
Dr Souham MESHOUL	**Examinateur**	**M. C**	**U. M. Constantine**

Mémoire préparé au Laboratoire LIRE, Université Mentouri, 25000 Constantine, Algérie

A mes parents...

Remerciements

Mes louanges et mes gratitudes intarissables vont en premier lieu à Dieu, le tout puissant qui ma prodigué le courage, la volonté et la patience afin d'accomplir ce présent travail.

Je tiens à remercier les membres de mon jury : **Mme Zizette Boufaïda**, qui a accepté la lourde tâche d'être président du jury ; **Mmes Faïza Belala** et **Souham Meshoul**, pour le temps qu'elles ont accordé à l'examen de ce travail.

Ma très grande reconnaissance va à mon encadreur, **Dr. Djamel-Eddine Saidouni** qui avec son noble esprit scientifique et sa modestie sans égale, m'a inculqué les vraies valeurs d'homme du savoir, qui par sa patience, sa loyauté scientifique et morale a su dirigé avec une si parfaite habileté ce présent travail.

Je remercie tous mes collègues et amis du laboratoire **LIRE** pour l'aide qu'ils m'ont apportés.

Adel Benamira

Résumé

La vérification formelle des systèmes complexes constitue aujourd'hui un enjeu majeur dans de nombreux domaines de la société humaine. En effet, l'employant des méthodes de spécification et vérification formelle, assistées par des outils informatiques performants rendre l'analyse de ces systèmes fiable et garantie, en plus, un bon compromis coût performances est assuré. La principale limitation de la vérification formelle basée sur les modèles est connue sous le nom de l'explosion combinatoire du graphe d'états. De multiples travaux sont en cours en vue de maîtriser cette explosion.

Nous allons plus particulièrement nous intéresser à l'explosion due à la représentation du parallélisme par l'entrelacement d'actions concurrentes, ce qui génère plusieurs séquences d'exécution commençant d'un même état et finissant dans un autre état, où l'ordre d'exécution est arbitraire. La technique de pas couvrants est une approche d'ordre partiel cherche à regrouper dans un seul pas, sous certaines conditions, les actions indépendantes qui font l'objet de ces séquences redondantes, nous avons proposé une méthode de réduction en combine cette approche avec le modèle des STEMs (Systèmes de Transitions Etiquetées Maximales), cela nous permet de calculer et prendre en considération toutes les relation d'indépendance entre les actions du système. Le graphe d'états obtenu est un graphe complet préservant les propriétés générales (les états de blocage et la vivacité). Nous avons aussi proposé une solution spécifique aux STEMs permet d'éliminer des redondances modulo la α-déquivalence, nous constatons que le taux de réduction obtenu est d'ordre exponentiel.

Mots-clés : Vérification formelle, Sémantique d'ordre partiel, Sémantique de maximalité, Graphe de pas couvrants.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Durant la dernière décennie, les méthodes formelles sont devenues une activité indispensable intégrée au processus de conception des systèmes complexes à caractère critique, telles que les protocoles de télécommunication, les systèmes répartis et les architectures multiprocesseurs. En effet, la complexité de ces systèmes rendant leur analyse classique (prototypage et test) extrêmement ardue, leur fiabilité ne saurait être garantie autrement qu'en employant des méthodes de spécification et vérification formelle, assistées par des outils informatiques performants. Une approche de vérification offrant un bon compromis coût performances est la vérification basée sur les modèles (model-checking), connue aussi sous le nom de la vérification énumérative.

Cette approche consiste à traduire le système, préalablement d´écrite dans un langage appropriée « Description du système», vers un modèle « sémantique », généralement un graphe d'états, sur lequel les propriétés de correction attendues « Spécification » sont vérifiées au moyen d'algorithmes spécifiques (Figure1.1). Bien que limitée aux applications ayant un nombre fini d'états, la présente approche est particulièrement utile dans les premières phases du processus de conception, permettant une détection rapide et économique des erreurs

La vérification formelle fondée précisément sur les quatre éléments suivants :

- Un langage de description formelle du système, c'est un formalisme du comportement de systèmes de haut niveau, il possède à la fois une syntaxe et une sémantique bien définie (dit formel). Parmi ces langages, nous pouvons citer les réseaux de Petri [Rei85], les algèbres de processus (ACP [BK85],CCS [Mil80][Mil83][Mil89], CSP [Hoa85],. . .) et les techniques de description formelle(ESTELLE [90788], LOTOS [BB87][ISO88], SDL [CCI88]).

- Un modèle sémantique du parallélisme (modèle de bas niveau), comme son nom l'indique, ce modèle est utilisé pour exprimer la sémantique du parallélisme des langages de description formelle. Nous distinguons deux grandes familles, les modèles d'entrelacement(Arbres de synchronisation [Mil80], STE [Arn92],. . .) et les modèles de non entrelacement (dit vrai parallélisme)(RDP [Rei85], SEP [NPW81], STA [Bed87][Shi85a]

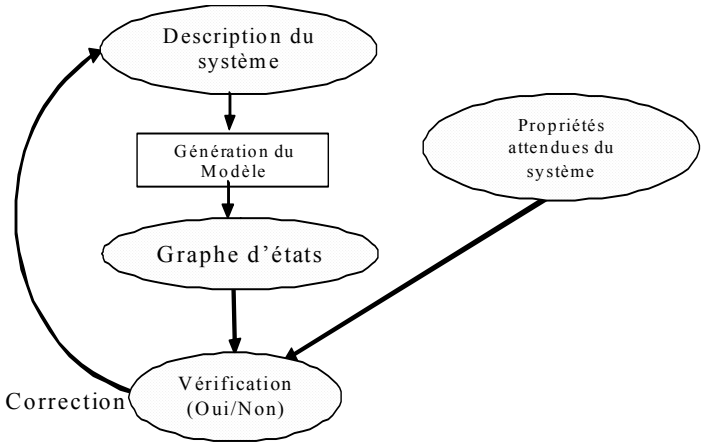


FIG. 1.1 – Vérification formelle

[Shi85b],...). Leur principale différence réside dans l'adoption ou non de l'hypothèse de l'atomicité temporelle et structurelle des actions (les actions sont de durée nulle et indivisibles). Les modèles de vrai parallélisme nous permettent d'éviter les difficultés liées aux modèles entrelacés d'une part, et d'une autre part nous permettent d'introduire une méthode de conception formelle par raffinement successif (remplacer une action par un processus).

- Un langage de spécification, c'est un formalisme dédié à la description de propriétés attendues du système. Nous trouvons dans la littérature deux types de langages de spécification : spécification logique et spécification comportementale. La spécification logique (HML [HM80], CTL [CE81] [CES83], CTL*[EH86],...) est généralement fondée sur des formules d'une logique temporelle modale qui sont interprétées sur le modèle sémantique sous-jacent. Cependant les propriétés du deuxième type sont exprimées dans le même modèle sémantique de celui de la description du système.

- La vérification, c'est une relation de satisfaction qui définit la comparaison entre la description du système et sa spécification. Selon le formalisme de la spécification employé, il existe généralement deux types de vérification ; le premier type consiste de vérifier les formules logiques sur le graphe d'états du système (dit vérification logique) et l'autre type est une comparaison entre les modèles sémantiques s'effectue au moyen d'une relation d'équivalence ou de préordre[Mil80][Par81] (dit vérification comportementale).

Problématique

Utiliser un modèle exhaustif pour exprimer le comportement du système entraîne deux conséquences majeures sur les techniques de vérification formelle, la première est considérée comme un avantage, avoir une technique de vérification complètement automatisables est devenue possible. Cependant, la deuxième conséquence est une forte limitation d'ordre opérationnel, effectivement, dans la majorité des cas le modèle est infini ou même trop gros pour être manipulé, il s'accroît très rapidement (accroissement exponentiel) avec la complexité du système. Ce phénomène est connu sous le nom de l'explosion combinatoire du graphe d'état.

De multiples travaux sont en cours en vue de maîtriser cette explosion, il existe en général deux stratégies complémentaires. La première consiste à contourner la difficulté de traiter les systèmes « gérer l'explosion », et la deuxième consiste à réduire la complexité du système « combattre l'explosion ».

La gestion de l'explosion cherche sur les effets du phénomène, les limites physiques de la mémoire sont vite atteintes. Parmi les approches introduites dans cet axe de recherche, la compression du graphe de comportement, c'est la technique la plus connue, elle est basée sur la notion de BDD(Binary Decision Diagramm), elle nous permet dans le meilleur des cas de compresser des graphes de 10^{20} états [BCM$^+$90]. Le taux de compression dépend de certains paramètres, et il peut être très faible pour certains graphes. Sur le même axe, on remarque une approche efficace dans le cas où la propriété à vérifier n'est pas satisfaite par le système, la vérification des propriétés ici se fait au cours de la construction du modèle, dite « vérification à la volée », et la non satisfaction des propriétés conduit directement à l'arrêt de la construction du modèle.

Les variables du système, les composants symétriques et la sémantique d'entrelacement sont jugés comme les racines du problème. Pour combattre ce phénomène, il faut éliminer les effets de chaque cause, toutes les techniques proposées dans cette approche sont basées sur la construction incomplète du modèle dont la partie construite doit être pertinente au vu des propriétés à vérifier.

Ce travail s'inscrit dans le cadre de la résolution du problème de l'explosion combinatoire du graphe d'états, nous allons plus particulièrement nous intéresser à l'explosion due à la représentation du parallélisme par l'entrelacement d'actions concurrentes, ce qui génère plusieurs séquences d'exécution commençant d'un même état et finissant dans un autre état, où l'ordre d'exécution est arbitraire. Les techniques d'ordre partiel cherchent à éliminer les entrelacements superflus en se basant sur les relations d'indépendance calculées directement à partir de la spécification formelle du système à analyser. On distingue en général deux stratégies : élimination de l'entrelacement et pas couvrant.

Les différentes techniques de la première stratégie tentent à obtenir un sous-graphe du graphe de comportement, contenant le moins de séquences équivalentes possibles[Val88] [Val89] [Val90]. Cette approche a été généralisée et décrite de manière plus dénotationnelle [God90] [GW91] [GW93] [WG93], ce qui a fait apparaître les notions d'ensembles persistants

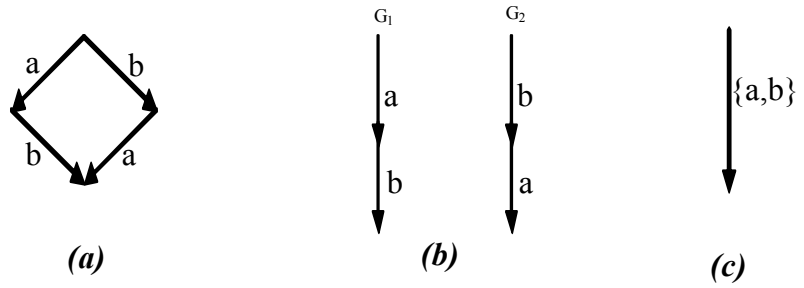


FIG. 1.2 – Le comportement de l'expression a|||b

et d'ensembles dormants, leur principal inconvénient est l'indéterminisme du résultat obtenu où on peut avoir plusieurs sous-graphes pour un même graphe de comportement[Rib05] (Figure 1.2.(b)). Une autre alternative a été proposée dans[VAM96]. Cette approche consiste à regrouper les événements indépendants dans un seul pas (Figure 1.2.(c)). Le graphe construit est le graphe de pas couvrants, qui est un graphe complet. La préservation des états de blocage et des propriétés de vivacité est assurée ; cependant, plusieurs versions différentes ont été proposées pour préserver l'équivalence observationnelle[VAM96], et la sémantique de refus[VM97].

On distingue deux principales limitations des approches d'ordre partiel. La première touche le facteur générique de l'approche dû au fait que la relation d'indépendance calculée est structurelle, ce qui nous interdit de prendre en considération quelques types de transitions indépendantes, par exemple le cas du conflit structurel (Figure 1.3). En conséquence, le parallélisme dynamique n'est pas pris en compte. Cependant, on trouve dans[RV02] une tentative connue sous le nom de « GPC vers une relation de conflit dynamique ». Or, cette dernière solution est spécifique aux réseaux de Petri.

La deuxième limitation se résume en l'impuissance d'exploiter toutes les relations d'indépendance. Par exemple, il existe des cas où il est impossible de prendre des transitions indépendantes dans le même pas (ou éliminer quelques chemins équivalents) au risque de perdre la préservation des états de blocage. Parmi ces cas de figure, on peut citer le conflit différé[Rei85] (Figure 1.4) où sa forte présence diminue le taux de réduction. En d'autres termes, les branchements ne sont plus considérés dans les réductions possibles.

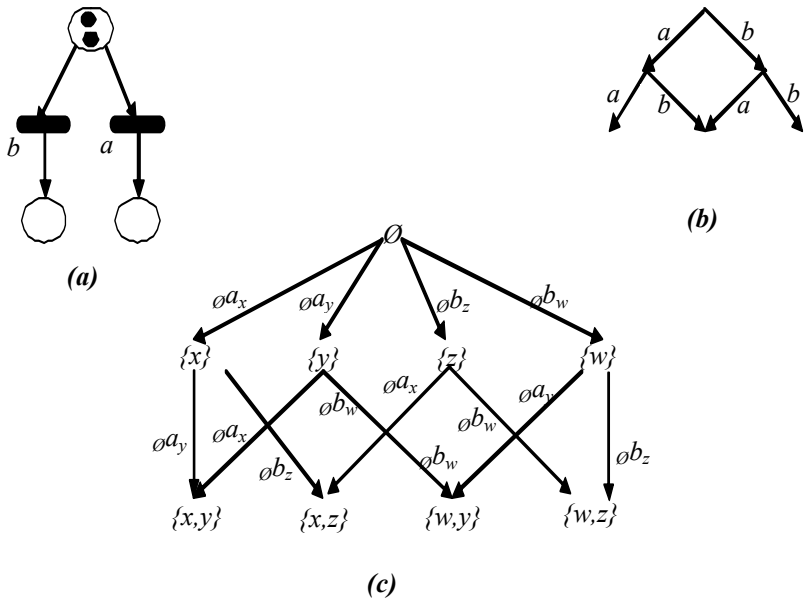


FIG. 1.3 – Conflit structurel

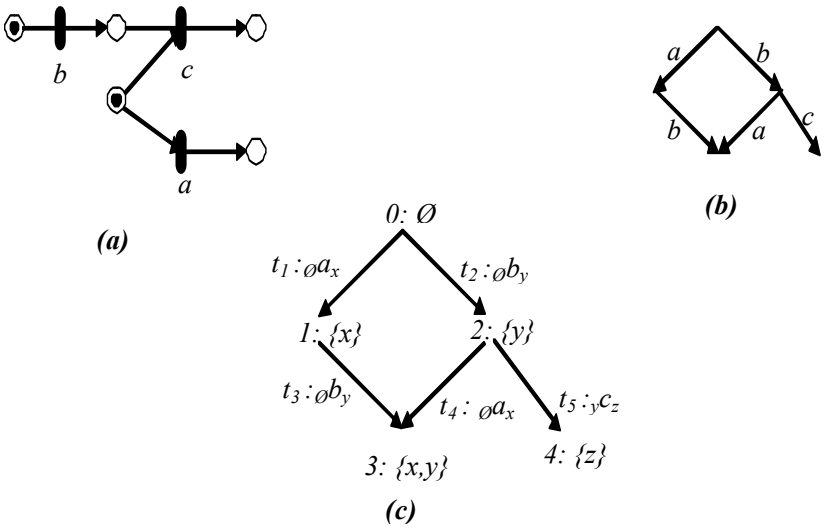


FIG. 1.4 – Le conflit différé

Contributions

Nous proposons, au niveau de ce présent mémoire, deux méthodes de réduction : la réduction basée sur l'approche d'ordre partiel et la réduction basée sur une relation d'équivalence. Les algorithmes et l'implémentation de chaque méthode proposée sont aussi donnés.

- **Méthode de réduction basée sur l'approche d'ordre partiel** :

  C'est une méthode qui combine l'utilisation du modèle des STEMs (Systèmes de Transitions Etiquetées Maximales)[Sai96] et la méthode de pas couvrants ce qui permettra de répondre aux limites citées précédemment.

  Le modèle des STEMs peut être utilisé comme une représentation sémantique du comportement du système étudié, d'où la possibilité d'utiliser différents modèles de spécification ; pour cela, il suffit de définir la sémantique en terme de STEMs pour chacun de ces modèles. Prenons par exemple le STEM de la Figure 1.4.(c) représentant le comportement de la Figure 1.4.(a). Dans l'état initial, aucune action n'a encore commencé son exécution. La transition t_1 (resp t_2) représente le début d'exécution (identifié par l'événement x (resp y)) de l'action a(resp b). A l'état 1 (resp 2) l'action a (resp b) est potentiellement en cours d'exécution, ceci est représenté respectivement par les événements (dits maximaux) x et y. Dans l'état 2, l'occurrence de c est conditionnée par la terminaison de b, ce qui est traduit par la présence de l'événement y au niveau de la transition t_5, donc z est le seul événement maximal dans l'état 4. Dans l'état 3, les événements x et y sont maximaux, c'est-à-dire que les actions correspondantes (a et b) peuvent s'exécuter simultanément (en parallèle).

  En utilisant la notion d'événements maximaux qui offre la possibilité de prendre en considération le parallélisme dynamique et d'exploiter au maximum les relations d'indépendance entre les actions, nous avons pu réduire le STEM de la Figure 1.4.(c) en la structure de la Figure 1.5.(a), et la figure 1.3.(c) en celle de 1.5.(b).

- **Méthode de réduction basée sur une relation d'équivalence :**

  Cette méthode, consiste d'éliminer les redondances modulo une relations d'équivalence spécifique au modèle des STEMs, connue sous le nom de la α-équivalence. Cette relation permet de regrouper des STEMs, qui décrivent le même comportement dans la seule différence réside dans le choix des noms des événements. Par exemple, les deux STEMs de la Figure 1.6 décrivent le même comportement (l'exécution parallèle de a et de b), nous pouvons obtenir le STEM de la Figure 1.6.(a) à partir de celui de la Figure 1.6.(b) en substituant les noms des événements e (resp z) par x (resp y).

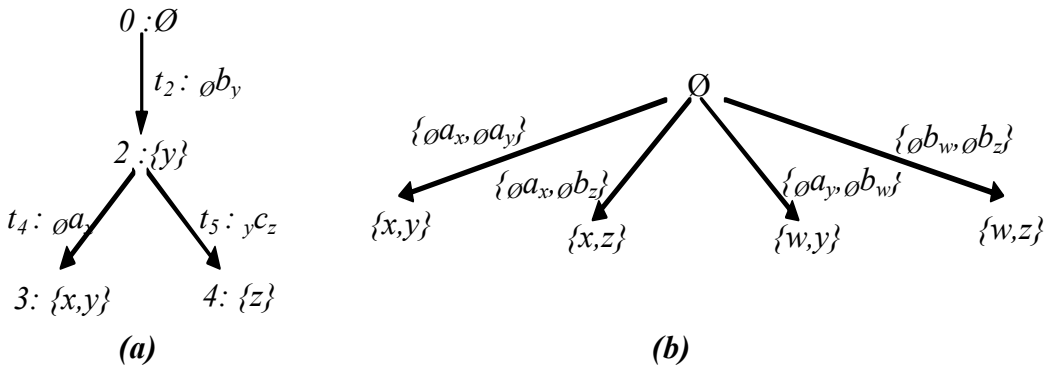


FIG. 1.5 – Graphes Réduits

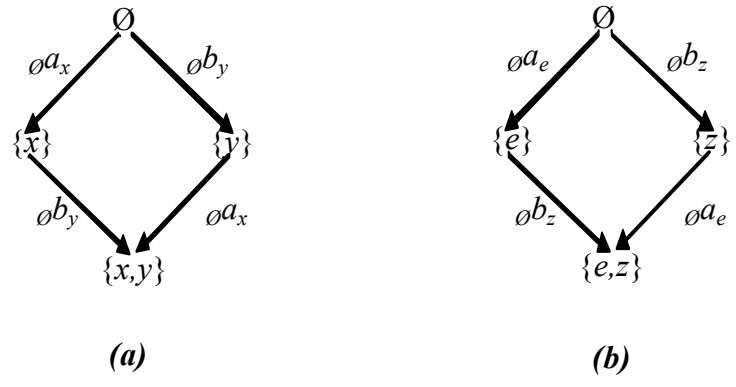


FIG. 1.6 – Deux STEMs α-équivalents

Plan du document

Le mémoire est organisé de la manière suivante :

- Le chapitre 2 introduit la sémantique de maximalité du langage LOTOS, et évoque quelques relations d'équivalence définies sur les systèmes de transitions étiquetées maximales, à savoir la α-équivalence et la bissimulation forte.
- Le chapitre 3 présente la technique de réduction du graphe d'états basée sur la notion d'ordre partiel. Les graphes persistants et les graphes de pas couvrants sont deux approches d'ordre partiel, en plus de l'introduction de ces deux la, l'approche qui les combine est également rappelée.
- Le chapitre 4 propose nos contributions ; deux méthodes de réduction du graphe d'états, la première basée sur la relation de la $\alpha-$équivalence et la deuxième combine l'utilisation du modèle des STEMs et la méthode de pas couvrants. Les algorithmes de réduction à la volée de ces chaque méthode sont présentés.
- Le chapitre 5 montre d'une manière expérimentale les bénéfices et les limitations de nos deux méthodes de réductions en utilisant des outils existants et les outils que nous avons développés dans ce présent travail.
- Le chapitre 6 évoque l'implémentation de nos méthodes de réduction
- Le chapitre 7 donne finalement quelques conclusions et perspectives des travaux présentés dans ce mémoire.

Chapitre 2

Sémantique de Maximalité du langage LOTOS

Ce chapitre est un rappel de la sémantique de maximalité de Basic LOTOS telle qu'elle a été définie en [CS95][Sai96] et la sémantique opérationnelle de Basic LOTOS, entre autre, quelques relations d'équivalence définies sur la maximalité sont ainsi données. La technique de description formelle LOTOS est donnée en premier lieu.

2.1 Technique de description formelle LOTOS

LOTOS [BB87][ISO88] est une technique de description formelle développée comme une norme internationale, qui pourrait être employée pour la description et analyse de systèmes complexes. Son développement était motivé par le fort besoin d'un langage avec une abstraction de haut niveau et une base mathématique solide.

Il existe en LOTOS une nette séparation entre données et contrôle, en d'autres termes, une spécification LOTOS est constituée de deux parties, la première partie est basée sur la théorie de types abstraits algébriques. La syntaxe et la sémantique employées sont celles du langage ACT ONE [EM85]. La deuxième partie (Basic LOTOS) décrit le comportement de la spécification, la syntaxe et la sémantique utilisées étant inspirées des deux algèbres de processus CCS [Mil80] et CSP [Hoa85]. Dans le cadre de ce présent travail, les données ne seront pas prises en compte.

Le comportement d'un système est considéré comme un ensemble de processus en interaction sur des portes de communication. L'interaction ici exprime la synchronisation par rendez-vous symétriques entre plusieurs processus.

2.1.1 Syntaxe formelle de Basic LOTOS

Soit PN l'ensemble des variables de processus parcouru par X et soit $\mathcal{G}$ l'ensemble des noms de portes définies par l'utilisateur (ensemble des actions observables) parcouru par $a, b, \ldots$.

Une porte observable particulière $\delta \notin \mathcal{G}$ est utilisée pour notifier la terminaison avec succès des processus. L dénote tout sous ensemble de $\mathcal{G}$, l'action interne est désignée par i. $\mathcal{B}$ parcouru par $E, F, ...$ dénote l'ensemble des expressions de comportement dont la syntaxe est :

$$
\begin{aligned}
E ::= \quad & stop \\
& \mid exit \\
& \mid E[L] \\
& \mid a; E \\
& \mid i; E \\
& \mid E[]E \\
& \mid E|[L]|E \\
& \mid hide\, L\, in\, E \\
& \mid E >> E \\
& \mid E [> E
\end{aligned}
$$

Opérateurs du langage

Soient E et F deux expressions de comportement et soit $L \subseteq \mathcal{G}$ un sous ensemble de portes

- Inaction : *stop* est une expression représente un processus inactif (un processus qui ne fait aucune interaction).
- Terminaison avec succès : *exit* est un processus de terminaison avec succès, son comportement se résume en une interaction sur la porte avant de se transformer dans le processus inactif.
- Préfixage : $a; E$ représente le comportement d'un processus qui interagit sur le port a et qui se comporte ensuite comme E.
- Intériorisation : *hide L in E* représente le comportement de E dont lequel toutes les interactions sur les portes de$L$ sont rendues invisibles.
- Choix : l'expression de comportement $E[]F$ représente le processus qui se comporte soit comme E soit comme F.
- Composition parallèle : l'expression de comportement $E|[L]|F$ représente la composition parallèle de E et F avec synchronisation sur les portes qui sont dans L ; c'est à dire que E et F se comportent de façon indépendante sauf sur les portes appartenant à L ; dans ce cas le premier comportement qui veut interagir sur l'une de ces portes doit attendre l'autre comportement pour qu'ils se synchronisent sur cette action, avant de pouvoir continuer à évoluer. Deux opérateurs sont utilisés pour désigner des cas particuliers de composition parallèle. Ce sont respectivement :
  * $E|||F$, qui représente le cas où l'ensemble des portes de synchronisation L est vide $(L = \varnothing)$

  * $E||F$, qui représente le cas où l'ensemble des portes de synchronisation L est égal à $\mathcal{G}$.

- Séquencement de processus : l'expression de comportement $E >> F$ dénote un processus qui se comporte d'abord comme E, et qui, dés que E s'est terminé avec succès, se comporte comme F.

- Interruption : l'expression de comportement $E[> F$ dénote un processus qui se comporte comme E, mais qui peut à tout moment, tant que E ne s'est pas terminé avec succès, être interrompu par F, ce dernier prenant la main et continuant à s'exécuter.

- Renommage de portes : soient $a_1, \ldots, a_n$ et $b_1, \ldots, b_n$ des éléments de $\mathcal{G}$. L'expression $E[b_1/a_1, .., b_n/a_n]$ représente un comportement identique à E sauf pour les occurrences des actions $a_1, \ldots, a_n$ qui sont renommées respectivement par les actions $b_1, \ldots, b_n$.

- Instanciation de processus : L'ensemble des noms des processus est noté PN, qui est parcouru par $P, Q, \ldots$. Une instanciation de processus est faite pour les occurrences des noms de processus dans une expression de comportement.

- Définition de processus : $P := E$ dénote la définition d'un processus P dont le comportement est décrit par l'expression de comportement E. L'ensemble des définitions de processus est appelé environnement de processus. Une expression de comportement est toujours considérée dans le contexte d'un environnement de processus.

2.2 Sémantiques opérationnelles structurées de Basic LOTOS

2.2.1 Principe de la sémantique de maximalité

La sémantique d'un système concurrent peut être caractérisée par l'ensemble des états du système et des transitions par lesquelles le système passe d'un état à un autre. Dans l'approche basée sur la maximalité, les transitions sont des événements qui ne représentent que le début de l'exécution des actions. En conséquence, l'exécution concurrente de plusieurs actions devient possible, c'est-à-dire que l'on peut distinguer exécutions séquentielles et exécutions parallèles d'actions.

Etant donné que plusieurs actions qui ont le même nom peuvent s'exécuter en parallèle (auto-concurrence), nous associons, pour distinguer les exécutions de chacune des actions, un identificateur à chaque début d'exécution d'action, c'est-à-dire à la transition ou à l'événement associé. Dans un état, un événement est dit maximal s'il correspond au début de l'exécution d'une action qui peut éventuellement être toujours en train de s'exécuter dans cet état là. Associer des noms d'événements maximaux aux états nous conduit à la notion de configuration qui sera formalisée dans la définition 2.2.

Pour illustrer la maximalité et cette notion de configuration, considérons les expressions de comportement E et F de la Figure 2.1. Dans l'état initial, aucune action n'a encore été

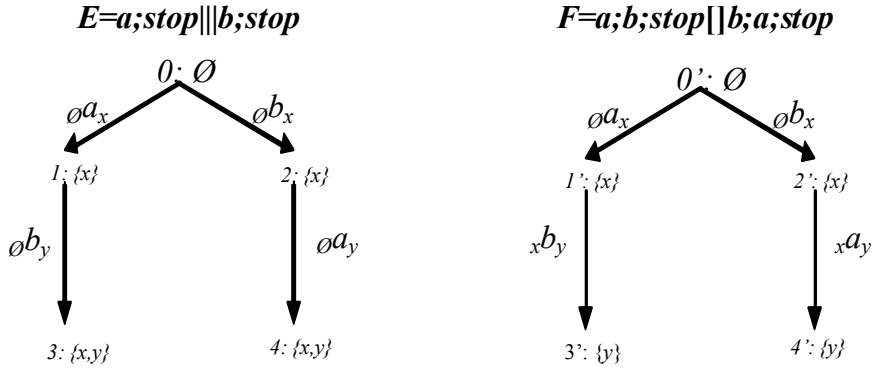


FIG. 2.1 – Arbres de dérivation de E et F

exécutée, donc l'ensemble des évenements maximaux est vide, d'où les configurations initiales suivantes associées à E et F : ${}_{\phi}[E]$ (état 0) et ${}_{\phi}[F]$ (état $0'$). En appliquant la sémantique de maximalité, les transitions suivantes sont possibles : ${}_{\phi}[E] \xrightarrow{\phi^a x}_m {}_{\{x\}}[stop] \;|||\; {}_{\phi}[b; stop] \xrightarrow{\phi^b y}_m {}_{\{x\}}[stop] \;|||\; {}_{\{y\}}[stop]$

x (resp y) étant le nom de l'événement identifiant le début de l'action a (respectivement b). Etant donné que rien ne peut être conclu à propos de la terminaison des deux actions a et b dans la configuration ${}_{\{x\}}[stop] \;|||\; {}_{\{y\}}[stop]$ (état 3), x et y sont alors maximaux dans cette configuration. Notons que x est également maximal dans l'état (état 1) intermédiaire représenté par la configuration ${}_{\{x\}}[stop] \;|||\; {}_{\phi}[b; stop]$.

Pour la configuration initiale, associée à l'expression de comportement F, la transition suivante est possible : ${}_{\phi}[F] \xrightarrow{\phi^a x}_m {}_{\{x\}}[b; stop]$. Comme précédemment, x identifie le début de l'action a et il est le nom du seul événement maximal dans la configuration ${}_{\{x\}}[b; stop]$ (état $1'$). Il est clair que, au vu de la sémantique de l'opérateur de préfixage, le début de l'exécution de l'action b n'est possible que si l'action a a terminé son exécution. Par conséquent, x ne reste plus maximal lorsque l'action b commence son exécution ; l'unique événement maximal dans la configuration résultante est donc celui identifié par y qui correspond au début de l'exécution de l'action b. L'ensemble des noms des événements maximaux a donc été modifié par la suppression de x et l'ajout de y, ce qui justifie la dérivation suivante : ${}_{\{x\}}[b; stop] \xrightarrow{\{x\}^b y}_m {}_{\{y\}}[stop]$

La configuration ${}_{\{y\}}[stop]$ est différente de la configuration ${}_{\{x\}}[stop] \;|||\; {}_{\{y\}}[stop]$, car la première ne possède qu'un seul événement maximal (identifié par y), alors que la deuxième en possède deux (identifiés par x et y). Les arbres de dérivation des expressions de comportement E et F obtenus par l'application de la sémantique de maximalité sont représentés dans la figure 2.1

2.2.2 Sémantique de maximalité de Basic LOTOS

Définition 2.1 *L'ensemble des noms des événements est un ensemble dénombrable noté $\mathcal{M}$. Cet ensemble est parcouru par $x, y, ...$ $M, N, ...$ dénotent des sous-ensembles finis de $\mathcal{M}$. L'ensemble des atomes de support Act est $Atm = 2^{\mathcal{M}}_{fn} \times Act \times \mathcal{M}$, $2^{\mathcal{M}}_{fn}$ étant l'ensemble des parties finies de $\mathcal{M}$. Pour $M \in 2^{\mathcal{M}}_{fn}$, $x \in \mathcal{M}$ et $a \in Act$, l'atome (M, a, x) sera noté ${}_M a_x$. Le choix d'un nom d'évenement peut se faire de manière déterministe par l'utilisation de toute fonction get : $2^{\mathcal{M}} - \{\phi\} \rightarrow \mathcal{M}$ satisfaisant $get(M) \in M$ pour tout $M \in 2^{\mathcal{M}} - \{\phi\}$.*

Définition 2.2 *"Configuration"*

L'ensemble $\mathcal{C}$ des configurations des expressions de comportement de Basic LOTOS est le plus petit ensemble défini par induction comme suit :

- $\forall E \in \mathcal{B}, \forall M \in 2^{\mathcal{M}}_{fn} : \quad {}_M[E] \in \mathcal{C}$
- $\forall P \in PN, \forall M \in 2^{\mathcal{M}}_{fn} : \quad {}_M[P] \in \mathcal{C}$
- *si $\mathcal{E} \in \mathcal{C}$ alors hide L in $\mathcal{E} \in \mathcal{C}$*
- *si $\mathcal{E} \in \mathcal{C}$ et $F \in \mathcal{B}$ alors $\mathcal{E} \gg F \in \mathcal{C}$*
- *si $\mathcal{E}, \mathcal{F} \in \mathcal{C}$ alors $\mathcal{E}$ op $\mathcal{F} \in \mathcal{C} \quad op \in \{ [] , ||| , || , |[L]| , [> \}$*
- *si $\mathcal{E} \in \mathcal{C}$ et $\{a_1, ..., a_n\}, \{b_1, ..., b_n\} \in 2^{\mathcal{G}}_{fn}$ alors $\mathcal{E}[b_1/a_1, ..., b_n/a_n] \in \mathcal{C}$*

Etant donné un ensemble $M \in 2^{\mathcal{M}}_{fn}$, ${}_M[\cdots]$ est appelée opération d'*encapsulation*. Cette opération est distributive par rapport aux opérations $[], |[L]|$, *hide*, $[>$ et le renommage des portes. Nous admettons aussi que ${}_M[E \gg F] \equiv {}_M[E] \gg F$. Une configuration est dite canonique si elle ne peut plus être réduite par la distribution de l'opération d'encapsulation sur les autres opérateurs. Par la suite, nous supposons que toutes les configurations sont canoniques.

Proposition 2.1 *[Sai96]"Configuration canonique"*

Toute configuration canonique est sous l'une des formes suivantes ($\mathcal{E}$ et $\mathcal{F}$ étant des configurations canoniques) :

${}_M[stop]$	${}_M[exit]$	${}_M[a; E]$	${}_M[P]$	$\mathcal{E}$ [] $\mathcal{F}$
$\mathcal{E}$ $\|[L]\|$ $\mathcal{F}$	hide L in $\mathcal{E}$	$\mathcal{E} \gg F$	$\mathcal{E}$ $[>$ $\mathcal{F}$	$\mathcal{E}[b_1/a_1, ..., b_n/a_n]$

Définition 2.3 *La fonction $\psi : C \rightarrow 2^{\mathcal{M}}_{fn}$, qui détermine l'ensemble des noms des événements dans une configuration, est définie récursivement par :*

$$\psi({}_M[E]) = M \qquad \psi(\mathcal{E} \ [] \ \mathcal{F}) = \psi(\mathcal{E}) \cup \psi(\mathcal{F}) \qquad \psi(\mathcal{E} \ |[L]| \ \mathcal{F}) = \psi(\mathcal{E}) \cup \psi(\mathcal{F})$$
$$\psi(\mathcal{E} \gg F) = \psi(\mathcal{E}) \qquad \psi(hide \ L \ in \ \mathcal{E}) = \psi(\mathcal{E}) \qquad \psi(\mathcal{E} \ [> \mathcal{F}) = \psi(\mathcal{E}) \cup \psi(\mathcal{F})$$
$$\psi(\mathcal{E}[b_1/a_1, ..., b_n/a_n]) = \psi(\mathcal{E})$$

Définition 2.4 *"Suppression"*

Soit $\mathcal{E}$ une configuration ; $\mathcal{E}\backslash N$ dénote la configuration obtenue par la suppression de l'ensemble des noms des événements N de la configuration $\mathcal{E}$. $\mathcal{E}\backslash N$ est définie récursivement sur la configuration $\mathcal{E}$ comme suit :

$$\begin{array}{ll}
({}_M[E])\backslash N = {}_{M-N}[E] & (\mathcal{E}\ []\ \mathcal{F})\backslash N = \mathcal{E}\backslash N\ []\ \mathcal{F}\backslash N \\
(\mathcal{E}\ |[L]|\ \mathcal{F})\backslash N = \mathcal{E}\backslash N\ |[L]|\ \mathcal{F}\backslash N & (hide\ L\ in\ \mathcal{E})\backslash N = hide\ L\ in\ \mathcal{E}\backslash N \\
(\mathcal{E} \gg F)\backslash N = \mathcal{E}\backslash N \gg F & (\mathcal{E}\ [>\ \mathcal{F})\backslash N = \mathcal{E}\backslash N\ [>\ \mathcal{F}\backslash N \\
(\mathcal{E}\,[b_1/a_1,...,b_n/a_n])\backslash N = \mathcal{E}\backslash N\,[b_1/a_1,...,b_n/a_n] &
\end{array}$$

Définition 2.5 *"Substitution"*

L'ensemble des fonctions de substitution des noms des événements est Subs (i.e. Subs $= \mathcal{M} \to 2^{\mathcal{M}}_{fn}$) ; $\sigma, \sigma_1, \sigma_2, ...$ désignent des éléments de Subs. Etant donnés $x, y, z \in \mathcal{M}$ et $M \in 2^{\mathcal{M}}_{fn}$, alors

- *L'application de σ à x sera écrite σx ;*
- *La substitution identité ι est définie par $\iota x = \{x\}$;*
- *$M\sigma = \cup_{x \in M}\ \sigma x$;*
- *$\sigma\,[y/z]$ est définie par : $\sigma\,[y/z]\,x = \begin{cases} \{y\} & si\ z = x \\ \sigma x & sinon \end{cases}$*

Soit σ une fonction de substitution, la substitution simultanée de toutes les occurrences de x dans $\mathcal{E}$ par σx, est définie récursivement sur la configuration $\mathcal{E}$ comme suit :

$$\begin{array}{ll}
({}_M[E])\,\sigma = {}_{M\sigma}[E] & (\mathcal{E}\ []\ \mathcal{F})\,\sigma = \mathcal{E}\sigma\ []\ \mathcal{F}\sigma \\
(\mathcal{E}\ |[L]|\ \mathcal{F})\,\sigma = \mathcal{E}\sigma\ |[L]|\ \mathcal{F}\sigma & (hide\ L\ in\ \mathcal{E})\,\sigma = hide\ L\ in\ \mathcal{E}\sigma \\
(\mathcal{E} \gg F)\,\sigma = \mathcal{E}\sigma \gg F & (\mathcal{E}\ [>\ \mathcal{F})\,\sigma = \mathcal{E}\sigma\ [>\ \mathcal{F}\sigma \\
(\mathcal{E}\,[b_1/a_1,...,b_n/a_n])\,\sigma = \mathcal{E}\sigma\,[b_1/a_1,...,b_n/a_n] &
\end{array}$$

Définition 2.6 *"Sémantiques opérationelles structurées de Basic LOTOS"*

La relation de transition de maximalité $\longrightarrow \subseteq \mathcal{C} \times Atm \times \mathcal{C}$ est définie comme étant la plus petite relation satisfaisant les règles suivantes :

1. $\dfrac{}{{}_M[exit] \xrightarrow{M^{\delta}_{x}} {}_{\{x\}}[stop]}\ x = get(\mathcal{M})$
2. $\dfrac{}{{}_M[a;E] \xrightarrow{M^{a}_{x}} {}_{\{x\}}[E]}\ x = get(\mathcal{M})$
3. (a) $\dfrac{\mathcal{E} \xrightarrow{M^{a}_{x}} \mathcal{E}'}{\mathcal{F}\ []\ \mathcal{E} \xrightarrow{M^{a}_{x}} \mathcal{E}' \qquad \mathcal{E}\ []\ \mathcal{F} \xrightarrow{M^{a}_{x}} \mathcal{E}'}$
4. (a) i. $\dfrac{\mathcal{E} \xrightarrow{M^{a}_{x}} \mathcal{E}' \qquad a \notin L \cup \{\delta\}}{\mathcal{E}\ |[L]|\ \mathcal{F} \xrightarrow{M^{a}_{x}} \mathcal{E}'[y/x]\ |[L]|\ \mathcal{F}\backslash M}\ y = get(\mathcal{M} - ((\psi(\mathcal{E}) \cup \psi(\mathcal{F})) - M))$

ii. $\dfrac{\mathcal{E} \overset{M^a x}{\longrightarrow} \mathcal{E}' \qquad a \notin L \cup \{\delta\}}{\mathcal{F} \ |[L]| \ \mathcal{E} \overset{M^a x}{\longrightarrow} \mathcal{F} \backslash M \ |[L]| \ \mathcal{E}'[y/x]} y = get(\mathcal{M} - ((\psi(\mathcal{E}) \cup \psi(\mathcal{F})) - M))$

(b) $\dfrac{\mathcal{E} \overset{M^a x}{\longrightarrow} \mathcal{E}' \qquad \mathcal{F} \overset{M^a y}{\longrightarrow} \mathcal{F}' \qquad a \in L \cup \{\delta\}}{\mathcal{E} \ |[L]| \ \mathcal{F} \overset{M \cup N^a z}{\longrightarrow} \mathcal{E}'[z/x] \backslash N \ |[L]| \ \mathcal{F}'[z/y] \backslash M} \ z = \begin{array}{l} get(\mathcal{M} - ((\psi(\mathcal{E}) \cup \\ \psi(\mathcal{F})) - (M \cup N))) \end{array}$

5. *(a)* $\dfrac{\mathcal{E} \overset{M^a x}{\longrightarrow} \mathcal{E}' \qquad a \notin L}{hide \ L \ in \ \mathcal{E} \overset{M^a x}{\longrightarrow} hide \ L \ in \ \mathcal{E}'}$

(b) $\dfrac{\mathcal{E} \overset{M^a x}{\longrightarrow}_m \mathcal{E}' \qquad a \in L}{hide \ L \ in \ \mathcal{E} \overset{M^i x}{\longrightarrow} hide \ L \ in \ \mathcal{E}'}$

6. *(a)* $\dfrac{\mathcal{E} \overset{M^a x}{\longrightarrow}_m \mathcal{E}' \qquad a \neq \delta}{\mathcal{E} >> F \overset{M^a x}{\longrightarrow} \mathcal{E}' >> F}$

(b) $\dfrac{\mathcal{E} \overset{M^\delta x}{\longrightarrow} \mathcal{E}'}{\mathcal{E} >> F \overset{M^i x}{\longrightarrow} {}_{\{x\}}[F]}$

7. *(a)* $\dfrac{\mathcal{E} \overset{M^a x}{\longrightarrow} \mathcal{E}' \qquad a \neq \delta}{\mathcal{E} \ [> \ \mathcal{F} \overset{M^a y}{\longrightarrow} \mathcal{E}'[y/x] \ [> \ \mathcal{F} \backslash M} y = get(\mathcal{M} - (\psi(\mathcal{E}) \cup \psi(\mathcal{F}) - M))$

(b) $\dfrac{\mathcal{E} \overset{M^\delta x}{\longrightarrow} \mathcal{E}'}{\mathcal{E} \ [> \ \mathcal{F} \overset{M^\delta y}{\longrightarrow} \mathcal{E}'[y/x] \ [> \ {}_{\psi(\mathcal{F}) - M}[stop]} y = get(\mathcal{M} - (\psi(\mathcal{E}) \cup \psi(\mathcal{F}) - M))$

(c) $\dfrac{\mathcal{F} \overset{M^a x}{\longrightarrow} \mathcal{F}'}{\mathcal{E} \ [> \ \mathcal{F} \overset{M^a y}{\longrightarrow} {}_{\psi(\mathcal{E}) - M}[stop] \ [> \ \mathcal{F}'[y/x]} y = get(\mathcal{M} - (\psi(\mathcal{E}) \cup \psi(\mathcal{F}) - M))$

8. *(a)* $\dfrac{\mathcal{E} \overset{M^a x}{\longrightarrow} \mathcal{E}' \qquad a \notin \{a_1, ..., a_n\}}{\mathcal{E}[b_1/a_1, ..., b_n/a_n] \overset{M^a x}{\longrightarrow} \mathcal{E}'[b_1/a_1, ..., b_n/a_n]}$

(b) $\dfrac{\mathcal{E} \overset{M^a x}{\longrightarrow} \mathcal{E}' \qquad a = a_i \ (1 \leq i \leq n)}{\mathcal{E}[b_1/a_1, ..., b_n/a_n] \overset{M^{bi} x}{\longrightarrow} \mathcal{E}'[b_1/a_1, ..., b_n/a_n]}$

9. $\dfrac{P := E \qquad {}_M[E] \overset{M^a x}{\longrightarrow} \mathcal{F}}{{}_M[P] \overset{M^a x}{\longrightarrow} \mathcal{F}}$

L'explication de ces règles-telle qu'elle a été présentée en[Sai96]-est donnée ci-dessous, :

La règle 1 implique que la terminaison avec succès ne peut commencer qu'après que toutes les actions référencées par l'ensemble M des noms d'événements aient terminé leur exécution.

La règle 2 caractérise la sémantique des opérateurs de préfixage des actions ; comme le début de l'exécution de l'action a dépend de la terminaison des actions associées aux événements de noms M, seul x, le nom de l'événement associé au début de l'action a, apparaît dans la configuration ${}_{\{x\}}[E]$.

Les règles 3 caractérisent la sémantique de l'opérateur de choix ; c'est une adaptation directe aux configurations de la règle d'inférence définie pour la sémantique opérationnelle structurée de l'entrelacement pour les expressions de comportement de Basic LOTOS.

Les règles 4(a)i, 4(a)ii et 4b caractérisent la sémantique de l'opérateur de composition parallèle. Pour l'explication de ces règles, considérons l'expression de comportement G .

$$G \equiv a; c; d, stop |[c]| b; c; e; stop$$

Dans la configuration ${}_{\varnothing}[G]$, les actions a et b sont sensibilisées et peuvent donc commencer leur exécution ; supposons que $x = get(\mathcal{M})$ et que $y = get(\mathcal{M} - \{x\})$, les dérivations suivantes peuvent être obtenues par l'application de la règle 4.a deux fois de suite :

$$ {}_{\varnothing}[G] \xrightarrow{\varnothing\, a_x}_m \ {}_{\{x\}}[c; d, stop]||[c]|_{\varnothing}[b; c; e; stop] \xrightarrow{\varnothing\, b_y}_m \underbrace{{}_{\{x\}}[c; d, stop]}_{S}||[c]|\underbrace{{}_{\{y\}}[c; e; stop]}_{S'} $$

Notons que, dans le cas où b commence son exécution en premier, le nom de l'événement qui lui est associé est x et le nom de l'événement associé à a est y ; la dérivation suivante est possible :

$$ {}_{\varnothing}[G] \xrightarrow{\varnothing\, b_x}_m \ {}_{\varnothing}[a; c; d, stop]||[c]|_{\{x\}}[c; e; stop] \xrightarrow{\varnothing\, a_y}_m \ {}_{\{y\}}[c; d, stop]||[c]|_{\{x\}}[c; e; stop] $$

Quand l'action c commence son exécution, elle est identifiée par le nom de l'événement égal à $get(\mathcal{M} - ((\psi(S) \cup \psi(S')) - \{x, y\}) = get(\mathcal{M}) = x$ qui devient le seul nom d'événement présent dans la configuration résultante, ce qui est montré dans la dérivation suivante obtenue par l'application de la règle 4.b :

$$ {}_{\{x\}}[c; d, stop]||[c]|_{\{y\}}[c; e; stop] \xrightarrow{\{x,y\}\, bcx}_m {}_{\{x\}} \ [d, stop]||[c]|_{\{x\}}[e; stop] $$

Pour cet état, le début de l'action d signifie que l'action c a terminé son exécution, et par conséquent x n'est plus le nom d'un événement maximal ; dans la configuration résultante, le nom x est supprimé des deux arguments de l'opérateur de composition parallèle. L'application de la règle 4.a mène à la dérivation suivante :

$$ {}_{\{x\}}[d, stop]||[c]|_{\{x\}}[e; stop] \xrightarrow{\{x\}\, d_x}_m \ {}_{\{x\}} \ [stop]||[c]|_{\varnothing}[e; stop] $$

Les règles 5a et 5b sont une adaptation directe aux configurations des règles correspondantes de la sémantique de l'entrelacement.

Les règles 6a et 6b traduisent la sémantique de l'opérateur de séquencement de processus. En fait, la règle 6a implique que le comportement de $\mathcal{E} >> \mathcal{F}$ est celui de $\mathcal{E}$ tant que $\mathcal{E}$ n'a pas terminé son exécution avec succès ; alors que la règle 6b stipule qu »une fois que $\mathcal{E}$ a terminé avec succès, l'unique événement maximal dans la configuration résultante, c'est-à-dire ${}_{\{x\}}[F]$, est alors celui identifié par x.

Les règles 7a, 7b et 7c donnent la sémantique de l'opérateur d'interruption. Afin de mieux comprendre ce mécanisme d'interruption, il est nécessaire de distinguer les différentes stratégies d'interruption qui peuvent être considérées en présence de la non atomicité des actions. Ces stratégies d'interruption sont :

- La première stratégie consiste à interrompre le processus, y compris les actions qui sont en cours. Cette stratégie est illustrée par la Figure 2.2.a.

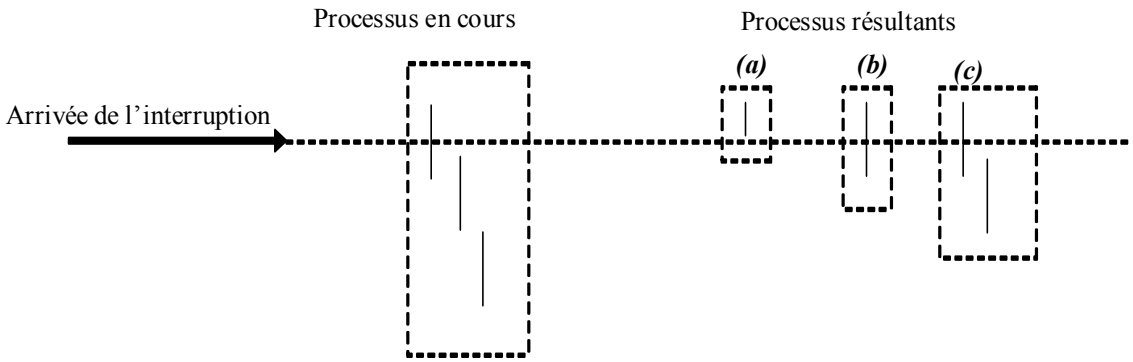


FIG. 2.2 – Stratégies d'interruption

- La deuxième stratégie consiste à empêcher les actions qui n'ont pas commencé leur exécution de s'exécuter tout en laissant les actions en cours continuer leur exécution. Cette stratégie est illustrée par la Figure 2.2.b.
- La troisième stratégie consiste à n'autoriser l'interruption que lorsque les actions qui sont en cours ne sont terminées. Cette stratégie a pour risque de ne jamais pouvoir interrompre le processus, dans le cas où il y a chevauchement entre le début el la fin des actions. Cette stratégie est illustrée par la Figure 2.2.c.

Dans[Sai96] c'est la deuxième stratégie qui a été prit en considéré.

Dans la sémantique de maximalité, l'ensemble des événements maximaux associé aux états est d'une importance capitale, car il représente les actions qui sont commencé leur exécution et qui peuvent encore être en train de s'exécuter dans cet état. De ce fait, lorsque l'un des processus $\mathcal{E}$ ou $\mathcal{F}$ termine son exécution, de son gré ou suite à une interruption, il est remplacé dans la configuration résultante par le processus qui n'offre plus d'actions nouvelles mais qui garde trace des action qui peuvent encore être en cours. Ceci implique, sous l'hypothèse que les actions ne sont pas atomiques, que l'interruption d'un processus n'affecte que les actions qui n'ont pas commencé leur exécution, les actions en cours terminant leur exécution.

Pour illustrer ces règles, nous prenons l'expression de comportement

$$H \equiv a; (b; exit[> c; exit)$$

En admettant que $x = get(\mathcal{M})$ et que $y = get(\mathcal{M} - \{x\})$, la dérivation suivante est obtenue par l'application de la règle 2

$$_{\varnothing}[H] \xrightarrow{\varnothing\, a_x}_{m\{x\}} [b; exit][>_{\{x\}} [c; exit]$$

De la configuration résultante, nous pouvons déduire que l'action a a commencé son exécution ; pour que les action b et c puissent commencer la leur, il faut que a ait terminé son exécution, ce qui explique les deux dérivations suivantes :

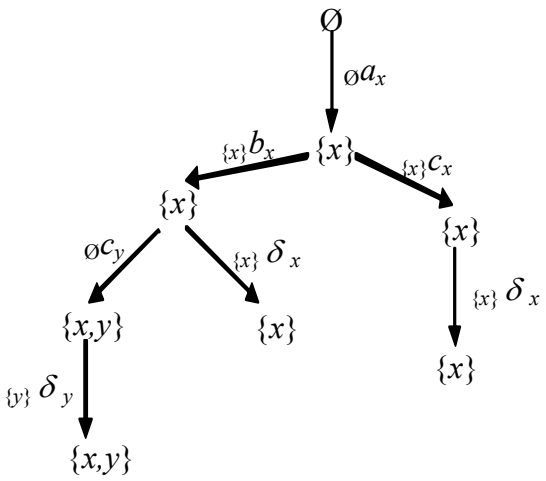


FIG. 2.3 – Arbre de dérivations de H

$$ {}_{\{x\}}[b; exit][>_{\{x\}}\ [c; exit] \quad \left\{ \begin{array}{ll} \xrightarrow{\{x\}b_x}_m & {}_{\{x\}}[exit][>_{\varnothing}\ [c; exit] \\ \xrightarrow{\{x\}c_x}_m & {}_{\varnothing}[stop][>_{\{x\}}\ [exit] \end{array} \right. $$

En continuant l'application des règles 1,2 et 7 nous obtenons l'arbre de dérivation de la Figure 2.3

Les règles 8 et 9 sont une adaptation directe aux configurations des règles correspondantes de la sémantique de l'entrelacement.

2.2.3 Systèmes de transitions étiquetées maximales

Comme nous l'avons vu dans la section précédente, les arbres de dérivation de la Figure 2.1 et la Figure 2.3 sont des structures obtenus en appliquant directement les règles de la Définition 2.6. Précisément, ces structures représentent le modèle des arbres maximaux [Sai96], c'est un modèle de vrai parallélisme, il nous permet de déterminer pour chacun de ses états les actions qui sont en train de s'exécuter en parallèle ; cela est assuré en associant les informations sur le parallélisme aux états et aux transitions, dont les états contiennent les événements des actions qui sont potentiellement en exécution, et les transitions sont des événements qui ne représentent que le début de l'exécution des actions. En conséquence, nous pouvons facilement différencier le comportement de l'expression " $a; stop|||b; stop$ " de celui de l'expression "$a; b; stop[]b; a; stop$ " (Voir Figure 2.1), ce qu'est impossible pour le cas des modèles d'entrelacement. La différence est entre, par exemple, l'état 3 et l'état 3', dans ce dernier état, il n'y a qu'un seul événement maximal (associé à l'action b). Cependant, l'état 3 contient deux événements maximaux représentant l'exécution parallèle de a et de b.

Définition 2.7 *"Arbre maximal"*

$\mathcal{M}$ *étant un ensemble dénombrable de noms d'événements, un arbre maximal de support* $\mathcal{M}$ *est un quintuplet* (T, l, μ, ξ, ψ) *avec* (T, l) *un arbre étiqueté de support L et*

- $\psi : T \longrightarrow 2^{\mathcal{M}}_{fn}$ *est une fonction qui associe à chaque noeud l'ensemble (fini) des noms des événements maximaux présents au niveau de ce noeud.*
- $\mu : \Theta(T) \longrightarrow 2^{\mathcal{M}}_{fn}$ *est une fonction qui associe à chaque transition l'ensemble (fini) des noms des événements correspondant aux actions qui ont commencé leur exécution et dont la terminaison sensibilise cette transition ; cet ensemble correspond aux causes directes de la transition.*
- $\xi : \Theta(T) \longrightarrow \mathcal{M}$ *est une fonction qui associe à chaque transition le nom de l'événement qui identifie son occurrence.*

tel que, pour tout $(t, t\prime) \in \Theta(T)$ les conditions suivantes sont satisfaites :

1. $\mu(t, t') \sqsubseteq \psi(t)$: exprime que l'occurrence d'un événement ne dépend qui de l'ensemble d'événements maximaux de noeud précédent
2. $\xi(t, t') \not\subseteq \psi(t) - \mu'(t, t')$: exprime que le nom d'événement attribué à une transition ne peut être le nom d'événement maximal du noeud origine que s'il est une cause directe de cette transition.
3. $\psi(t') = (\psi(t) \backslash \mu(t, t')) \cup \{\xi(t, t')\}$: L'ensemble de nom d'événements maximaux du nœud cible est l'union du nom d'événement associé à la transition et l'ensemble résultant de la restriction de l'ensemble d'événements qui correspondent aux causes directes, sur l'ensemble d'événements maximaux du nœud origine.

une transition de maximalité sur les arbres maximaux peut être représentée par $t \xrightarrow{{}^{N}a_{x}}_{m} t\prime$avec : $t\prime = te, l(e) = a, \mu(t, t') = \mu({}_{N}a_{x}) = N, \zeta(t, t') = \zeta({}_{N}a_{x}) = x$, et $\psi(t') = \psi(t) - N) \cup \{x\}$; les deux conditions suivantes sont satisfaites par la transition (t, t') :$N \subseteq \psi(t)$ et $x \notin \psi(t) - N$.

Les arbres maximaux sont des structures acycliques, en pratique, les graphes d'états sont des graphes cycliques, c'est pour cette raison, les systèmes de transitions étiquetées maximales sont introduits.

Définition 2.8 *"Système de Transitions Etiquetées Maximales "*

$\mathcal{M}$ *étant un ensemble dénombrable des noms des évènements, un Système de Transitions Etiquetées Maximales (STEM) de support* $\mathcal{M}$ *est un quintuplet* $(\Omega, A, \mu, \xi, \psi)$ *avec :*

- $\Omega :< S, T, \alpha, \beta >$ *est un système de transitions tels que :*
  - *S : est un ensemble d'états qui peut être fini ou infini.*

- T : *est un ensemble de transitions qui peut aussi être fini ou infini.*
- α *et* β *sont deux applications de* T *dans* S *tels que pour toute transition* t *de* T *on a :* $\alpha(t)$ *l'origine de la transition et* $\beta(t)$ *son but.*

- $< \Omega, A >$ *est un système de transitions étiquetées par un alphabet* A.

2.2.4 Notions d'équivalence

Dans cette section, deux relations d'équivalence sont évoquées. La première équivalence est une technique de substitution spécifique aux STEMs, connue sous le nom de la $\alpha-$équivalence, et la deuxième relation cherche de regrouper les STEMs qui font les mêmes choix au même moment "bissimulation maximale".

$\alpha-$équivalence

Cette relation permet de regrouper des STEMs, qui décrivent le même comportement dans la seule différence réside dans le choix des noms des événements

Par exemple, les deux STEM de la Figure 1.6 décrivent le même comportement (l'exécution parallèle de a et de b), nous pouvons obtenir le STEM de la Figure 1.6.(a) à partir de celui de la Figure 1.6.(b) (modulo l'isomorphisme $\cong$)en substituant les noms des événements e (resp z) par x (resp y).

Définition 2.9 *"α-relation" Soit $=_\alpha$ la plus petite relation sur les STEMs telle que $S =_\alpha T$ ssi*

- $S \cong T$,*ou bien*
- $S \cong \sum_{i \in I} {}_{M_i}a_{i\ x_i}\ T_i, T \cong \sum_{j \in J} {}_{Mj}a_{j\ xj}\ T_j$, *et*

$-\psi(S) = \psi(T)$, *et*
$-$ *il existe une bijection* $f : I \longrightarrow J$ *telle que, pour tout* $i \in I, M = M_{f(i)}, a^i = a^{f(i)}$, *et*
- $x_i = x_{f(i)}$ *and* $Ti =_\alpha T_{f(i)}$
- $x_{f(i)} \notin \psi(T_i)$ *and* $T_i[x_{f(i)}/x_i] =_\alpha T_{f(i)}$

La α-relation est une relation d'équivalence où $=_\alpha \subseteq \cong$

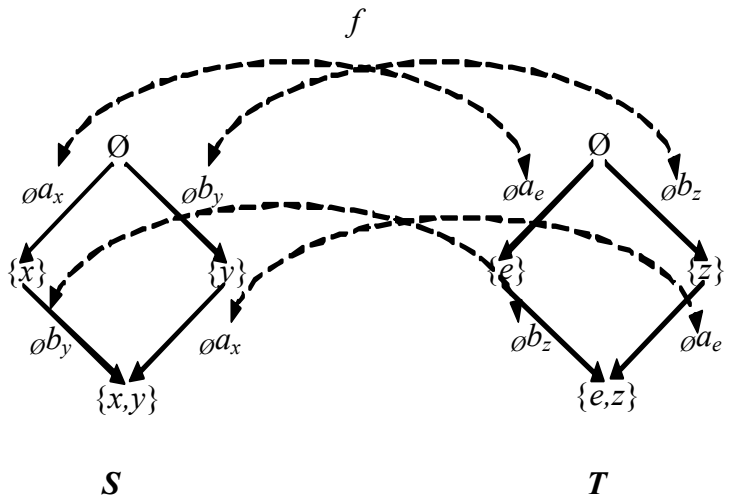


FIG. 2.4 – Une bijenction

Propriété 2.1 *Soient T et T' deux STEMs, et soit σ une fonction de substitution ; alors les propriétés suivantes sont vérifiées :*

1. *Si ${}_N a_x T$ est un STEM et $y \notin \psi(T)$, alors ${}_N a_x T =_\alpha {}_N a_x T[y/x]$*
2. *$T =_\alpha T\ \iota$*
3. *$T =_\alpha T' \Longleftrightarrow T\iota \cong T'\iota$*
4. *$T =_\alpha T' \Longleftrightarrow T\sigma \cong T'\sigma$*

La définition 2.9 cherche de trouver une fonction (bijection) entre les deux STEMs afin de décider c'est ils sont $\alpha-$équivalents ou non. En appliquant cette définition sur l'exemple de la Figure 1.6, la bijection est illustrée dans la Figure 2.4. Une alternative consiste de réduire le test de la $\alpha-$équivalence des STEMs T et T'à un test d'égalité (isomorphisme) des STEMs $T\sigma$ et $T'\sigma$ (Propriété2.1.(4)). Cette dernière possibilité que nous avons adopté dans la technique de réduction modulo la $\alpha-$équivalence (Section 4.1). Prenons le même exemple, il est certain que les STEMs S$[x/x, y/y]$ et T$[x/e, y/z]$ de la Figure 2.4 sont deux STEMs égaux.

Relations de bissimulation maximale

Dans[Sai96], deux relations de bissimulation sont définies, à savoir la relation de bissimulation maximale forte et la relation de bissimulation maximale faible, sont deux relations compatible avec le rafinement d'actions. La première relation est présentée dans cette section.

Définition 2.10 *Soit* $\mathfrak{R} \subseteq STEM \times STEM \times \mathfrak{F}$ *une relation entre STEMs, et soit* $\mathbb{F}_m$ *:*

Rel(STEM) $\rightarrow$*Rel(STEM) une fonction définie comme suit :*$(S,T,F) \in \mathbb{F}^m(\mathfrak{R})$ *ssi :*

1. $Dom(f) \subseteq \psi(S)$ *et* $Cod(f) \subseteq (T)$,
2. *Si* $S \xrightarrow{M} {}^{a_x}_{m} S'$ *, alors il existe* $T \xrightarrow{N} {}^{a_y}_{m} T'$*tel que :*
   - **(a).** *Pour tout* $(u,v) \in f$*, si* $u \notin M$ *alors* $v \notin N$ *; et*
   - **(b).** $(S',T',f') \in \mathfrak{R}$*, avec* $f' = (f\lceil(\psi(S') - \{x\}))\lfloor((T') - \{y\}) \cup \{(x,y)\}$.
3. *Si* $T \xrightarrow{N} {}^{a_y}_{m} T'$ *, alors il existe* $S \xrightarrow{M} {}^{a_x}_{m} S'$*tel que :*
   - **(a).** *Pour tout* $(u,v) \in f$*, si* $v \notin N$ *alors* $u \notin M$ *; et*
   - **(b).** $(S',T',f') \in \mathfrak{R}$*, avec* $f' = (f\lceil(\psi(S') - \{x\}))\lfloor((T') - \{y\}) \cup \{(x,y)\}$.

$\mathcal{R}$ est une relation de bissimulation maximale forte ssi $\mathfrak{R} \in \mathbb{F}^m$ $(\mathfrak{R})$. Si $(S\iota, T\iota, F) \in \mathfrak{R}$ pour une certaine relation de bissimulation maximale forte $\mathfrak{R}$, alors les STEMs S et T sont dits fortement maximalement bissimilaires, ce qui est noté symboliquement par $S \sim_m T$.

- Dans la définition précédente STEM dénote l'ensemble des systèmes de transitions étiquetées maximales, $\mathfrak{F}$ dénote l'ensemble des fonctions ayant pour domaine et codomaine des sous ensembles des évènements maximaux.
- $\mathbb{F}^m$ est une fonction qui a pour domaine et codomaine des ensembles de relations entre STEMs. Elle est utilisée pour la définition par point fixe de la relation de bissimulation maximale.
- La condition (a) stipule que si deux évènements maximaux u et v sont reliés par la fonction f, $(u,v) \in f$; alors si u peut rester maximal dans le futur, alors v doit aussi avoir une possibilité de rester maximal dans le futur. C'est cette condition qui préserve la maximalité .
- La condition (b) montre que f' doit être une extension de f, on enlève x du domaine def', y du codomaine de f' et on ajoute le couple (x,y) à f' .
- L'équivalence de bissimulation maximale est une relation réflexive, symétrique et transitive. L'union de plusieurs relations de bissimulations maximales est une bissimulation maximale. L'union de toutes les bissimulations maximales qui existent entre deux arbres maximaux est le point fixe de $\mathbb{F}^m$.

En appliquant la définition 2.10, nous prouvons que les deus STEMs S et T de la Figure 2.5 sont fortement maximalement bissimulaire, où :
$\mathfrak{R} = \{(1,1',\varnothing),(2,2',\{(x,x)\}),(3,3',\{(x,x),(y,y)\}),(4,3',\{(y,x)\})\}$.

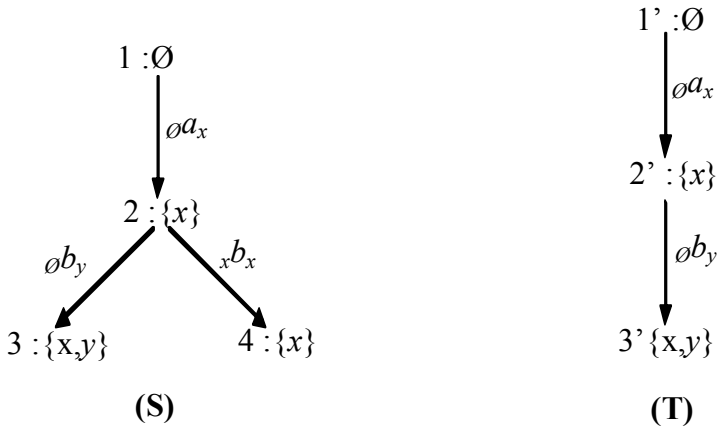


FIG. 2.5 – S et T sont fortement maximalement bissimulaire

2.3 Conclusion

Dans ce chapitre, nous avons présenté l'intuition derrière la sémantique de maximalité et nous avons présenté aussi les systèmes de transitions étiquetées maximales. Le modèle des systèmes de transitions étiquetées maximales peut être utilisé comme une représentation sémantique du comportement du système étudié, d'où la possibilité d'utiliser différents modèles de spécification ; pour cela, il suffit de définir la sémantique en terme de STEMs pour chacun de ces modèles.

Les systèmes de transitions étiquetées maximales sont des modèles de vrai parallélisme. En échappant à l'hypothèse d'atomicité spatiale et temporelle des actions, les transitions de ce modèle sont considérées comme des événements représentant que le début de l'exécution des actions, et l'ensemble d'événements maximaux associé à chaque état correspond aux actions qui sont potentiellement en cours d'exécution. Cela rendre possible d'une part, l'introduction de la sémantique de maximalité aux méthodologies de conception basées sur le raffinement d'actions[CS95] [Sai96], et dans une autre part, la prise en compte du temps[SB05] [BS05a].

En plus, l'ensemble d'événements maximaux associé à chaque état représente explicitement les relations d'indépendance « exactes » entre les actions, en inspirant de méthodes de réduction du graphe d'états basées sur l'approche d'ordre partiel, on peut regrouper dans un seul pas, sous certaines conditions, les actions qui correspondent à cet ensemble, ce qui est effectivement réalisé au niveau de la section 4.2 .

Chapitre 3

Méthodes d'ordre partiel pour la réduction du graphe d'état

Les approches d'ordre partiel résoudrent le problème de l'explosion combinatoire du graphe d'état en éliminant l'une de ses causes, à savoir la représentation du parallélisme par l'entrelacement des actions.

Dans le cas général, l'exécution parallèle de deux (n) actions est représentée par un losange (hybercube) où les différents chemins convergent vers un même état (Figure 3.2.(a)). Grâce aux travaux de Mazurkiewicz [Maz86], il est possible de définir une relation d'équivalence entre ces chemins, au but de n'explorer qu'un chemin particulier parmi eux, ceci définit l'idée directrice des approches d'ordre partiel dont la réduction est modulo cette relation d'équivalence. Informellement, deux chemins sont équivalents ssi ils peuvent être obtenus l'un à partir de l'autre par une suite de permutations d'actions adjacentes et **indépendantes**.

Relation d'indépendance

On peut définir la relation d'indépendance soit directement sur les système de transitions étiquetées [GW93] ou soit en travaillant directement sur la description du système et sur la sémantique opérationnelle du formalisme dans lequel elle est écrite.

Définition 3.1 *"Système de Transitions Etiquetées (STE)"*

Un STE est un quadruplet $< S, s_0, T, \longrightarrow >$ *où* S *est un ensemble d'états,* s_0 *un état initial.* T *un ensemble de labels de transitions, et* $\longrightarrow$ *est un ensemble de transitions* ($\longrightarrow \subset S \times T \times S$). *On note :*

$s \longrightarrow$ *ssi* $\exists s' \in S : (s, t, s') \in \longrightarrow$

$s \not\longrightarrow$ *ssi non* $(s \longrightarrow)$

$s \xrightarrow{t} s'$ *pour signifier que* $(s, t, s') \in \longrightarrow$

$s_0 \stackrel{w}{\Longrightarrow} s_n$ *ssi* $w = t_1.t_2...t_n$ *et* $s_0 \xrightarrow{t_1} s_1$, $s_1 \xrightarrow{t_2} s_2$, ..., $s_{n-1} \xrightarrow{t_n} s_n$

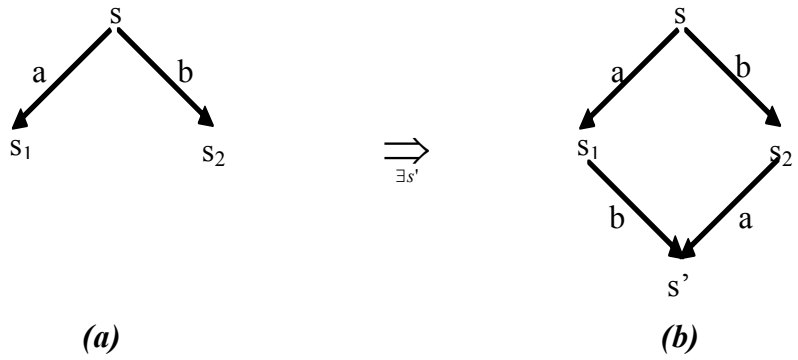


FIG. 3.1 – Deux transition indépendantes

Définition 3.2 *"Relation d'indépendance dans les STE"*

Soit un STE $\sum =< S, s_0, T, \longrightarrow >$ *et* $\imath$ *une relation binaire sur l'ensemble des transitions de* $\sum (\imath \subset T \times T)$: $\imath$ *est une relation d'indépendance pour* $\sum$ *ssi :* $\forall s, s_1, s_2 \in S, \forall a, b \in T$: $\Big[a \neq b, s \xrightarrow{a} s_1, s \xrightarrow{b} s_2$ *et* $a \ \imath \ b\Big] \Longrightarrow \exists s' \in S : s_1 \xrightarrow{b} s'$ *et* $s_2 \xrightarrow{a} s'$

Si deux transitions indépendantes a et b sont possibles en s (Figure 3.1.(a)), alors si $s \xrightarrow{a} s_1$ (resp $s \xrightarrow{b} s_2$) alors b est possible en s_1(resp a est possible en s_2), de plus il existe un unique état s' vérifiant à la fois $s_1 \xrightarrow{b} s'$ et $s_2 \xrightarrow{a} s'$ (Figure 3.1.(b)).

La relation complémentaire de $\imath$ est appelée relation de conflit ou de dépendance. On la note # ou ($\imath^c$).

Définition 3.3 *"Approximation structurelle de* $\imath$ *dans les réseaux de Petri[Rei85]"*

Soit un Réseau $R =< P, T, Pre, Post >$ *(Place/Transition),* m_0 *un marquage initial et* $G(R, m_0)$ *le graphe des marquages accessible associé au réseau marqué* $< R, m_0 >$*. Pour chaque transition* $t \in T$*, on définit :*

${}^*t =_{def} \{p \in P : \Pr e(p,t) \neq 0\}$, $t^* =_{def} \{p \in P : Post(p,t) \neq 0\}$ *et* ${}^*t^* =_{def} t^* \cup^* t$

et soit $\imath_{P/T} \subset T \times T$ *définie par* $t_1 \ \imath_{P/T} t_2$ *ssi* ${}^*t_1 \cap^* t_2 = \varnothing$

Pour $< R, m_0 >$ un réseau Pl/Tr marqué : $\imath_{P/T}$ est une relation d'indépendance pour $G(R, m_0)$

La notion d'indépendance vue dans la définition 3.2, est directement définie sur le STE. Et comme le but des approches d'ordre partiel est d'éviter la construction des STEs, l'approximation structurelle de $\imath$ est la plus convenable d'être utilisée.

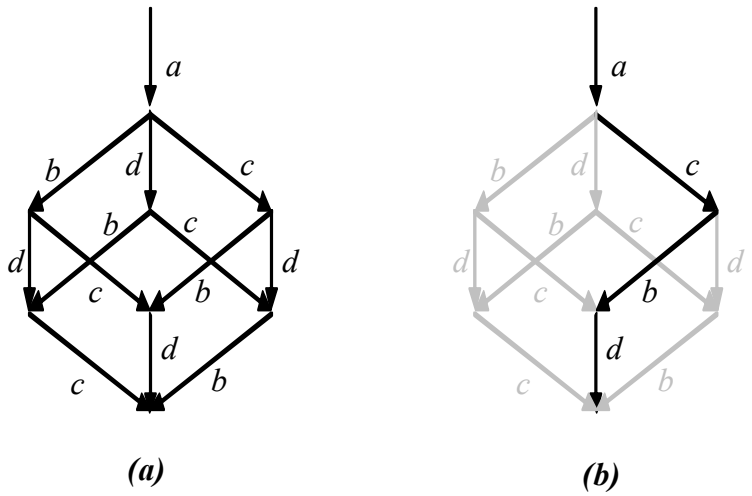


FIG. 3.2 – Traces de Mazurkiewicz

Traces de Mazurkiewicz

Définition 3.4 *"Alphabet concurrent"*

Un alphabet concurrent est un couple $\varepsilon = (\alpha, \#)$ *où* α *est un alphabet et* $\#$ *une relation de dépendance dans* α ($\#$ *est réflixive et symétrique*). *La relation complémentaire* $\#^c$ *est applée relation d'indépendance dans* α. *La fermeture réflexive et transitive de la relation de conflit est notée* $[\#]$.

Définition 3.5 *"Transition libre"*

Une transition t est dite libre si elle n'est en conflit avec aucune autre transition qu'elle même

t libre ssi $[\#]t = \{t\}$ *où* $[\#](t) = \{t' \in T : t[\#]t'\}$

Définition 3.6 *"Traces de Mazurkiewicz"*

Pour un alphabet concurrent $\varepsilon = (\alpha, \#)$, *on considère* $\approx_\varepsilon$*la relation définie sur* $\alpha^* \times \alpha^*$ *par* $U.a.b.V \approx_\varepsilon U.b.a.V$ *ssi* $(a, b) \in \#^c$. *Par construction,* $\approx_\varepsilon$*est réflexive et symétrique.*

L'équivalence de traces de Mazurkiewicz, notée $\equiv_\varepsilon$, peut être définie comme la fermeture transitive de la relation $\approx_\varepsilon$, ses classes d'équivalence sont appelées des traces sur ε. $[w]_\varepsilon$ dénote la trace sur ε générée par le mot w.

Propriété 3.1 *"Fondement des approches d'ordre partiel"*

Pour un STE $\sum$, $\imath$ *une relation d'indépandance sur* $\sum$ *et l'alphabet concurret* $(T, \imath^c)$:
$\forall s \in S,\ w_1, w_2 \in T^* : (s \stackrel{w_1}{\Longrightarrow} s_1\ \&\ s \stackrel{w_2}{\Longrightarrow} s_2\ et\ [\ w_1]_\varepsilon = [w_2]_\varepsilon) \Longrightarrow s_1 = s_2$

La propriété 3.1 exprime que les chemins possédant la même trace conduisent au même état. Donc on peut sous certaines conditions réduire l'explosion combinatoire du graphe d'états en ne prenant en compte qu'un seul chemins par trace. Sur cette propriété qu'ont été fondues les approches d'ordre partiel.

Pour expliquer cela, prenons par exemple le graphe de la Figure 3.2.(a), ce graphe représente le comportement de l'expression « $a; (b; stop|||c; stop|||d; stop)$ » ; les séquences $[a.b.c.d]$, $[a.b.d.c]$, $[a.c.b.d]$, $[a.c.d.b]$, $[a.d.b.c]$ et $[a.d.c.b]$ sont équivalentes ; donc le graphe réduit se peut être une de ces séquences (Figure 3.2.(b)).

Mais, le problème est loin d'être résolu, en effet, selon cette restriction, rien ne permet d'assurer que les états intermédiaires constituant les chemins équivalents soient eux même équivalents (selon la propriété que l'on souhaite vérifier). Cette situation est appelée situation de confusion « conflit ». Pour mieux expliquer ce problème, prenant l'exemple de la Figure 1.4, les transitions a et b sont indépendantes ($^*a \cap^* b = \varnothing$), ce qui nous permet d'écrire $a.b \approx_\varepsilon b.a$. Prendre arbitrairement un de ces deux chemins n'est pas évident, puisque, les états qui constituent ces deux chemins ne sont pas équivalents du point de vu la possibilité de blocage. La réduction donc, ne peut être effectuée que sous certaines conditions qui nous assurent les propriétés à vérifier. De multiples travaux sont proposés en vue de préciser ces conditions, en général, il existe deux types d'approches d'ordre partiel :

- Approches par choix : au niveau de ce type d'approches, on ne choisit et on n'exploite qu'un sous-ensemble des entrelacements possibles. Parmi ces ensembles on peut citer les ensembles persistants [God90], dormants [God90], proviso[God90][WG93], amples [Pel98],... etc.
- Approches par pas : ces approches construisent des pas qui sont des ensembles de transitions que l'on peut tirer simultanément afin d'éviter toutes les combinaisons d'entrelacement possibles des transitions du pas. On peut citer les graphes de pas couvrants [VAM96] et SRA (Simultaneous Reachability Analysis) [Wes86].

Dans ce qui suit, nous présenterons deux approches complémentaires : **les graphes persistants** et **les graphes de pas couvrants**, cette présentation évoque seulement les versions qui préservent uniquement les propriétés d'accessibilité générales à savoir les états de blocage et la vivacité. L'approche hybridée de ces deux là est connue sous le nom de « **les graphes de pas persistants** ». Et pour compléter ce chapitre, l'introduction de l'outil `Tina`[TIN] est donnée.

3.1 Graphes persistants

Les ensembles persistants sont un cas particulier des ensembles stubborns [Val88]. L'exploration standard par l'ensembles persistants préserve uniquement l'états de blocage, de multiples extension ont été proposées dans le but de préserver d'autres classes de propriétés [God90][GW93][WG93].

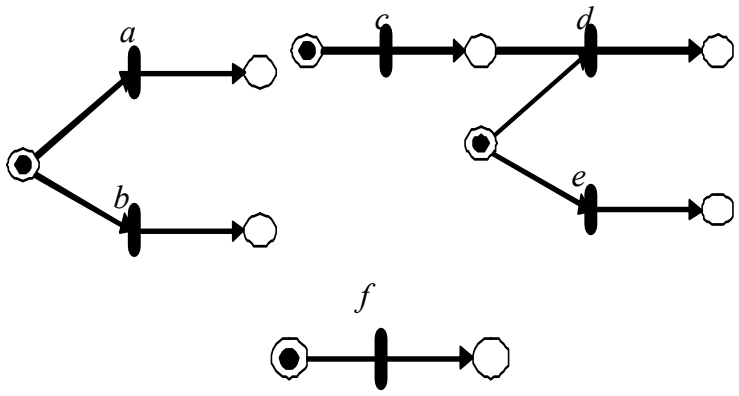


FIG. 3.3 – Réseau de Petri d'étude

Informellement, un ensemble d'actions T est persistant dans un état s ssi toute action sur une trace, partant de s et composée d'action hors de T, est indépendante de toute action dans T.

Définition 3.7 *"Ensemble persistant"*
Un ensemble P de transitions est persistant dans un état s ssi
$\forall t \in P : s \xrightarrow{t}$ *et* $\forall w \in \bar{P}^* : s \stackrel{w}{\Longrightarrow} s' \Longrightarrow w \imath P$
avec $\bar{P} = T \backslash P$ *et* $\forall t_1 \in w$ *et* $\forall t_2 \in P, t_1 \imath t_2$

Propriété 3.2 *[God90] si $[\#]t \in enabled(s)$, alors $[\#]t$ est persistant dans s.*

La propriété 3.2 exprime la possibilité de dériver un ensemble persistant directement de la description formelle du système. Cela permet de déduire aisément un algorithme de construction de graphe persistant (dénoté GP) directement à partir de la description formelle. Effectivement, en ajoutant la règle suivante à n'importe quel algorithme d'énumération classique (Algorithme 3.1) : pour chaque état atteint s, calculer un ensemble persistant associe à s et n'exécuter que les actions dans cet ensemble. Précisément, nous pouvons obtenir l'algorithme de GP en ajoutant, après la ligne α de l'algorithme 3.1, l'instruction $T \longleftarrow Persistant(T)$, où $Persistant(T)$ est la fonction qui choisit un ensemble persistant dans s.

Pour donner une idée sur le taux de réduction obtenu, prenant le système de la Figure 3.3. Le graphe de comportement exhaustif de ce système comporte 30 états et 65 transitions avec 4 états de blocage, en revanche, son graphe persistant comporte 21 états et 27 transitions.

Selon le système de la Figure 3.3 , on peut écrire $[\#]d = [\#]e = \{d, e\}, [\#]a = [\#]b = \{a, b\}, [\#]f = \{f\}$et $[\#]c = \{c\}$. Et comme $Enabled(s_0) = \{a, b, f, c, e\}$(actions sensibilisées dans s_0) alors les ensembles $\{a, b\}, \{f\}$ et $\{c\}$ sont persistants. La question maintenant est : *quel est l'ensemble persistant à prendre ?*.

Le calcul d'ensemble persistant est local, en conséquence, l'influence réelle sur la taille globale du graphe n'est pas connue. Donc il est impossible de savoir localement quel est le

meilleur ensemble persistant. Différentes heuristiques ont été proposées [God90][Val89][Ove81] pour choisir un ensemble persistant. Une stratégie fréquemment utilisée est de prendre le plus petit ensemble persistant que l'on sache calculer. Cette instance du graphe persistant est dénotée GP_{min}. En plus du problème du bon choix, nous pouvons avoir pour le même critère de choix, plusieurs ensembles persistants qui le satisfait " $\{f\}$ et $\{c\}$ de l'exemple précédent", cela conduit à l'indéterminisme, construire des graphes différents pour le même modèle si on exécute le même algorithme plusieurs fois [Rib05].

Algorithme 3.1 *"Algorithme général de construction du graphe d'états"*

$Pile \longleftarrow s_0$
$S \longleftarrow \{s_0\}$
$Graphe \longleftarrow \varnothing$
Tantque *Pile non vide* ***Faire***
  $s \longleftarrow dépiler(Pile)$
  $T \longleftarrow enabled(s).....(\alpha)$
// *Développement Exhaustive*
  Pourchaque $(s_1, a, s_2) \in T$ ***Faire***
    $Graphe \longleftarrow (s_1, a, s_2)$
    Si $s_2 \notin S$ ***alors***
      $S \longleftarrow S \cup \{s_2\}$
      $empliler(s_2, Pile)$
    FinSi
  FinPourChaque
Fintantque
FinAlgo

Où *Pile* contient l'ensemble des états à développer, et S contient l'ensemble d'états déjà développés. *Graphe* contient le STE et (s_1, a, s_2) une transitions.

Description *Tant qu'il y a un état s à développer, on construit l'ensemble « T » de transitions sensibilisées dans s. Et pour chaque transition (s_1, a, s_2) de cet ensemble, on ajout cette transition à Graphe et on ajout aussi s_2 à S et à Pile si et seulement si s_2 est un nouveau état.*

3.2 Graphes de pas couvrants

Comme dans l'entrelacement de n événements, l'ordre d'occurrence de chaque événement dans la séquence est arbitraire, et ainsi les états intermédiaires qui la constituent sont ausi arbitraires, l'approche des graphes de pas couvrants, introduite dans [VAM96], représente sous certaines conditions, le parallélisme de n événements par un seul pas de calcul (Figure 3.4.(b)), dont la réalisation est un événement atomique.

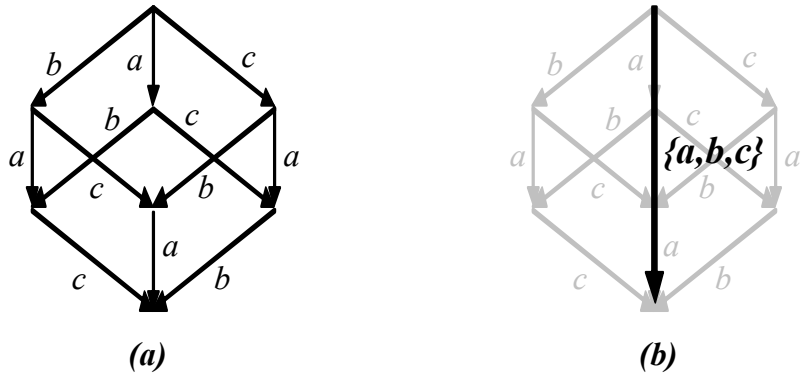


FIG. 3.4 – Dérivation de 3 actions indépendantes

Le graphe de pas couvrant est un modèle qui couvre, modulo l'équivalence de traces de Mazurckiewicz, le graphe initial. L'approche de pas couvrants est une technique assez générale qui peut être déclinée suivant le type de propriétés que l'on cherche à vérifier. Effectivement, la version de graphe de pas couvrants présentée dans cette section préserve uniquement les états de blocage et la propriété de vivacité[VAM96] ; plusieurs autres versions ont été proposées pour préserver l'équivalence observationnelle[VAM96], et la sémantique de refus[VM97].

Avant de donner la définition de graphes de pas couvrants et l'algorithme de construction associé, nous présentons les notions de base utilisées dans cette approche.

Définition 3.8 *"Pas de transition"*
Un ensemble de transitions E définit un pas de transitions pour $\imath$ ssi $\forall t_1, t_2 \in E : t_1 \ \imath \ t_2$

A titre d'exemple, les transitions a, b et c de la Figure 3.4 forment un pas de transitions.
Pour la suite, $Step(T, \imath)$ désigne l'ensemble de tous les pas de transitions sur T pour $\imath$.

Définition 3.9 $\forall E \in Step(T, \imath), F \subset E \Longrightarrow F \in Step(T, \imath)$

Définition 3.10 *"Ensemble de séquences associées à un pas de transitions"*
Soit Seq l'application de $\rho(T) \longmapsto \rho(T)$ *définie par :*

- $Seq(\varnothing) =_{def} \{\varnothing\}$
- $Seq(E) =_{def} \bigcup_{e \in E} Seq(E \backslash \{e\}) \otimes e$

où pour $\Omega \in \rho(T^*)$ et $w \in T^* : \Omega \otimes w =_{def} \{\omega.w : \omega \in \Omega\}$

Propriété 3.3 $\forall E \in Step(T, \imath), \omega_1$ *et* $\omega_2 \in Seq(E) \Longrightarrow [\omega_1]_{(T, \imath^c)} = [\omega_2]_{(T, \imath^c)}$

Prenant le même exemple, nous trouvons que
$Seq(\{a, b, c\}) = \{a.b.c; a.c.b; b.a.c; b.c.a; c.a.b; c.b.a\}$, et $[a.b.c] = [a.c.b] = [b.a.c] = [b.c.a] = [c.a.b] = [c.b.a]$

Définition 3.11 *"Extention de la relation d'accessiilité initiale aux pas de transitions"*
Soit $\twoheadrightarrow$, *l'extension de* $\longrightarrow$ *aux pas de transitions.*$\forall s, s' \in S, \forall E \in Step(T)\ s \overset{E}{\twoheadrightarrow} s'$ *ssi* $\forall \omega \in Seq(E)\ \ s \overset{\omega}{\Longrightarrow} s'$.

Propriété 3.4 $s \overset{E}{\twoheadrightarrow}$ *ssi* $\forall e \in E : s \xrightarrow{e}$

Définition 3.12 *"Graphe de pas couvrant[VAM96]"*
Soit un STE $\Sigma =< S, s_0, T, \longrightarrow >$, $\imath$ *une relation d'indépendance pour* Σ *(on note* $\#=\imath^c$*),* $\Sigma_p =< S_p, s_0, \Pi, \longrightarrow_c >$ *est un graphe de pas couvrants de* Σ *pour* $\imath$ *ssi :*

1. $S_p \subset S$
2. $\Pi \subset Step(T, \imath)$
3. $\exists E \in \Pi, \forall s, s' \in S_p : s \xrightarrow{E}_c s' \Longrightarrow s \overset{E}{\twoheadrightarrow} s'$
4. $\forall s \in S_p, \forall \omega \in T^*, \forall s' \in S : s \overset{\omega}{\Longrightarrow} s' \Longrightarrow \left\{ \begin{array}{c} \exists s'' \in S_p, \exists \omega' \in T^*, \exists P_{\omega.\omega'} \in \Pi^* \\ s' \overset{\omega'}{\Longrightarrow} s'', s \overset{P_{\omega.\omega'}}{\longrightarrow}_c \text{ et } [\omega.\omega']_{(T,\#)} = \lfloor P_{\omega.\omega'} \rfloor_{(T,\#)} \end{array} \right.$

où :

1. impose que tout état du GPC est un état du graphe initial,
2. chaque sous-ensemble de transitions consiste un pas de transitions,
3. chaque séquence associée à un pas de transitions dans GPC correspond à une séquence de transitions du graphe initial
4. toute séquence d'actions du graphe initial est préfixe (couverte) d'une séquence de pas possédant la même trace dans le GPC.

Exemple 3.1 *"Ensemble de conflit indépendants"*
Le réseau de Petri de la Figure 3.5.(a) présente un ensemble de conflits indépendants, la Figure 3.5.(b) est son STE associé et la Figure 3.5.(c) est le GPC calculé.

3.2.1 Algorithme général de construction du graphe de pas couvrants

Cette section présente un schéma général d'algorithme (Algorithme 3.2) pour la dérivation des graphes de pas couvrants [VAM96]. Comme dans le cas des graphes persistants, la génération des graphes de pas couvrants directement de la description formelle du système est possible .

Le calcul consiste, pour un état donné s, à partitionner les transitions sensibilisées en deux parties, le premier ensemble T_u contient les transitions qui seront explorées de façon classique et l'autre ensemble T_m contient celles qui seront explorées par pas. Cette partition est conditionnée par quatre exigences :

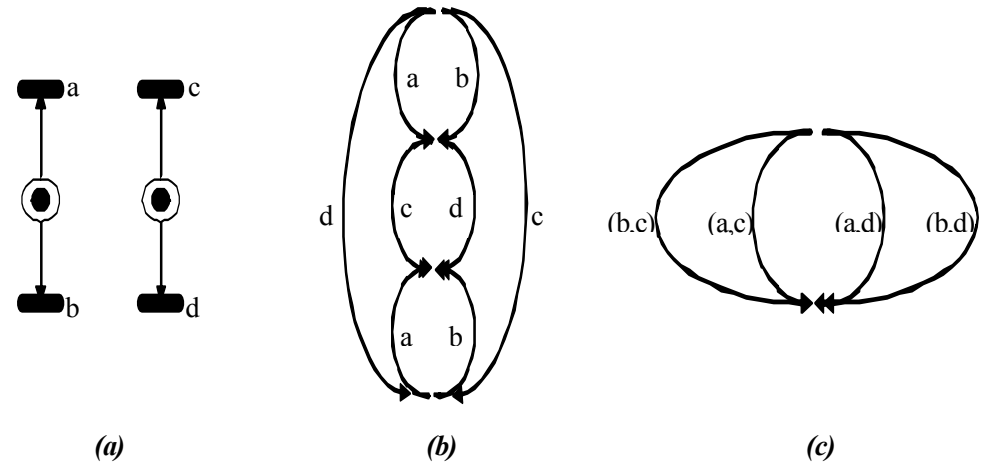


FIG. 3.5 – Le Graphe de pas couvrants

- $T_m \cup T_u = Enabled(s)$ et $T_m \cap T_u = \varnothing$: impose que T_u et T_m forment une partition de l'ensemble des transitions sensibilisées dans l'états s.
- si $t \in T_m$ alors $t'[\#]t \Longrightarrow t' \in Enabled(s)$: Exprime que toute transition en conflit avec une transition fusionnable doit être elle-même sensibilisée.
- $\forall p \in \Lambda_{T_m} : p \in Step(Enabled(s), [\#]^C)$: Assure que les pas de transitions du graphe couvrant sont aussi des pas pour la relation $[\#]$.
- $\forall p \in Step(Enabled(s), [\#]^C), \exists p' \in \Lambda_{T_m} : p \subseteq p'$: Montre que tout pas de transition pour $[\#]$ est couvert par un pas de transition appartenant à Λ_{T_m}.

La justification du choix de ces conditions est donnée dans [VAM96].

Construction de pas

La fonction cherche Λ_C à définir un sous-ensembles constitués d'événements indépendants recouvrant l'ensemble T_m. L'ensemble T_m est défini de façon qu'il doit être cohérent avec la relation de conflit. Pour ce faire, les deux fonctions T_U et T_M sont définies comme suit :

- $T_M(T, \imath) = \{t \in T : t'[\#]t \Longrightarrow t' \in T\}$
- $T_U(T, \imath) = T \backslash T_M(T, \imath)$

Pour obtenir des pas, il suffit ensuite d'effectuer le produit croisé Λ_C sur la partition $T_M(T, \imath)$, L'ensemble de pas construit $\Lambda_C(T_m, \imath)$ connu sous le nom de conflits croisés. Dans [VAM96], ont preuvé que ce type de pas assure les quatre conditions citées ci-dessus.

où $E = \{E_1, E_2, ...E_n\}$ et

$\Lambda_C(E) =_{def} \{\{e_1, e_2, ...e_n\} : (e_1, e_2, ...e_n) \in E_1 \times E_2 \times ... \times E_n\}$

Par exemple, dans la Figure 3.5.(a), $a[\#]b$ et $c[\#]d$, ce qui nous permet de trouver au niveau de l'état initial $s_0 : T_M(\{a, b, c, d\}, \imath) = \{\{a, b\}, \{c, d\}\}$ où $Enabled(s_0) = \{a, b, c, d\}$. donc l'application de Λ_C sur l'ensemble $\{\{a, b\}, \{c, d\}\}$ donne les pas suivants $\{\{a, b\}$, $\{a, d\}$, $\{b, c\}$, $\{b, d\}\}$ (Figure 3.5.(c)).

Algorithme 3.2 *"Algorithme général de construction du graphe de pas couvrants"*

Pile $\longleftarrow s_0$
$S \longleftarrow \{s_0\}$
Graphe $\longleftarrow \varnothing$
Tantque *Pile non vide* ***Faire***
  $s \longleftarrow$ *dépiler*$(Pile)$
  $T \longleftarrow$ *enabled*(s)
// Partition de l'ensemble des transitions senibilisées
  $T_m \longleftarrow T_M(T, \imath)$
  $T_u \longleftarrow T_U(T, \imath)$
// Développement Exhaustive
  Pourchaque $(s_1, a, s_2) \in T_u$ ***Faire***
    Graphe $\longleftarrow (s_1, a, s_2)$
    Si $s_2 \notin S$ ***alors***
      $S \longleftarrow S \cup \{s_2\}$
      empliler$(s_2, Pile)$
    FinSi
  FinPourChaque
// Développement par pas
  $\Lambda_{T_m} \longleftarrow \Lambda_C(T_m, \imath)$ *// Construction des pas*
  Pourchaque $p \in \Lambda_{T_m}$ ***Faire***
    Graphe $\longleftarrow (s_1, p, s_3)$ *// où* $s_1 \overset{p}{\twoheadrightarrow} s_3$
    Si $s_3 \notin S$ ***alors***
      $S \longleftarrow S \cup \{s_3\}$
      empliler$(s_3, Pile)$
    FinSi
  FinPourChaque
Fintantque
FinAlgo

Modèle	Exhaustif	GP_{min}	GPC	GPP
Scheduler 300	$2^n * n = 6.10^{92}$	1394	301	301
Philosophe 8	103681	233	31231	227

TAB. 3.1 – *Evaluation des GPs et GPCs*

3.3 Graphes de pas persistants

La table 3.1 donne quelques exemples qui montrent que parfois le GPC est beaucoup plus performant (cas de Scheduler), alors que sur d'autres exemples le graphe persistant parvient à construire un graphe plus petit (cas des Philosophes). L'idée présentée dans cette section, est de faire collaborer les graphes persistants et les graphes de pas couvrants pour en cumuler les bénéfices [Rib05]. Les graphes obtenus sont appelés graphes de pas persistants et noté GPP.

Le bénéfice qui peut être obtenu n'est qu'au niveau de taux de réduction (Table 3.1), en effet, cette collaboration affaiblit les graphes de pas couvrants, en d'autre terme le graphe de pas persistants ne préserve que les états de blocage [Rib05].

Le GPC utilise l'indépendance entre les transitions pour ne faire qu'un seul pas entre l'état de départ et l'état d'arrivée, les ensembles persistants sont employées pour ne pas tirer toutes les transitions sensibilisées mais uniquement un sous-ensemble de celles-ci. L'idée de l'approche de pas persistants est d'utiliser les ensembles persistants pour ne pas tirer toutes les transitions et les pas pour regrouper les transitions que l'on a décidé de tirer.

Propriété 3.5

- *Les explorations GPP conservent les blocages.*

- *GPP généralise GP.*

3.4 L'outil `Tina`

Dans le cadre de la réduction basée sur l'approche d'ordre partiel, plusieurs outils logiciels ont été développés, on peut citer par exemple trois logiciels (distribution libre) :

- `EXP.OPEN 2.0`, un outil de la boîte à outils `CADP`, Cette boîte a été développé par l'équipe VASY de l'INRIA Rhônes-Alpes. `EXP.OPEN 2.0` est disponible depuis *Août 2004* à l'adresse "http ://www.inrialpes.fr/Vasy/Cadp".

- `SPIN`, c'est une propriété de ULg (Université de Liège.USA) *1992-1994*, la version *4.2.6* est disponible depuis le *27 Octobre 2005* à l'adresse " http ://Spinroot.com/Spin".

- `Tina`, il est disponible sur internet à l'adresse "http ://www.laas.fr/tina". Nous avons utilisé dans cette section la version *2.7.4* (disponible depuis le *18 Juin 2005*)

Cette section présente un environnement de vérification formelle développé par LAAS/CNRS, connu sous le nom de `Tina` (TIme Petri Net Analyser).

`Tina` est un outil logiciel permettant l'édition et l'analyse de réseaux de Petri et Temporels [MF76], il offre les fonctions classiques d'édition et d'analyse énumérative (les graphes de marquages, les arbres de couverture) ou structurelle (semi-flots).

`Tina` n'est pas un model-checker au sens où il ne permet pas de vérifier la satisfaction d'une certaine propriété, sauf les propriétés générales d'accessibilité (le blocage et la vivacité). En effet, `Tina` fournit un graphe réduit en utilisant les approches présentées dans ce chapitre, `Tina` peut intervenir en amont du model-checher pour l'aider à effectuer plus efficacement la vérification. Effectivement, `Tina` est couplé avec `MEC` [Arn92] pour la vérification de formules du $\mu-$calcul et avec `ALdebaran` [FM91] pour la vérification de pré-ordres ou d'équivalences de comportement.

`Tina` n'impose l'utilisation d'aucun model-checher spécifique, car `Tina` produit des résultats dans divers formats, ainsi, l'extension de cet ensemble de formats est possible, il suffit de développer un modèle spécifique traduisant les formats existants au format en question.

`Tina` est une boîte à outils qui contient trois logiciels qui peuvent être utilisés de façon combinée ou indépendante :

1. Editeur graphique, fournit des fonctions de dessin de réseaux de Petri et d'automates.
2. Outil de construction de comportement et d'analyse structurelle
3. Outil de simulation de réseaux de Petri et de Petri Temporels, il est très utile pédagogiquement .

Cette variété de construction, permet à `Tina` de se positionner dans l'enseignement et également dans l'industrie, à titre d'exemple, `Tina` est figuré parmi les environnement de vérification retenus dans le cadre du projet RNTL COTRE (Composants Temps Réels http ://www.laas.fr/cotre).

Description "COTRE" : *L'objectif de ce projet était de définir une démarche logiciel pragmatique de modélisation et de validation d'architecture de logiciels temps réel permettant de combiner les approches formelles et semi-formelles dans un processus industriel garantissant la continuité de la phase de conception jusqu'à celle d'implantation du logiciel. COTRE a coûté deux ans de travail (Janvier 2002-Janvier 2004). Les participants à ce projet sont : la fédération Feria (ONERA-DTIM, LAAS et IRIT), EADS, TNI-Valiosys, l'ENST de Bretagne et AIRBUS-France(maître d'oeuvre).*

Exemple 3.2 *Dans cet exemple, nous montrons quelques types de formats en utilisant comme exemple le système de la Figure 3.6, où cette figure représente l'interface de l'éditeur graphique. L'outil de construction de comportement et d'analyse structurelle nous fournit le graphe de*

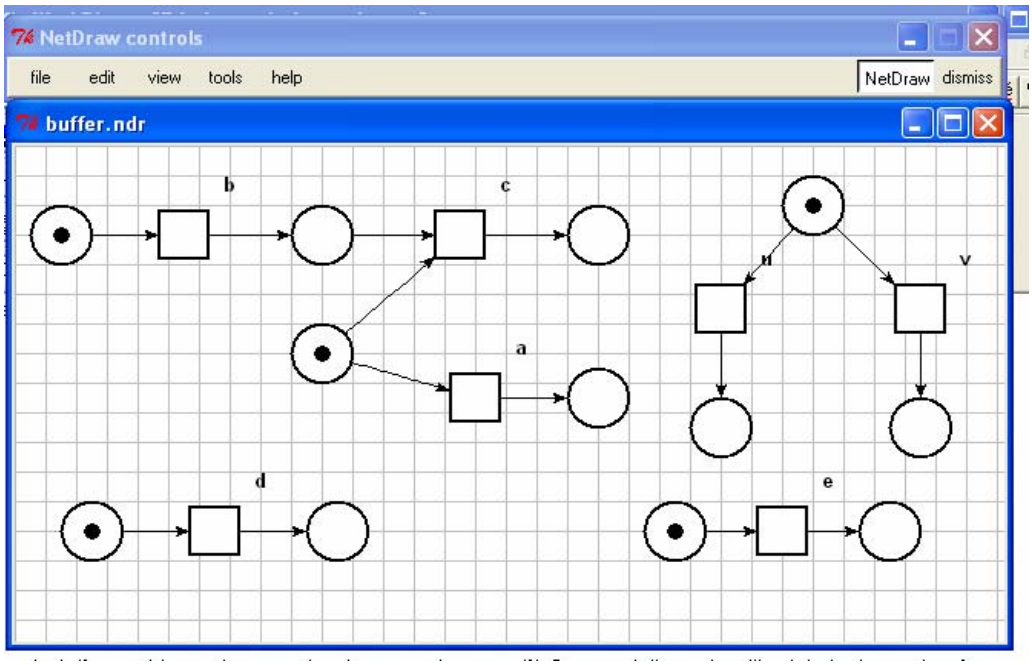
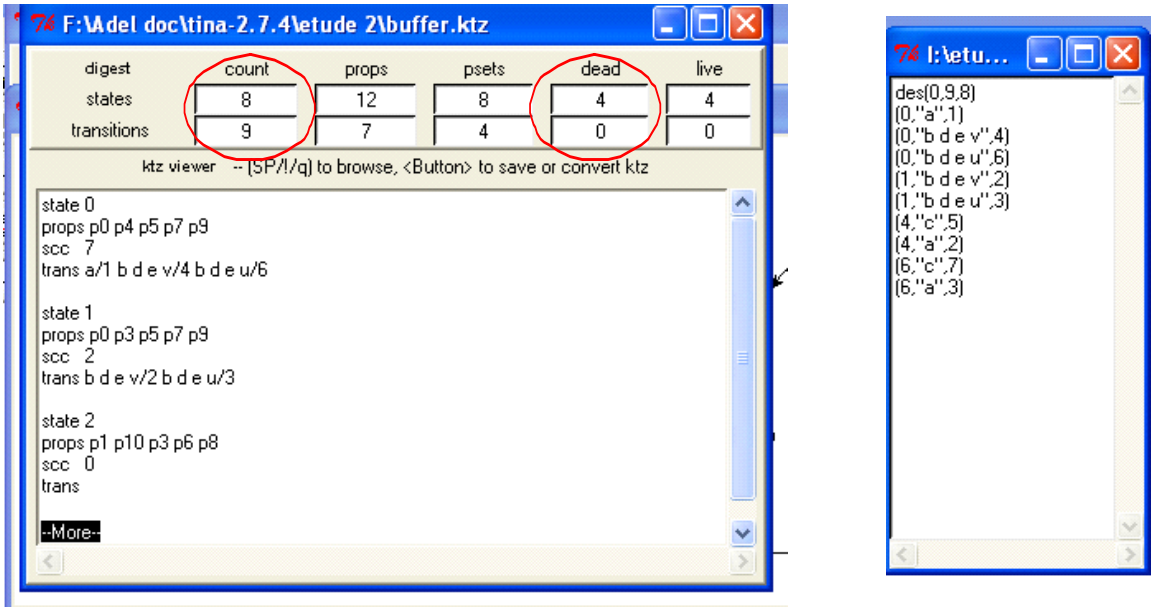


FIG. 3.6 – L'environnement graphique de Tina

*comportement (graphe exhaustif, GPC, GP, ...) en plusieurs formats ; par exemple, la Figure 3.7.(b) est le format textuel du graphe de pas couvrants, la Figure 3.7(a) présente les différentes statistiques sur ce GPC, à savoir le nombre d'états (*08*), le nombre de transitions (*09*) et le nombre d'états de blocage (*04*).*



(a) ***(b)***

FIG. 3.7 – Statistiques sur le graphe de pas couvrants

```
Tina version 2.7.4 -- 05/18/05 -- LAAS/CNRS
mode -P
INPUT NET ----------------------------------------------------

parsed net buffer
12 places, 7 transitions
0.000s

REACHABILITY ANALYSIS ----------------------------------------

possibly bounded
31 marking(s), 39 transition(s)
0.000s + 0.000s

LIVENESS ANALYSIS --------------------------------------------

not live
4 dead marking(s), 4 live marking(s)
0 dead transition(s), 0 live transition(s)
dead marking(s): 12 11 6 5
0.000s
ANALYSIS COMPLETED -------------------------------------------
```

FIG. 3.8 – Analyse accomplie par Tina

3.5 Conclusion

Il existe essentiellement deux types d'approches d'ordre partiel, à savoir les approches par choix (élimination de l'entrelacement) et les approches par pas. Nous avons présenté dans ce chapitre deux approches complémentaires ; la première est une méthode par choix dont le graphe obtenu préserve seulement les états de blocage. En revanche, le graphe de pas obtenu par la deuxième méthode préserve les états de blocage et la propriété de vivacité.

Les résultats expérimentaux obtenus montrent qu'il existe des systèmes où les graphes persistants sont eux qui ont un taux de réduction plus réduit à celui de graphes de pas couvrants, et l'inverse aussi reste vrai. Nous avons présenté une approche qui combine ces deux approches, elle a été introduite au but d'obtenir un taux de réduction le plus réduit de ces deux approches. Mais cela n'est pas toujours assuré, en effet, la taille du graphe obtenu dépend de l'approche et également du système à analysé. Pour montrer ceci, le graphe persistant du système de la Figure 3.3 contient 21 états et 27 transitions, et son graphe de pas couvrant possède 8 états et 9 transitions, normalement et selon les graphes de pas persistants on doit trouver un taux de réduction le plus réduit. Cependant, le GPP obtenu a la même taille que celle du GPC. En d'autres termes, dans ce présent exemple l'approche par pas persistants ne fait qu'affaiblir les propriétés à préserver.

Chapitre 4

Méthodes de réduction proposées

Nous proposons, au niveau de ce chapitre, deux méthodes de réduction : la réduction basée sur la $\alpha-$équivalence et la réduction basée sur l'approche d'ordre partiel. Nous allons proposer aussi l'algorithme de construction à la volée d'un graphe réduit de chacune de ces deux méthodes.

4.1 Systèmes de transitions étiquetées maximales $\alpha-$réduits

Avant d'introduire la α-réduction, il est très important de rappeler les trois points essentiels pour juger la qualité d'une technique de réduction donnée :

- Taux de réduction fort : le rapport taille de graphe complet et la taille de graphe réduit doit être le plus grand possible.

- Capacité d'analyse forte : signifie que l'ensemble des propriétés satisfaites dans le graphe complet et non vérifiables dans le graphe réduit est le plus petit possible.

- Simple à réaliser : deux critères doivent être pris en compte : le temps et l'espace mémoire nécessaires à la construction du graphe réduit par rapport à ceux nécessaires à la construction du graphe complet.

Au cours de cette section, nous montrons que ces trois points sont largement assurés, en effet, un STEM α-réduit vérifie exactement le même ensemble de propriétés vérifié dans le STEM initial ($\approx_\alpha \subseteq \cong$), avec un prix linéaire en temps et en espace mémoire. Effectivement, l'algorithme proposé pour la construction à la volée des STEMs $\alpha-$réduits (Section 4.1.2)est obtenu en ajoutant qu'une simple fonction de substitution à l'algorithme général de construction du graphe d'états. Le taux de réduction de la $\alpha-$réduction est relativement important, ceci sera montré d'une manière expérimentale dans le chapitre 5.

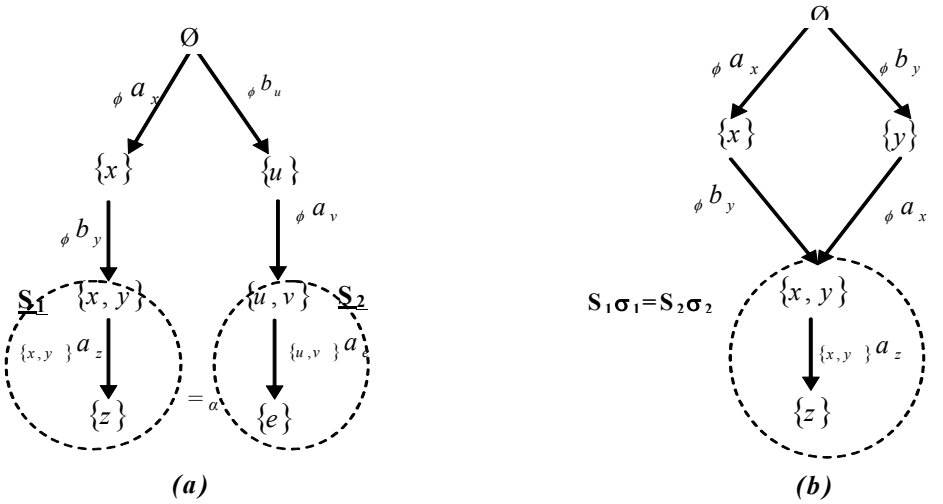


FIG. 4.1 – Réduction modulo α-équivalence

4.1.1 Idée de base

La réduction consiste d'éliminer les redondances modulo certaines relations en préservant les propriétés à vérifier. Dans cette section nous allons utiliser la α-équivalence comme critère de redondance. A titre d'illustration, le STEM de la Figure 4.1.(a) représente le comportement de l'expression "$a; d; stop||[d]|b; d; stop$", il a été généré par l'application directe des règles de la sémantique opérationnelle de maximalité (Définition 2.6). Les deux sous-STEMs S_1 et S_2 de la Figure 4.1.(a) sont α-équivalents, en effet il existe deux fonctions de substitution $\sigma_1 = \{x/x, y/y, z/z\}$ et $\sigma_2 = \{x/v, y/u, z/e\}$ tel que $S_1\sigma_1 \cong S_2\sigma_2$. Pour supprimer une telle redondance, on doit, en premier lieu, substituer le STEM de 4.1.(a) par $\sigma_1 \cup \sigma_2$, et en suite, on supprime $S_1\sigma_1$ ou $S_2\sigma_2$. Le STEM de 4.1.(b) est le STEM α-réduit construit

Ce que nous venons de voir est appliqué à un STEM déjà généré, cependant, notre but est la génération à la volée des STEMs α-réduits. L'alternative consiste de vérifier pour chaque configuration générée si elle est α-équivalente ou non avec une autre configuration, c'est le cas, nous supprimons l'une des deux en substituant le STEM en construction.

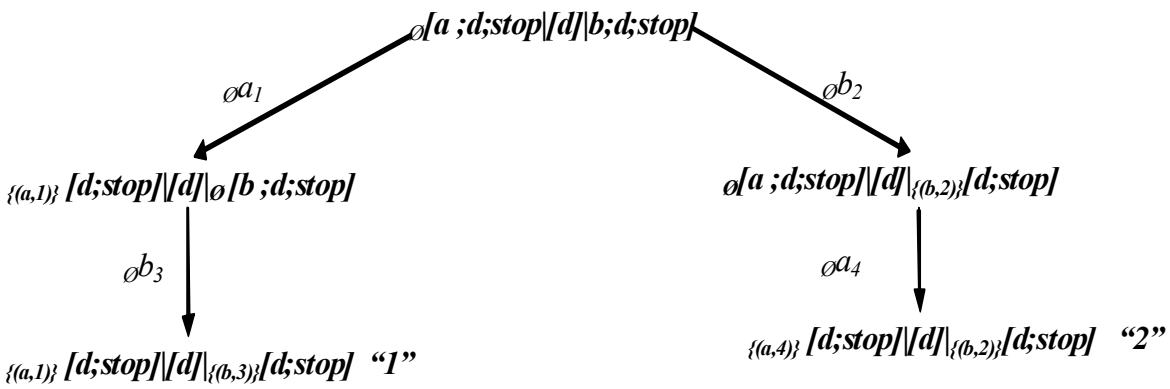


FIG. 4.2 – STEM en cours de génération

Définition 4.1 *" La α-équivalence "*

La α-équivalence est définie récursivement sur les configurations comme suite :

- ${}_M[E] =_\alpha {}_N[E]$, *ssi il existe* σ *où* $M\sigma = N\sigma$
- $\mathcal{E}|[L]|\mathcal{F} =_\alpha \mathcal{E}'|[L]|\mathcal{F}'$, *ssi* $\mathcal{E} =_\alpha \mathcal{E}'$ *et* $\mathcal{F} =_\alpha \mathcal{F}'$
- $\mathcal{E}$ *op*$\mathcal{F} =_\alpha \mathcal{E}$*'op*$\mathcal{F}'$, *ssi* $\mathcal{E} =_\alpha \mathcal{E}'$ *et* $\mathcal{F} =_\alpha \mathcal{F}', op = \{';', '[]', '[>', '>>'\}$
- *hide* L *in* $\mathcal{E} =_\alpha$*hide* L *in* $\mathcal{E}'$, *ssi* $\mathcal{E} =_\alpha \mathcal{E}'$
- ${}_M[P] =_\alpha {}_N[P]$, *ssi il existe* σ *où* $M\sigma = N\sigma$

Dans le reste de cette section et pour le but de simplifier les explications :

- nous avons associé, pour chaque événement maximal, son action appropriée.

- et nous avons considéré l'ensemble $\mathcal{M}$ comme l'ensemble des entiers N .

Dans l'exemple de la Figure 4.2, les configurations 1 et 2 sont α-équivalentes, dont les événements $(a, 1)$ et $(a, 4)$ (resp $(b, 2)$ et $(b, 3)$) représentent le même événement, supposons $(a, 5)$ (resp $(b, 6)$), donc nous pouvons écrire : $\sigma = \{(a, 5)/(a, 1), (a, 5)/(a, 4), (b, 6)/(b, 2), (b, 6)/(b, 3)\}$

4.1.2 Algorithme de construction à la volée d'un STEM$_{/\approx_\alpha}$

L'Algorithme 4.1 nous permet à partir d'une expression de comportement LOTOS donnée, de générer un STEM α-réduit. Le calcul est basé sur les configurations, où pour chaque nouvelle configuration, nous tirons de nouvelles configurations et de nouvelles transitions, en utilisant les règles de la sémantique opérationnelle de maximalité.

Une configuration est dite nouvelle si elle n'est α-équivalente à aucune autre configuration générée précédemment.

Au but de rendre la présentation de l'algorithme simple et intuitive, nous allons procéder par raffinement successif, où à chaque étape nous détaillerons les composants jugés essentiels.

Algorithme 4.1 *"Algorithme de construction à la volée d'un STEM$_{/\approx_\alpha}$ "*

Données : *Spécification LOTOS ;*
Résultats : *Un STEM α-Réduit ;*
Variables :
Liste_Confs : liste de configurations non traitées ;
Liste_Confs_traitéés : liste de configurations déjà traitées ;
Sub : liste de liste des couples (action,événement)
représentant un même événement (la fonction σ)
Début
Construire la configuration initiale ;
Initialiser la liste de configurations Liste_Confs par la configuration initiale ;
Tantque *liste_Confs non vide* ***Faire***
Sub$\longleftarrow \varnothing$
Sélectionner et supprimer un élément Conf de Liste_Confs ;
Traiter_configuration *Conf ;*
Ajouter Conf à la liste Liste_Confs_traitées ;
Ajouter les nouvelles configurations résultantes à Liste_Confs ;
Ajouter les transitions résultantes à STEM ;
Substituer *STEM,Liste_Confs et Liste_Confs_traitées en utilisant Sub ;*
FinTant
FinAlgo.

Description : Tant qu'il y a une configuration Conf (état s) à développer, on construit l'ensemble de transitions maximales sensibilisées dans s (**Traiter_configuration**). Et pour chaque transition $(s,_M a_x, s_i)$ de cet ensemble :

- on ajoute cette transition à STEM et
- la configuration s_i est intégrée dans liste_Confs si et seulement si s_i n'est pas un nouveau état.

La liste Sub reste vide jusqu'à la détection d'un état s_j existant ($\exists s' \in$ *liste_ Confs*$\cup$ Liste_Confs_traitées : $s_j \approx_\alpha s'$), dans ce cas, le STEM, *liste_ Confs* et Liste_Confs_traitées doivent être substitués par Sub, ce qu'est effectivement réalisé au niveau de la fonction **Substituer.**

Pour mieux expliquer cela, prenons l'expression précédente "$a; d; stop|[d]|b; d; stop$", sa configuration initiale est :

$$(_{\emptyset}[a; d; stop|[d]|b; d; stop]) \text{ et } Liste\_Confs = [(_{\emptyset}[a; d; stop|[d]|b; d; stop])].$$

- Dans la première itération de la boucle **Tantque**, nous pouvons générer par le biais de la fonction **Traiter_configuration** les deux configurations suivantes :

$(_{\{(a,1)\}}[d; stop]||[d]|_{\emptyset}[b; d; stop])$ et $(_{\emptyset}[a; d; stop]||[d]|_{\{(b,2)\}}[d; stop])$
où :
$$\begin{cases} \text{Liste\_Confs\_traitées} = \{(_{\emptyset}[a; d; stop|[d]|b; d; stop])\} \\ \text{Liste\_Confs} = \{(_{\{(a,1)\}}[d; stop]||[d]|_{\emptyset}[b; d; stop]);\ (_{\emptyset}[a; d; stop]||[d]|_{\{(b,2)\}}[d; stop])\} \end{cases}$$
avec aucun cas d'équivalence détecté.

- Dans la deuxième itération, nous pouvons générer à partir de la configuration $(_{\{(a,1)\}}[d; stop]||[d]|_{\emptyset}[b; d; stop])$ la configuration $(_{\{(a,1)\}}[d; stop]||[d]|_{\{(b,3)\}}[d; stop])$ où :
$$\begin{cases} \text{Liste\_Confs\_traitées} = \{(_{\emptyset}[a; d; stop|[d]|b; d; stop]);\ (_{\{(a,1)\}}[d; stop]||[d]|_{\emptyset}[b; d; stop])\} \\ \text{Liste\_Confs} = \{(_{\emptyset}[a; d; stop]||[d]|_{\{(b,2)\}}[d; stop]);\ (_{\{(a,1)\}}[d; stop]||[d]|_{\{(b,3)\}}[d; stop])\} \end{cases}$$
Toujours avec aucun cas d'équivalence à signaler.

- Cependant, et dans la troisième itération, le développement de la configuration $(_{\emptyset}[a; d; stop]||[d]|_{\{(b,2)\}}[d; stop])$ génère $(_{\{(a,4)\}}[d; stop]||[d]|_{\{(b,2)\}}[d; stop])$ qu'est une configuration α-équivalente avec $(_{\{(a,1)\}}[d; stop]||[d]|_{\{(b,3)\}}[d; stop])$,

  ce qui nous donne la liste de substitution suivante Sub= $\{\{(a,1),(a,4)\},\{(b,2),(b,3)\}\}$ où $\{(a,1),(a,4)\}/(a,5)$ et $\{(b,2),(b,3)\}/(b,6)$

Après la substitution :
$$\begin{cases} \text{Liste\_Confs\_traitées} = \{(_{\emptyset}[a; d; stop|[d]|b; d; stop]);\ (_{\{(a,5)\}}[d; stop]||[d]|\emptyset[b; d; stop]); \\ \qquad\qquad (_{\emptyset}[a; d; stop]||[d]|_{\{(b,6)\}}[d; stop])\} \\ \text{Liste\_Confs} = \{(_{\{(a,5)\}}[d; stop]||[d]|_{\{(b,6)\}}[d; stop])\} \end{cases}$$

- Au niveau de la quatrième itération, la configuration $(_{\{(d,7)\}}[stop]||[d]|_{\{(d,7)\}}[stop])$ est générée. Le STEM construit jusqu'à là est représenté dans la Figure 4.3 , avec :
$$\begin{cases} \text{Liste\_Confs\_traitées} = \{(_{\emptyset}[a; d; stop|[d]|b; d; stop]);\ (_{\{(a,5)\}}[d; stop]||[d]|\emptyset[b; d; stop]); \\ \qquad\qquad (_{\emptyset}[a; d; stop]||[d]|_{\{(b,6)\}}[d; stop]); \\ \qquad\qquad (_{\{(a,5)\}}[d; stop]||[d]|_{\{(b,6)\}}[d; stop])\} \\ \text{Liste\_Confs} = \{(_{\{(d,7)\}}[stop]||[d]|_{\{(d,7)\}}[stop])\} \end{cases}$$

- La cinquième itération est le point d'arrêt de cet algorithme, en effet la seule configuration restée à développer est un état puits, donc le STEM α-réduit est le même STEM obtenu dans l'itération précédente.

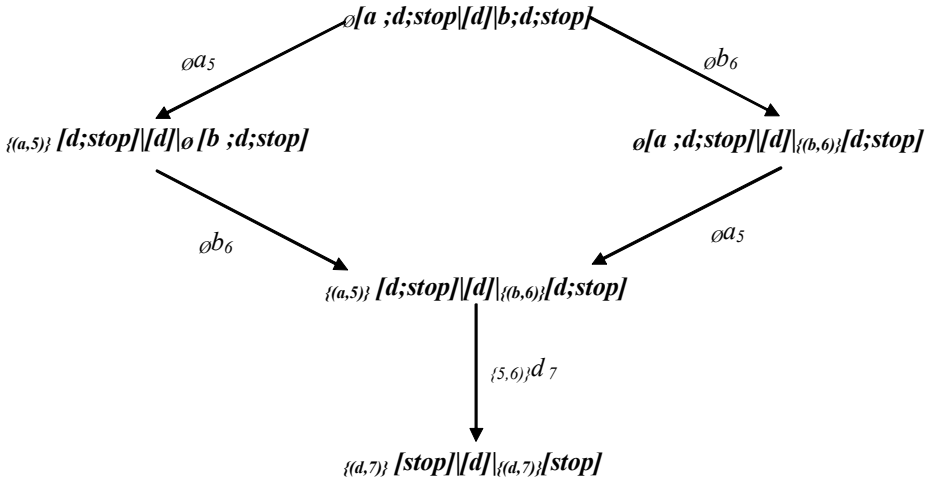


FIG. 4.3 – STEM α-réduit

Raffinement de l'algorithme La génération d'un STEM$_{/\approx_\alpha}$ est basée principalement sur deux fonctions, à savoir la fonction **Substituer** et la fonction **Traiter_configuration**. La substitution cherche à remplacer l'occurrence d'un événement par un autre événement donné, cette modification touche le STEM, *liste_ Confs* et Liste_Confs_traitées.

Algorithme 4.2 *"Substituer"*

Données : *Une liste de substitutions Sub, un STEM, liste_ Confs et Liste_ Confs_ traitées ;*
Résultats : *Un STEM ;*
Début
Attribuer à chaque bloc $\beta_i \in Sub$ un couple $(\pi(\beta_i), get)$ "identificateur" ;
PourChaque $\beta_i \in Sub$ ***Faire***
PourChaque $(a, é) \in \beta_i$ ***Faire***
// La substitution du STEM
Remplacer les occurrences de (a,é) dans le STEM par son identificateur ;
// La substitution de liste_ Confs et Liste_ Confs_ traitées
Remplacer les occurrences de (a,é) dans liste_ Confs et Liste_ Confs_ traitées par son identificateur ;
FinPour
FinPour
FinAlgo

où : $\pi(\beta_i) = a$, le nom de l'action représentée par la liste d'événements β_i.

La fonction **Traiter_configuration** nous permet de construire l'ensemble de transitions sensibilisées dans Conf, où on vérifie pour chacune de ces transitions si sa configuration cible Conf ' est α-équivalent avec une autre configuration Conf" ou non, si oui

- on met leurs événements maximaux en relation, ceci se traduit par Sub.
- on remplace l'état cible de transition par l'état attribué à Conf ".

Algorithme 4.3 *"Traiter Configuration"*
Données : *Une configuration Conf, Liste_ Confs et Liste_ Confs_ Traitées ;*
Résultats : *Sub, liste de transitions, liste de nouvelles configurations ;*
Début
A partir de Conf, ***tirer des transitions*** *en utilisant les sémantiques opérationnelles de maximalité ;*
PourChaque *transition t résultante* ***Faire***
// Soit Conf' la configuration cible de t;
Vérifier, si il existe une Conf"=$_\alpha$Conf' appartient à Liste_ Confs ou à Liste_ Confs_ Traitées ;
Si *Conf" existe* ***Alors***
Construire la liste de substitution Sub ;
Remplacer l'état cible de t par l'état attribué à Conf " ;
Sinon
Ajouter Conf' à la liste de nouvelles configurations ;
Ajouter t à la liste de transitions ;
FinAlgo

A titre d'indication, la complexité de l'algorithme 4.1 est comparable à celle de l'algorithme général de construction du graphe d'états, car la seule différence entre ces deux là est résumée par la fonction **Substituer** qui est un calcul lié au nombre d'actions qui peuvent être potentiellement en cours d'exécution, dans le pire des cas cet ensemble contient tous les événements du système.

4.2 Graphe de pas maximaux

S'inspirant de la technique de pas couvrants, nous ne considérons pas tous les entrelacements possibles. Par contre, nous construisons, sous certaines conditions, un pas permettant d'atteindre directement l'état final qui aurait été atteint par chacune des séquences entrelacées. La Figure 4.4 montre le bénéfice obtenu dans le cas de la dérivation de trois actions parallèles (a, b et c) en présence du conflit différé. Le graphe de la Figure 4.4.(b) est le graphe de pas du STEM de la Figure 4.4.(a) dans lequel tous les entrelacements ont été convertis en deux pas (p_1 et p_2) ; le premier pas exprime le début d'exécution de c et l'autre exprime celui de l'exécution parallèle de a et de b.

Le graphe de pas construit couvre le STEM initial modulo l'équivalence de traces de Mazurckiewicz. Il préserve également les états de blocage et la propriété de la vivacité, sa génération à la volée est ainsi possible.

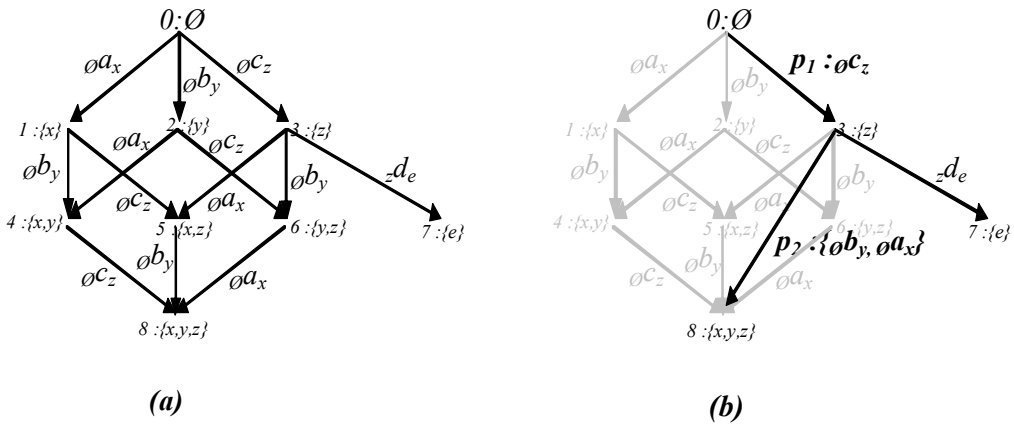


FIG. 4.4 – Un STEM et son graphe de pas maximaux

4.2.1 Définions préliminaires

L'entrelacement censé être représenté par un pas (ou plusieurs pas), est un chemin (dit chemin maximal) dont l'état cible contient tous les événements maximaux associés aux actions qui le constituent. Les différentes définitions présentées dans cette section ont pour but l'introduction de la notion de pas (dit pas maximal).

Définition 4.2 *« Séquence d'événements »*

Soit Atm un ensemble d'atomes et $\mathcal{M}$ un ensemble dénombrable de noms des événements, $< \_ >$ est l'application Atm $\longrightarrow \mathcal{M}^*$, définie intuitivement par

- $< \epsilon >=_{def} \epsilon$

- $<_M a_x.p >=_{def} x. < p >$

Définition 4.3 *« Support d'une séquence »*

Soit $||\ ||$, l'application de $E \longrightarrow 2^E_{fn}$ définie par :

- $||\epsilon|| =_{def} \varnothing$

- $||u.w|| =_{def} \{u\} \cup ||w||$

Définition 4.4 *« Extension de trace de Mazurckiewicz sur les STEMs »*

Soit $G =< S, s_0, T, \psi, \mu, \xi >$ un STEM, et soient $U._M a_x._N b_y.V$ et $U._{N'} b_y._{M'} a_x.V$ deux chemins de G. On considère $\approx$ la relation définie sur $T^* \times T^*$ par $< U._M a_x._N b_y.V > \approx < U._{N'} b_y._{M'} a_x.V >$ ssi $x \notin N$ et $y \notin M'$ (cette condition exprime aib), par construction, $\approx$ est réflexive et symétrique.

L'équivalence de trace $\equiv$ peut être définie par la fermeture transitive de la relation $\approx$, ses classes appelées des traces. $[< w >]$ dénote la trace générée par le chemin w.

Définition 4.5 *« Chemin maximal »*

Soit $G =< S, s_0, T, \psi, \mu, \xi >$ *un STEM et soit* $w \in T^*$, w *est un chemin maximal ssi* $\exists s, s' \in S, s \stackrel{w}{\Longrightarrow} s'$:

- $|| < w > || \subseteq \psi(s')$

- et $(s' \nrightarrow) \vee (\exists t \in T : wt$ n'est pas un chemin maximal)

Définition 4.6 *« Le petit chemin »*

Soit C_s *un ensemble de chemins maximaux associe à l'état* s, $Min(C_s) = \{c \backslash\ \nexists c' \in C_s : || < c' > || \subset || < c > ||\}$

Définition 4.7 *« Équivalence des chemins maximaux »*

Deux chemins maximaux w *et* w' *sont équivalents* $(w \approx_c w')$ *ssi :* $s \stackrel{w}{\Longrightarrow} s'$ *et* $s \stackrel{w'}{\Longrightarrow} s'$.

C'est un cas particulier de la relation d'équivalence de trace de Mazurckiewicz dont tous les événements sont indépendants $w \approx_c w' \Longrightarrow [< w >] \equiv [< w' >]$,*cette conséquence est évidente, car* $|| < w > || = || < w' > || \subseteq \psi(s')$ *et* $\forall t, t' \in ||w|| : t \imath t'$.

Preuve. $w \approx_c w' \Longrightarrow || < w > || = || < w' > ||$

Soient w et w'deux chemins équivalents, ce qui nous permet d'écrire

$$\begin{cases} || < w > || \subseteq \psi(s') \\ || < w' > || \subseteq \psi(s') \end{cases} \stackrel{?}{\Longrightarrow} || < w > || = || < w' > ||$$

on va procéder par l'absurde, donc on suppose que $\exists x \in || < w > ||$ et $x \notin || < w' > ||$,

cependant, la supposition $x \notin || < w' > ||$ nous conduit directement vers la conclusion $x \notin \psi(s')$, ce qui signifie que w n'est pas un chemin maximal !. Même raisonnement pour $\exists x \in || < w' > ||$ et $x \notin || < w > ||$ ■

Par exemple, pour l'état initial de la Figure 4.5 les chemins maximaux possibles sont $C_0 = \{_\varnothing b_y ;_\varnothing b_y .{}_\varnothing a_x .{}_\varnothing c_z ;_\varnothing a_x .{}_\varnothing c_z .{}_\varnothing b_y\}$, où $_\varnothing b_y .{}_\varnothing a_x .{}_\varnothing c_z$ et $_\varnothing a_x .{}_\varnothing c_z .{}_\varnothing b_y$ sont deux chemins équivalents. Le chemin $_\varnothing b_y$ est le petit chemin de l'ensemble C_0.

Définition 4.8 *« Pas de transitions maximales»*

Soit $G =< S, s_0, T, \psi, \mu, \xi >$, *et soit* $w \in T^*$, $||w||$ *définit un pas ssi*

$\exists s, s' \in S, w \in T^*, s \stackrel{w}{\Longrightarrow} s' : \forall x \in || < w > || \Longrightarrow x \in \psi(s')$

N'est qu'une autre écriture de la définition de pas couvrant (Définition 3.8) en utilisant la terminologie de la maximalité.

Propriété 4.1 *soit* $s \stackrel{w}{\Longrightarrow} s'$ *un chemin maximal :*

1. *w est un chemin fini.*

2. *les transitions d'un chemin maximal constituent un pas de transitions*

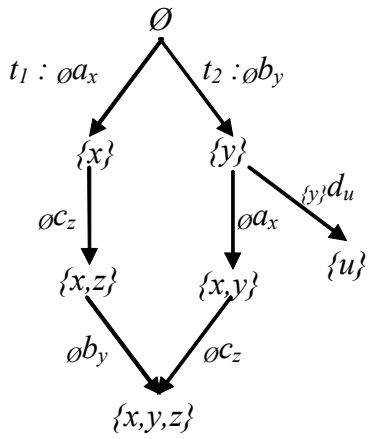


FIG. 4.5 – un STEM

La première propriété est une conséquence directe de la Définition 4.5, en d'autres termes, s' ne peut pas être un état intermédiaire dans w.

Pour un chemin maximal donné w, la deuxième propriété exprime que l'ensemble de transition $||w||$ peut constituer un pas c-à-d $\forall x \in || < w > || \Longrightarrow x \in \psi(s')$, cela est vérifie car w est un chemin maximal où $|| < w > || \subseteq \psi(s')$.

Définition 4.9 *« Extension de la relation d'accessibilité aux pas de transitions maximales »*

Soit $\longrightarrow_p$*, l'extension de* $\longrightarrow$ *aux pas maximaux, et* w *un chemin maximal* $s \stackrel{w}{\Longrightarrow} s'$*. Le pas associé est* $\stackrel{||w||}{\longrightarrow_p}$ *.*

La méthode de réduction proposée consiste à remplacer tous les chemins maximaux équivalents par un seul chemin (en préservant les chemins maximaux), où ce dernier doit être à son tour remplacé par un pas maximal. A la fin, le graphe construit est un graphe de pas maximaux.

4.2.2 Préservation de chemins maximaux

La préservation des chemins maximaux consiste, pour chaque état, à ne prendre en compte que ses transitions qui couvrent toutes ses traces de chemins maximaux possibles. On note le STEM qui préserve les chemins maximaux par STEM^{op}. A titre d'illustration, dans la Figure 4.5, la transition t_1 sortant de l'état 0 préserve la trace $[x.z.y]$, alors que l'autre transition t_2 préserve les traces $[x.z.y]$ et $[y]$. En conséquence, c'est la deuxième transition qui doit être prise en compte. La Figure 4.6 représente le STEM^{op} de celui de la Figure 4.5.

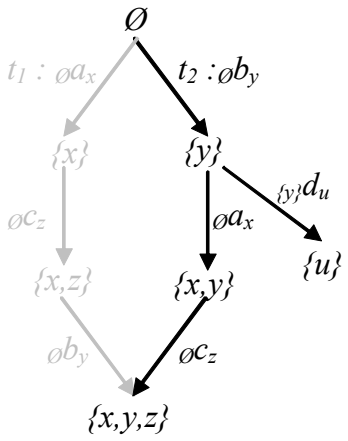


FIG. 4.6 – un STEM^{op}

Proposition 4.1 *Soit s un état et soit C_s son ensemble des chemins maximaux. l'ensemble $Min(C_s)$ assure toutes les bronchements qui couvrent l'ensemble C_s*

$\forall c \in C_s \backslash Min(C_s), \exists c_m \in Min(C_s)$ *et* $\exists w \in T^* : c_m.w \in C_s$ *tel que* $c \approx_c c_m.w$

Preuve. Selon la Définition 4.6

$Min(C_s) = \{c_m \backslash \nexists c \in C_s : || < c > || \subset || < c_m > ||\}$, ce qui nous permet d'écrire

$\forall c \in C_s \backslash Min(C_s), \exists c_m \in Min(C_s)$ où $|| < c_m > || \subset || < c > ||$,ceci implique que l'état cible de c_m est un état intermédiaire dans c, en d'autres termes, $\exists w \in T^* : c_m.w \in C_s$ tel que l'état cible de c et l'état cible de $c_m.w$ son le même état. ■

Définition 4.10 *"STEM^{op}"*

Soit G=$< S, s_0, T, \psi, \mu, \xi >$ et G'=$< S', s_0, T', \psi', \mu', \xi' >$ deux STEMs, G' un STEM^{OP} de G ssi :

1. $\forall s' \in S' : s' \in S$,
2. $\forall t' \in T' : t' \in T$ *et*
3. $\forall s \in S', s \xrightarrow{M a_x} s' \in T, \forall s" \in S', \forall w \in T^* : s' \xRightarrow{w} s"$

$$\Longrightarrow \left\{ \exists w' \in T'^*, s \xRightarrow{w'} s" : [<_M a_x.w >] \equiv [< w' >] \right\}$$

La Condition 1 impose que tout état du STEM^{op} soit un état du STEM. La Condition 2 impose que toute transition du STEM^{op} soit une transition du STEM, et la Condition 3 exprime une condition de couverture modulo les traces de Mazurckiewcz entre les séquences du STEM et celles associées au STEM^{op}. Par conséquent, le STEM^{op} est un graphe complet du STEM initial qui préserve les états de blocage et la propriété de la vivacité.

Proposition 4.2

1. *Etant donné un STEM G, G^{op} préserve les chemins maximaux de G.*
2. *Si G' un $STEM^{op}$ de G alors DeadLock(G')=DeadLock(G).*
3. *Soit G' un $STEM^{op}$ de G, pour tout événement é de G', é est accessible de n'importe qu'il état de G' (dit vivant) ssi é est accessible de n'importe qu'il état de G.*

Preuve. "Proposition 4.2.(1)"

Soit $G =< S, s_0, \mathrm{T}, \psi, \mu, \xi >$ un STEM et $G' =< S', s_0, \mathrm{T'}, \psi, \mu, \xi >$ un STEM^{OP} de G , et soit C_s^G ($\mathrm{resC}_s^{G'}$) l'ensemble de tous les chemins maximaux de l'états s de G (resp G').

Une transition $t \in T : s \xrightarrow{{}_M a_x} s'$ est supprimée ssi tous ses chemins maximaux sont jugés redondants

$\forall s'' \in S', \forall w \in C_{s'}^G : s' \overset{w}{\Longrightarrow} s'' \Longrightarrow \left\{ \exists w' \in C_s^{G'}, s \overset{w'}{\Longrightarrow} s'' :_M a_x.w \approx_c w' \right\}$

Et comme ${}_M a_x.w \approx_c w' \Longrightarrow [<_M a_x.w >] = [< w' >]$ alors

$\forall s'' \in S', \forall w \in \mathrm{T}^* : s' \overset{w}{\Longrightarrow} s'' \Longrightarrow \left\{ \exists w' \in \mathrm{T'}^*, s \overset{w'}{\Longrightarrow} s'' : [<_M a_x.w >] \equiv [< w' >] \right\}$

n'est que la condition 3 de la Définition 4.10, en d'autres termes G' préserve les chemins maximaux de G. ■

Preuve. "Proposition 4.2.(2)"

1. $DeadLock(G') \subset DeadLock(G)$ par définition
2. $DeadLock(G) \subset DeadLock(G')$, soit $\forall d \in DeadLock(G) : s_0 \overset{w}{\Longrightarrow} d$, d'après la Définition.4.10.(3) il existe $s' \in S'$ où $s_0 \overset{w}{\Longrightarrow} d \overset{w'}{\Longrightarrow} s'$ et comme d est un état de blocage alors $w' = \epsilon$ et $s' = d$. Ce qui nous donne $d \in DeadLock(G')$

■

Preuve. "Proposition 4.2.(3)"

$G =< S, s_0, \mathrm{T}, \psi, \mu, \xi >$ est E-vivant pour E $=\{\xi\ (\mathrm{t})/\mathrm{t} \in \mathrm{T}\}$ssi

$$\forall s \in S, \forall t \in \mathrm{T} : \xi(t) \in E, \exists w \in \mathrm{T}^* \text{ tel que } s \overset{wt}{\Longrightarrow} .$$

D'après la Définition 4.10.(3) nous pouvons écrire

$\forall s' \in S', s' \xrightarrow{t'} s \overset{wt}{\Longrightarrow} s'', \exists w' \in T'^*, s' \overset{w'}{\Longrightarrow} s$" :

$$[< t'.wt >] \equiv [< w' >] \Longrightarrow \xi(t) \in || < w' > ||.$$

En d'autres termes $\forall s' \in S', \forall t \in \mathrm{T'} : \xi(t) \in E, \exists w \in \mathrm{T''}^*$ tel que $s' \overset{wt}{\Longrightarrow}$ .donc G'est E-vivant. ■

```
/*Spécification LOTOS : Deux ensembles de conflits indépendants */
system Conflits[a,b,c,d] := P[a,b]|||P[c,d]
where
Process P[a,b] :=a ;stop[]b ;stop endproc
endsys
```

TAB. 4.1 – *Spécification LOTOS : Deux ensembles de conflits indépendants*

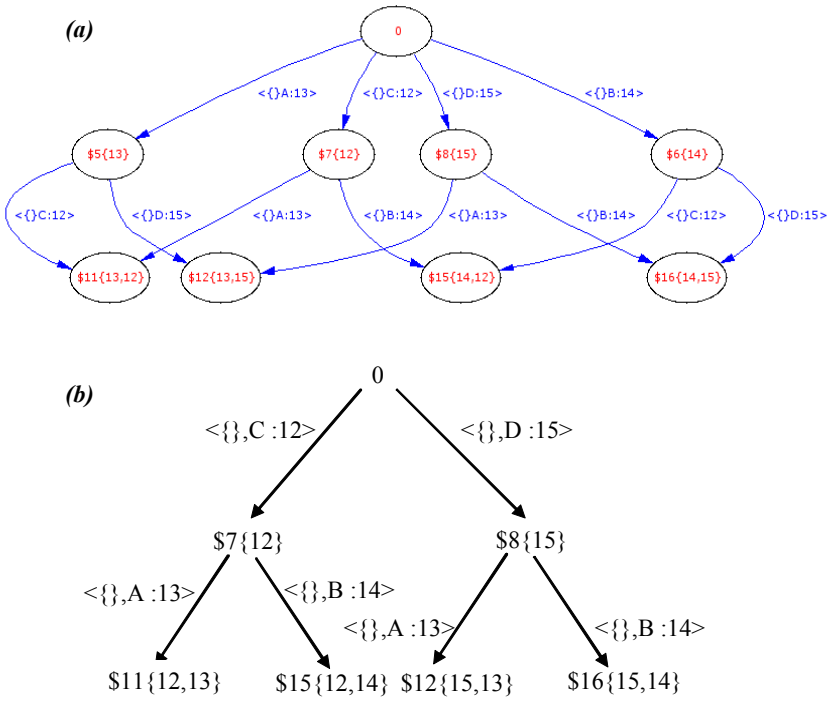


FIG. 4.7 – Un STEM et son STEM^{op}

Exemple 4.1 *La spécification de la table 4.1 présente deux ensembles de conflits indépendants, la Figure 4.7.(a) représente son STEM associé généré par l'outil* `LotoSTEM`$_{V_{2.0}}$ *[SB04]. Ce dernier nous permet de générer à partir d'une expression LOTOS son STEM* α*-réduit. La Figure 4.7.(b) représente le STEM*^{op} *correspondant. Nous remarquons par exemple que les chemins* ${}_{\emptyset}A_{13}.{}_{\emptyset}C_{12}$ *et* ${}_{\emptyset}C_{12}$ *.*${}_{\emptyset}A_{13}$ *sont couverts dans la Figure 4.7.(b) par le chemin* ${}_{\emptyset}C_{12}$ *.*${}_{\emptyset}A_{13}$*, et même chose pour les états de blocage* $\$11, \$12, \$15$ *et* $\$16$ *qui sont eux aussi préservés.*

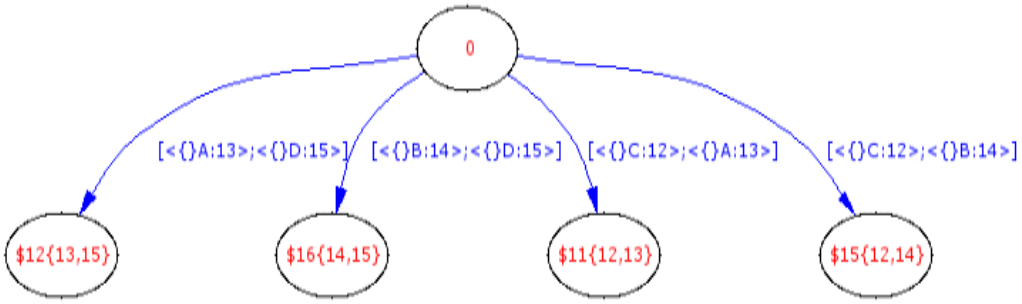


FIG. 4.8 – Un Graphe de pas

4.2.3 Graphe de pas maximaux

Intuitivement, un STEM^{op} n'est qu'un graphe de pas maximaux (GPM) en remplaçant chacun de ses chemins maximaux par leur pas correspondant. À titre d'illustration, le GPM de la Figure 4.8 a été obtenu en remplaçant l'ensemble de chemins maximaux $\{_{\emptyset} C_{12.\emptyset} A_{13}$; $_{\emptyset} C_{12.\emptyset} B_{14}$; $_{\emptyset} D_{15.\emptyset} A_{13}$;$_{\emptyset} D_{15.\emptyset} B_{14}\}$ du STEM^{op} de la Figure 4.7.(b) par leur ensemble de pas correspondant.

Remarque 4.1 *Le STEM de la Figure 4.8 est généré par l'outil* `LotosGPM[BSO5b]`.

Définition 4.11 *"Graphe de Pas Maximaux"*

Soit $G{=}< S, s_0, T, \psi, \mu, \xi >$ *un STEM*^{op}, $G'{=}< S, s_0, \Xi, \psi, \mu', \xi' >$ *est un graphe de pas maximaux de* T *ssi :*

1. $\forall s' \in S' : s' \in S$,
2. $\forall t' \in \Xi : t'$ *est un pas, où* $||t'||$ *constituent un chemin maximal dans* T.
3. $\forall s \in S', s \xrightarrow{M a_x} s' \in T, \forall s" \in S', \forall w \in T^* : s' \overset{w}{\Longrightarrow} s"$

$$\Longrightarrow \left\{ \exists w' \in \Xi^*, s \overset{w'}{\Longrightarrow}_p s" : [<_M a_x.w >] \equiv [< w' >] \right\}$$

tel que :

- $\xi' : 2^T_{fn} \rightarrow 2^{\mathcal{M}}_{fn}$
  $\xi'(\epsilon) =_{def} \epsilon$
  $\xi'(\{t\} \cup E) =_{def} \xi(t) \cup \xi'(E)$

- $\xi(\overset{M}{\longrightarrow}{}^{a_x}) =_{def} x$
- $\mu' : 2^T_{fn} \rightarrow 2^{\mathcal{M}}_{fn}$
  $\mu'(\epsilon) =_{def} \epsilon$
  $\mu'(\{t\} \cup E) =_{def} \mu(t) \cup \mu'(E)$
- $\mu(\overset{M}{\longrightarrow}{}^{a_x}) =_{def} M$

Où pour tout pas $s \xrightarrow{E}_p s'$, les conditions suivantes sont satisfaites :

- $\psi(s') = (\psi(s) \backslash \mu'(E)) \cup \xi'(E)$
- $\xi'(E) \not\subseteq \psi(s) - \mu'(E)$
- $\mu'(E) \subseteq \psi(s')$

Proposition 4.3 *Les graphes de pas maximaux préservent les états de blocage et la vivacité.*

Preuve. Soit $G =< S, s_0, \mathrm{T}, \psi, \mu, \xi >$ un STEM^{op} et $G' =< S, s_0, \Xi, \psi', \mu', \xi' >$ son GPM

- préservation des états de blocage :

1. DeadLock (G') $\subset$DeadLock (G) par définition
2. DeadLock (G) $\subset$DeadLock (G'), soit $\forall d \in$DeadLock(G) : $s_0 \overset{w}{\Longrightarrow} d$ d'après la Définition 4.11.(3) il existe $s' \in S'$ où $s_0 \overset{w}{\Longrightarrow} d \overset{w'}{\Longrightarrow} s'$ et comme d est un état de blocage alors $w' = \epsilon$ et $s' = d$ donc $d \in$ DeadLock (G')

- préservation de la vivacité

  $G =< S, s_0, \mathrm{T}, \psi, \mu, \xi >$ est E-vivant pour E $=\{\xi$ (t)/t$\in$ T$\}$ssi
  $\forall s \in S, \forall t \in \mathrm{T} : \xi(t) \in E, \exists w \in \mathrm{T}^*$ tel que $s \overset{wt}{\Longrightarrow}$.

  D'après la Définition 4.11.(3) nous pouvons écrire $\forall s' \in S', s' \overset{t'}{\longrightarrow} s \overset{wt}{\Longrightarrow} s'', \exists w' \in \Xi^*, s' \overset{w'}{\Longrightarrow} s$" :

  $$[< t'.wt >] \equiv [< w' >] \Longrightarrow \xi(t) \in || < w' > ||.$$

  En d'autres termes $\forall s' \in S', \forall t \in \Xi : \xi(t) \in E, \exists w \in \Xi^*$ tel que $s' \overset{wt}{\Longrightarrow}$,donc G'est E-vivant

■

4.2.4 Algorithme de construction à la volée d'un graphe de pas maximaux

Le graphe de pas maximaux de la Figure 4.8 a été construit à la volée à partir d'une spécification LOTOS. Ceci est effectué en se servant de l'extension directe (algorithme 4.4) de l'algorithme de génération à la volée d'un STEM α-réduit présenté dans la Section 4.1.2. La différence réside dans la ligne # (La construction des pas) , qui consiste à vérifier, pour chaque transition développée, si elle peut faire partie (Condition.1) d'un pas maximal ou elle est elle-même un pas.

Algorithme 4.4 *"Algorithme de construction à la volée d'un GPM"*

Données : *Spécification LOTOS ;*
Résultats : *Un GPM ;*
Variables :
Liste_Confs : liste de configurations non traitées ;
Liste_Confs_traitées : liste de configurations déjà traitées ;
T : liste de les transitions
Sub : liste de liste des couples (action,événement)
représentant un même événement (la fonction σ)
Début
Construire la configuration initiale ;
Initialiser la liste de configurations Liste_Confs par la configuration initiale ;
Tantque *liste_Confs non vide* ***Faire***
Sub$\longleftarrow \varnothing$
Sélectionner et supprimer un élément Conf de Liste_Confs ;
Traiter_configuration *Conf ;*
Ajouter Conf à la liste Liste_Confs_traitées ;
Ajouter les nouvelles configurations résultantes à Liste_Confs ;
Ajouter les transitions résultantes T à STEM ;
Substituer *STEM,Liste_Confs et Liste_Confs_traitées en utilisant Sub ;*
Construire_pas ..... #
FinTant
FinAlgo.

Condition 4.1 *Soit $G_{s'} =< S_{s'}, s_0, \Xi_{s'}, \psi, \mu, \xi >$ un GPM en cours de génération(au niveau de l'état s'), et soit $s \xrightarrow{p}_p s' \in \Xi, T_{s'} = \{t/\forall t : s' \xrightarrow{t} s'' \}$ généré :*

si $p \in Min(C_s)$ alors $t \in T_{s'}$ est un pas de GPM si non pt est un pas, donc dans tous les cas on a :

$\exists w \in (\Xi_{s'} \cup T_{s'})^* : s.\overset{w}{\Longrightarrow}_p s'' \Longrightarrow [< pt >] = [< w >]$.

Cette condition assure la construction d'un GPM préserve les chemins maximaux. L'idée derrière cette condition est la proposition 4.1,

La construction des pas

Afin d'assure la préservation des chemins maximaux, le calcul (Algorithme 4.5)est effectué en cherchant pour chaque un état s (l'antécédent de l'état en cours de dérivation) à transformer ses petits chemins maximaux en pas (proposition 4.1).

Algorithme 4.5 *"Construire_ pas"*

Données : *s : l'état en cours de dérivation,GPM en cours de génération ;*
Résultats : *Un GPM ;*
Variables :
S : liste des antécédents de l'état s ;
Début
PourChaque $s' \in S$ *:* $s' \xrightarrow{p}_p s$ ***Faire***
Construire $C_{s'}$ *et* $Min(C_{s'})$;
Transformer $Min(C_{s'})$ *en pas ;*
EndPour
EndAlgo

Pour mieux expliquer cela, prenons l'expression "$a; stop|||b; stop|||c; stop$" :

- Dans la première itération de la boucle **Tantque** de l'algorithme 4.4 , l'état en cours de dérivation est l'état initial 0. Les transitions générées son présentées dans la Figure 4.9.*étape1.* Et comme $C_0 = Min(C_0) = \{_{\varnothing} a_1;_{\varnothing} b_2;_{\varnothing} c_3\}$ le GPM reste le même.

- Au niveau de la deuxième itération (l'état en cours est l'état1), les transitions qui prouvent être tirées son présentées dans la Figure 4.9.*étape2.*(a), où il y a un seul état antécédent qui est l'état 0, dont $C_0 = Min(C_0) = \{_{\varnothing} a_1._{\varnothing} c_5;_{\varnothing} a_1._{\varnothing} b_4 ;_{\varnothing} b_2;_{\varnothing} c_3\}$.Le GPM est donné dans Figure 4.9.*étape2.*(b).

- Cependant, dans la troisième itération (l'état en cours est 2 qui a un seul état antécédent 0), on signale un cas de α-équivalence. Le graphe obtenu après la substitution est présenté dans la Figure 4.9.*étape3.*(a), où $C_0 = Min(C_0) = \{_{\varnothing} a_6._{\varnothing} c_5;_{\varnothing} a_6._{\varnothing} b_7 ;_{\varnothing} b_7._{\varnothing} c_8;_{\varnothing} c_3\}$. la Figure 4.9.*étape3.*(b) est le GPM obtenu.

- Dans la la quatrième itération, on détecte aussi deux cas d'équivalence (Figure 4.9.*étape4.*(a)), l'état en cours est l'état 3 qui a un seul état antécédent 0. L'ensemble de chemins maximaux de 0 est $C_0 = Min(C_0) = \{_{\varnothing} a_{10}._{\varnothing} c_{11};_{\varnothing} a_{10}._{\varnothing} b_9 ;_{\varnothing} b_9._{\varnothing} c_{11}\}$. Le GPM construit est la Figure 4.9.*étape4.*(b).

- Dans la cinquième itération, le GPM obtenu est présenté dans la Figure 4.9.*étape5.*(b) où $C_0 = Min(C_0) = \{_{\varnothing} a_{10}._{\varnothing} c_{11};_{\varnothing} a_{10}._{\varnothing} b_9._{\varnothing} c_{12};\ _{\varnothing} b_9._{\varnothing} c_{11}\}$.

- Au niveau de l'itération avant dernière, l'état en cours est 6 son antécédent est toujours l'état 0. où $C_0 = Min(C_0) = \{_{\varnothing} a_{13}._{\varnothing} c_{14}._{\varnothing} b_{15} ;_{\varnothing} a_{13}._{\varnothing} c_{14}\}$.

- Dans la dernière itération on a $C_0 = Min(C_0) = \{_{\varnothing}a_{13}._{\varnothing}c_{14}._{\varnothing}b_{16}\}$, donc le GPM final est le graphe de la Figure 4.10.étape7.(c).C'est un graphe a un seul pas !.

Proposition 4.4 *Le graphe généré par l'algorithme 4.4 est un graphe de pas maximaux.*

Preuve. On va procéder par induction sur le nombre d'itérations nécessaire pour générer un GPM .

- Dans la première itération, le graphe construit (un STEM) est de profondeur 1, il est évident qui il est un GPM.

- Pour la $n+1$ ^{ième}*itération* :

soit $G_n = \{S_n, s_0, \Xi_n, \psi, \mu, \xi >$ un GPM construit par supposition dans la n ^{ième}*itération*, et soit s l'état à traiter dans la $n+1$ ^{ième}*itération* où

$$\begin{cases} T_s = \{t / \forall s' \in S_n : s \xrightarrow{t} s'\} \\ S_s = \{\beta(t) : \forall t \in T_s\} \end{cases}$$

le GPM construit est donc $G'_{n+1} = \{S_n \cup S_s, s_0, \Xi_n \cup T_s, \psi, \mu, \xi >$ et selon la condition4.1 :

$\forall p : s_1 \xrightarrow{p}_p s \in \Xi_n, \forall t \in {T_s}^*, s_1. \overset{pt}{\Longrightarrow}_p s_2 :$

$$\Longrightarrow (\exists w \in \Xi^*_{n+1}, s_1. \overset{w}{\Longrightarrow}_p s_2 : [< pt >] \equiv [< w >]).$$

où $G_{n+1} = \{S_{n+1}, s_0, \Xi_{n+1}, \psi, \mu, \xi >$ le graphe construit suivant la condition4.1

cela nous permet de déduire que

$\forall s_1 \in S_{n+1}, s_1 \xrightarrow{p}_p s_2 \in (\Xi_n \cup T_s), \forall s' \in S_{n+1}, \forall w \in (\Xi_n \cup T_s)^* : s_2 \overset{w}{\Longrightarrow} s'$

$\Longrightarrow (\exists w' \in \Xi^*_{n+1}, s_1 \overset{w}{\Longrightarrow}_p s' : [< p.w >] \equiv [< w' >]$

en d'autres termes, le graphe G_{n+1} est un GPM qui couvre le graphe $G'_{n+1} = \{S_n \cup S_s, s_0, \Xi_n \cup T_s, \psi, \mu, \xi >$.

■

Pour donner une idée sur la complexité de l'algorithme 4.4, le nombre d'itérations nécessaires pour générer un GPM est exactement le même nombre d'itérations nécessaires pour générer un STEM. Le plus est dans le calcul de pas maximaux.

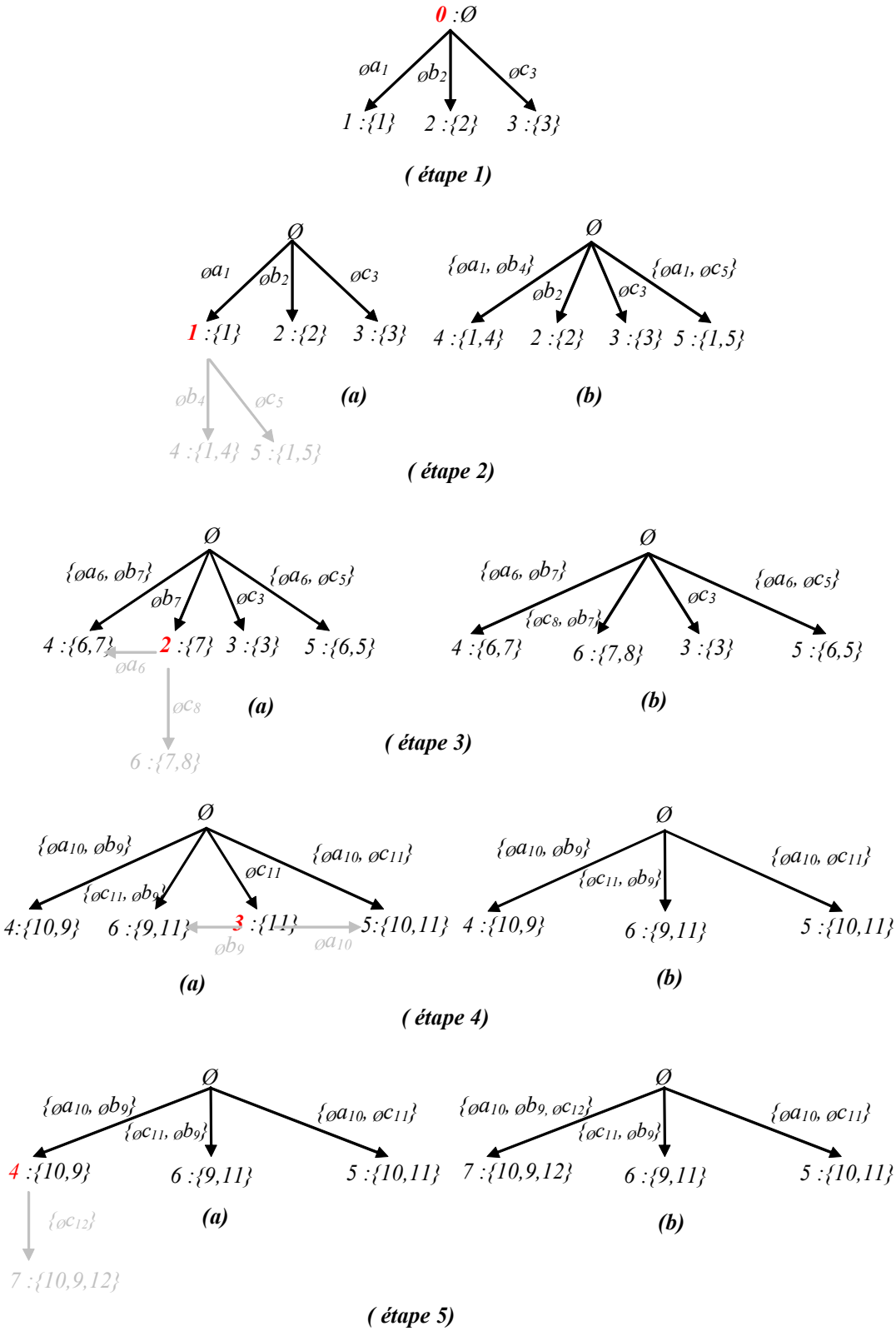


FIG. 4.9 – GPM en cours de génération

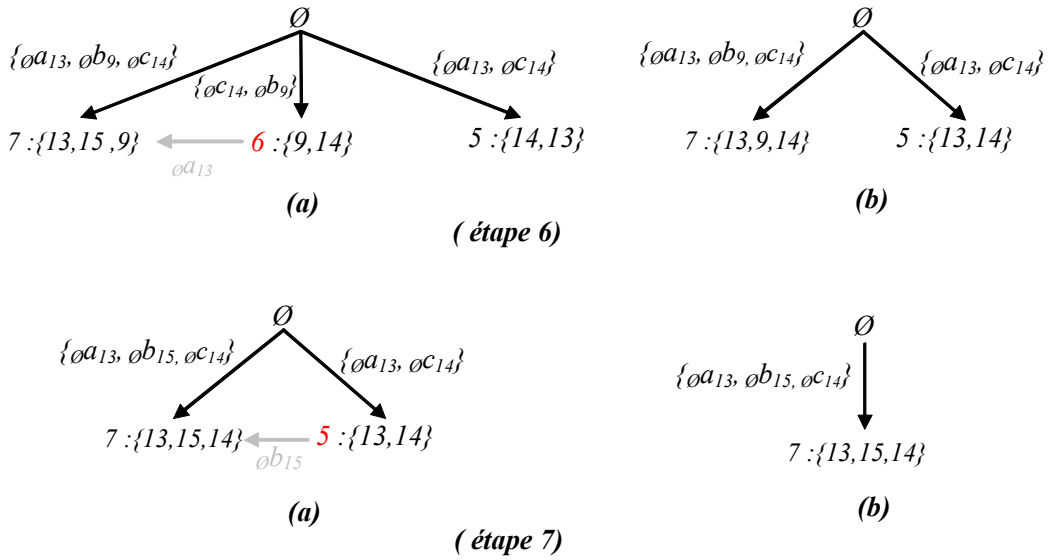


FIG. 4.10 – GPM en cours de génération "*suite*"

4.3 Conclusion

Ce chapitre présente notre contribution à la résolution du problème de l'explosion combinatoire du graphe d'états. Précisément, nous avons proposé deux méthodes de réduction. La première cherche à éliminer, dans un STEM donné, les redondances module la $\alpha-$équivalence, nous constatons que le taux de réduction est relativement considérable. Dans la deuxième méthode, nous avons abordé la notion de pas couvrant dans les systèmes de transitions étiquetées maximales, cela nous permet de répondre aux limites des approches d'ordre partiel présentées dans le Chapitre 3, en effet, au niveau de ces approches il y a des cas où nous ne pouvons pas calculer toutes les relations d'indépendance du système à analyser, en plus, les relations d'indépendance calculées ne sont pas forcement toutes prises en considération.

L'étude de complexité des algorithmes proposés n'est pas été évoquée dans ce mémoire, en effet l'objectif de ce présent travail est de voire l'impacte de ce deux méthodes sur la taille du graphe d'états obtenu par apport à celle du graphe initial, de cela vient la nécessité d'avoir des outils implantant ces deux méthodes ; et comme la durée réservée à la réalisation de ce mémoire est relativement courte, nous avons préféré de développer seulement des outils prototypes. Donc l'étude de complexité est laissée aux futurs travaux au but de proposer des outils efficaces.

Chapitre 5

Resultats

Ce chapitre introduit deux études comparatives, cherchant d'exposer d'une manière expérimentale l'apport de nos méthodes de réduction proposées.

5.1 Apport de STEM$_{/\approx_\alpha}$

Au but de montrer, par des exemples, le gain obtenu en utilisant la $\alpha-$réduction, nous présenterons dans cette section, une étude comparative de STEMs $\alpha-$réduits avec les STEMs classiques (générés directement par les règles de la sémantique opérationnelle de maximalité), les STEs et la ST-sémantique [GV87].

Les outils utilisés dans cette section sont :

1. `FoCoVE-CCS`$_{v_{1.0}}$: outil développé dans [BB01], outil permet de générer un STE à partir d'une spécification CCS donnée.

2. `FoCoVE-LotoSTEM`$_{v_{1.0}}$: outil développé dans [BDS02]. Il nous offre la possibilité d'avoir un STEM classique à partir d'une spécification LOTOS donnée.

3. `FoCoVE-LotoSTEM`$_{\mathtt{V}_{2.0}}$: la version `2.0` de `LotoSTEM`[BS05b] a été développé pour générer à la volée un STEM $\alpha-$réduit à partir d'une spécification LOTOS.

5.1.1 Comparaison avec les STEMs classiques

Cette étude consiste d'étudier le ratio $Taux = Taille(STEM\ classique)/Taille(STEM\ \alpha - réduit)$ en augmentant le degré de parallélisme (Tableau 5.1). La Figure 5.1 montre d'une façon assez claire le bénéfice de la $\alpha-$réduction en fonction du degré de parallélisme, où le taux de réduction est relativement grand (on peut même dire d'ordre exponentiel).

Où
$A \equiv a; A$, $B \equiv b; B$, $C \equiv c; C$, $D \equiv d; D$ et $E \equiv e; E$

	STEM classique		*STEM* $\alpha - réduit$		Taux		*STE*	
	états	Trns	états	Trns	états	Trns	états	Trns
Deux sous-processus $P_1 \equiv A\|\|\|B$	5	10	4	8	1.25	1.25	1	2
Trois sous-processus $P_2 \equiv A\|\|\|B\|\|\|C$	16	48	8	24	2	2	1	3
Quatre sous-processus $P_3 \equiv A\|\|\|B\|\|\|C\|\|\|D$	65	260	16	64	4.06	4.06	1	4
Cinq sous-processus $P_4 \equiv A\|\|\|B\|\|\|C\|\|\|D\|\|\|E$	326	1630	32	160	10.18	10.18	1	5

TAB. 5.1 – *Etude comparative avec les STEMs classiques et avec les STEs*

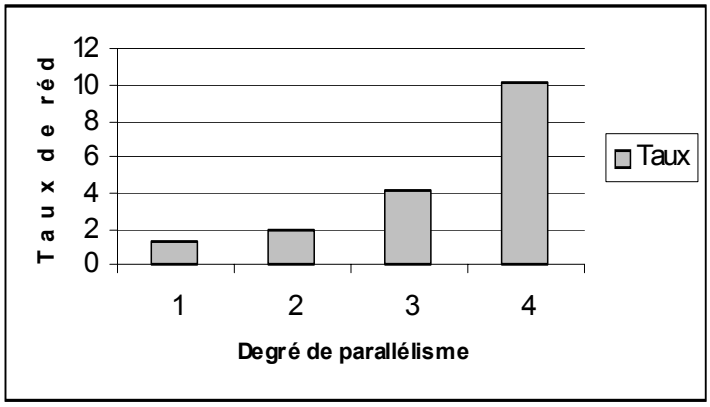


FIG. 5.1 – Taux de réduction en fonction du degré de parallélisme

5.1.2 Comparaison avec les STEs

Dans le Tableau 5.1, il est bien clair que la taille des STEMs est plus importante que celle des STEs correspondants, cela est évident, car les STEs ont moins d'informations à représenter dans le sens où ils ne distinguent pas les actions indépendantes. En revanche les STEMs sont de modèle du vrai parallélisme, où on a besoin de plus d'informations pour représenter le parallélisme, ce qui est traduit par l'introduction explicite des événements maximaux, en conséquence, on a plus d'états à générer, donc la comparaison ne peut être effectuée que de point de vue pouvoir d'expression. Effectivement, la Table 5.2 expose cette idée, car au niveau du STE de P_2 il n'y a aucune information sur le parallélisme, en d'autres termes on ne peut pas analyser l'exécution parallèle de a, b et c.

5.1.3 Comparaison avec la ST-sémantique

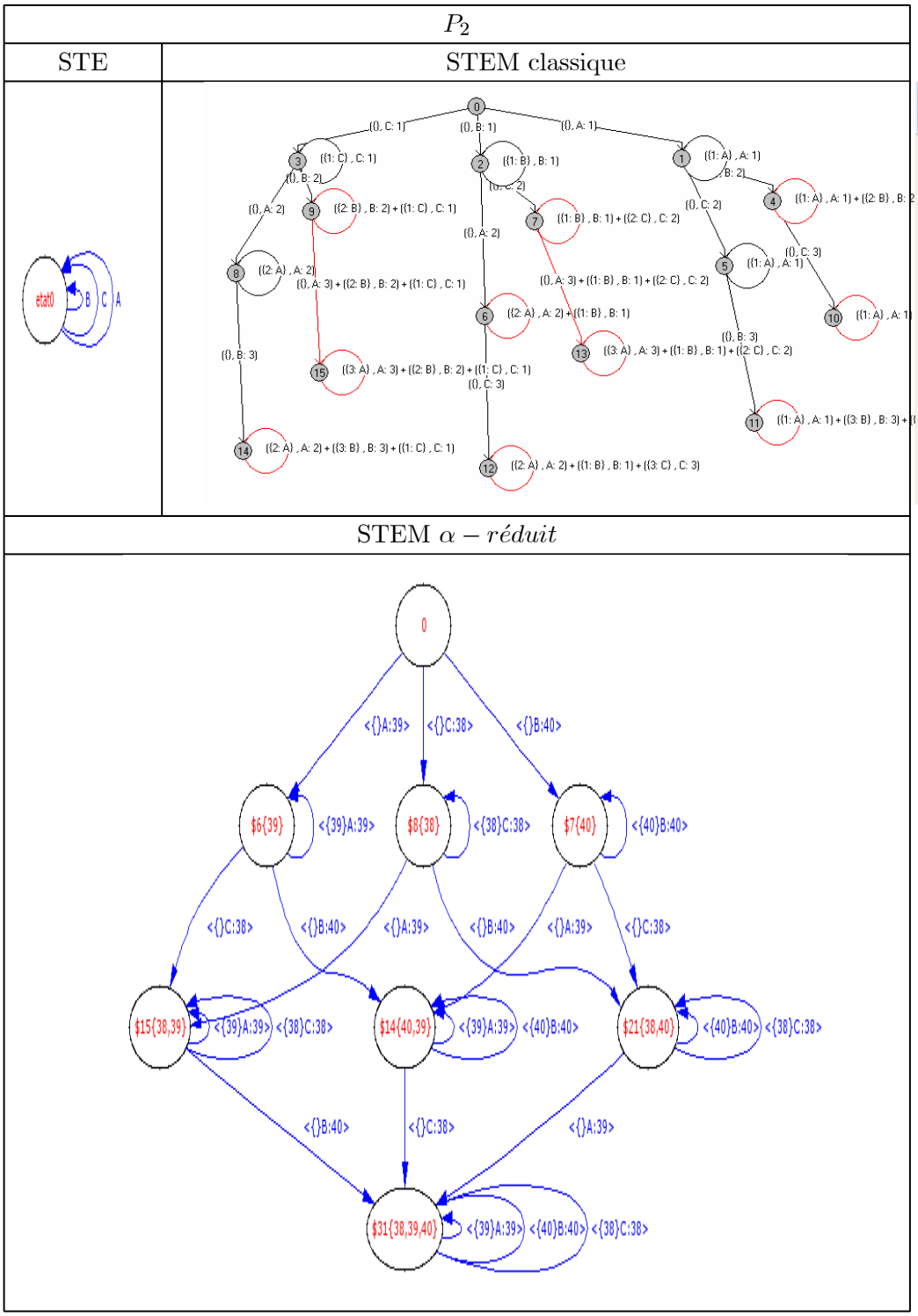
Afin de bien juger la $\alpha-$réduction, la comparaison doit être faite avec un modèle du vrai parallélisme, l'une des sémantiques les plus intéressantes vis-à-vis de l'hypothèse d'atomicité des actions est la ST-sémantique [GV87], c'est un modèle qui représente la non atomicité de l'action par l'exécution séquentielle de deux actions atomiques, qui déterminent respectivement le début et la fin de cette action, en plus, il a été montré qu'il est équivalent au maximalité [Vog91]. Dans ce qui suit, on va comparer la taille de ces deux modèles selon deux stratégies, dans la première stratégie, nous augmentons le nombre d'actions (voir Tableau 5.3) et dans la deuxième, nous augmentons le degré de parallélisme (Voir Tableau 5.5)

Où

- $A'_1 \equiv a_{1_{début}}; a_{1fin}; Stop$, et
- $A'_2 \equiv a_{1_{début}}; a_{1_{fin}}; a_{2_{début}}; a_{2_{fin}}; Stop, \ldots$
- $B'_1 \equiv b_{1_{début}}; b_{1_{fin}}; Stop$,
- $B'_2 \equiv b_{1_{début}}; b_{1_{fin}}; b_{2_{début}}; b_{2_{fin}}; Stop, \ldots$
- $A_1 \equiv a_1; Stop$,
- $A_2 \equiv a_1; a_2; Stop, \ldots$
- $B_1 \equiv b_1; Stop$,
- $B_2 \equiv b_1; b_2; Stop, \ldots$

À titre d'illustration, nous donnons dans le Tableau 5.4 le comportement de P'_1 et P_1.

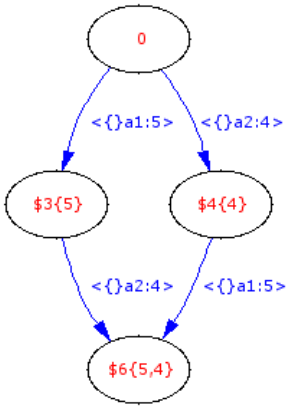
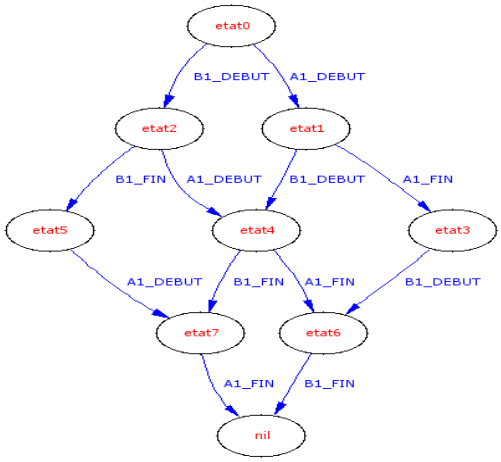
La Figure 5.2 et la FIgure 5.3 montrent que la taille de STEM α-réduit est intéressante devant celle de la ST-sémantique, soit on augmente le nombre d'actions ou soit on augmente le degré de parallélisme.



TAB. 5.2 – *le STEet le STEM de P2*

ST-sémantique	états	Transitions	Maximalité avec $\alpha - réduction$	états	Transitions
Uneseule action $P_1 \equiv A'_1 \|\|\| B'_1$	9	12	Uneseule action $P_1 \equiv A_1 \|\|\| B_1$	4	4
Deux actions $P_2 \equiv A'_2 \|\|\| B'_2$	25	40	Deux actions $P_2 \equiv A_2 \|\|\| B_2$	9	12
Trois actions $P_3 \equiv A'_3 \|\|\| B'_3$	49	84	Trois actions $P_3 \equiv A_3 \|\|\| B_3$	16	24
Quatre actions $P_4 \equiv A'_4 \|\|\| B'_4$	81	144	Quatre actions $P_4 \equiv A_4 \|\|\| B_4$	25	40
Cinq actions $P_5 \equiv A'_5 \|\|\| B'_5$	121	220	Cinq actions $P_5 \equiv A_5 \|\|\| B_5$	36	60

TAB. 5.3 – *Etude comparative avec la ST-sémantique "1"*

P_1 en STEM	P'_1 en ST-sémantique
	

TAB. 5.4 – *Le graphe de P'1 et P1*

ST-sémantique	états	Transitions	Maximalité avec $\alpha - réduction$	états	Transitions
Deux sous-processus $P1 \equiv A'_3 \|\|\| B'_3$	49	84	Deux sous-processus $P1 \equiv A_3 \|\|\| B_3$	16	24
Trois sous-processus $P_2 \equiv A'_3 \|\|\| B'_3 \|\|\| C'_3$	343	882	Trois sous-processus $P_2 \equiv A_3 \|\|\| B_3 \|\|\| C_3$	64	144
Quatre sous-processus $P_3 \equiv A'_3 \|\|\| B'_3 \|\|\| C'_3 \|\|\| D'_3$	2401	8232	Quatre sous-processus $P_3 \equiv A_3 \|\|\| B_3 \|\|\| C_3 \|\|\| D_3$	256	768

TAB. 5.5 – *Etude comparative avec la ST-sémantique "2"*

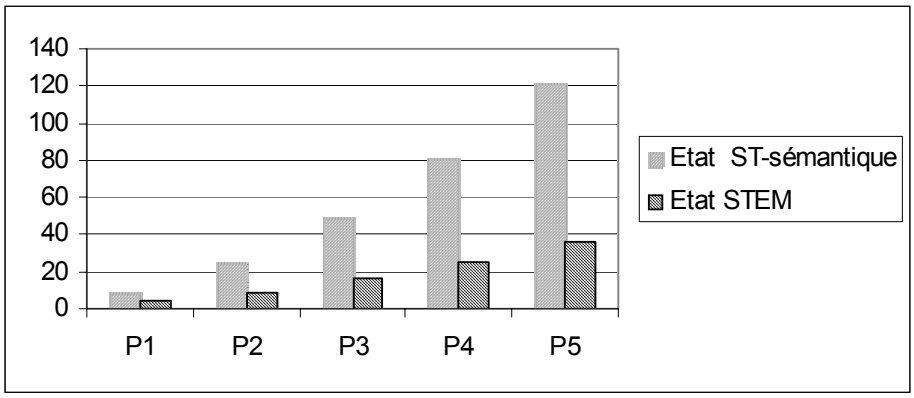


FIG. 5.2 – Le nombre d'états en fonction de nombre d'actions

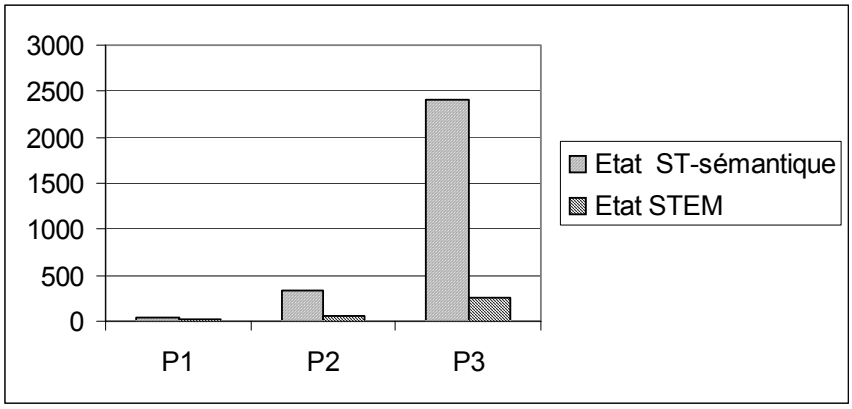


FIG. 5.3 – Le nombre d'états en fonction du degré de parallélisme

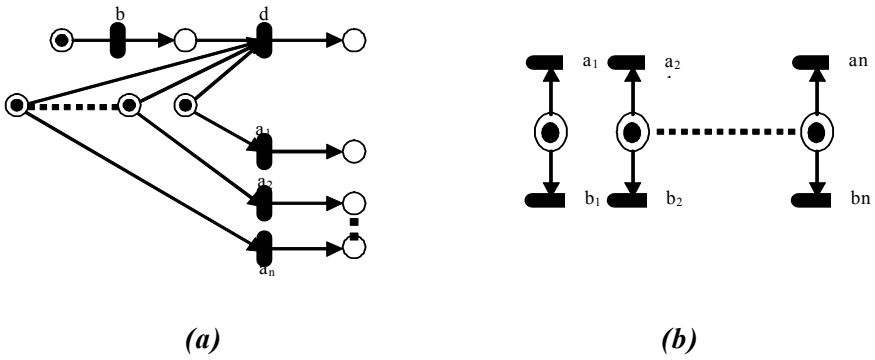


FIG. 5.4 – Les systèmes à étudier

	Graphe d'états		GPC		GP		GPP		STEM		GPM	
n	Trns	états	Trns	états	Trns	états	Trns	états	Trns	états	Trns	états
2	13	9	13	9	10	8	6	6	13	9	3	4
3	33	17	33	17	20	13	14	10	33	17	3	4
4	81	33	81	33	42	22	34	18	81	33	3	4
5	193	65	193	65	92	39	82	34	193	65	3	4

TAB. 5.6 – *Des résultats pour le système (a)*

5.2 Apport de GPM

Nous présentons dans cette section deux systèmes à étudier dans le but de confirmer le fait qu'il est très difficile de connaître au préalable quelle est l'approche d'ordre partiel la plus efficace. Cette étude consiste à comparer la taille des GPMs et le taux de réduction avec ceux des autres approches citées dans le chapitre 3.

Les outils utilisés sont :

1. `Tina` [TIN] : pour générer les graphes de pas couvrants « GPC », les graphes persistants « GP » et les graphes de pas persistants « GPP ».
2. $\texttt{LotoSTEM}_{\texttt{V}_{2.0}}$ [SB04] : pour générer les STEMs
3. `LotosGPM`[BS05b] : pour générer les GPMs.

Le système de la Figure 5.4.(a) présente un cas où les GPMs sont plus efficaces que les autres graphes. Le tableau 5.6 résume les résultats obtenus en nombre de transitions et en nombre d'états selon le nombre de transitions pouvant être tirées en parallèle (n). Nous constatons que la taille du GPM reste la même quelque soit la valeur de n.

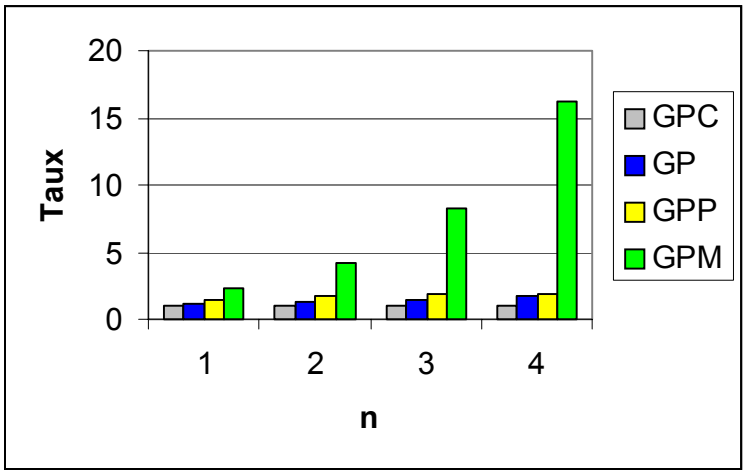


FIG. 5.5 – Taux de réduction pour les états (premier système)

	Graphe d'états		GPC		GP		GPP		STEM		MSG	
n	Trns	états	Trns	états	Trns	états	Trns	états	Trns	états	Trns	états
2	8	4	4	2	8	4	4	2	12	9	4	5
3	24	8	8	2	16	7	8	2	54	27	8	9
4	64	16	16	2	26	11	16	2	216	81	16	17
5	160	32	32	2	38	16	32	2	810	243	32	33

TAB. 5.7 – *Des résultats pour le système (b)*

La Figure 5.5 exprime l'avantage de l'approche des graphes de pas maximaux. Incontestablement, le taux de réduction obtenu par cette approche est largement supérieur à celui des autres approches.

En revanche, pour le système de la Figure 5.4.(b) nous avons remarqué que la taille d'un GPM est largement inférieure à celle du STEM correspondant ; cependant, elle est plus importante que celle des autres graphes. En dépit de ces résultats(Tableau 5.7 et Figure 5.6), les GPMs restent relativement privilégiés car ils représentent implicitement plus d'informations sur l'exécution parallèle des actions.

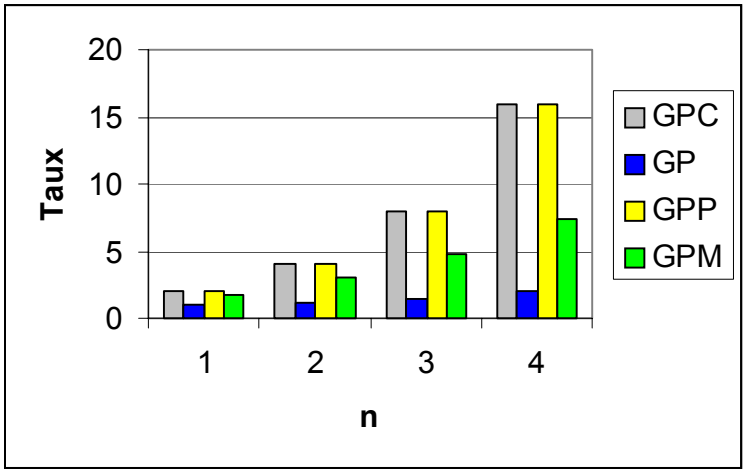


FIG. 5.6 – Taux de réduction pour les états (deuxième système)

Chapitre 6

Implémentation

6.1 L'environnement FoCoVE

FoCoVE est une boîte à outils pour la compilation, la vérification et la validation de programmes LOTOS. La conception de l'architecture globale de FoCoVE a été élaborée par Dr. Djamel-Eddine SAIDOUNI du département d'informatique de l'université de Constantine - Algérie, en Août 2000.

FoCoVE intègre un ensemble cohérent d'outils permettant de traiter des spécifications comportementales et logiques, en utilisant les méthodes basées sur les modèles. L'environnement comprend divers outils6.1, nous pouvons les classer en deux :

Les compilateurs : utilisés en amont, leur rôle est de traduire le programme à vérifier vers un modèle sémantique du parallélisme. FoCoVE contient troix compilateurs :

- ICC [BB01](Interleaving CCS Compiler) : En entrée, il prend le programme CCS à vérifier et il nous fournit comme sortie un STE.

- ILC [BBB01](Interleaving LOTOS Compiler) : nous permet d'obtenir un STE à partir d'une spécification LOTOS.

- LotoStem[BDS02] : au niveau de ce compilateur, nous avons adopté le modèle des arbres maximaux, ces arbres son dérivés à partir d'une spécification LOTOS

les vérificateurs : utilisées en aval, ils effectuent des vérifications sur les graphes engendrés par les compilateurs. FoCoVE couvre les quatre types de vérifications :

- la vérification comportementale : BSTE[BB01] et DBRE[Bel02] deux outils permettent respectivement la réduction et la comparaison de STEs par rapport à des relations d'équivalence comportementales à savoir la bissimulation forte et faible

- la vérification logique : LotoStem contient un model-checker qui permet de vérifier des propriétés exprimées en logique temporelle arborescente (CTL) sur le STEM généré.

- la vérification symbolique : `MSMC` [Lab02](Maximality based Symbolic Model Checking), il prend comme arguments une spécification de système à vérifier modélisé par un système de transitions étiquetées maximal, et une spécification des propriétés à vérifiées exprimées en logique temporelle arborescente. L'implémentation de cet outil est basée sur l'utilisation des diagrammes de décision binaire (BDDs) qui permettent d'éviter la construction explicite du graphe d'états.

- la vérification par test : `TEST`[DG02] est un outil de mise en oeuvre de l'approche test, apportant une certitude mathématique par génération de tests de conformité sous forme d'une structure appelée : graphe de refus, et par extension et adaptation du calcul de bissimulation pour la vérification des relations de conformité d'une implémentation par rapport à sa spécification.

La boîte à outils `FoCove` fournit aussi deux outils permettant la visualisation graphique du STE (`STEViewer` [BBB01]) et du STEM (`STEMViewer`[BDS02]).

6.1.1 Nouveautés de `FoCoVE`

Dans ce présent travail, nous avons enrichi l'environnement `FoCoVE` par deux outils compilateur/vérificateur (Figure 6.2) à savoir $\texttt{LotoSTEM}_{V_{2.0}}$ et `LotosGPM`. Les deux outils sont une contribution à la maîtrise du problème de l'explosion combinatoire du graphe d'états selon deux approches. Le premier outil cherche à générer à la volée un STEM réduit dont la vérification consiste à détecter et supprimer les états $\alpha-$équivalents. `LotosGPM` est un outil d'ordre partiel, il nous fournit à la volée un graphe qui couvre le STEM initial modulo l'équivalence de traces de Mazurckiewicz.

Afin de réutiliser les éditeurs graphiques existants tels que l'éditeur graphique de `Tina` et l'outil `DOT`, nous avons développé un composant assurant la transformation vers le format spécifique à chacun de ces éditeurs.

Description : L'outil `Dot` a été développé par AT&T Labs, au sein du projet GraphViz (http ://www.research-att.com/sw/tools/graphviz/), c'est un ensemble de logiciels libres qui permettent le calcul et l'affichage de graphes.

L'objectif principal du développement de $\texttt{LotoSTEM}_{V_{2.0}}$ et `LotosGPM` est la nécessité de prouver de façon expérimentale l'impacte des deux approches présentées dans le Chapitre 4 sur le taux de réduction. En conséquence, le développement d'un outil prototype fait la faire. Dans ce sens, nous allons dans ce qui suit présenter l'implémentation prototype de ces deux outils. OCaml c'est le langage de programmation choisi comme langage de cette implémentation, avant de plonger dans les détails de l'implémentation, une brève introduction sur OCaml est donnée .

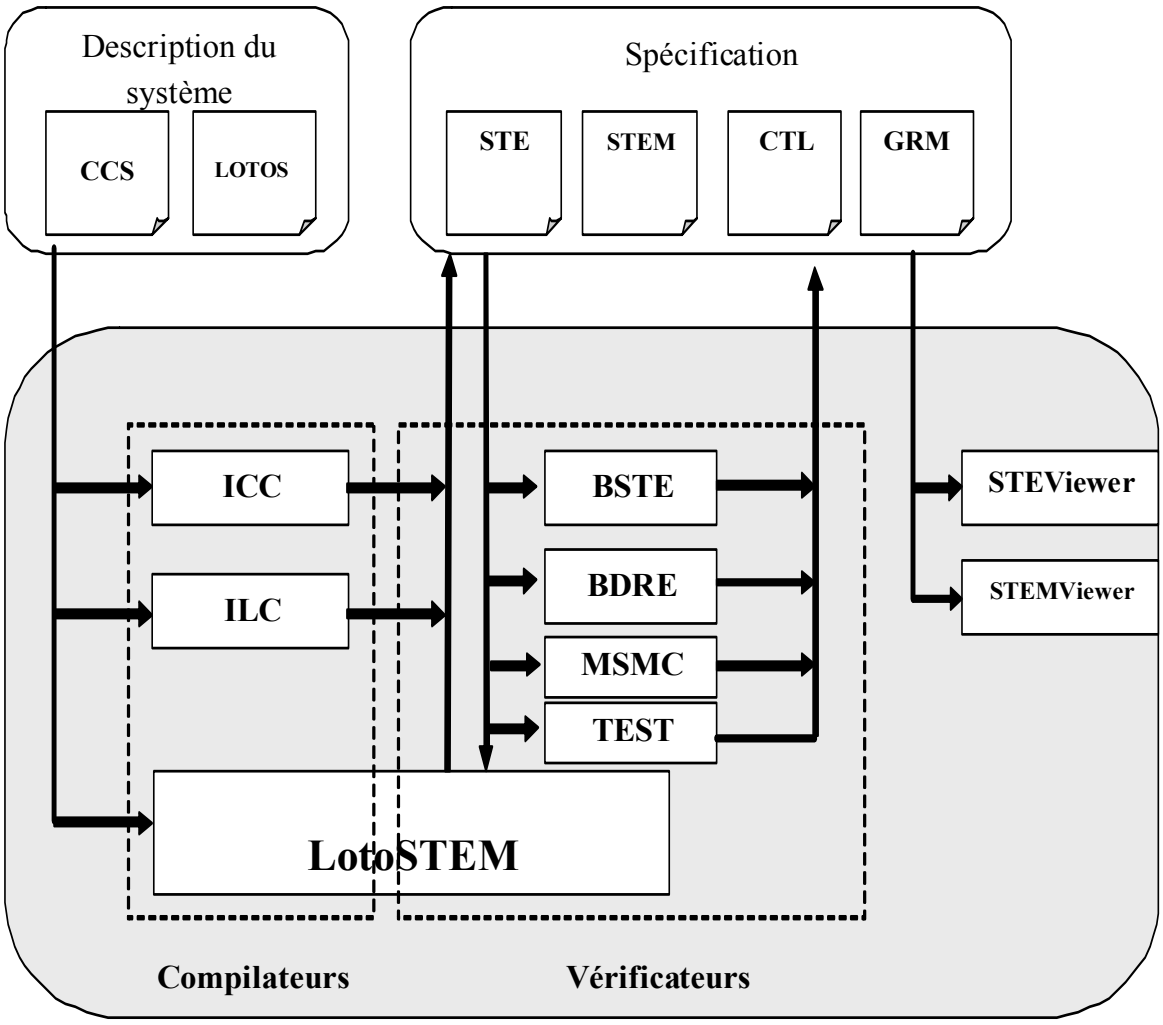


FIG. 6.1 – Architecture Globale de FoCoVE

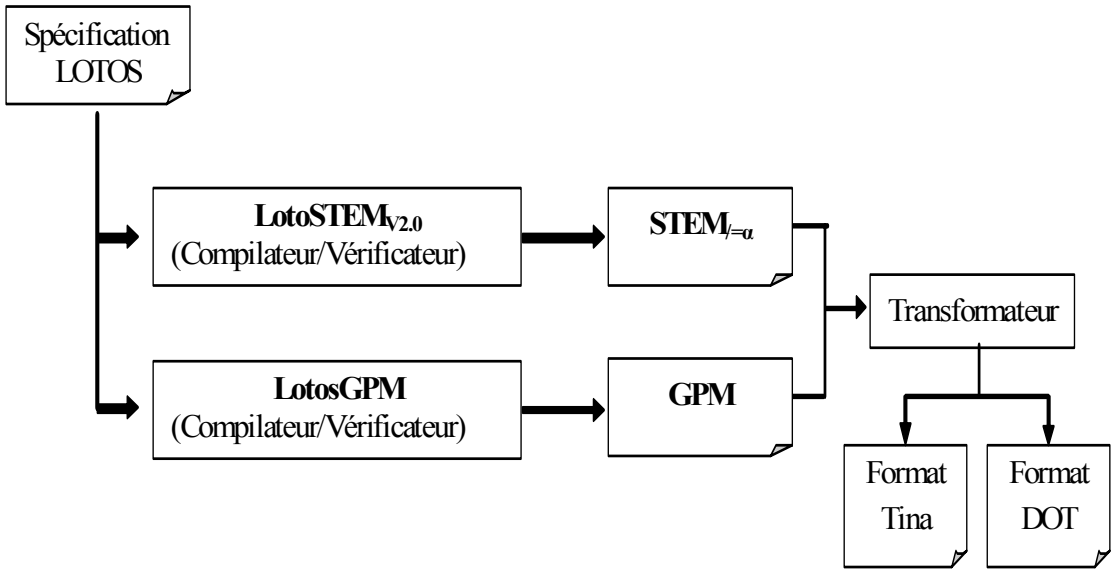


FIG. 6.2 – Nouveautés de FoCoVE

OCaml "Objective Caml"

OCaml est une implémentation du langage ML, basée sur la syntaxe du Caml light, avec un système d'objets basé sur les classes, et un système de modules proche de celui de Standard ML. Il se présente comme un lointain successeur de Lisp. Il est développé à l'INRIA.

OCaml possède les caractéristiques suivantes :

- OCaml est un langage fonctionnel, qui manipule les fonctions comme étant des valeurs de langage, pouvant être utilisées en tant que des paramétres d'autres fonctions ou être retournées comme résultats d'un appel de fonctions.

- C'est un langage typé statiquement : la vérification de compatibilité entre les types des paramétres formels et des paramétres d'appel est effectuée au moment de la compilation, ce qui fait accroître l'efficacité et la sûreté de l'exécution.

- C'est un langage polymorphe paramétrique : une fonction peut avoir un paramètre non entièrement déterminé, ce qui permet de développer un code générique utilisable pour des structures de données différentes.

- Il est muni d'un mécanisme d'exception, qui permet de rompre l'exécution d'un programme à un endroit et de le reprendre à un autre.

- Un programme OCaml peut se structurer à partir de deux modèles : le modèle des modules paramétrés et le modèle objets.

OCaml est un langage sûr, d'abord grâce à son typage fort, par son récupérateur automatique de mémoire, et enfin grâce au mécanisme d'exception. OCaml muni d'une syntaxe

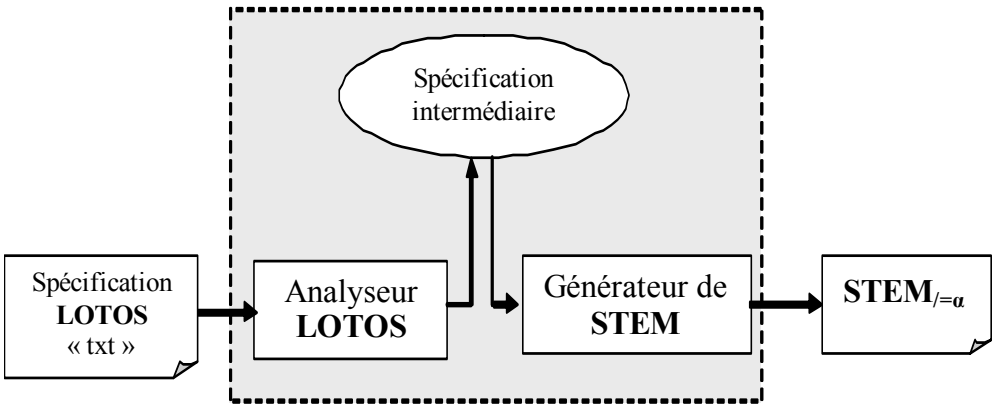


FIG. 6.3 – La fonction `LotoSTEM`

simple et intuitive, facile à prendre. Tous cela nous a conduit à juger OCaml comme le langage idéale pour nos besoins

6.2 L'outil $\texttt{LotoSTEM}_{v_{2.0}}$

Dans cette section, nous allons présenter l'implémentation de $\texttt{LotoSTEM}_{v_{2.0}}$ comme une fonction (Figure 6.3) à un seul paramètre d'appel qui est une spécification LOTOS et un seul paramètre de sortie représentant un STEM réduit en format textuel.

La Figure 6.3 montre que la fonction LotoSTEM est constituée de deux fonctions principales : la fonction Analyseur et la fonction Générateur, dont la sortie de la première fonction est une entrée de la deuxième.

La fonction `Analyseur` est une réutilisation de la pile Analyseurs (Analyseur Lexical, Analyseur Syntaxique et Analyseur Sémantique) développée dans [BDS02] en ajoutant une sous-couche qui transforme la sortie de cette pile vers une spécification LOTOS intermédiaire écrite en code OCaml. La Table 6.1 présente le type de spécification LOTOS en OCaml.

Au but de rendre simple et intuitive la présentation de l'implémentation de $\texttt{LotoSTEM}_{v_{2.0}}$, nous allons procéder par raffinement successif, où à chaque étape nous détaillerons les composants jugés essentiels.

Générateur de $\textbf{STEM}_{/=_\alpha}$

La fonction `Générateur` est le noyau de `LotosSTEM`, en effet c'est elle qui a la tache de dériver à partir d'une spécification intermédiaire un STEM réduit. Le `Générateur` est l'implémentation de l'algorithme 4.1. Selon ce dernier, il existe trois fonctions principales (Figure 6.4) :

- Caml_STEM : nous permet de dériver à partir d'une configuration donnée des transitions et des configurations.

```
type
action=string
and appel_Effectif=action list
and paramètres_Formels=action list
and id_Processus=int
and expression=                    (*  Structure Récursive *)
Stop
| Exit
| Préfixe of  action*expression
| Hide of  action list*expression
| Choix of  expression*expression
| Parallélisme of  expression*expression*action list
| Séquence of  expression*expression
| Interruption  of expression*expression
| Appel of  appel_Effectif*id_Processus
and processus=id_Processus*paramètres_Formels*expression
and système=processus list ; ;
```

TAB. 6.1 – *Type de spécification LOTOS en OCaml*

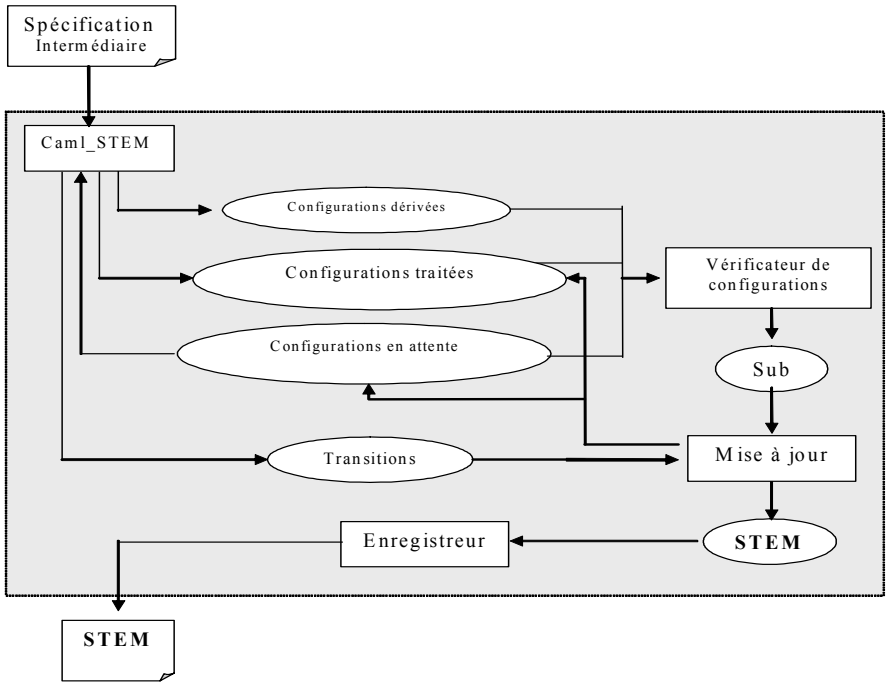


FIG. 6.4 – Raffinement du Générateur

- Vérificateur : fonction cherche pour chaque configuration qui appartient à l'ensemble Configurations dérivées si elle existe une configuration générée précédemment (l'ensemble configuration en attente et l'ensemble configuration traitées).

- Mise à jour : elle substitue les ensembles : transitions, configurations en attente et configurations traitées en utilisant l'ensemble *Sub*.

Au niveau de cette fonction nous avons manipulé trois structures de données : l'ensemble de configurations, Sub et un STEM. La Table 6.2 donne en OCaml le type de ces trois structures.

Remarque 6.1 *('a,'b,'c)graphe est un type paramétré, la raison de ce choix se résume sous le fait que la fonction Générateur définie pour* `lotoSTEM` *est presque la même utilisée dans l'outil* `LotosGPM`*, la seul différence réside dans le type STEM et le type GPM, et comme les deux sont des systèmes de transitions étiquettes, et au but de réutiliser cette fonction nous avons paramétré le type graphe.*

```
type
action_Max=action*int
and configuration=                          (*  Structure Récursive *)
StopConf  of action_Max list
| ExitConf  of action_Maxlist
| PréfixeConf of action_Max list*action*expression
| HideConf  of action list*configuration
| ChoixConf  of action_Max list*expression*action_Max list*expression
| ParallélismeConf  of configuration*configuration*action list
| SéquenceConf   of configuration*configuration
| InterruptionConf  of configuration*configuration
| AppelConf  of action_Max list*appel_Effectif*id_Processus
and état=int
and élément_Configuration=configuration*état
and liste_Configuration=élément_Configuration list
and ('a,'b,'c)graphe=('a*'b*'c)list
and sub=((action*int) list *(action* int)) list;;
```

TAB. 6.2 – *Types configuration, graphe et sub*

```
Let Caml_STEM(élément_Configuration ⟶ ('a,'b,'c)graphe * liste_Configuration)=
    fun (conf) ⟶
        match conf with
                (StopConf(max),état)->([],[])
               |(ExitConf(max),_,_)->tirer_action_exit(conf)
               |(PréfixeConf(_,_,_),_,_)->tirer_action_préfixe(conf)
               |(ChoixConf(_,_,_,_),_,_)->tirer_action_choix(conf)
               |(ParallélismeConf(_,_,_),_,_)->tirer_action_parallèlisme(conf)
               |(SéquenceConf(_,_),_,_)->tirer_action_séquence(conf)
               |(InterruptionConf(_,_),_,_)->tirer_action_interruption(conf)
               |(AppelConf(_,_,_),_,_)->tirer_action_appel(conf)
               |(HideConf(_,_),_,_)->tirer_action_hide(conf)
);;
```

TAB. 6.3 – *La fonction CamlSTEM*

Caml_STEM

Caml_STEM est une fonction permet de dériver à partir d'une configuration donnée un ensemble de configurations et un ensemble de transitions. La Table 6.3 donne le schéma général de cette fonction dont les fonctions **tirer_action_exit**, **tirer_action_préfixe**, **tirer_action_choix**, ..., **tirer_action_hide** représentent l'implémentation des règles de la sémantique opérationnelle de maximalité (Définition 2.6).

Vérificateur

Cette fonction consiste la deuxième étape du calcul, le Vérificateur est une fonction récursive (Table 6.4) qui cherche d'éliminer chaque configuration, appartienne à l'ensemble de configurations dérivées, jugée $\alpha-$équivalente avec une autre configuration générée précédemment. Les configurations résultantes constituent l'ensemble des nouvelles configurations. Pour ne pas compliquer la présentation, nous exposons seulement la sous-fonction **alpha_équivalence** qui implémente la définition de la $\alpha-$équivalence (Définition 4.1).

La fonction alpha_équivalence nous fournit la liste de substitution en comparant syntaxiquement deux configurations données. Pour explique ceci, prenons comme exemple $conf = ((\{(a,4)\},"a",Stop),10)$ et supposons $conf' = ((\{(a,8)\},"a",Stop),20)$ une configuration générée antérieurement, selon la Table 6.5 $conf$ et $conf'$ sont équivalentes, en effet, $conf.action = conf'.action = "a"$ et $conf.exp = conf'.exp = Stop$ dont la correspondance $sub = \{(a,4),(a,8)\}$ est possible, où l'identificateur de $conf$ est devenu 20.

```
Let rec vérificateur(liste_Configuration*liste_Configuration ⟶ sub)=
    fun (confs_dérivées,liste_Conf)  ⟶
        match confs_dérivées with
            []− > []
          | conf : :queue-> alpha_équivalence(conf,liste_Conf)
                            @vérificateur(queue,liste_Conf)
);;
```

TAB. 6.4 – *La fonction Vérificateur*

```
Let rec alpha_quivalence(élément_Configuration*liste_Configuration ⟶ sub)=
fun(conf,liste_confs)->
 match liste_confs with
    []− > []
| conf' : :queue->match conf' with
                  (StopConf(max),état)->stop(max,état,conf)
                 |(ExitConf(max),état)->exit(max,état,conf)
                 |(PréfixeConf(max,a,exp),état)->préfixe(max,a,exp,état,conf)
                 |(HideConf(act_liste,conf),état)->hide(act_liste,conf,état,conf)
                 |(ChoixConf(max1,exp1,max2,exp2),état)->choix(max1,exp1,max2,exp2,état,conf)
                 |(ParallélismeConf(c1,c2,act_liste),état)->parallèlisme(c1,c2,act_liste,état,conf)
                 |(SéquenceConf(c1,c2),état)->séquence(c1,c2,état,conf)
                 |(InterruptionConf(c1,c2),état)->interruption(c1,c2,état,conf)
                 |(AppelConf(max,apl_ef,nom),état)->appel(max,apl_ef,nom,état,conf)
);;
```

TAB. 6.5 – *La fonction alpha équivalence*

Les fonctions **stop**, **exit**, **préfixe**, .., **appel** (Figure 6.6 et Figure 6.7) sont introduites au but de :

- Vérifier l'égalité syntaxique entre les expressions qui constituent la configuration $conf$ avec celles de la configuration $conf'$
- Etablir une correspondance entre les événements de la première configuration avec ceux de la deuxième configuration, la correspondance doit être effectuée selon la structure statique de configurations.

où **égalité_max** une fonction logique donne vrai ssi il existe le même ensemble d'action dans max et max' en ignorant le nom des événements et **égalité_liste** une fonction logique donne vrai ssi il existe le même ensemble dans les deux listes.

Exemple 6.1 *pour mieux illustrer ces deux points, prenons comme exemple :*

- $conf = ParallélismeConf(\Pr éfixeConf([("a", 10)], "b", Stop),$
  $\Pr éfixeConf([("c", 24)], "d", Exit), ["b"])$
- *et* $conf' = ParallélismeConf(\Pr éfixeConf([("a", 12)], "b", Stop),$
  $\Pr éfixeConf([("c", 16)], "d", Exit), ["b"])$

l'appel $alpha\_quivalence(conf :: [], conf')$ *provoque dans l'ordre les appels suivants :*

1. $Parallélisme(\Pr éfixeConf([("a", 12)], "b", Stop),$
   $\Pr éfixeConf([("c", 16)], "d", Stop), ["b"], conf)$ *on a*
   $égalité\_listes = true$
2. $\Pr éfixe([("a", 12)], "b", Stop, \Pr éfixeConf([("a", 10)], "b", Stop))$ *on trouve*
   $égalité\_ \max = true$ *,*$a = a' = "b"$*,* $exp' = exp = Stop$ *et* $sub = [("a", 12); ("a", 10)]$
3. $\Pr éfixe([("c", 16)], "d", Exit, \Pr éfixeConf([("c", 24)], "d", Exit))$ *où*
   $égalité\_ \max = true$ *,*$a = a' = "d"$*,* $exp' = exp = Exit$ *et* $sub = [("c", 16); ("c", 24)]$

à la fin de cette série des appels, on obtient $sub = [[("c", 16); ("c", 24)]; [("a", 12); ("a", 10)]]$

```
let
stop(max',état')=match conf with
                    StopConf(max)->
                          if (égalité_max(max,max')) then
                             gréer_sub(max,max')
                          else []
                   | _ -> []
and exit(max',état')=match conf with
                         ExitConf(max)->
                            if (égalité_max(max,max')) then
                              gréer_sub(max,max')
                            else []
                       | _ -> []
and préfixe(max',a',exp',état')=match conf with
                                    PréfixeConf(max,a,exp)->
                                        if(égalité_max(max1,m2))
                                            &(a'=a)&(exp=exp) then
                                                gréer_sub(max,max')
                                         else []
                                       | _ -> []
and appel(max',effectif',nom',état')=match conf with
                                   AppelConf(max,effectif,nom1)->
                                     if(égalité_max(max',max))
                                        &(égalité_listes(apl_ef1,eff2))
                                        &(nom'=nom) then
                                         gréer_sub(max,max')
                                      else []
                                    | _ -> []
```

TAB. 6.6 – *Les fonctions stop, exit, prefixe et appel*

```
and hide(acts_liste',conf'1,état')=match conf with
                                    HideConf(acts_liste,conf1)->
                                            if égalité_listes(acts_liste',acts_liste) then
                                               alpha_quivalence((conf'1,0),(conf1,0) : :[])
                                             else []
                                  | _ -> []
and choix(max'1,exp'1,max'2,exp'2,état')=match conf with
                                                        ChoixConf(max1,exp1,max2,exp2)->
                                                         if(égalité_max(max'1,max1))
                                                           &(égalité_max(max'2,max2))
                                                           &(exp1=exp'1)&(exp2=exp'2) then
                                                                gréer_sub(max2,max'2)
                                                              @ gréer_sub(max1,max'1)
                                                           else []
                                                       | _ -> []
and parallélisme(c'1,c'2,acts_liste',état')=match conf with
                                                    ParallélismeConf(c1,c2,acts_liste)->
                                                      if égalité_listes(acts_liste',acts_liste) then
                                                         alpha_quivalence((c'1,0),(c1,0) : :[])
                                                        @ alpha_quivalence((c'2,0),(c2,0) : :[])
                                                       else []
                                                     | _ -> []
and séquence(c'1,c'2,état')=match conf with
                                        SéquenceConf(c1,c2)->
                                                 alpha_quivalence((c'1,0),(c1,0) : :[])
                                                @ alpha_quivalence((c'2,0),(c2,0) : :[])
                                      | _ -> []
and interruption(c'1,c'2,état')=match conf with
                                        InterruptionConf(c1,c2)->
                                                 alpha_quivalence((c'1,0),(c1,0) : :[])
                                                @ alpha_quivalence((c'2,0),(c2,0) : :[])
                                       | _ -> []
```

TAB. 6.7 – *Les fonctions choix,hide et parallélisme, exit et prefixe*

```
type
état= int and états=état list
and action=string
and action_Max=string* int
and événement=int and événements=événement list
and atm=action_Max list*action*événement and atms=atm list
and chemin=(état*action_Max list)*atm list*état
chemins=chemin list
('a,'b,'c)gpm=('a,'b,'c)graphe
```

TAB. 6.8 – *Type de Chemins maximaux et type de pas*

6.3 L'outil `LotosGPM`

Dans la section 4.2.4, nous avons montré que l'algorithme de génération d'un GPM est une extension directe de l'algorithme de génération d'un STEM $\alpha-$réduit, la différence réside dans la fonction **construire_ pas**. A partir de cela, la fonction `GénérateurGPM` est donnée comme dans la Figure 6.5. Assimilable à ce que nous avons fait dans la section précédente, nous allons présenter les sous-fonctions principales de la fonction construire_ pas et par la suite nous détaillerons chacune d'elles.

Selon la Figure 6.6 et la Table 6.9 la fonction construire_ pas est une fonction récursive qui contient les quatre sous-fonctions suivantes :

- Construire_chemins_max : fonction introduite au but de bâtir pour chaque état s, état antécédent de $conf$ ($conf$ la configuration en cours) l'ensemble de chemins maximaux C_s.

- Construire_petits_chemins : c'est l'implémentation de la définition de petit chemin (Définition 4.6).

- Transformer_petit_pas : fonction transforme l'ensemble de petits chemins à un ensemble de pas correspondant.

- Mise à jour : la mise à jour consiste de remplacer les transitions constituants un pas par ce dernier.

nous détaillerons dans ce qui suit la fonction Construire_chemins_max et la fonction onstruire_petits_chemins, mais avant cela , nous révélons les types de données utilisés qui sont le type de pas de transitions, le type de chemins maximaux et le type de GPM (Table 6.8)

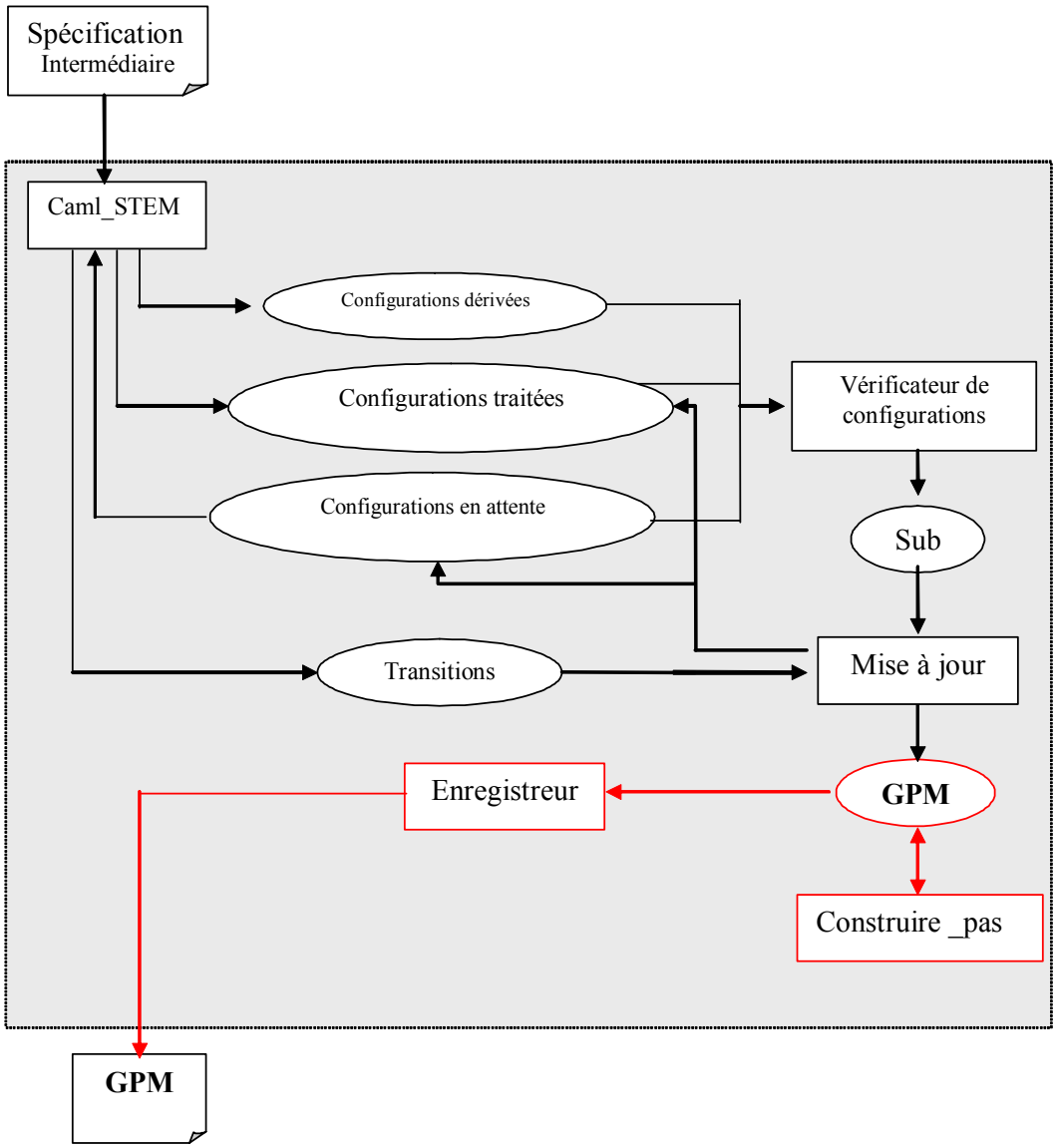


FIG. 6.5 – La fonction GénérateurGPM

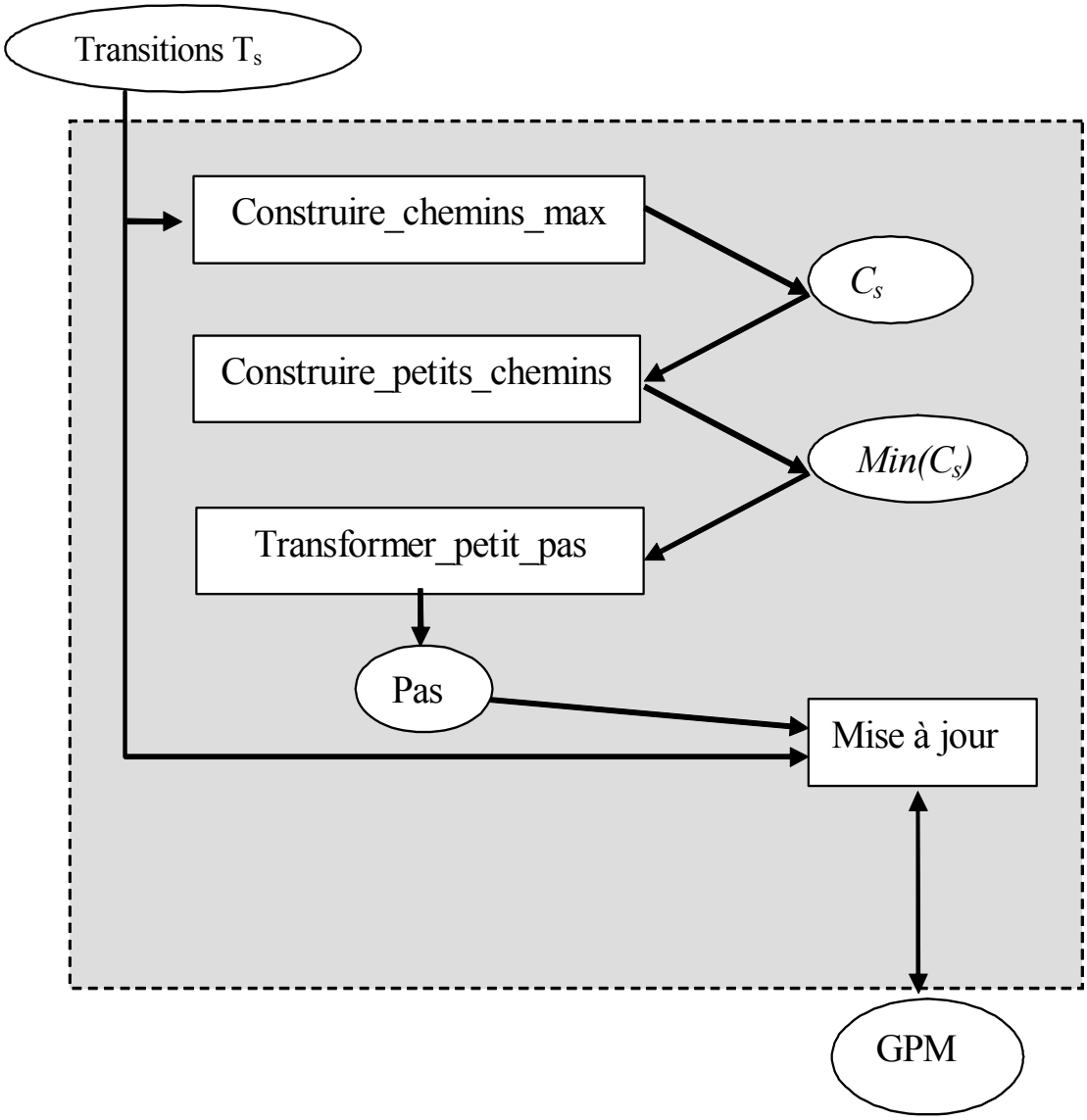


FIG. 6.6 – Raffinement de construire_ pas

```
let rec construire_ pas(états*('a,'b,'c)gpm->('a,'b,'c)gpm)=
fun(liste_antécédents,gpm)->
    match liste_antécédents with
          []->[]
        |s : :queue->let c=construire_chemins_max(s,gpm)
                     in( let petits=construire_petits_chemins(c)
                     in(let pas=transformer_petit_pas(petits)
                     in(construire_ pas(queue,Mise à jour(pas,gpm))
)));;
```

TAB. 6.9 – *construire pas en OCaml*

```
let rec construire_chemins_max(chemin*('a,'b,'c)gpm->chemins)=
fun(c,liste_transition)->
        match liste_transition with
            []->c : :[]
           |t : :queue->if psi(s') ∩ xi(t) <> [] then
                              c : :construire_chemins_max(c,queue)
                          else construire_chemins_max(ct,queue);;
```

TAB. 6.10 – *construire chemins maximaux*

Construire_chemins_max

Le calcul d'un chemin maximal est une fonction récursive qui basée sur le test suivant :

soit c un chemin en cours de construction (état s') où $c = ((s, événements\_max), liste\_atms, s')$

et soit $t \in liste\_transitions$, $t = ((s', événement\_max\_s'), atm\_t, s")$, c est un chemin maximal ssi :

$$psi(s') \cap xi(t) \neq \varnothing \ (psi(s')=\psi(s') \text{ et } xi(t) = \xi(t))$$

sinon on va faire le même calcul pour le chemin $ct = ((s, événements\_max), atm\_t :: liste\_atms, s")$. Le schéma global de la fonction Construire_chemins_max est donné dans la Table 6.10.

Construire_petits_chemins

cette fonction cherche à trouver un chemin $c = ((s, événements\_max), liste\_atms, s')$ qui appartient à l'ensemble C_s de façon qu'il n'existe pas un autre chemin $c' = ((s, événements\_max), liste\_atms', s")$ du même ensemble où : $inclus(trace\_max(liste\_atms'), trace\_max(liste\_atms)) = true$

inclus est une fonction d'inclusion sur les ensembles et $trace\_max$ implémente$|| < _ > ||$. La table 6.11 donne le shéma globale de Construire_petits_chemins.

```
let rec Construire_petits_chemins(chemins->chemins)=
fun(liste)->
let rec(chemins_inclus :chemin*chemins->chemin*chemins)=
fun(c,liste)->match liste with
                []->(c,[])
               |élément : :queue->if inclus(c,élément) then
                                    chemins_inclus(c,queue)
                                 else if inclus(élément,c) then
                                      chemins_inclus(élément,queue)
                                      else match chemins_inclus(c,queue) with
                                           (c1,reste)->(c1,élément : :reste)
in(   match liste with
      []->[]
     |c : :queue->match chemins_inclus(c,queue) with
                  (c1,reste)->c1 : :chemins_Mins(reste)
);;
```

TAB. 6.11 – *construire un ensemble de petits chemins maximaux*

6.4 Transformateur

Au but de réutiliser les éditeur graphiques existants, nous avons introduire la fonction Transformateur qui traduit les graphes obtenus par nos deux outils vers le format aut de Tina et vers le format Dot

6.4.1 Format Dot

les STEMs et les GPMs présentés dans le chapitre 4 comme des exemples sont tous générés en format dot de la Table 6.12. A titre d'illustration, le STEM de la Table 5.4 est généré à partir du code de la Table 6.13.

6.4.2 Format aut

L'outil DOT nous permet seulement la visualisation des STEMs et des GPMs, en revanche nous constatons que l'outil Tina va nous permet, plus à la visualisation des STEMs et des GPMs, la possibilité réutiliser ses analyseurs pour vérifier, par exemple, la présence des états de blocage. La Table 6.14 présente la format aut.

digraph G{
edge [color = blue,fontname = verdana,fontsize = 8, fontcolor = blue] ;
node [fontname = verdana, fontsize = 8, fontcolor = red] graphe }
où
graph->edge graph|node graphe|edge|node
edge-> état "->" état [label=sting] ;
node->état [label=string] ;
état->string

TAB. 6.12 – *Format Dot*

digraph G{
edge [color = blue,fontname = verdana,fontsize = 8, fontcolor = blue] ;
node [fontname = verdana, fontsize = 8, fontcolor = red]où
0->3[label="[<{}a1 :5>]"] ;
3[label="$3{5}"] ;
0->4[label="[<{}a2 :4>]"] ;
4[label="$4{4}"] ;
3->6[label="[<{}a2 :4>]"] ;
6[label="$6{4,5}"] ;
4->6[label="[<{}a1 :5>]"] ;}

TAB. 6.13 – *STEM en format Dot*

des(état_Initial,nb_transition,nb_états)graphe
où
graphe -> (état,string,état) graphe|(état,string,état)

TAB. 6.14 – *Format aut*

Begin(nb_transition,nb_états)graphe
où
graphe->grapheSTEM|grapheGPM
grapheSTEM -> transition ; grapheSTEM| transition
transition->liste_événements état"<" atm">" état
atm->liste_événements action événement
grapheGPM->pas ;grapheGPM|pas
pas->liste_événements état"<"[atms]">" état
atms->atm ;atms|atm

TAB. 6.15 – *Format STEM et format GPM*

Afin de terminer cette section nous présentons dans la Table6.15 le format STEM et le format GPM

		lignes	sous-fonctions
`Générateur`	Caml_STEM	640	47
	Vérificateur	218	17
	Mise à jourEnregistreur	49	5
fonctions générales		120	15
`construire_ pas`	Construire_chemins_max	60	10
	Construire_petits_chemins	40	6
	Transformer_petit_pas	30	3
	Mise à jour	70	9
Transformateur	vers Dot	55	7
	vers aut	40	6
	vers STEM	65	8
	vers GPM	60	10
	Total	1447	137

TAB. 6.16 – *Quelques statistiques sur le code source*

Chapitre 7

Conclusion et perspectives

Le travail présenté dans ce mémoire est une contribution à la résolution du problème de l'explosion combinatoire du graphe d'états. Précisément, nous avons proposé deux méthodes de réduction du graphe d'états. La première méthode cherche d'éliminer, dans un STEM donné, les redondances module la $\alpha-$équivalence qu'est une relation spécifique aux STEMs, entre autre, nous avons trouvé que le taux de réduction est relativement considérable. Dans la deuxième méthode, nous avons propulsé la notion de pas couvrant aux systèmes de transitions étiquetées maximales, cela nous permet de répondre aux limites des approches d'ordre partiel présentées dans ce mémoire, en effet, au niveau de ces approches il y a des cas où nous pouvons pas calculer toutes les relations d'indépendance du système à analyser, en plus, les relations d'indépendance calculées ne sont pas forcement toutes prises en considération.

Après une brève présentation de la technique de vérification formelle basée sur les modèles donnée dans le premier chapitre, nous avons présenté, dans le chapitre 2, la sémantique opérationnelle de maximalité de Basic LOTOS. La relation de la $\alpha-$équivalences et la bessimulation forte sont aussi données.

Dans le chapitre 3, Les graphes persistants, les graphes de pas couvrants et les graphes de pas persistants sont arborés, ce dernier type de graphe est une combinaison entre les deux premiers. Nous avons vu dans ce chapitre que malgré cette combinaison de plusieurs approches, il reste très difficile de connaître au préalable quelle est l'approche d'ordre partiel la plus efficace.

Dans le chapitre 4, nous avons proposé nos méthodes de réductions. Nous avons montré que la génération à la volée de graphes réduits est possible.

Pour compléter ce présent travail et pour le rendre plus expérimental, nous avons donné, dans le chapitre 5, quelques résultats en utilisant l'environnement `FoCoVE` et l'outil `Tina`.

Dans le chapitre 6, les algorithmes proposés au niveau du chapitre 4 sont implémentés dans l'environnement FoCoVE, en d'autre terme, nous avons enrichi l'environnement `FoCoVE` par deux outils : $\texttt{LotoSTEM}_{v2.0}$ et `LotosGPM`.

Perspectives

Ce travail est loin d'être achevé. Effectivement, il nous reste à proposer au niveaux des SETMs des algorithmes de réduction des systèmes de transitions étiquetées maximales, les réductions envisagées doivent préserver les relation de maximalité forte et faible.

En ce qui concerne les GPMs, les résultats obtenus dans le chapitre 6, nous donne la volonté de compléter ce travail en terme de préservation de propriétés spécifiques tels que l'équivalence observationnelle, la sémantique de refus,... etc. Il est intéressant aussi d'étudier les relations d'équivalence comportementale entre les GPMs, bien évidemment, combiner l'approche des graphes de pas maximaux avec les approches de la gestion de l'explosion combinatoire du graphe d'états tels que la vérification à la volée et les structures des BDDs, et l'extension de ceux-ci pour la prise en compte du temps comme il a été déjà fait pour les STEMs [BS05a][SB05].

Il reste enfin à étudier la complexité des algorithmes proposés dans le chapitre 4 au but d'enrichir l'environnement FoCoVE par des outils efficaces en temps de réponse et en espace mémoire

Bibliographie

[90788] ISO 9074. "Estelle, a Formal Description Technique Based on an Extended State Transition Model". ISO (November 1988).

[Arn92] A. Arnold. Systèmes de transitions finis et sémantique des processus communicants. (1992).

[BB87] T. Bolognesi and E. Brinksma. "Introduction to the ISO Specification Language LOTOS", volume 14. Computer Networks and ISDN Systems (1987).

[BB01] A. Benamira and A. Bouziane. Conception et implémentation d'un environnement de vérification pour basic CCS : Approche d'Entrelacement. Mémoire d'Ingénieur, Université Mentouri - Constantine (2001).

[BBB01] S. Boulkera, M. T. Bourekab, and C. Berdoudi. Conception et implémentation d'un environnement de vérification pour basic LOTOS : Approched'Entrelacement. Memoire d'Ingénieur, Université de Constantine (2001).

[BCM+90] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking : 10 e20 states and beyond. In "Proceeding of 5th IEEE Symposium on Logic in Computer Science" (1990).

[BDS02] N. Belala, A. Delimi, and M. N. Seghir. Développement d'un environnement deVérification formelle basée sur la sémantique de maximalité: Approche logique. Mémoire d'Ingénieur, Université de Constantine (2002).

[Bed87] M. A. Bednarczyk. "Categories of Asynchronous Systems". PhD thesis, Univ Sussex (1987 1987). Available as CS R 1/88.

[Bel02] Meriem Belguidoum. Conception et implémentation d'un outil pour la vérification formelle de relation de bissimulation entrelacée. Memoire d'Ingénieur, Université de Constantine (2002).

[BK85] J. A. Bergstra and J. W. Klop. Algebra of communicating processes with abstraction. *TCS* **37**, 77–121 (1985).

[BS05a] N. Belala and D. E. Saidouni. Non-atomicity in timed models. In "Proceedings of ACIT'2005" (December 2005).

[BS05b] A. Benamira and D. E. Saidouni. Contribution à la résolution de l'explosion du graphe d'état par l'utilisation conjointe des graphes de pas couvrants et de la sémantique de maximalité. Technical Report, , Laboratoire LIRE, Equipe PRAI, Université Mentouri, Constantine (Octobre 2005).

[CCI88] CCITT88. SDL, recommendation z.100-z.104. *CCITT* (1988).

[CE81] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In "Logics of Programs Workshop", volume 131, pages 52–71. LNCS (May 1981).

[CES83] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specification: A practical approach. In "10th ACM Symp. Principles of Programming Language", pages 117–126, Austin, Texas (January 1983).

[CS95] J. P. Courtiat and D. E. Saidouni. Relating maximality-based semantics to action refinement in process algebras. In "D. Hogrefe and S. Leue, Editors, IFIP TC/WG6.1, 7th Int. Cof of Formal Description Techniques(FORTE'94)", pages 293–308. Chapman Hall (1995).

[DG02] Linda Deraredj and Afifa Ghenai. Conception et implémentation d'un outil pour la vérification formelle selon l'approche test basée sur la sémantique d'entrelacement. Memoire d'Ingénieur, Université de Constantine (2002).

[EH86] E. A. Emerson and J. Y. Halpern. "sometims" and "not never" revisited: On branching versus linear time temporal logic. *Journal of the ACM* **33(1)**, 151–178 (1986).

[EM85] H. Ehrig and B. Mahr. "Fondamentals of Algebraic Specification". Springer-Verlag (1985).

[FM91] Jean-Claude Fernandez and Laurent Mounier. A toolset for deciding behavioral equivalences. In "CONCUR'91". LNSC (1991).

[God90] P. GodeFroid. Using partial orders to improve automatic verification methods. In "Proceedings of CAV'90", volume 3, pages 321–340. ACM, DIMACS (1990).

[GV87] R. J. Van Glabbeek and F. W. Vaandrager. Petri net models for algebraic theories of concurrency. In "PARLE'87, Vol II (Parallel Language)", volume 259 of LNCS, page 224 242. Spring-Varlag (1987).

[GW91] P. Godefroid and P. Wolper. A partial approach to model cheking. In "Proceedings 6th Symp. On Logic in Computer Science", volume 531, pages 406–415, Amsterdam (July 1991).

[GW93] P. Godefroid and P. Wolper. Using partial orders for the efficient verification of deadlock free-dom and safety properties. In "Formal Methods in Systems Design", pages 2(2):149–164 (April 1993).

[HM80] M. Hennessy and R. Miner. On observing nondeterminism and concurrency. In "Proc 7th ICALP", pages 299–309. Spriner-Verlag (July 1980).

[Hoa85] C. A. R. Hoare. "Communicating Sequenctial Processes". Prentice Hall (1985).

[ISO88] ISO8807. LOTOS, a formal description technique based on the ordering of observation behaviour. *ISO* (November 1988).

[Lab02] Ouassi Labbani. Conception et implémentation d'un outil pour la vérification symbolique basée sur la sémantique de maximalité. Memoire d'Ingénieur, Université de Constantine (2002).

[Maz86] A. Mazurckiewicz. Trace theory. in petri nets: Applications and relationships to other model of concurrency. In "Advances in Petri Nets 1986, Part II; Proceedings of an Advanced Course", pages 279–324. Springer Verlag, LNCS 255 (1986).

[MF76] P. M. Merlin and D. J. Farber. Recoverability of communication protocols: Implications of theoretical study. *IEEE Trans. Comm* **24(9)**, 1036–1043 (September 1976).

[Mil80] R. Milner. "Communication and Concurrency", volume 92 of LNCS. Springer Verlag (1980).

[Mil83] R. Milner. Calculus for synchrony and asynchrony. *TCS* **25**, 267–310 (1983).

[Mil89] R. Milner. "Communication and Concurrency". Prentice Hall (1989).

[Ove81] Overman. "Verification of Concurrent Systems: Function and Timing". PhD thesis, university of California (1981).

[Pel98] D. Peled. Ten years of partial ordre reduction. In "Proceedings of CAV". LNCS 1427, Spriner-Verlag (1998).

[Rei85] W. Reisig. Petri nets. In "EATCS Monofraphs on Theoretical Computer Science", Berlin, Heidelberg, New York, Tokyo (1985). Springer Verlag.

[Rib05] P. O. Ribet. "Vérification Formelle de Systèmes, Contribution À la Réduction de L'explosion Combinatoire". PhD thesis, LAAS-CNRS, CNRS 7 av. du Colonel Roche, 31077 Toulouse Cedex France (2005).

[RV02] P. O. Ribet and F. Vernadat. Graphes de pas couvrants vers une relation de conflit dynamique. Technical Report 02065, LAAS (Mars 2002).

[Sai96] D. E. Saidouni. "Sémantique de Maximalité: Application Au Raffinement D'actions Dans LOTOS". PhD thesis, LAAS, Université Paul Sabatier, Toulouse (Mars 1996).

[SB04] D. E. Saidouni and A. Benamira. La alpha-réduction à la volée des systèmes de transitions étiquetées maximales. Technical Report, Laboratoire LIRE, Equipe Vision et Infographie Université Mentouri, Constantine, Algérie (Avril 2004).

[SB05] D. E. Saidouni and N. Belala. Impact des durées d'actions dans les modèles temporisés. Technical Report, Laboratoire LIRE, Université Mentouri, Constantine, Algérie (Avril 2005).

[Shi85a] M. W. Shields. Concurrent machines. *The Computer Journal* **28(5)**, 449–465 (1985).

[Shi85b] M. W. Shields. Deterministic asynchronous automata. In "Formal Methods in Programming", North-Holland (1985).

[TIN] TINA. : Time petri net analyzer. The Tina Toolbox is Property of LAAS-CNRS. 7, avenue du Colonel Roche,31077, Toulouse.

[Val88] A. Valmari. Error detection by reduced reachability graph generation. In "Proceedings of Application and Theory of Petri Nets". Springer Verlag, LNCS (1988).

[Val89] A. Valmari. Sets for reduced state space generation. In "Proceedings of the Tenth International Conference on Application and Theory of Petri Nets", volume II, Bohn (1989).

[Val90] A. Valmari. A stubborn attack on state explosion. In "Proceedings of CAV'90", volume ACM, DIMACS, 3, pages 25–42 (1990).

[VAM96] F. Vernadat, P. Azéma, and F. Michel. Covering step graph. In "Proceedings of Application and Theory of Petri Nets 96", volume LNCS 1091. Springer Verlag (1996).

[VM97] F. Vernadat and F. Michel. Covering step graph preserving failure semantics. In "Proccedings of Appplication and Theory of Petri Nets 97". Springer Verlag, LNCS (1997).

[Vog91] W. Vogler. Bisimulation and action refinement. In "STACSz91", volume 480 of LNCS, pages 309–321. Springer-Verlag (1991).

[Wes86] C. H. West. Protocol verification by random state exploration. In "PSTV VI", pages 233–242 (1986).

[WG93] P. Wolper and P. Godefroid. Partial-order methods for temporal verification. In "Proceedings of Concur's93", volume LNCS 575 (1993.)