Université Paris 7 - Denis Diderot – UFR d'informatique –
Année 2004

Thèse
pour l'obtention du diplôme de
docteur de l'université Paris 7, spécialité informatique
présentée par

Blaise GENEST

le 29 Novembre 2004

L'Odyssée des Graphes de Diagrammes de Séquences (MSC-Graphes)

soutenue publiquement devant le jury suivant :

Paul Gastin	*Examinateur*
Claude Jard	*Rapporteur*
Anca Muscholl	*Directrice de thèse*
Philippe Schnœbelen	*Président du Jury*
P.S. Thiagarajan	*Rapporteur*
Igor Walukiewicz	*Rapporteur*

Remerciements

Je tiens tout d'abord à remercier Anca, bien plus qu'une directrice de Thèse. Pour ses conseils, sa patience, son enthousiasme. Sa gentillesse aussi. Tout ce qu'elle m'a appris enfin. Et puis pour les noms imaginatifs et amusants des macros et autres labels -que personne ne verra jamais- dans les papiers que nous avons écrit (et écrirons) ensemble.

Merci ensuite à Philippe Schnœbelen pour avoir accepté d'être le président du jury. Cette thèse serait sans doute différente sans ce qu'il m'a appris lors d'un stage sous sa (co)direction ("ce que tu as fait ne sera pas perdu" avait-il alors dit avec prémonition). Mes remerciements vont également à Paul Gastin pour accepter de participer au jury, mais aussi pour ses exposés fréquents qui m'ont permis de mieux appréhender les logiques temporelles (concurrentes). J'ai honteusement recopié dans la bibliographie ([DG04]) un de ses abstracts, mais comment faire autrement vu la limpidité de la description ?

Je dois aussi beaucoup aux membres de la défunte ARC FISC, les partenaires francais de choix concernant les scénarios. Je pense notament à Claude Jard pour avoir accepté la lourde tâche de rapporter cette thèse. Merci aussi aux membres bordelais des nombreuses ACI nous liant, en particulier Igor Walukiewicz, qui a dù se battre avec mon francais en acceptant d'être rapporteur. Enfin, P.S. Thiagarajan a eu la gentillesse de faire un rapport sur la thèse, et je lui en suis reconnaissant.

Le lecteur attentif aura sans doute déjà compris que j'accorde un place importante à toutes les personnes avec lesquelles j'ai travaillé pendant ses années de thèse, qu'il y ait eu ou non un article écrit à la clef. Pour moi, c'est cette coopération qui forme la recherche. Je souhaite les remercier tous collectivement ici, la liste serait trop longue et je risquerais d'en oublier. Merci enfin à Jack Allgood (ou pas), sans qui cette thèse ne serait pas en Français.

Enfin, je souhaite remercier les amis du Liafa, de Cachan et d'ailleurs, qui m'ont supporté pendant cette période.

Table des matières

Introduction

Les Enjeux

Les protocoles de communication sont des règles d'échange entre plusieurs processus, plus ou moins proches. Avec le développement des réseaux de communication, tels Internet et les réseaux de téléphonie (mobile ou non), la complexité des protocoles mis en jeux s'accroît de manière vertigineuse. De même, la miniaturisation des composants électroniques permet d'intégrer dans la même puce de plus en plus de possibilités, concrétisées par des protocoles de communication nombreux et constamment mis à jour. Par exemple, les transactions sur internet sont régies par des protocoles bien précis. De même, dans un ordinateur, un contrôleur mémoire accède aux données stockées d'une manière très spéciale.

Les communications se séparent en deux grandes catégories :

- *Synchrones* lorsque les données s'échangent entre les différents processus à des instants bien déterminés (les *tops d'horloge*). La grande spécificité de ces échanges est que le temps pour communiquer une donnée précise d'un processus à un autre est connu. Ce type de communication ne peut être mis en œuvre que si la topologie exacte du réseau est connue, les canaux de communication sûrs et de débit constant, par exemple à l'intérieur d'un processeur. De plus, la vitesse de communication est décidée par le processus le plus lent.
- *Asynchrones* lorsque les données s'échangent entre les différents processus lorsque ceux-ci sont prêts à les envoyer. Le temps de transmission d'un message ne peut alors pas être connu a priori. Ce type de communication peut être mis plus facilement en œuvre.

La seconde catégorie de communication semble prendre le pas sur le première à cause de sa simplicité à mettre en œuvre et de sa meilleure performance. Nous nous intéresserons ainsi à cette seconde catégorie de communi-

cation. Nous choisirons également d'étudier les communications asynchrones par *passage de messages*, plutôt que par *variables partagées*.

Comprendre. Le grand problème des protocoles asynchrones est qu'ils sont intrinsèquement plus compliqués à comprendre que les protocoles synchrones[1]. Ainsi, une des premières tâches est de comprendre quel genre de protocole il est possible de décrire avec quel genre de structure. Ce travail a été commencé dans [HMNK$^+$04] pour des protocoles restreints, puis repris par [Kus03, Mor02]. Dernièrement, [BL04] ont obtenu des résultats partiels sans restriction sur les protocoles.

Construire. En fait, les protocoles de communication mettent en œuvre des processus en parallèle, c'est à dire faisant des choix dépendant uniquement de leur connaissance, et non de la connaissance de l'état des autres processus. Nous dirons de tels protocoles qu'ils sont à contrôle *local*. Une croyance partagée par beaucoup est que notre mode de pensée est intrinsèquement séquentielle, et que penser d'une manière parallèle est plus compliqué. Ainsi, un autre but est d'aider à construire des protocoles de communication, par exemple en distribuant automatiquement des algorithmes trop séquentiels en des algorithmes avec un contrôle local, comme l'ont commencé [AEY00, AEY01, BM03] en supposant qu'il ne pouvait y avoir des données additionnelles de contrôle. Une approche différente a été menée par [HJ00] en exhibant une classe de langages toujours implantable, et par [CDHL00] qui supposaient que plus de libertés étaient laissées aux ingénieurs.

Vérifier. Enfin, la complexité grandissante des protocoles de communication a amené à de nombreuses erreurs par le passé. Tous les protocoles de communication de ces 5 dernières années ont vu leurs premières implémentations sur PC ponctuées de problèmes. Les premiers contrôleurs intégrant l'usb étaient incapables de communiquer en l'utilisant, les premiers contrôleurs mémoire RDRAM étaient instables (intel i820,i840) quand plus de deux bancs mémoires étaient accédés, et le protocole agp 8x des premières puces graphiques ATI R300 générait de nombreux problèmes. Enfin, la dernière nouveauté en terme de communication, le protocole PCI-Express connaît des problèmes sur le chipset serveur intel Lindenhurst quand trop de périphériques accèdent aux canaux point à point. Vérifier les protocoles de communication assez tôt permettrait d'économiser des sommes très impor-

[1]Un nouveau genre de communication pour le matériel embarqué, plus facile à appréhender, fait la jonction entre les deux, en étant globalement asynchrone et localement synchrone.

tantes consacrées à rappeler puis remplacer les composants défectueux. Cette idée, nous la retrouvons dans les travaux de [AY99, MP99] qui modélisent la propriété par un ensemble d'exécutions à éviter, puis de [Pel00, MM01] qui modélisent la propriété par des logiques d'ordre partiel.

Les Outils

Les outils dont nous allons nous servir dans le cadre de cette thèse sont les suivants :

L'algorithmique Notre but est de donner des algorithmes qui construisent un modèle à partir d'un autre, donnant une preuve constructive de l'inclusion d'une classe de structures dans une autre. De même, on donnera également des algorithmes qui vérifient que certaines propriétés sont satisfaitent (par exemple être un élément d'une classe de structures). Quand cela est connu, nous donnons une preuve que la transformation est optimale, dans ce sens qu'il y a des objets de la classe qui requièrent cette transformation. Pour plus de précisions concernant l'algorithmique, nous renvoyons le lecteur à [CLRS01].

La théorie des automates Une structure très étudiée par l'informatique est celle des automates, et plus généralement de plusieurs structures qui modélisent les langages réguliers. Nous nous servirons à de nombreuses reprises des algorithmes connus pour de telles structures. Ainsi, nous essayerons souvent d'exprimer nos systèmes et propriétés sous forme de langage réguliers. Pour plus de précisions concernant la théorie des automates, nous renvoyons le lecteur à [HU79].

Décidabilité et complexité Connaître des algorithmes qui calculent ce que nous voulons est nécessaire, mais pas suffisant. En effet, il se pourrait qu'un algorithme ne termine pas toujours, c'est à dire ne nous donne jamais la réponse. De plus, même si l'algorithme termine, il se peut que le temps pris soit démesuré par rapport au problème posé. Si de nombreuses questions sont en général indécidables, il suffit souvent de réduire le problème à une sous classe. Cette sous classe doit permettre d'exprimer les modèles intéressants, mais en ayant un algorithme de complexité plus faible. Ainsi, un algorithme n'est souvent pas suffisant pour résoudre un problème. Il vaut mieux avoir un algorithme rapide pour une sous classe, et avoir un algorithme plus lent mais qui donne une réponse correcte dans tous les cas intéressants. Pour plus

de précisions concernant la théorie de la complexité, nous renvoyons le lecteur à [Pap94, GJ78].

Les traces de Mazurkiewicz Comme nous l'avons remarqué précédemment, les protocoles de communication mettent en œuvre des processus en parallèle. Il se trouve que les traces de Mazurkiewicz modélisent des événements concurrents. Il serait donc judicieux de découvrir quand nous pouvons appliquer les résultats nombreux de cette théorie aux protocoles de communication. Pour plus de précisions concernant les traces de Mazurkiewicz, nous renvoyons le lecteur à [DR95].

Vérification Enfin pour vérifier qu'un protocole donné est conforme à une spécification fixée, nous utiliserons la théorie prolifique du model-checking. Ainsi, nous considérerons souvent le système donné par un modèle abstrait, et une propriété donnée soit par une formule logique, soit par un ensemble d'exécutions d'un autre modèle. Le but est de savoir si toutes les exécutions du système sont incluses dans les exécutions de la propriété (model-checking positif), ou bien si aucune exécution du modèle n'intersecte celles de la propriété (model-checking négatif). Parfois, les modèles mis en œuvre sont trop imposants pour être vérifiés directement, et ils doivent être abstraits en des modèles beaucoup plus petits, qui ne contiennent pas les détails inutiles à la vérification. Pour plus de précisions concernant la vérification, nous renvoyons le lecteur à [CGP00, BBF$^+$01].

Pour décrire des scénarios de communication, un modèle visuel largement utilisé, *les diagrammes de séquences (time-space diagrams) [LL90, Lyn96]* a été réintroduit sous le nom de *Message Sequence Chart (MSC)* par la communauté du langage SDL. On les retrouve également dans UML (Unified Modeling Language) [BJR97]. Afin de décrire des protocoles complets de communication, un réseau d'*automates communicants* via des files de communications point à point (CFM) sont utilisés [BZ83]. Une autre solution est d'utiliser des sortes d'automates de MSCs, appelés MSC-graphes [itu96].

Cette thèse est articulée autour de deux grandes parties qui reflètent les deux équipes du laboratoire LIAFA entre lesquelles j'étais partagé, l'équipe Automate et Applications et l'équipe Modélisation et Vérification. La première partie de cette thèse est centrée sur les automates communicants (comprendre leur pouvoir d'expression etc.), et essaie de les éclairer sous la lumière d'autres structures, telles les MSC-graphes. La seconde partie est centrée quant à elle autour des MSC-graphes. En effet, ils sont a priori plus simples

à comprendre de par leur caractère visuel. De plus, le contrôle global des MSC-graphes rend des restrictions décidables plus faciles à mettre en œuvre.

Plan de la Partie Automates, Communication et Diagrammes

Cette partie s'intéresse à montrer les relations entre plusieurs structures communicantes par messages. Nous commençons par définir les outils dont nous avons besoin dans les trois premières sections du premier chapitre : automates et langages réguliers, décidabilité et complexité puis les traces de Mazurkiewicz. Nous donnons les théorèmes qui unissent les structures ainsi exposées, les théorèmes de Kleene, de Büchi et d'Ochmanski. Puis nous introduisons dans les autres sections du premier chapitre les différentes structures communicantes qui nous intéressent, logiques, automates communicants (contrôle local) [BZ83] et MSC-graphes (contrôle global) [MR97, GMP04]. En particulier, nous expliquons en quoi modéliser avec des traces la communication par message est en général difficile.

Dans le second chapitre, nous expliquons les relations entre les automates communicants et les autres structures de la communication. Pour cela, nous avons besoin de la structure de CMSC-graphe introduite par [GMP01, GMP04]. Nous terminons en démontrant un théorème central qui explique l'expressivité des automates communicants en terme de logique du second ordre et de MSC-graphes dans un cas un peu restreint, celui des canaux existentiellement bornés. En fait, cette restriction permet de modéliser assez bien la communication par message avec des traces, et ainsi d'appliquer les théorèmes concernant les traces, comme l'avait fait [Kus03] dans le cas des canaux universellement bornés. Nous étendons ainsi avec [GKM04] des résultats de [HMNK$^+$04, Mor02, Kus03].

Plan de la Partie Vérification

La première application des MSC-graphes est que, de part leur caractère globalement séquentiel, ils sont plus proches de la pensée qu'un automate communicant modélisant le même langage. Ainsi, le problème que nous nous posons dans le troisième chapitre est de savoir quand il est possible de transformer, et comment, un MSC-graphe en un automate communicant sans blocage. C'est ce qu'on appelle l'implémentation, qui correspond à distribuer

le contrôle global d'un MSC-graphe en un contrôle local. Nous donnons une solution efficace au problème, mais malheureusement incomplète, qui peut se trouver également dans [GMSZ02, GM03, Gen04]. Nous demandons ainsi quels protocoles peuvent être exprimés par un CMSC-graphe à choix local, classe toujours implémentable sans blocage. D'autres approches, elles aussi incomplètes, peuvent être trouvées dans [AEY01, BM03].

Ainsi, les MSC-graphes apparaissent très tôt dans la modélisation des protocoles de communication. Il serait alors judicieux de vérifier les protocoles dès cette phase de la modélisation, pour ne pas perdre du temps à implémenter un protocole défectueux [AHP96]. De plus, les détails sont souvent non présents au début de la modélisation, ce qui évite d'avoir des systèmes de taille trop grande. Ainsi, nous donnons la complexité exacte de la vérification des MSC-graphes par rapport à de nombreux formalismes dans le chapitre quatre, issu de travaux précurseurs de [MP99, AY99, MM01, Pel00], ainsi que de nos travaux [GMSZ02, GKM04, GMMP04].

Enfin, la taille des systèmes peut tout de même être trop importante pour que les algorithmes donnent une réponse en un temps raisonnable. Ainsi, nous nous intéressons dans le chapitre cinq à réduire la taille de ces MSC-graphes, à la manière de [AY98, AKY99]. Pour ce faire, nous proposons d'abstraire certaines parties qui ne concernent pas la propriété à vérifier. Nous proposons également de se servir de l'aspect généralement modulaire des protocoles pour gagner du temps. En effet, ces modules expriment une redondance dans le protocole (on peut voir une description modulaire comme une sorte de compression). Nous montrons comment il est parfois possible, parfois difficile, de se servir de cette information de redondance pour éviter de calculer plusieurs fois les mêmes opérations, et ainsi accélérer les algorithmes. Cet aspect n'a pas été beaucoup considéré pour les MSC-graphes, à part dans [GM02, GHM03]. On notera tout de même que compresser des systèmes concurrents a été considéré dans [ACE+03].

Résultat Principal

Les contributions principales de cette thèse concernent trois axes déterminants pour l'utilisation des specifications basées sur les diagrammes de séquence : cerner le pouvoir expressif en termes de langages, examiner les limites de la validation algorithmique et proposer des heuristiques pour accélérer les algorithmes.

Dans cette thèse, nous allons présenter un panorama de classes de systèmes communicants par passage de messages. Cependant, toutes ces classes ne sont pas aussi intéressantes les unes que les autres.

Un soin particulier a été apporté aux systèmes à canaux existentiellement bornés, parce qu'ils apparaissent être les plus intéressants. Considérer des systèmes non existentiellement bornés a le défaut d'amener l'indécidabilité de beaucoup de problèmes. En revanche, ça n'est pas le cas pour les systèmes à canaux existentiellement bornés grâce à une représentation régulière des comportements non réguliers de ces systèmes.

En fait, la classe des CMSC-graphes sûrs semble être des plus robuste :

1. Elle généralise les MSC-graphes et les automates communicants existentiellement bornés (Chapitre 2).
2. Une restriction structurelle (globalement coopératif) des CMSC-graphes sûrs capture exactement les automates communicants existentiellement bornés (Chapitre 2).
3. Une heuristique générale peut être décidée pour implémenter de tels systèmes en algorithmes distribués sans blocage (Chapitre 3).
4. On peut vérifier de tels scénarios (Chapitre 4) par rapport à des spécifications exprimées par
   - formule de la logique MSO,
   - formule de la logique temporelle 'template MSC',
   - automates communicants,
   - CMSC-graphes globalement coopératif,
5. Contrairement aux MSC-graphes, les CMSC-graphes sûrs sont clos par projection (Chapitre 5).

Les comparaisons de langages, et dans une moindre mesure le model-checking de beaucoup de spécifications découlent du théorème 5 page 84, véritable pierre angulaire de la thèse, aussi important que compliqué à prouver. Ce théorème montre l'équivalence entre logique MSO, automates communicants et CMSC-graphes globalement coopératif dés que les canaux sont existentiellement bornés. Les résultats présentés avant ce théorème sont souvent déjà connus, alors que les résultats suivants sont issus d'une collaboration de l'auteur (sauf indication contraire).

Première partie

Automates, Communication et Diagrammes.

Chapitre 1

Structures Multiples

οὔτως οὐχ ἅμα πάντα θεοὶ χαρίεντα διδοῦσιν
ἀνδράσιν, οὔτε φυὴν οὔτ᾽ ἄρ φρένας οὔτ᾽ ἀγορητύν[1]
Homère in l'Odyssée, Chant VIII, vers 167-168

Nous présentons dans les trois premières sections des définitions et notations pour les structures de base que nous allons utiliser. De plus, on rappelle le Théorème de Kleene-Büchi pour les mots en section 1, qui unit les langages reconnaissables aux rationnels aux formules de la logique MSO. En section 3, nous donnons une nouvelle démonstration pour le théorème d'Ochmański, extension du théorème de Kleene aux traces.

Puis nous motivons le besoin d'une extension du théorème de Kleene-Büchi liant les différentes structures communicantes que nous définissons dans les sections 4. Enfin, nous présentons les MSC-graphes en section 5.

1.1 Langages Réguliers

L'informatique théorique a débuté par l'étude de la décidabilité, ainsi que par celle du monoïde libre, structure très peu étudiée par les mathématiciens auparavant. L'étude du monoïde libre par les informaticiens s'explique en ce qu'un élément du monoïde libre représente exactement un mot sur un ensemble fini de lettres, appelé *alphabet* fini. Le comportement d'une machine

[1] "Beauté, raison, bien dire ;
on voit qu'en un même homme, les dieux jamais ne mettent tous les charmes."
On pourrait reformuler cela en "Expressivité, compléxité, concision ; on voit qu'en une même structure, les dieux jamais ne mettent tous les charmes."

peut être décrit par la succession de ses actions, c'est à dire un mot sur l'alphabet des actions. L'ensemble de ses actions sera appelé le *langage* d'une machine.

Définition 1 *Le monoïde libre $\Sigma^\star$ sur l'alphabet (fini) Σ admet un neutre que l'on notera ϵ. Sa loi de composition interne sera appelée concaténation. Il est défini par sa représentation par son ensemble Σ de générateurs et par une absence de règles (à l'exception de l'associativité), absence à laquelle il doit son nom de libre.*

Dans le monoïde libre, on note la concaténation de u par v par $u \cdot v$, ou plus simplement par uv quand il n'y a pas ambiguïté. Pour $a_i \in \Sigma$ et $b_j \in \Sigma$, $1 \leq i \leq n$, $1 \leq j \leq m$, on a $a_1 \cdots a_n = b_1 \cdots b_m$ si et seulement si $n = m$ (la longueur des mots est la même), et $a_i = b_i$ pour tout $1 \leq i \leq n$ (les i-ème lettres des deux mots coïncident). Ainsi, $\Sigma^\star$ représente l'ensemble des mots finis sur l'alphabet Σ, et chaque $u \in \Sigma^\star$ est défini de manière unique par sa décomposition $u = a_1 \cdots a_n$ pour $a_i \in \Sigma$. On notera l'ensemble des lettres de u par $\Sigma(u)$.

On se bornera à utiliser les lettres grecques capitales $(\Sigma, \Omega, \ldots)$ pour désigner les alphabets, et on essaiera de garder les premières lettres de l'alphabet $(a, b, \ldots)$ pour désigner les lettres, ainsi que les lettres de la fin de l'alphabet pour les mots $(u, v, \ldots)$.

On dit qu'un mot $v \in \Sigma^\star$ est facteur d'un mot $u \in \Sigma^\star$ si il existe $y, z \in \Sigma^\star$ avec $u = yvz$. On appelle v facteur strict si $u \neq v$. De plus, si $y = \epsilon$, alors v est un préfixe de u, alors qu'il est appelé suffixe si $z = \epsilon$.

On notera par $|u|$ la taille d'un mot u, c'est à dire son nombre de lettres, et par $|u|_a$ le nombre de lettres $a \in \Sigma$ dans u.

1.1.1 Un Modèle de Machines

Une machine a un certain nombre de cellules où elle peut stocker de l'information. On appelle ces cellules la mémoire de la machine. Cette mémoire, bien que pouvant être très grande, est finie. C'est à dire qu'on ne peut pas stocker plus d'information qu'une certaine borne fixée. Une manière simple de modéliser le comportement d'une machine est alors d'utiliser un *automate* à états finis, où les états représentent les différentes configurations de la mémoire. Les actions de la machine seront quant à elles modélisées par les lettres de l'alphabet.

Définition 2 *Un automate fini $\mathcal{A}$ sur un alphabet fini Σ est un quintuplet $(Q, \Sigma, \rightarrow, Q_0, Q_f)$, avec*

- *Σ est un ensemble fini de lettres.*
- *Q est un ensemble fini d'états.*
- *$\rightarrow \subseteq Q \times \Sigma \times Q$ est une relation décrivant les transitions de l'automate.*
- *$Q_0 \subseteq Q$ est l'ensemble fini des états initiaux.*
- *$Q_f \subseteq Q$ est l'ensemble fini des états finaux.*

Exemple 1 *On donnera toutefois souvent un automate sous sa forme graphique, avec des cercles symbolisant les états de l'automates, ainsi que des flèches entre des états symbolisant la relation de transition $\rightarrow$. Un état destinataire d'une flèche sans état initial représente un état initial. Un état source d'une flèche sans état final représente un état final. Voilà ci-dessous la représentation graphique de l'automate $\mathcal{A} = (Q, \Sigma, \rightarrow, Q_0, Q_f)$, avec*

- *$\Sigma = \{a\}$ est composé d'une seule lettre.*
- *$Q = \{p, q\}$ est composé de deux états.*
- *$\rightarrow = \{(p, a, q); (q, a, p)\}$.*
- *$Q_0 = \{p\}$ et $Q_f = \{q\}$.*

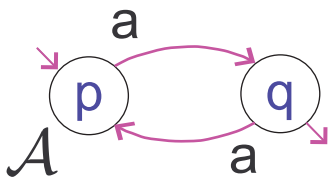


FIG. 1.1 – L'automate $\mathcal{A}$

Comme mentionné plus tôt, un des aspects intéressants d'une machine est la suite des comportements qu'elle peut exhiber. Ainsi, un automate $\mathcal{A}$ génère le langage $L(\mathcal{A})$, c'est à dire l'ensemble des suites de comportements entre un état final et un état initial.

Définition 3 *Soit $\mathcal{A} = (Q, \Sigma, \rightarrow, Q_0, Q_f)$ un automate fini.*

On note $p \xrightarrow{a} q$ quand $(p, a, q) \in \rightarrow$. De même, pour un mot $w = w_1 \cdots w_n \in \Sigma^\star$, on note $p \xrightarrow{w} q$ quand il existe $q_0 \cdots q_n$ avec $q_0 = p, q_n = q$

et pour tout i, $q_{i-1} \xrightarrow{w_i} q_i$. Si $p \in Q_0$ et $q \in Q_f$, alors on dit que w est un mot engendré par $\mathcal{A}$. Le langage $L(\mathcal{A})$ est l'ensemble des mots engendrés par $\mathcal{A}$. De même, on appelle $q_0 \cdots q_n$ un chemin dans $\mathcal{A}$. Il est initial si $q_0 \in Q_0$, final si $q_n \in Q_f$, et acceptant si $q_0 \in Q_0, q_n \in Q_f$. On appelle ce chemin un cycle si $q_0 = q_n$. Il est sans cycle si pour tout $i \neq j$, $q_i \neq q_j$. Si un cycle ne contient pas de cycles strictement plus petits, alors on dire que c'est un cycle simple.

Exemple 2 *Dans l'exemple de la figure 1.1 page précédente, le langage généré par $\mathcal{A}$ est l'ensemble des mots de longueur impaire, c'est à dire $\{a^{2i+1} \mid i \in \mathbb{N}\}$.*

Il est important de savoir quel est le pouvoir d'expression d'une classe de langages. C'est à dire, quels sont les langages que l'on peut représenter et quels sont ceux que l'on ne peut pas représenter avec cette classe. On appellera un automate $\mathcal{B}$ équivalent à un automate $\mathcal{A}$ si $L(\mathcal{B}) = L(\mathcal{A})$.

Il est important de savoir que beaucoup de variantes des automates ont le même pouvoir expressif.

Remarque 1 *Les automates déterministes sont ceux qui n'ont qu'un seul état initial et pour lesquels la relation de transition $\rightarrow$ satisfait que si $p \xrightarrow{a} q$ et $p \xrightarrow{a} q'$, alors $q = q'$. Ils ont le même pouvoir expressif que les automates (non déterministes).*

Si on oublie les états initiaux et finaux ainsi que les étiquettes des transitions, alors un automate peut être décrit par un couple $(Q, \rightarrow)$. On appellera un tel automate un *graphe*, que l'on notera alors plutôt par (V, E), où V est l'ensemble des arêtes et $E \subseteq V \times V$ l'ensemble des transitions entre arêtes. Puisque les transitions ne sont plus étiquetées, on ne s'intéresse qu'aux chemins dans un graphe.

On peut avoir une vue grammaticale des automates :

Définition 4 *Un langage régulier sur l'alphabet Σ est défini par une grammaire sur l'alphabet de terminaux Σ et de non terminaux Var avec des règles de la forme $X = \epsilon$, ou $X = aY$ avec $X \in Var$, $a \in \Sigma$, et $Y \in Var$, ainsi que par un non terminal initial X_0.*

Une grammaire régulière et un automate à un seul état initial sont isomorphes, en voyant les non terminaux de la grammaire en tant qu'états, et

les règles comme transitions. Les états finaux de l'automate correspondent aux non-terminaux X où $X = \epsilon$ est une règle de la grammaire.

Proposition 1 *Un langage L est régulier si et seulement si il existe un automate $\mathcal{A}$ dont il est le langage, ie $L = L(\mathcal{A})$.*

Exemple 3 *Supposons que L soit un langage régulier décrit par une grammaire G, avec pour variable initiale X_0. Alors le langage $L^\star = \{u_1 \cdots u_n \mid n > 0, \forall i, u_i \in L\}$, appelé étoile de Kleene de L, est régulier. On utilise les règles et non-terminaux de G, et on ajoute pour toute règle $X = \epsilon$ une règle $X = X_0$.*

Ainsi, les réguliers sont clos par étoile de Kleene.

1.1.2 Représentations Différentes

Il existe d'autres formes de représentations pour spécifier des langages, qui permettent d'enrichir les connaissances sur les automates.

Un modèle aussi classique que les automates est celui des *langages rationnels*.

Définition 5 *Un langage rationnel est défini par une expression de la forme $L := \emptyset \mid a \mid L \cup L \mid LL \mid L^\star$, pour $a \in \Sigma$. Ainsi, la classe des rationnels pour un alphabet Σ est la plus petite classe qui contient Σ, et qui est stable par union, concaténation et par l'étoile de Kleene.*

Le langage de l'automate de la figure 1.1 page 19 est un rationnel, donné par l'expression $a(aa)^\star$.

On peut aussi avoir une vue plus algébrique des réguliers,

Définition 6 *Un langage L sur l'alphabet Σ est reconnu par monoïde fini si il existe un monoïde fini R, un partie S de R et un morphisme f de $\Sigma^\star$ dans R tel que $f^{-1}(S) = L$.*

Exemple 4 *Le langage $a(aa)^\star$ est reconnaissable par le monoïde $(\mathbb{Z}/2\mathbb{Z}, +)$, avec $f(\epsilon) = 0, f(a) = 1$, et $S = \{1\}$.*

Enfin, un langage peut être défini par le biais d'une logique. Les mots composants ce langage seront ceux qui satisferont la formule logique donnée. On donne ici la définition de la logique monadique du second ordre (MSO).

Définition 7 *Une formule MSO sur l'alphabet* Σ *est de la forme* $\varphi := v_a(x) \mid \neg\varphi \mid \varphi \wedge \varphi \mid x < y \mid \exists X\varphi \mid \exists x\varphi \mid x \in X$*, pour* $a \in \Sigma$*. Une formule close est une formule sans variable libre.*

On définit maintenant par induction les mots de $\Sigma^\star$ qui satisfont une formule MSO close φ.

Définition 8 *Un mot* $w = w_1 \cdots w_n$ *satisfait une formule MSO* φ *sur l'alphabet* Σ *si* $w, \emptyset \models \varphi$*, où* $w, f \models \varphi$ *est défini par*

- *Soit Var l'ensemble des occurrences des variables du premier ordre et* $\mathcal{S}$ *les occurrences de variable du second ordre qui apparaissent dans* φ*.*
- f *est une fonction (partielle) qui associe des symboles* $x \in Var$ *à un nombre de* $\{1, \dots n\}$*, et des symboles* $X \in \mathcal{S}$ *à un sous-ensemble de* $\{1, \dots n\}$*.*
- $w, f \models v_a(x)$ *ssi* $w_{f(x)} = a$*,*
- $w, f \models \neg\varphi$ *ssi* $w, f \not\models \varphi$*,*
- $w, f \models \varphi \wedge \psi$ *ssi* $w, f \models \varphi$ *et* $w, f \models \psi$*,*
- $w, f \models x < y$ *ssi* $f(x) < f(y)$*,*
- $w, f \models x \in X$ *ssi* $f(x) \in f(X)$*,*
- $w, f \models \exists x\varphi$ *ssi* $\exists i \in \{1, \cdots, n\}$ *tel que* $w, f' \models \varphi$*, avec* $f'(y) = i$ *si* $y = x$*,* $f'(y) = f(y)$ *sinon,*
- $w, f \models \exists X\varphi$ *ssi* $\exists S \subseteq \{1, \cdots n\}$ *tel que* $w, f' \models \varphi$*, avec* $f'(y) = S$ *si* $y = X$*,* $f'(y) = f(y)$ *sinon.*

L'ensemble des mots satisfaisant une formule close φ *représente le langage* $L(\varphi)$ *de* φ*.*

Exemple 5 *On peut définir plusieurs formules "macros" qui seront utiles par la suite.* $\varphi \vee \psi$ *est définie par* $\neg(\neg\varphi \wedge \neg\psi)$*. De même,* $\forall x \in X, \varphi$ *est définie par* $\neg(\exists x \in X, \neg\varphi)$*. On peut aussi définir* $x \neq y$ *par* $x < y \vee y < x$*, et* $x \notin X$ *par* $\forall z \in X, z \neq x$*. On peut définir enfin* $first(x) = \neg\exists z, z < x$*, c'est à dire* x *est le premier élément de* w*. De même,* $last(x) = \neg\exists z, x < z$ *De plus, on peut noter* $x \lessdot y$ *pour* y *successeur direct de* x*, qui est définie en MSO par* $x < y \wedge \forall z, z = x \vee z = y \vee z > y \vee z < x$*.*

Ainsi, le langage $a(aa)^\star$ *est le langage des mots de* $\{a\}^\star$ *satisfaisant la formule* $\exists X, [(\exists x \in X, first(x)) \wedge (\exists x \notin X, last(x)) \wedge \forall x \forall y (x \lessdot y \implies (x \in X \iff y \notin X))]$*.*

On a montré que le langage $a(aa)^\star$ était en même temps un langage rationnel, un langage satisfaisant une formule MSO, un régulier et un re-

connaissable. En fait, le théorème de Kleene-Büchi donne un résultat plus général.

Théorème 1 *[Kle56, Büc60][Kleene-Büchi] Les formules MSO, les reconnaissables, les rationnels et les réguliers ont le même pouvoir expressif pour tout monoïde libre $\Sigma^\star$.*

Définition 9 *Le complémentaire d'un langage $L \subseteq \Sigma^\star$ est l'ensemble de tous les mots de $\Sigma^\star$ qui n'appartiennent pas à L. On note par $\bar{L}$ le complémentaire de L.*

Pour ne donner qu'un exemple de la force de ce théorème, il est facile de complémenter un automate déterministe. Ainsi, les réguliers, donc les rationnels, sont clos par complémentation, ce qui est difficile à démontrer directement.

Bien entendu, il existe de nombreux formalismes qui ne sont pas aussi expressifs que les langages réguliers. On donne ici l'exemple d'une autre logique, la *logique temporelle linéaire*, encore appelée *LTL* [Pnu77].

Définition 10 *Une formule LTL sur l'alphabet Σ est de la forme $\varphi := a \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathcal{X}\varphi \mid \varphi\mathcal{U}\varphi$, où $a \in \Sigma$.*

On définit maintenant par induction les mots de $\Sigma^\star$ qui satisfont une formule LTL φ.

Définition 11 *Un mot $w = w_1 \cdots w_n$ satisfait une formule LTL φ sur l'alphabet Σ si $w, 0 \models \varphi$, où $w, i \models \varphi$ est défini pour $i \in \{0, \ldots, n\}$ par :*

- *$w, i \models a$ ssi $w_i = a$.*
- *$w, i \models \neg\varphi$ ssi $w, i \not\models \varphi$.*
- *$w, i \models \varphi \wedge \psi$ ssi $w, i \models \varphi$ et $w, i \models \psi$.*
- *$w, i \models \mathcal{X}\varphi$ ssi $i \neq n$ et $w, i+1 \models \varphi$.*
- *$w, i \models \varphi\mathcal{U}\psi$ ssi $\exists k \leq n$ tel que $\forall i \leq j < k$, $w_j \models \varphi$ et $w_k \models \psi$.*

L'ensemble des mots satisfaisant une formule φ représente le langage $L(\varphi)$ de φ.

Tout langage d'une formule LTL est aussi le langage d'une formule MSO. En revanche, il est possible de montrer qu'aucune formule LTL ne peut engendrer le langage $a(aa)^\star$.

Exemple 6 *On peut définir plusieurs formules "macros" de la même manière que pour MSO, tout comme* $\varphi \vee \psi = \neg(\neg\varphi \wedge \neg\psi)$*, ou* $tt = \bigcup_{a\in\Sigma} a$ *et* $ff = \neg tt$*. On peut également définir la formule* $F\varphi$ *par* $tt\mathcal{U}\varphi$*, ce qui représente 'un jour* φ*'. De plus, on peut définir la formule 'toujours* φ*' par* $\neg F(\neg\varphi)$*, que l'on notera* $G\varphi$*.*

Le langage $(ab)^\star$ *est engendré par la formule LTL suivante :* $a \wedge G((a \wedge \mathcal{X}b) \vee (b \wedge \mathcal{X}a)) \wedge F(b \wedge \mathcal{X}ff)$*.*

En fait, on peut montrer que les formules LTL ont le même pouvoir expressif qu'une sous classe de MSO, appelé logique du *premier ordre*, ou encore *FO*. Une formule MSO est un formule du fragment FO si elle ne contient pas de variable du deuxième ordre.

Si les langages réguliers (et leurs équivalents) sont très bien compris et étudiés, ils sont néanmoins très insuffisants pour spécifier des langages plus évolués. Par exemple, le langage qui teste si un mot a un nombre égal de a et de b n'est pas régulier. Pour décrire de tels langages, nous allons définir une autre structure de machine, bien plus expressive.

1.2 Les Machines de Turing

1.2.1 Définition

Les automates ne permettent pas d'exprimer des langages plus complexes parce qu'ils semblent avoir deux lacunes. Premièrement, ils n'ont pas le droit de revenir lire une lettre qu'ils ont déjà lu. De plus, ils ont une mémoire finie, codée dans leurs états finis. [2]

Afin d'augmenter le pouvoir expressif des automates, on va leur permettre d'avoir une mémoire, et de pouvoir aller et venir sur le mot.

Définition 12 *Une machine de Turing* M *déterministe sur un alphabet* Σ *est un quadruplet* (Q, Σ, δ, q_0) *avec*

- Q *est un ensemble fini d'états.*
- $q_0 \in Q$ *est l'état initial.*

[2] En fait, il est bien connu qu'en présence d'une mémoire finie, la première restriction n'en est pas une, c'est à dire que le pouvoir expressif des automates boustrophédons (qui peuvent lire dans les deux sens) n'est pas plus élevé que celui des automates.

– *La fonction δ est définie par $\delta : Q \times \Sigma \cup \{\sharp, \$\} \rightarrow (Q \cup \{$ oui, non$\}) \times \Sigma \cup \{\sharp, \$\} \times \{-1, 0, 1\}$, où oui et non sont respectivement les états d'arrêt, d'acceptation et de rejet et n'appartiennent pas à Q.*

Les machines de Turing représentent exactement les programmes déterministes (écrits en C par exemple). Le langage $L(M)$ de M est l'ensemble des mots acceptés par M.

Exemple 7 *Le langage des mots qui contiennent autant de a que de b est reconnu par une machine de Turing.*

1.2.2 Décidabilité et Complexité

On dira qu'un langage L est décidable si il existe une machine de Turing M qui reconnaît ce langage et qui s'arrête sur toutes les entrées. Autrement, il sera appelé indécidable. Il est évidemment important que M rejette le complémentaire, sinon, il se pourrait que la machine boucle indéfiniment. Dans ce cas, un utilisateur extérieur serait incapable de savoir si il doit attendre une réponse plus tard, ou si la machine boucle.

Grâce aux machines de Turing, on peut facilement exprimer si il existe un algorithme qui donne une réponse à un problème ou non. Par exemple, il existe un algorithme pour dire si un mot a le même nombre de a que de b, mais il n'existe pas d'algorithme qui teste si une machine de Turing s'arrête sur un mot $w \in \Sigma^\star$.

Proposition 2 *Le langage des codages de machines de Turing qui s'arrêtent sur l'entrée vide est indécidable.*

On dira alors que le problème de l'arrêt est indécidable pour les machines de Turing. Si un tel résultat a l'air rédhibitoire pour trouver un programme qui calcule ce que l'on veut, il ne faut pas perdre de vue qu'il concerne uniquement le pire des cas. C'est à dire qu'il peut être très facile d'écrire un algorithme terminant pour certains sous cas, et très dur pour d'autres. L'approche que l'on suivra lorsque l'on se trouvera en face de tels cas sera d'exhiber des sous classes pour lesquelles le problème est décidable.

Si le fait de savoir s'il existe un algorithme qui termine ou non est fondamental, il ne faut pas perdre de vue que le temps de calcul de l'algorithme est non moins important. En effet, à quoi servirait un programme qui a besoin

de la durée de vie de la Terre pour donner la réponse ? Pour formaliser ce problème, on peut également se servir des machines de Turing.

Définition 13 *Soit f une fonction de $\mathbb{N}$ dans $\mathbb{N}$. Une machine de Turing déterministe est de temps f si pour toute entrée u, la machine s'arrête après au plus $f(|u|)$ transitions. La classe de ces machines est notée TIME(f).*

Elle sera appelée d'espace f si pour toute entrée u, la machine ne va jamais lire ou écrire de symbole après les $f(|u|)$ premiers symboles de l'espace de travail. La classe de ces machines est notée SPACE(f).

Pour X une classe de complexité, on notera NX la classe des machines de même complexité, mais qui sont non déterministes. De même, on notera co-X pour le complémentaire de la classe X, c'est à dire les langages qui ne sont pas dans X.

Les classes utilisées dans cette thèse seront :

- *LOGSPACE, la classe des machines en espace $P(\log)$, où P est un polynôme.*
- *PTIME, la classe des machines en temps P, où P est un polynôme.*
- *PSPACE, la classe des machines en espace P, où P est un polynôme.*
- *EXPSPACE, la classe des machines en espace 2^P, où P est un polynôme.*

Le non déterminisme peut être considéré comme le fait pour la machine de pouvoir deviner une certaine information, qu'elle devra vérifier par la suite. On rappelle ici quelques propositions fondamentales de la théorie de la complexité.

Proposition 3 *Pour toute fonction $f \geq \log$,*

- *TIME(f) $\subseteq$ SPACE(O(f))*
- *SPACE(f) $\subseteq$ TIME($2^{O(f)}$)*
- *NSPACE(f) $\subseteq$ SPACE(O(f^2)) (théorème de Savitch)*
- *co-SPACE(f) = SPACE(O(f))*
- *co-TIME(f)= TIME(O(f))*
- *co-NSPACE(f)=NSPACE(O(f)) (théorème de Immerman-Szelepscényi)*

Ainsi, on dira qu'un test est en temps polynomial si le langage des mots satisfaisant le test est dans PTIME. Si il est dans TIME(x) il sera linéaire, et quadratique si il est dans TIME(x^2). Il est facile de montrer qu'un problème est dans une classe X, en exhibant un algorithme dans X qui résout le problème.

Exemple 8 *Savoir si le langage d'un automate $\mathcal{A}$ est vide est co-NLOG SPACE, donc NLOGSPACE par le théorème de Immerman-Szelepscényi. Ainsi, $\mathcal{A}$ reconnaît un mot si et seulement si il existe un chemin d'un état initial à un état final. L'algorithme consiste à partir de l'état initial, et à chaque étape, de deviner un successeur. La machine de Turing n'a besoin que de garder l'état courant et le successeur, ce qui est bien de taille logarithmique. Elle accepte si et seulement si elle rencontre l'état final.*

Savoir si deux automates $\mathcal{A}, \mathcal{B}$ reconnaissent le même langage est dans PSPACE. On a $L(\mathcal{A}) = L(\mathcal{B})$ si et seulement si $L(\mathcal{A}) \cap \overline{L(\mathcal{B})} = \emptyset$ et $L(\mathcal{B}) \cap \overline{L(\mathcal{A})} = \emptyset$. Il est connu qu'on peut à partir de tout automate $\mathcal{C}$ obtenir un automate déterministe $\mathcal{D}$ de taille exponentielle. Il est facile de complémenter un automate déterministe. Enfin, en utilisant le produit des automates, on sait creér un automate qui a pour langage $L(\mathcal{A}) \cap L(\mathcal{B})$ pour deux automates $\mathcal{A}, \mathcal{B}$. Ainsi, on peut créer un automate de taille exponentielle en $|\mathcal{A}| + |\mathcal{B}|$ qui reconnaît $L(\mathcal{B}) \cap \overline{L(\mathcal{A})}$, et sur lequel on peut tester le vide en espace logarithmique en la taille du modèle, c'est à dire en PSPACE.

Une complexité plus précise qu'à un polynôme près n'est pas toujours très représentative, donc on se bornera à considérer les classes à un polynôme près. L'avantage d'une telle approche est que l'on peut prouver qu'un certain problème est intrinsèquement dans une classe de complexité, c'est à dire qu'il n'y a aucune chance qu'il soit dans une classe strictement moins expressive.

Définition 14 *On dit qu'un problème B se réduit à un problème A si il existe une fonction f qui associe à toute instance J de B une instance I de A en temps polynomial en $|B|$ telle que I est vraie pour A si et seulement si J est vraie pour B.*

Un problème A est X-hard si pour tout problème B de la classe X, B se réduit à A. Si de plus A appartient à X, alors A est X-complet.

Remarque 2 *Il est facile de montrer que A est X-hard si et seulement si il existe un problème B qui est X-complet tel que B se réduit à A.*

Un problème A qui est X-hard n'a alors aucune chance d'appartenir à une classe strictement moins expressive que X.

Exemple 9 *On définit ici le problème* (1-in-3) SAT. *Une instance de (1-in-3) SAT est une formule $\varphi = \wedge_{j=1}^{m} C(a_j, b_j, c_j)$ sur n variables $(x_i)_{i=1,n}$, où chaque clause $C(a_j, b_j, c_j)$ est vraie si exactement un littéral a_j, b_j, c_j est vrai.*

Le problème consiste à savoir si il y a une valuation v qui à chaque variable x assigne une valeur de $\{0,1\}$, c'est à dire 0 si le littéral $\neg x$ est vrai, et 1 si le littéral x est vrai.

Il est bien connu que (1-in-3) SAT est un problème NP-complet [Sch78].

De même, on peut définir (1-in-3) QBF *par une formule φ de (1-in-3) SAT. La question est alors de savoir si il existe un arbre de valuation telle que la formule $\forall x_n \exists x_{n-1} \forall x_{n-2} \cdots \exists x_1 \varphi$ est vraie. Il est connu que (1-in-3) QBF est un problème PSPACE-complet [Sch78], voir aussi [Dal00].*

De plus, savoir si deux automates sont équivalents est PSPACE-complet.

La classe de complexité d'un problème est souvent représentative de ce qu'il est possible de faire. Plutôt que de donner une borne de complexité à partir de laquelle il n'est plus réaliste d'implanter le problème en machine, il est plus sage de considérer la classe de complexité comme un bon indicateur des limites sur la taille de l'entrée. En effet, un algorithme linéaire pourra sans doute traiter des tableaux de plusieurs millions de champs, alors qu'un algorithme EXPSPACE ne sera utilisable que si l'entrée est très petite.

Exemple 10 *On peut décider si deux automates, ou deux rationnels, ou deux formules LTL, ou deux formules MSO ont une intersection non vide. Cependant, la complexité de ces tests est très différente. Elle est NLOGSPACE-complète pour les automates et les rationnels, PSPACE pour les formules LTL, et non élémentaire pour les formules MSO. Non élémentaire signifie en temps f, où $f(n) = 2^{2^{2^{\cdot}}}$, où la tour d'exponentielles a pour hauteur n. Comment est ce possible alors que tous ces formalismes sont équivalents ? En fait, c'est parce que pour passer d'une formulation à une autre, il faut souvent payer un prix assez élevé. On utilisera ainsi souvent les automates, qui ont une complexité plus exploitable que les autres formalismes.*

Remarque 3 *Si on réduit un problème A indécidable à un problème B, alors B est lui aussi indécidable.*

Exemple 11 *On définit ici le* problème de correspondance de Post (PCP) *[Pos46]. Soit un alphabet Σ. On se donne n dominos (u_i, v_i) où $u_i, v_i \in \Sigma^\star$. Le problème est de savoir s'il existe une suite non vide d'indices $\alpha_1 \cdots \alpha_k$ tel que $u_{\alpha_1} \cdots u_{\alpha_k} = v_{\alpha_1} \cdots v_{\alpha_k}$. Il est bien connu que le problème de l'arret d'une machine de Turing se réduit à PCP [Pos46]. Ainsi, PCP est lui aussi indécidable.*

1.3 Traces de Mazurkiewicz

Parce que les machines de Turing décrivent tous les programmes en C, on pourrait penser qu'elles sont suffisantes pour modéliser n'importe quel modèle. Cependant, il existe de nombreux cas pour lesquels les machines de Turing ne sont pas adaptées. Trois grandes classes restent à étudier plus en détail. Si l'on veut étudier les comportements transfinis, on a besoin de nouvelles structures [PP02, Mul63]. De même si l'on veut ajouter la modélisation du temps (continu) aux possibilités, cela pose problème [AD94]. Enfin, et c'est le point que nous allons développer ici, considérer plusieurs machines travaillant de manière concurrente (pour modéliser des protocoles de communication par exemple) n'est pas aisé.

Une manière assez facile d'introduire de la concurrence au niveau algébrique des structures considérées est de modifier légèrement le monoïde libre en lui ajoutant des règles de commutation (voir [DR95]). Soit $I \subseteq \Sigma \times \Sigma$ une relation antiréflexive, symétrique, appelée relation de commutation (ou d'indépendance, d'où elle tire son symbole). On notera par $D = \Sigma \times \Sigma \setminus I$ la relation de dépendance. On peut représenter un alphabet (Σ, D) de dépendance par son *graphe de dépendance*, avec une arête entre deux lettres qui ne commutent pas.

Exemple 12 *L'*alphabet de dépendance $D = (\{a, b, c, d\}, \{(a, a), (b, b), (c, c), (d, d), (a, b), (b, a), (d, c), (c, d)\})$ *est représenté par le* graphe de dépendance *ci dessous.*

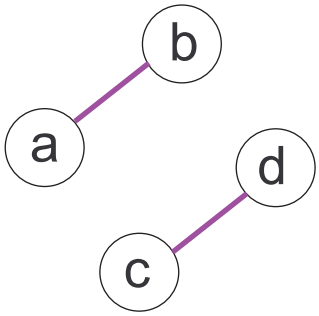


FIG. 1.2 – L'alphabet de dépendance de (Σ, D)

Comme D et I sont symétriques, on pourra parfois donner pour I simplement une des deux relations $(a,b) \in I$ plutôt que $(a,b) \in I$ et $(b,a) \in I$. La relation I induit une relation d'équivalence $\sim_I \subseteq \Sigma^\star \times \Sigma^\star$: deux mots u, v sont équivalents si et seulement si il existe une suite de mots $(w_i)_{i=1}^n$ telle que $w_1 = u$, $w_n = v$ et pour tout $i < n$, il existe $(a_i, b_i) \in I$ et deux mots x_i, y_i telles que $w_i = x_i a_i b_i y_i$ et $w_{i+1} = x_i b_i a_i y_i$. C'est à dire que u, v sont équivalents si il existe une suite de commutations de lettres adjacentes. On notera la classe d'equivalence de u par $[u]_I$. Comme $\sim_I$ est compatible avec la concaténation, on peut définir une concaténation sur les classes d'équivalence de $\sim_I$ par $[u]_I[v]_I = [uv]_I$.

Définition 15 *On appelle monoïde de traces le monoïde $\mathbb{M}(\Sigma, I)$ des classes d'équivalence de $\sim_I$ sur $\Sigma^\star$, muni de la concaténation définie ci-dessus.*

En fait, $\mathbb{M}(\Sigma, I)$ est le quotient de $\Sigma^\star$ par $\sim_I$. On appelle un élément de $\mathbb{M}(\Sigma, I)$ une trace de Mazurkiewicz [DR95, Maz77]. A partir de tout mot u, on peut connaître la trace $[u]_I$ qui lui correspond, et de toute trace $[u]_I$, on peut connaître l'ensemble $\{v \in \Sigma^\star \mid u \sim_I v\}$ des mots qui lui correspondent. On assimilera donc une trace à un ensemble de mots. On peut remarquer que pour $u \sim_I v$, $|u| = |v|$. On a même que le nombre $|u|_a$ de lettres a dans u est égal à $|v|_a$.

Exemple 13 *Reprenant l'alphabet de dépendance $(\{a,b,c,d\}, \{(a,a),(b,b),(c,c),(d,d),(a,b),(b,a),(d,c),(c,d)\})$, la classe d'équivalence de $[bacd]_I$ est $\{bacd, bcad, cbad, bcda, cbda, cdba\}$.*

On peut étendre la relation I à $[u]_I, [v]_I \in \mathbb{M}(\Sigma, I)$ par $[u]_I I [v]_I$ si toutes les lettres de u commutent avec toutes les lettres de v. De même, on notera uIS ou $u \in I(S)$ pour $u \in \Sigma^\star, S \subseteq \Sigma^\star$ si toutes les lettres de u commutent avec toutes les lettres de tous les mots de S.

Remarque 4 *Pour décider si deux traces $[u]_I, [v]_I$ sont égales, on peut tester si les ensembles correspondant à $[u]_I$ et $[v]_I$ sont égaux. Plus simplement, il suffit de tester si u appartient a l'ensemble $[v]_I$, ou l'inverse. C'est à dire qu'on a juste besoin de calculer un des deux ensembles. Une autre méthode est de s'assurer que pour tout couple aDb, les projections de u et v sur l'alphabet $\{a,b\}$ sont égales. Cette méthode est la plus rapide, car de complexité polynomiale.*

On utilisera la même notation $[S]_I$, où $S \subseteq \Sigma^\star$ est un ensemble de mots, pour désigner la clôture par commutation de S, c'est à dire $[S]_I = \bigcup_{u \in S} [u]_I = \{v \in \Sigma^\star \mid \exists u \in S, v \sim_I u\}$. On appelle $[S]_I$ un langage de traces.

Remarque 5 *De même, pour décider si pour deux ensembles de traces $[S]_I$, $[T]_I$, on a $[S]_I \subseteq [T]_I$, on peut tester si $S \subseteq [T]_I$. Pour tester si $[S]_I \cap [T]_I = \emptyset$, on peut juste tester si $S \cap [T]_I = \emptyset$ (ou l'inverse). Il est à noter qu'on ne peut plus utiliser l'idée de la projection dans ce cas.*

1.3.1 Langages Réguliers et Rationnels

On peut évidemment définir des automates, que nous appellerons automates I-diamant, sur un alphabet de traces afin de décrire des machines concurrentes.

Définition 16 *Soit (Σ, I) un alphabet de commutation. Un automate $\mathcal{A}$ I-diamant est un automate fini $(Q, \Sigma, \to, Q_0, Q_f)$ avec la propriété additionnelle suivante. Si $q \xrightarrow{a} q_1 \xrightarrow{b} q_2$ pour deux lettres $a, b \in \Sigma$ telles que aIb, alors il existe q_1' avec $q \xrightarrow{b} q_1' \xrightarrow{a} q_2$. Un langage de traces est régulier si il est reconnu par un automate I-diamant.*

On peut remarquer que la propriété I-diamant assure que l'on puisse écrire un fonction de transitions sur les traces. Par exemple, si $u \sim_I v$, alors $q \xrightarrow{u} q'$ si et seulement si $q \xrightarrow{v} q'$, que l'on notera par $q \xrightarrow{[u]_I} q'$. Ainsi, pour un automate I-diamant $\mathcal{A}$, on a $L(\mathcal{A}) = [L(\mathcal{A})]_I$, c'est à dire que $L(\mathcal{A})$ est *clos par commutation*. Par abus de terminologie, on dira également que $\mathcal{A}$ est clos par commutation si son langage l'est. Réciproquement, si on a un automate $\mathcal{A}$ tel que $L(\mathcal{A})$ est clos par commutation, alors on peut créer un automate I-diamant $\mathcal{B}$ qui est équivalent[3] à $\mathcal{A}$.

Proposition 4 *Les langages de traces réguliers sont exactement les langages de mots reconnus par monoïdes finis, où les morphismes $h : \Sigma \to S$ satisfont $h(ab) = h(ba)$ pour tout $(a, b) \in I$.*

Exemple 14 *Soit l'alphabet d'indépendance $(\Sigma, I) = (\{a, b, c\}, (a, c), (c, a))$. Parmi les trois automates $\mathcal{A}, \mathcal{B}, \mathcal{C}$ de la figure 1.3 page suivante, l'automate*

[3]En fait, si $\mathcal{A}$ est clos par commutation, l'automate déterministe minimal de $L(\mathcal{A})$ est I-diamant.

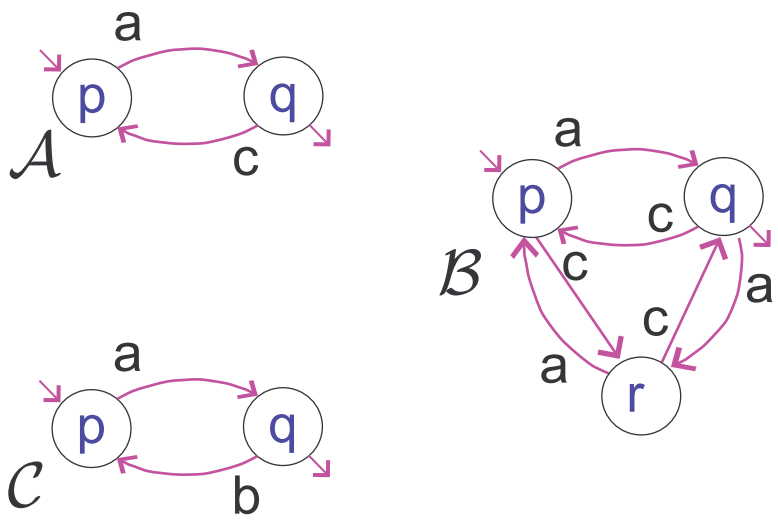


FIG. 1.3 – Trois automates $\mathcal{A}, \mathcal{B}, \mathcal{C}$

$\mathcal{A}$ n'est pas I-diamant parce que de l'état p, on peut tirer la suite d'actions ac, avec aIc, mais pas la suite équivalente ca. De plus, il n'est pas non plus clos par commutation parce que $aca \in L(\mathcal{A})$, mais $aac \notin L(\mathcal{A})$.

L'automate $\mathcal{B}$ est I-diamant parce que de tout état, on peut tirer ac et ca et arriver au même état. L'automate $\mathcal{C}$ est I-diamant parce que aDb.

De même, on peut définir des langages rationnels de traces $[L]_I$ par des expressions rationnelles $L := \emptyset \mid a \in \Sigma \mid L \cup L \mid LL \mid L^\star$. Une trace $[w]_I$ appartient à un langage rationnel de traces $[L]_I$ si il existe un mot $u \in L$ avec $u \sim_I w$. Ainsi, un langage rationnel de traces n'est clos par commutation que de manière sémantique. Ceci fait que les langages rationnels et réguliers ne sont pas égaux.

Par exemple, si on prend le langage rationnel $[(ab)^\star]_I$ sur l'alphabet de commutation $I = (\{a, b\}, \{(a, b), (b, a)\})$, alors l'ensemble de traces reconnu par ce langage est $\{u \in \Sigma^\star \mid |u|_a = |u|_b\}$, qui n'est pas un régulier (de mots), donc pas non plus un régulier de traces. Comme un automate I-diamant est un automate sur les mots, et qu'on a l'équivalence avec les rationnels sur les mots, on peut affirmer que

Proposition 5 *Les langages réguliers de traces sont strictement inclus dans la classe des langages rationnels de traces.*

1.3.2 Théorème d'Ochmański

Il est particulièrement intéressant de se demander quelle est la sous classe des rationnels qui correspond aux réguliers. Le théorème d'Ochmański répond à cette question. Le précédent exemple suggère que les boucles concernant un alphabet non connecté posent problème. Soit (Σ, I) un alphabet de commutation, avec D la relation de dépendance. On appelle un automate à boucles I-connexes si pour chaque ensemble V de lettres qui apparaissent sur une boucle (pas forcément simple), le graphe $(V, D|_{V\times V})$ est connexe.

Exemple 15 *Reprenant les exemples de la figure 1.3 page ci-contre, sur l'alphabet d'indépendance* $(\Sigma, I) = (\{a, b, c\}, \{(a, c), (c, a)\})$*, les automates* $\mathcal{A}$ *et* $\mathcal{B}$ *ne sont pas à boucles* I*-connexes parce qu'ils autorisent des boucles de* ac*, avec* aIc*. En revanche, l'automate* $\mathcal{C}$ *est à boucles* I*-connexes, puisque la seule boucle concerne les lettres* a, b*, qui sont dépendantes.*

Théorème 2 *[Och85] Soit* L *un langage rationnel de traces sur l'alphabet* (Σ, I)*. Alors* L *est régulier si et seulement si* L *est reconnu par un automate à boucles* I*-connexes*[4]*.*

Ce théorème est excessivement important, pour les traces comme dans la suite de cette thèse. Nous allons donc en donner une preuve. Cependant, nous ne donnerons pas la preuve originale due à Ochmański parce qu'elle est inefficace et utilise la fonction de rang (voir aussi [DR95]), que nous n'avons pas défini et dont nous n'avons pas besoin. La complexité des algorithmes que nous donneront en deuxième partie est fondamentale, ce qui explique que l'on ait besoin de preuves constructives.

Pour être plus précis, nous allons montrer les deux propriétés suivantes, qui suffisent pour montrer le théorème d'Ochmański.

Proposition 6 *Si* $\mathcal{A}$ *est un automate à boucles* I*-connexes, alors il existe un automate* I*-diamant* $\mathcal{B}$ *tel que* $[L(\mathcal{A})]_I = L(\mathcal{B})$*, et* $|\mathcal{B}| = O((2^{|\Sigma|}|\mathcal{A}|^2)^{|\mathcal{A}||\Sigma|})$ *[MP99].*

[4]Notons que l'automate I-diamant reconnaissant L n'est en général pas I-connexe

Si $\mathcal{A}$ est un automate avec $L(\mathcal{A})$ clos par I-commutation, alors il existe un automate à boucles I-connexes $\mathcal{B}$ tel que $[L(\mathcal{B})]_I = L(\mathcal{A})$, et $|\mathcal{B}| = O(|\Sigma|! \times |\mathcal{A}|)$.

La construction d'un automate I-diamant à partir d'un automate à boucles I-connexes est due à [MP99], et prouve d'une autre façon que les langages rationnels I-connexes sont réguliers.

La construction réciproque d'un automate à boucles I-connexes à partir d'un automate dont le langage est clos par commutation est nouvelle (voir [Gen04]). Cette construction reprend toutefois les idées d'Ochmański sur la *forme normale lexicographique (LNF)*, mais en donnant une construction à la volée de cette forme normale, évitant ainsi de passer par une construction implicite, et donc de connaître la taille de l'automate obtenu.

Nous allons ici donner la preuve (assez longue) de la proposition 6 page précédente sous forme d'une suite de propositions et de lemmes.

On va tout d'abord montrer que si $\mathcal{A} = (Q, \Sigma, \to, Q_0, Q_f)$ est un automate à boucles I-connexes, alors il existe un automate $\mathcal{B}$ avec $L(\mathcal{B}) = [L(\mathcal{A})]_I$. L'idée est que, pour un mot u qu'il est en train de reconnaître, $\mathcal{B}$ va deviner un réordonnancement de u en un chemin acceptant de $\mathcal{A}$. En effet, $u \in [L(\mathcal{A})]_I$ si et seulement si un tel réordonnancement existe.

Un automate travaille à la volée, donc il va avoir à deviner le mot petit à petit. Ainsi, il va devoir deviner le chemin correspondant au préfixe d'un mot. Le problème est que si $u = vw$, alors pour $u' \sim_I u$, la partie v' qui correspond à v dans u' n'a aucune raison d'etre un préfixe de u'. Pire, v' n'a aucune raison d'être un facteur de u. Ainsi v' n'a pas raison de correspondre à un chemin dans $\mathcal{A}$. En fait, v' peut correspondre à un chemin à trous dans $\mathcal{A}$. Nous définissons maintenant les *chemins à trous* d'un automate.

Définition 17 *Un chemin de $\mathcal{A}$ à n trous est un n-uplet de triplets $\tau = (q_{2i-1}, \Sigma_i, q_{2i})_{i=1}^n$ où pour tout $1 \leq i \leq n$, $q_{2i-1}, q_{2i} \in Q$ et $\Sigma_i \subseteq \Sigma$.*

Un chemin à trous représente l'ensemble des suites $(\rho_i)_{i=1}^n$ de chemins ρ_i de q_{2i-1} à q_{2i} étiquetés par des mots de $\Sigma_i^\star$. Il est important de noter qu'il n'y a pas de raison que $q_{2i} = q_{2i+1}$ (d'où le nom de trous), parce qu'il peut arriver qu'on lise plus tard dans l'automate $\mathcal{B}$ des lettres qui commutent avec certaines déjà lues et qui se trouvent plus tôt dans le chemin associé dans $\mathcal{A}$. Ces lettres lues plus tard viendront reboucher les trous en formant un chemin entre q_{2i} et q_{2i+1}. Pour savoir si une lettre a peut être écrite au début d'un chemin à trous, il faut qu'elle commute avec toutes les lettres déjà lues

de ce chemin, c'est à dire $a\, I\, \Sigma(\tau)$, pour $\Sigma(\tau) = \bigcup_{i=1}^{n} \Sigma_i$. On notera par Γ_n l'ensemble des chemins à au plus n trous. On note ϵ pour le chemin à trous qui est vide. On montrera un peu plus tard qu'on peut borner le nombre de trous par un certain n_0 en fonction de $|\mathcal{A}|$ et $\Sigma|$.

L'ensemble des états de $\mathcal{B}$ est Γ_{n_0}. L'état initial de $\mathcal{B}$ est ϵ, alors qu'un chemin à trous τ est final dans $\mathcal{B}$ si et seulement si c'est un chemin sans trou acceptant dans $\mathcal{A}$, c'est à dire si $\tau = (q_0, \Omega, q_f)$ est un singleton avec $q_0 \in Q_0$ et $q_f \in Q_f$. De plus, ϵ est final si $Q_0 \cap Q_f \neq \emptyset$.

On définit $\rightarrow_{\mathcal{B}}$ par $\tau \xrightarrow{a}_{\mathcal{B}} \tau'$ si et seulement si $\tau = AXYB$ et $\tau' = AZB$ avec A, B, X, Y, Z des chemins à trous possiblement vides. On demande de plus que $a\, I\, \Sigma(YB)$ et que $X = (q_1, \Sigma_1, q_2)$, $Y = (q_3, \Sigma_2, q_4)$, $q_2 \xrightarrow{a}_{\mathcal{A}} q_3$ et $Z = (q_1, \Sigma_1 \cup \{a\} \cup \Sigma_2, q_4)$. Il est possible que X, Y soient vides, auquels cas $q_1 = q_2$, $q_3 = q_4$, $\Sigma_1 = \Sigma_2 = \emptyset$.

Exemple 16 *Soit $I = \{(a,b),(a,c),(b,c)\}$ (c'est à dire toutes les lettres commutent). On donne l'exemple en figure 1.4 page suivante d'un automate I-diamant $\mathcal{B}$ construit à partir de l'automate à boucles I-connexes $\mathcal{A}$ de la figure 1.4 page suivante reconnaissant $L(\mathcal{A}) = \{abc\}$. On a $[L(\mathcal{A})]_I = \{abc, acb, bac, bca, cab, cba\}$. Nous avons représenté uniquement les états accessibles de b, plus un état non coaccessible pour illustrer les chemins à trous à considérer.*

Proposition 7 *[MP99] Soit $\mathcal{A}$ un automate tel que pour toute suite de mots non vides $x_1y_1 \cdots x_ky_k$ étiquetant un chemin de $\mathcal{A}$ avec $\forall i < j, x_i I y_j$, on a $k < N$. Alors l'automate $\mathcal{B}$ construit sur l'ensemble d'états Γ_N est tel que $L(\mathcal{B}) = [L(\mathcal{A})]_I$. Le nombre d'états de $\mathcal{B}$ est au plus $O(2^{|\Sigma|}|\mathcal{A}|^2)^N$.*

Preuve. On montre par récurrence sur la longueur $|u|$ de u que pour tout $\epsilon \xrightarrow{u}_{\mathcal{B}} (q_{2i-1}, \Sigma_i, q_{2i})_{i=1}^{n}$, alors il existe une famille de chemins $(\rho_i)_{i=1}^{n}$ de $\mathcal{A}$. De plus, pour $1 \leq i \leq n$, le chemin ρ_i part de q_{2i-1} et arrive à q_{2i}, et est étiqueté par un mot $x_i \in \Sigma_i^\star$, avec $x_0 \cdots x_{n-1} \sim_I u$.

Si $u = \epsilon$, alors c'est vrai. Si $u = va$, alors on a $\epsilon \xrightarrow{v}_{\mathcal{B}} AXYB \xrightarrow{a}_{\mathcal{B}} AZB$, avec $a\, I\, \Sigma(YB)$ et $X = (q_1, \Sigma_1, q_2)$, $Y = (q_3, \Sigma_2, q_4)$, $q_2 \xrightarrow{a}_{\mathcal{A}} q_3$ et $Z = (q_1, \Sigma_1 \cup \{a\} \cup \Sigma_2, q_4)$. Il est possible que X, Y soient vides, auquels cas $q_1 = q_2$, $q_3 = q_4$, $\Sigma_1 = \Sigma_2 = \emptyset$.

En appliquant l'hypothèse de récurrence à v, on a des chemins de $\mathcal{A}$ pour A, B étiquetés par u_1 et u_2. On a un chemin de q_1 à q_2 dans $\mathcal{A}$ étiqueté par u_3 correspondant à X, et un de q_3 à q_4 étiqueté par u_4 correspondant à Y.

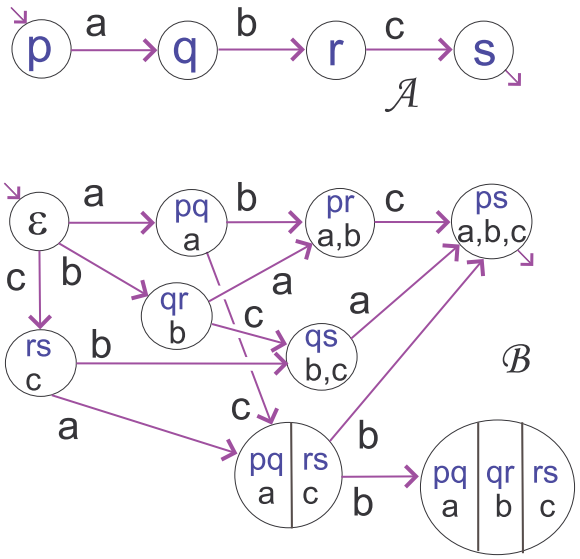


FIG. 1.4 – L'automate I-diamant $\mathcal{B}$ génère la clôture par commutation (totale) de l'automate à boucles I-connexes $\mathcal{A}$.

On a ainsi facilement un chemin de q_1 à q_4 correspondant à Z, étiqueté par u_3au_4. De même, $u = va \sim_I u_1u_3u_4u_2a \sim_I u_1u_3au_4u_2$ parce que aIu_4u_2.

Ainsi, si $u \in L(\mathcal{B})$, alors il existe un chemin de $q_0 \in Q_0$ à $q_f \in Q_f$ étiqueté par $v \sim_I u$, c'est à dire $u \in [L(\mathcal{A})]_I$.

Pour la réciproque, on montre par récurrence sur la taille de u que si $(uw) \in [L(\mathcal{A})]_I$, alors il existe un chemin τ à N trous avec $\epsilon \xrightarrow{u}_{\mathcal{B}} \tau$.

Le cas $u = \epsilon$ est trivial. Si $u = va$, alors il existe un chemin ρ acceptant de $\mathcal{A}$ étiqueté par un mot équivalant à uw, c'est à dire $\rho = \rho_1\rho_2 \cdots \rho_{2k}$ avec pour tout i, ρ_{2i} étiqueté par x_i et ρ_{2i+1} étiqueté par y_i, avec $u = x_1 \dots x_l$ et $w = y_1 \dots y_l$, et $\forall i < j, v_iIu_j$.

D'après l'hypothèse sur $\mathcal{A}$, on a $l < N$. Ainsi, en se servant de l'hypothèse de récurrence, on a $\epsilon \xrightarrow{v} \tau'$ avec τ' un chemin à N trous. D'après la définition de $\rightarrow_{\mathcal{B}}$, en posant τ le chemin à au plus N trous $\tau = (\rho_{2i})_{i=1}^{l}$, nous avons $\epsilon \xrightarrow{v} \tau' \xrightarrow{a} \tau$, ce qui prouve la récurrence. Prenant $w = \epsilon$, on a $[L(\mathcal{A})]_I \subseteq L(\mathcal{B})$, ce qui clôt la preuve de la proposition.

□

Il nous reste maintenant à montrer le lemme suivant pour clore cette direction du théorème d'Ochmański, montrant qu'on peut choisir $N = |\Sigma||\mathcal{A}|$.

Lemme 1 *Si $\mathcal{A}$ est un automate à boucles I-connexes sur l'alphabet de traces (Σ, I), alors pour toute suite de mots non vides $x_1y_1 \cdots x_ky_k$ étiquetant un chemin de $\mathcal{A}$ avec $\forall i < j, x_iIy_j$, on a $k \leq |\Sigma||\mathcal{A}|$. C'est à dire que $\mathcal{A}$ satisfait les conditions de proposition 7 page 35 avec $N = |\Sigma||\mathcal{A}|$.*

Preuve. La preuve est très aisée. Par l'absurde, supposons qu'il y ait un chemin $\rho_1\sigma_1 \cdots \rho_k\sigma_k$ étiqueté par $x_1y_1 \cdots x_ky_k$ avec $k \geq |\Sigma||\mathcal{A}| + 1$, et $x_i, y_j \neq \epsilon$, sauf peut être x_1 et y_k. D'après le lemme des chaussettes de Dirichlet, il existe un ensemble J de $|\Sigma|$ indices tel que ρ_j contient une certaine transition t_0 de $\mathcal{A}$ pour tout $j \in J$. On peut supposer sans restriction que cette transition arrive au début des ρ_j. Ainsi, $\rho_j \cdots \rho_l$ est une boucle, pour $j \neq l \in J$ consécutifs. On appelle $u_1 \dots u_{|\Sigma|}$ les étiquettes de ces boucles, avec $u_i \sim_I v_iw_i$ la décomposition où les v_i correspondent à $\rho_j \dots \rho_l$ et les w_i correspondent aux $\sigma_j \dots \sigma_l$. Parce que les boucles sont I-connexes, il existe a_iDb_i avec $a_i \in \Sigma(v_i)$ et $b_i \in \Sigma(w_i)$ pour chaque boucle. Comme b_j doit commuter avec a_i pour tout $i > j$, nous avons $b_j \neq b_i$. Ainsi, les $(b_i)_{i=1}^{|\Sigma|}$ sont deux à deux différents et différents de $a_{|\Sigma|}$, ce qui n'est pas possible parce qu'on n'a pas tant de symboles dans l'alphabet Σ. Cette contradiction clôt la preuve du lemme.

□

Nous passons maintenant à l'autre direction de la preuve du théorème d'Ochmański, à savoir que tout automate $\mathcal{A}$ dont le langage $L(\mathcal{A})$ est clos par commutation admet un automate à boucles I-connexes $\mathcal{B}$ avec $[L(\mathcal{B})]_I = L(\mathcal{A})$.

L'idée est que si l'on élimine les chemins de $\mathcal{A}$ qui ne sont pas en forme normale lexicographique, alors on obtiendra un automate $\mathcal{B}$ équivalent qui est à boucles I-connexes.

Définition 18 *Soit $\prec$ une relation d'ordre totale sur Σ. Elle définit récursivement une relation d'ordre, totale sur $\Sigma^\star$, par $u \prec v$ si soit $a \prec b$, soit $a = b$ et $u_1 \prec u_2$, avec $u = au_1 \in \Sigma^\star$ et $v = bv_1 \in \Sigma^\star$. De plus $\epsilon \prec u$ pour tout $u \neq \epsilon$. La* forme normale lexicographique *d'une trace $[w]_I$ est le mot $LNF([w]_I)$ minimal pour $\prec$ dans $[w]_I$.*

Proposition 8 *Pour tout mot $v \in \Sigma^\star$, on a $v = LNF(w)_I$ pour un mot $w \in \Sigma^\star$ ssi pour toute décomposition $v = xaybz$ avec $ayIb$, $a \prec b$.*

On montre ici que l'on peut tester à la volée si un mot est en LNF, et donc qu'on peut construire notre automate $\mathcal{B}$ équivalent à $\mathcal{A}$ et à boucles I-connexes.

On définit une fonction de mémoire $f : \Sigma \to 2^\Sigma$ qui à chaque lettre a associe l'ensemble des lettres qui ont été lues depuis la dernière occurrence de a (on y inclut a). En mémoire, on pourra représenter f comme une relation d'ordre sur Σ. Le nombre de relations d'ordre sur Σ est $|\Sigma|!$. En effet, $b \in f(a)$ est une relation d'ordre. Si $b \in f(c)$ et $c \in f(a)$, alors b apparaît après c qui apparaît après a, donc on a la transitivité. Si $a \in f(b)$, alors a est arrivé depuis la dernière occurrence de b, donc depuis a, on n'a pas vu de b, ce qui satisfait l'antisymétrie. On se sert aussi de f pour coder si on a déjà vu une lettre ou non, avec $f(b) = \emptyset$ si on a jamais vu de b. On notera Mem pour l'ensemble des fonctions de mémoire.

On définit $\mathcal{B}$ comme l'automate $(Q \times Mem, \Sigma, \to_\mathcal{B}, Q_0 \times (\emptyset), Q_f \times Mem)$, où $\to_\mathcal{B}$ est défini par $(q_1, f_1) \xrightarrow{a}_\mathcal{B} (q_2, f_2)$ si
- $q_1 \xrightarrow{a} q_2$
- $f_2(a) = \{a\}$ et $f_2(b) = f_1(b) \cup \{a\}$ pour $b \neq a$.
- Pour tout $f_1(b) \neq \emptyset$, soit $b \prec a$, soit $\exists c \in f_1(b)$, cDa.

La dernière propriété assure que les chemins sont en forme normale lexicographique, parce que pour tout b lu avant a, soit $b \prec a$, soit il existe c lu entre b et a avec cDa. Utilisant cet automate $\mathcal{B}$, on peut montrer que

Proposition 9 *Pour tout automate $\mathcal{A}$ clos par I-commutation, on peut construire un automate à boucles I-connexes $\mathcal{B}$ avec $[L(\mathcal{B})]_I = L(\mathcal{A})$, de taille $|\mathcal{B}| = |\Sigma|! \times |\mathcal{A}|$ (qui n'est donc exponentiel qu'en la taille de l'alphabet Σ).*

Preuve. Par construction de $\mathcal{B}$, on a $L(\mathcal{B}) \subseteq L(\mathcal{A})$, donc $[L(\mathcal{B})]_I \subseteq L(\mathcal{A})$ parce que $L(\mathcal{A})$ est clos par I-commutation.

Soit $u \in L(\mathcal{A})$. On montre que $v = \text{LNF}([u]_I) \sim_I u$ est reconnu par $\mathcal{B}$. Par l'absurde, supposons que $v = v_1 \dots v_n$ ne soit pas reconnu par $\mathcal{B}$. Comme il était reconnu par $\mathcal{A}$ et que les états finaux de $\mathcal{A}$ et $\mathcal{B}$ sont les mêmes, il existe j tel que $v_1 \cdots v_j$ n'étiquette pas un chemin initial de $\mathcal{B}$ (On choisit j minimal). Ainsi, il existe un lettre $b = v_i$ avec $v_j = a \prec b$, $i < j$ et $aIf(b)$. Cela veut dire que $v_1 \cdots v_{i-1} v_j v_i \cdots v_{j-1}$ est équivalent à $v_1 \dots v_j$, ce qui contredit le fait que v soit en LNF. Ainsi, $[L(\mathcal{B})]_I = L(\mathcal{A})$.

Montrons que $\mathcal{B}$ est à boucles I-connexes. Par l'absurde, supposons qu'une boucle de $\mathcal{B}$, étiquetée par w, ne soit pas I-connexe, ie $w \sim_I uv$ avec uIv et $u, v \neq \epsilon$. On écrit $w = u_1 v_1 \cdots u_n v_n$ avec u_i, v_i des mots non vides (sauf peut

être u_1 et v_n),tels que $u_1 \cdots u_n = u$, $v_1 \cdots v_n = v$. Entre autre, ww étiquette un chemin de $\mathcal{B}$. On pose a_i, b_i les premières lettres de u_i, v_i respectivement. Après avoir lu u_i, on a $f(a_i) \subseteq u_i$. Ainsi, quand on lit b_i, comme $b_i I f(a_i)$, on a $a_i \prec b_i$. De même, $b_i \prec a_{i+1}$. Quand on recommence la boucle, on va avoir $a_1 \prec b_1 \cdots \prec b_n \prec a_1 \prec b_1$, ie $b_1 \prec b_1$, une contradiction (on préfère utilisé b_1 plutôt que a_1 car a_1 peut ne pas exister).

□

On peut concevoir une trace $[u]_I \in \mathbb{M}(\Sigma, I)$ en tant qu'un *pomset*, c'est à dire une structure $(E, \leq_I, \lambda)$ avec

- E est un ensemble d'événements en bijection avec $\{0, \cdots, |u|\}$. On note γ cette bijection.
- $\lambda(e) = u_{\gamma(e)}$.
- $\leq$ est une relation d'ordre partiel sur $E \times E$ est la clôture transitive de :
  - $e \leq_I f$ si $\gamma(e) \leq \gamma(f)$ et $\lambda(e) D \lambda(f)$.

Ainsi, on peut définir des formules MSO sur les traces avec exactement le même formalisme que pour les mots. Une trace satisfait une formule MSO si le pomset associé satisfait la formule MSO, où $<$ dans la formule est interprété sur l'ordre partiel $\leq_I$.

Théorème 3 *[EM93] Un langage de traces L est définissable en MSO si et seulement si le langage de mots $[L]_I$ est définissable en MSO. Ainsi, la logique MSO sur les traces a le même pouvoir expressif que les langages réguliers de traces.*

Comme pour les mots, on peut définir des logiques moins expressives (mais aussi moins coûteuse), comme les logiques du premiers ordre (FO) qui est la restriction de MSO aux formules n'ayant pas de variables du second ordre. Il a été prouvé dans [EM93] que la logique FO sur les traces a le même pouvoir expressif que les langages réguliers sans étoile de traces.

De même, il est possible de définir des logiques temporelles (LTL). Au contraire des mots, deux notions différentes de logiques peuvent être définies avec des opérateurs propres. En effet, on peut assimiler un événement d'un mot au préfixe de ce mot qui termine dans cette état. En revanche, il y a des préfixes de traces qui ont plusieurs événements maximaux.

- Les logiques Temporelles globales parlent des préfixes de trace (ou configuration) [APP95, AMP98]. La plus connue est LTrL, équivalente à FO [TW02]. En fait, cette logique permet l'utilisation de modalité dans le

passé. Il a été prouvé dans [DG02] qu'en fait, ces modalités passées ne rajoutent pas d'expressivité à LTL.
– Les logiques Temporelles locales parle des événements des traces. Là encore, il a été prouvé que ces traces sont équivalentes à FO, même sans utilisation de modalités passées [DG04].

Si les logiques temporelles locales et globales sont expressivement équivalentes, les logiques globales semblent plus faciles à utiliser, mais la complexité de la satisfaisabilité de telles logiques est bien plus grande que celle des logiques locales. [Wal98] montre que la satisfaisabilité de LTrL est Non élémentaire, alors que [GK03] montre que la complexité pour LTL local est PSPACE.

1.3.3 Traces et Communication

On peut distinguer deux grands types de communication. La première est celle en cours sur internet (protocole TCP/IP), par passage de messages (ou plus précisément de paquets IP dans le cas d'internet).

Ainsi, deux ordinateurs communiquent entre eux en envoyant des informations avec un destinataire unique. L'autre type de communication est celle en cours à l'intérieur d'un système multiprocesseur, par variables partagées. Ainsi, deux processeurs communiquent entre eux en écrivant dans la mémoire certaines informations, que l'autre processeur pourra ensuite aller lire.

Quel genre de communication peut on représenter par les traces ? Il semble que le second type de communication, par variables partagées, soit assez simple à modéliser par un alphabet de traces (Σ, I). Ainsi, pour a, b des actions de deux processus différents sur des variables locales, on aura aIb, et si a ou b est une action concernant une variable partagée par les deux processus, alors aDb. Si a, b concernent le même processus, alors aDb.

Concernant le premier type de communication, qui nous intéresse plus particulièrement parce qu'il concerne les protocoles de communication, la tâche semble plus ardue.

Exemple 17 Prenons l'exemple très simple d'un protocole mettant en scène deux processus. Le premier envoie au second un nombre quelconque de messages. Nous appellerons ce protocole le protocole *asynchrone*.

Formellement, il est aisé de définir un alphabet de trace tel qu'une trace sur cet alphabet corresponde aux exécutions ayant les mêmes actions, mais

dans un ordre différent. L'idée est de modéliser chaque action (envoi ou réception) par une lettre différente, et de demander que aIb si et seulement si a, b sont un envoi et une réception ne formant pas un message. Néanmoins, cet alphabet est infini, ce qui ne nous satisfait pas. Pouvons nous trouver une représentation analogue, mais avec un alphabet fini ?

La première idée est de considérer un message en tant qu'action atomique α. Ainsi, l'alphabet contient juste α. Il n'y a donc pas de commutation. D'un certain point de vue, ce n'est pas très satisfaisant parce que on voudrait pouvoir modéliser que certaines actions peuvent commuter avec d'autres (les envois peuvent toujours arriver avant toutes les réceptions), alors que l'alphabet de traces fixe l'exécution a être toujours "envoi réception envoi $\cdots$". Pire, certains comportements ne peuvent pas être modélisés. Changeons le protocole légèrement, avec les mêmes messages α du processus 1 au processus 2, mais avec en plus un message β du processus 2 au processus 1 qui est envoyé avant les réceptions de α, et reçu après les envois de α. Alors on ne peut plus considérer les messages comme actions atomiques, parce que α et β arrivent en même temps. On ne peut pas non plus considérer l'exécution en entier, parce que il y a une infinité d'exécutions différentes, et on a un alphabet fini.

La seconde idée est de modéliser chaque envoi ou réception comme action atomique. Reprenant notre protocole asynchrone, on peut penser à un alphabet de traces à deux lettres $\{a, b\}$. La première réception ne doit pas commuter avec le premier envoi, parce qu'un message ne peut être reçu avant d'être envoyé. Ainsi aDb, ce qui ne nous satisfait pas. Sinon, il faudrait une action différente par événements, ce qui n'est pas possible sur un alphabet fini.

La dernière idée est de choisir une semi-trace, c'est à dire une trace pour laquelle la relation d'indépendance n'est pas symétrique. La relation de semi-commutation serait donc bSa, c'est à dire $ba \to ab$, les réceptions peuvent toujours être repoussées après n'importe quel envoi. Evidemment, comme la relation n'est pas symétrique, on peut avoir $u \to_S v$ mais pas $v \to_S u$. Par exemple, partant du mot $abababab\cdots$, on peut obtenir tous les réordonnancements possibles en appliquant S. Cependant, partant du mot $u = a\cdots ab\cdots b$, on n'obtient rien d'autre que u en appliquant S. On peut penser qu'une telle situation est correcte tant qu'il existe un mot qui génère tous les autres par applications de la relation de semi-commutation. Ce n'est toutefois pas le cas en général.

Prenons l'exemple de trois processus $1, 2, 3$. Le premier envoie deux mes-

sages au second, puis reçoit deux messages du troisième. Le second envoie deux messages au troisième puis reçoit les messages du premier. Enfin, le troisième processus envoie deux messages au premier processus, puis reçoit les messages venus du second processus. On suppose donc avoir 6 lettres $\{s_1, s_2, s_3, r_1, r_2, r_3\}$, correspondant respectivement aux envois du processus 1, 2, 3, puis aux réceptions des processus 1, 2, 3. La relation d'indépendance est qu'un envoi est indépendant d'un envoi d'un autre processus, une réception est indépendante des réceptions des autres processus. De plus, on a la relation de *semi-commutation* $r_2 s_1 \rightarrow s_1 r_2$, $r_3 s_2 \rightarrow s_2 r_3$ et $r_1 s_3 \rightarrow s_3 r_1$. Les relations d'indépendance et de semi-commutation forment la relation S. Nous montrons qu'aucun mot décrivant une exécution ne peut être réordonné par S en tous les mots corrects. Soit un mot u. Prenons le premier événement dans u qui correspond à une réception. On peut supposer que c'est r_1 par symétrie. Ainsi, les deux envois de s_1 ont déjà eu lieu, et les réceptions r_2 n'ont pas eu lieu. Il sera donc impossible de faire commuter le second s_1 avec le premier r_2, ce qui devrait normalement pouvoir se produire.

Ces exemples nous montrent combien il est difficile (voir impossible) de modéliser de manière satisfaisante les systèmes à passage de messages par des alphabets finis de traces. Les traces correspondent plus à une communication par variables partagées.

1.4 Des Machines Communicantes

Nous allons donc définir un modèle de machine qui répond à nos attentes. En un sens, nous allons étendre les automates finis, en leur permettant d'envoyer des messages, et en leur permettant d'avoir un contrôle distribué.

Définition 19 *Un* automate communicant $\mathcal{A}$ *(appelé aussi* Communicating Finite-state Machine *CFM) sur un ensemble de processus* $\mathcal{P}$ *est une famille d'automates* $\mathcal{A} = (\mathcal{A}^p)_{p\in\mathcal{P}}$*, plus deux ensembles* V_0, V_f*, où* $V_0, V_f \subseteq \prod_{p\in\mathcal{P}} V^p$ *sont les ensembles finis d'états initiaux et finaux respectivement. Pour tout processus* $p \in \mathcal{P}$*,* $\mathcal{A}^p = (V^p, \Sigma^p, \rightarrow_p)$*, où* Σ^p *contient des symboles* $p!q(a)$ *pour tout* $q \in \mathcal{P}$*, désignant un envoi du message* a *du processus* p *vers le processus* q*. De même, les symboles* $p?q(a)$ *désignent la réception par* p *d'un message* a *venu du processus* q*. Enfin, les symboles* $p(a)$ *désignent une action locale sur* p*. On notera* $\Sigma^p = S^p \uplus R^p \uplus L^p$ *la partition en envois, réceptions et actions*

locales sur p. Usuellement, V^p est l'ensemble fini des états de $\mathcal{A}_p$. $V_0, V_f \subseteq \prod_{p\in\mathcal{P}} V^p$ sont les ensembles finis d'états initiaux et finaux respectivement.

Les transitions de $\mathcal{A}^p$ sont de la forme $v \xrightarrow{a,m} v'$ où $v, v' \in S^p$ sont les états initiaux et finaux de la transition, $a \in \Sigma^p$ est l'étiquette de la transition, et $m \in \mathbb{N}$ est une donnée additionnelle de contrôle.

On dira que $\mathcal{A}$ n'a pas de données ajoutées si $m = 0$ pour toute transition $v \xrightarrow{a,m} v'$. Il sera à données ajoutées linéaires si $m \leq |V^p|$ pour tout $v \xrightarrow{a,m} v'$.

On comprend un automate communicant en tant qu'un réseau de machines qui communiquent via des canaux infinis FIFO, dont l'ensemble de configuration est $B_{p,q}$. C'est à dire qu'il n'y a pas de borne sur le nombre de messages que les canaux peuvent contenir, et ils délivrent les messages dans leur ordre d'arrivée. Formellement, $B_{p,q}$ est l'ensemble des files sur l'alphabet $\mathcal{C}^{p,q} \times \mathbb{N}$, où $\mathcal{C}^{p,q}$ est l'alphabet des lettres a telles qu'il existe $p!q(a)$ ou $p?q(a)$ $\in \Sigma_p$, et la composante de $\mathbb{N}$ contient les données de contrôle.

Définition 20 *On définit le comportement d'un automate communicant $\mathcal{A} = (\mathcal{A}^p)_{p\in\mathcal{P}}$ en tant q'un automate* **infini** *$\mathcal{B} = (V, \Sigma, \to, V_0, V_f)$ avec*

- *$V = \prod_{p\in P} V^p \times \prod_{p,q\in P} B_{p,q}$ est l'ensemble des configurations globales de $\mathcal{A}$.*
- *$\Sigma = \bigcup_{p\in\mathcal{P}} \Sigma^p$*
- *$((v_p)_{p\in\mathcal{P}}, (b_{p,q})_{p,q\in\mathcal{P}}) \xrightarrow{e} ((w_p)_{p\in\mathcal{P}}, (c_{p,q})_{p,q\in\mathcal{P}})$, $e \in \Sigma^p$ si $v_p \xrightarrow{e,m}_p w_p$, $v_q = w_q$ pour tout $q \neq p$ et l'une des propositions suivantes est satisfaite :*
  - *$e \in L^p$, $b_{r,s} = c_{r,s}$ pour tout $r, s \in \mathcal{P}$,*
  - *$e = p!q(a)$, $c_{p,q} = b_{p,q}(a, m)$, et $b_{r,s} = c_{r,s}$ pour tout $(r, s) \neq (p, q)$*
  - *$e = p?q(a)$, $b_{q,p} = (a, m)c_{q,p}$, et $b_{r,s} = c_{r,s}$ pour tout $(r, s) \neq (q, p)$.*

Intuitivement, si une action e est exécutée, alors le processus p change son état v_p en w_p conformément à $\to$. De plus, si $e = p!q(a)$, le message a est ajouté à la fin de la file $b_{p,q}$ avec l'information de contrôle m donnée par la transition. Si $e = p?q(a)$, le message a doit être au début de la file $b_{q,p}$ dont il est retiré, et on vérifie que l'information de contrôle est égale à celle donnée par la transition. Initialement, le CFM débute avec des canaux vides, et les automates sont sur une position initiale. Le CFM termine si les canaux sont vides et tous les processus sont dans un état final. On note comme d'habitude $L(\mathcal{A}) \subseteq \Sigma^\star$ pour le langage des exécutions reconnues par $\mathcal{A}$.

Reprenons l'exemple qui posait problème dans le cas des traces. On montre qu'on peut le modéliser aisément avec des CFM.

Exemple 18 *Reprenons le protocole asynchrone, mettant en scène les processus 1 et 2 (voir figure 1.5). Le premier envoie au second un nombre quelconque de messages α.*

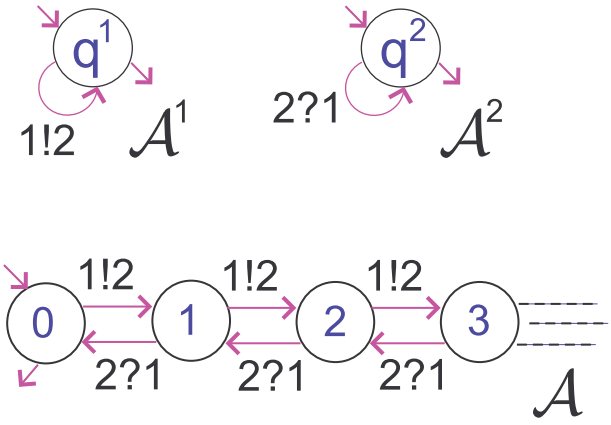


FIG. 1.5 – Un automate communicant $(\mathcal{A}^1, \mathcal{A}^2)$ et l'automate infini $\mathcal{A}$ décrivant son comportement.

Le CFM sans donnée additionnelle associé est un réseau de deux automates $\mathcal{A}^1, \mathcal{A}^2$. Les automates $\mathcal{A}^1, \mathcal{A}^2$ n'ont qu'un état q^1 et q^2 respectivement, qui sont initiaux et finaux. L'automate $\mathcal{A}^1$ a une transition $q^1 \xrightarrow{1!2(\alpha)} q^1$, et l'automate $\mathcal{A}^2$ a une transition $q^2 \xrightarrow{2?1(\alpha)} q^2$. L'automate $\mathcal{A}$ des comportements a des états qui sont des entiers, correspondant aux nombres de messages en transit dans le canal. Le langage est bien $\{u \in \{a,b\}^\star \mid |u|_a = |u|_b$ et $|v|_a \geq |v|_b$ pour tout préfixe v de $u\}$, où $a = 1!2(\alpha)$ et $b = 2?1(\alpha)$. En effet, quand il y a autant de b que de a, le canal $b_{1,2}$ est vide, donc le processus 2 ne peut plus recevoir. Notons au passage que ce langage n'est pas régulier, donc un automate fini ou un CFM à canaux bornés ne suffirait pas à décrire ce langage.

On appellera une configuration globale d'un CFM un *blocage* si elle n'est pas coaccessible dans le graphe des configurations globales. Un CFM $\mathcal{A}$ est *sans blocage* si aucune configuration accessible n'est un blocage. Cette notion est très importante, parce qu'on ne veut pas avoir de blocage pour un protocole de communication. De plus, il est difficile dans un protocole réel

de tester des états finaux globaux (et dans une moindre mesure des états initiaux globaux). On appellera un ensemble d'états globaux $T \subseteq \prod_{p\in\mathcal{P}} V^p$ un ensemble local si il existe des ensembles d'états $T^p \in V^p$ avec $\prod_{p\in\mathcal{P}} T^p = T$. Ainsi, on impose qu'un automate sans blocage ait des ensembles d'états initiaux et finaux qui sont locaux. On donne ici un exemple qui montre que le pouvoir expressif des automates sans blocage est strictement plus petit que celui des automates avec blocage.

Exemple 19 *Considérons le CFM $\mathcal{A}$ sur deux processus 1 et 2, qui ont chacun un état initial v_0 et deux états finaux v_a, v_b. De l'état v_0, chacun envoie soit a à l'autre et va dans l'état v_a, soit envoie b et va dans l'état v_b. Il y a une boucle sur v_a pour recevoir des messages a, et une boucle sur v_b pour recevoir des messages b, voir figure 1.6. Ainsi, le langage $L(\mathcal{A})$ contient* $1!2(a)2!1(a)1?2(a)2?1(a)$, $1!2(a)2!1(a)2?1(a)1?2(a)$, $2!1(a)1!2(a)1?2(a)2?1(a)$, $2!1(a)1!2(a)2?1(a)1?2(a)$ *et de même avec b qui remplace a.*

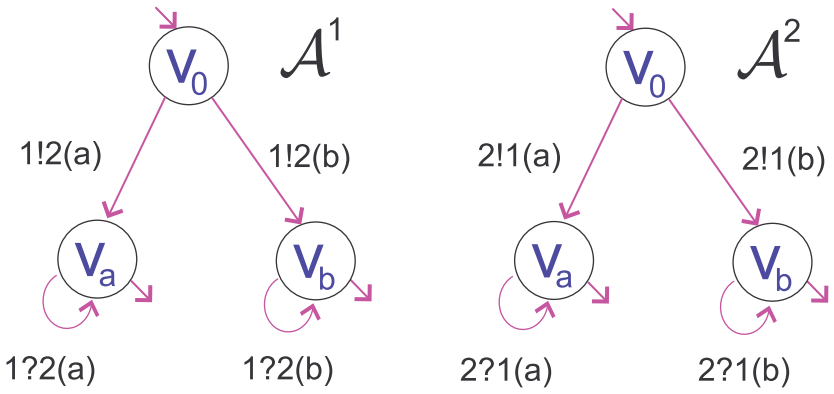


FIG. 1.6 – Un automate communicant $(\mathcal{A}^1, \mathcal{A}^2)$ avec blocage.

Cependant, ce langage n'est pas le langage d'un automate sans blocage. En effet, les automates $\mathcal{A}^1$ et $\mathcal{A}^2$ peuvent tous les deux choisir d'envoyer l'un a et l'autre b, puisqu'ils n'ont aucune information de l'autre automate. De cette configuration, on doit pouvoir atteindre une configuration finale. Cependant, $1!2(a)2!1(b)$ *n'est pas le préfixe d'un mot de $L(\mathcal{A})$, donc il n'est pas possible que $L(\mathcal{A})$ soit un langage de CFM sans blocage.*

Proposition 10 *Les CFM sans blocage sont strictement moins expressifs que les CFM. Savoir si un CFM $\mathcal{A}$ est sans blocage est indécidable [BZ83].*

En fait, les CFM sont un excellent modèle pour décrire les protocoles distribués. Il est facile de modéliser un protocole distribué par un CFM, et réciproquement, il est facile d'implémenter un algorithme distribué qui exhibe les comportements d'un CFM.

Brand et Zafiropulo [BZ83] ont démontré que les CFM étaient trop puissant pour permettre la décidabilité.

Proposition 11 *[BZ83] On peut simuler toute machine de Turing via un CFM, même si on dispose uniquement de deux processus déterministes. On dira que les CFM sont* Turing-équivalents. *Les CFM sans blocage ne sont pas Turing-équivalents, puisqu'on peut décider le vide d'un CFM sans blocage.*

On montre ici un résultat plus faible, qui est qu'il est indécidable de savoir si le langage d'un CFM est non vide, c'est à dire si un état final est accessible. On réduit pour cela PCP à ce problème.

Preuve. Soit une instance $(u_i, v_i)_{i=1}^n$ du PCP. On considère une variante de PCP, qui demande en plus à une suite d'indices $\alpha_1, \ldots, \alpha_k$ de satisfaire $|u_{\alpha_1} \cdots u_{\alpha_i}| \geq |v_{\alpha_1} \cdots v_{\alpha_i}|$ pour tout i, pour être une solution.

On fabrique à partir de cette instance un CFM à trois processus $\mathcal{P} = \{1, 2, 3\}$. Le premier processus devine l'indice i à utiliser et l'envoie au processus 2. Ensuite, il envoie au processus 3 les lettres correspondantes à u_i, puis il recommence.

Le processus 2 reçoit les messages de 1 et les renvoie à 3 immédiatement.

Le processus 3 reçoit l'indice i depuis le processus 2, puis attend de recevoir les messages venus du premier processus et étiquetés par v_i.

On a $L(\mathcal{A}) \neq \emptyset$ si et seulement si il y a une solution pour PCP. Cette solution est donnée en regardant la file $b_{1,2}$.

Evidemment, si $\mathcal{A}$ n'a pas de blocage, alors il suffit de savoir si une transition est tirable pour savoir si $L(\mathcal{A}) \neq \emptyset$.

□

Ainsi, on peut montrer facilement que l'équivalence de deux CFM et la recherche de motif dans un CFM sont indécidables. Le problème de recherche d'un motif v dans A est de savoir si il existe un mot $uvw \in L(A)$.

Preuve. Si on savait décider de l'égalité des CFM, on pourrait décider si un CFM est égal au CFM vide, ce qui est indécidable vu qu'on ne sait pas décider si $L(A) \neq \emptyset$.

De même, si on savait décider s'il existe une exécution qui contient un mot w, alors on pourrait décider de PCP. Reprenons le CFM défini ci-dessus pour l'indécidabilité du vide d'un CFM. On le change légèrement en rajoutant certaines actions. Le processus 1 peut décider de s'arrêter à un moment, auquel cas il envoie un message c au processus 2, qui le renvoie au processus 3 dans le cadre de son comportement normal. Après avoir lu c, les processus exécutent exactement w et finissent dans un état final.

Ainsi, w est exécutable dans ce CFM si et seulement si un état final était accessible dans le CFM décrit avant (pour peu que l'on fasse attention à utiliser des lettres pour la construction du CFM disjointes avec celle de w.

□

Il existe cependant une classe évidente de CFM qui permet la décidabilité : ceux ayant des canaux bornés. En effet, un CFM borné a un comportement d'automate fini. On peut cependant remarquer que cet automate est de taille exponentielle dans la taille de la borne sur les canaux.

Définition 21 *On appellera par* $\forall$-b-CFM *la classe des CFM à canaux universellement bornés par* b*. L'union de ses classes pour toute borne* b *sera notée* $\forall$-CFM*.*

Proposition 12 *Soit* $\mathcal{A}$ *un CFM.*

1. *Savoir si* $\mathcal{A} \in \forall$-*CFM est indécidable.*
2. *Savoir si* $\mathcal{A} \in \forall$-b-*CFM est indécidable.*
3. *Savoir si* $\mathcal{A} \in \forall$-b-*CFM est décidable si* $\mathcal{A}$ *est sans blocage.*

Preuve. On réduit les deux premiers problèmes au vide d'un CFM. Soit un CFM $\mathcal{A}$ dont on se demande si son langage est vide.

On va transformer légèrement $\mathcal{A}$ en un automate $\mathcal{B}$. On pose $1, \ldots, p$ les processus de $\mathcal{A}$. On lui rajoute un processus $p+1$ qui ne fait que recevoir des messages du processus 1 et son seul état est acceptant. On modifie le processus 1 en rajoutant un état, successeur de tout état final. Le seul état final dans $\mathcal{B}$ du processus 1 est ce nouvel état, qui peut envoyer des messages au processus $p+1$ (il y a une boucle étiquetée par $1!(p+1)$ sur ce nouvel état final). Si $L(\mathcal{A})$ est vide, alors $\mathcal{B}$ est universellement borné. Si $L(\mathcal{A})$ n'est pas vide, alors il est possible de déduire des exécutions de $\mathcal{B}$ qui finissent en bouclant sur l'état final de 1, c'est à dire qui sont non universellement bornées. Ainsi $L(\mathcal{A}) = \emptyset$ si et seulement si $\mathcal{B} \in \forall$-CFM.

De même, si on fixe un entier b, et qu'on enlève la boucle sur l'état final du processus 1 dans $\mathcal{B}$ et qu'on la remplace par $b+1$ transitions successives, alors $L(\mathcal{A}) = \emptyset$ si et seulement si $\mathcal{B} \in \forall\text{-}b\text{-}CFM$.

Si $\mathcal{A}$ est sans blocage, alors il suffit de construire l'automate fini des comportements de $\mathcal{A}$ avec des canaux b-bornés, et de voir si il y a une transition de $\mathcal{A}$ qui en sort. Si c'est le cas, comme $\mathcal{A}$ est sans blocage, alors on pourra l'étendre à une exécution qui est dans le langage de $\mathcal{A}$, mais pas b-bornée.

□

1.4.1 Diagrammes de Séquences (MSC)

On peut se demander si il est judicieux de représenter dans le langage d'un CFM toutes les exécutions alors qu'il y a beaucoup de redondance (à commutation près). De plus, considérer une linéarisation 1!2 2?1 1!2 1!2 2?1 n'est pas très visuel. Plutôt que de choisir arbitrairement une exécution pour représenter la classe qui le contient, il est plus naturel d'utiliser une représentation visuelle, les *Diagrammes de Séquences*, encore appelés *MSC* pour Message Sequence Charts. Notons que les Diagrammes de Séquences sont l'analogue des Diagrammes Temporels pour l'informatique distribuée.

Exemple 20 *Voila un MSC qui représente une exécution du protocole asynchrone avec 3 messages envoyés. Les deux lignes verticales représentent les deux processus, chaque flèche représente un message, la base de la flèche représentant l'action d'envoi du message et la pointe de la flèche sa réception. L'étiquette du message représente son contenu.*

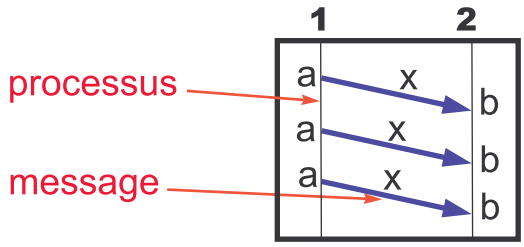


FIG. 1.7 – Un Diagramme de Séquence (MSC)

On définit ici plus formellement ce qu'est un MSC [itu96, LL95, Mau95].

Définition 22 *Un MSC M est un sextuplet $M = (\mathcal{P}, E, \mathcal{C}, \lambda, m, <)$ avec*
- *$\mathcal{P}$ est l'ensemble fini des processus*
- *E_p est l'ensemble fini des événements sur le processus p, avec $E = \bigcup_{p\in\mathcal{P}} E_p$.*
- *$\mathcal{C}$ est un ensemble d'étiquettes des messages.*
- *$\lambda : E \to \mathcal{T} = \{p!q(a), p?q(a), p(a) \mid p \neq q \in \mathcal{P}, a \in \mathcal{C}\}$ est la fonction de type, qui a un événement associe son type. On partitionne $E = S \uplus R \uplus L$ en envois $p!q(a)$, réceptions $p?q(a)$ et événements locaux $p(a)$.*
- *$m : S \to R$ la bijection de message qui à chaque envoi associe une réception, tel que si $m(s) = r$, alors $\lambda(s) = p!q(a)$ et $\lambda(r) = q?p(a)$ avec des certains $p, q \in \mathcal{P}, a \in \mathcal{C}$.*
- *$<\subseteq E \times E$ est une relation sur les événements, formée de*
  - *un ordre total $<_p$ sur E_p, pour tout processus $p \in \mathcal{P}$*
  - *$s < r$ pour tout message $m(s) = r$.*

On demande en outre que la clôture réflexive et transitive $\leq$ de $<$ soit un ordre partiel (donc en particulier, cette clôture est acyclique). De plus, on demande que si $s_1 <_p s_2$ et $\lambda(s_1) = \lambda(s_2) = p!q$, alors $m(s_1) <_q m(s_2)$ (FIFO).

Une fonction P qui à chaque événement $e \in E$ associe son processus $p \in \mathcal{P}$ est ainsi définie implicitement par λ. En effet, $P(e) = p$ ssi $\lambda(e) = p!q(a)$ ou $\lambda(e) = p?q(a)$ ou $p(a)$ pour $q \in \mathcal{P}, a \in \mathcal{C}$.

On peut voir alternativement un MSC comme un *pomset* $(E, \leq, \lambda)$. On définit le langage de mot $Lin(M)$ (ou $L(M)$) de M comme l'ensemble des linéarisations de M, c'est à dire les ordres totaux sur E compatible avec $\leq$. On note par une lettre arrondie $\mathcal{M}$ un ensemble de MSC. Pour un langage de MSC $\mathcal{M}$, on note par $Lin(\mathcal{M})$ ou $L(\mathcal{M})$ l'union des linéarisations des MSC $M \in \mathcal{M}$.

Exemple 21 *Reprenant le MSC de l'exemple 20 page précédente, l'ensemble de ses linéarisations est $\{ababab, aabbab, aababb, aaabbb\}$, avec $a = 1!2(x)$ et $b = 2?1(x)$.*

On définit maintenant les exécutions possibles d'une structure communicante en tant qu'un mot u de $\mathcal{T} = \{p!q(a), p?q(a), p(a) \mid p \neq q \in \mathcal{P}, a \in \mathcal{C}\}$, avec les propriétés que pour tout $p, q \in \mathcal{P}, a \in \mathcal{C}$, et pour tout préfixe v de u, on a $|v|_{q?p(a)} \leq |v|_{p!q(a)}$, c'est à dire qu'il n'y a jamais plus de messages reçus

qu'envoyés. De plus, $|u|_{q?p(a)} = |u|_{p!q(a)}$ signifie qu'à la fin, tous les messages envoyés on été reçus. Remarquons que toute linéarization d'un MSC satisfait ces conditions. Réciproquement, pour tout mot respectant ces conditions, on peut définir un MSC dont ce mot sera une linéarisation. Par abus de langage, nous appellerons donc ces exécutions des linéarisations.

Ainsi, pour un langage de mots $L \subseteq \mathcal{T}^\star = \{p!q(a), p?q(a), p(a) \mid p \neq q \in \mathcal{P}, a \in \mathcal{C}\}$, on note par $Lin(L)$ l'ensemble des mots $u \in L$ qui sont des "linéarisations", c'est à dire avec pour tout $p, q \in \mathcal{P}, a \in \mathcal{C}$, $|u|_{q?p(a)} = |u|_{p!q(a)}$ et pour tout préfixe v de u, $|v|_{q?p(a)} \leq |v|_{p!q(a)}$.

Exemple 22 *Pour un CFM $\mathcal{A}$, on a facilement $Lin(L(\mathcal{A})) = L(\mathcal{A})$, que l'on notera $Lin(\mathcal{A})$, c'est à dire que tout mot généré par un CFM est une linéarisation.*

Réciproquement, pour un langage de mots $L \subseteq \mathcal{T}^\star$, on notera par $\mathcal{L}(L)$ le langage des MSC qui ont **toutes** leurs linéarisations dans $Lin(L)$. Plus généralement, on notera le langage de MSCs générés par une structure K avec un $\mathcal{L}(K)$, et son langage de mot par $L(K)$.

Exemple 23 *Pour un CFM $\mathcal{A}$, on notera $\mathcal{L}(\mathcal{A})$ pour le langage $\mathcal{L}(L(\mathcal{A}))$ des MSC généré par $\mathcal{A}$. On peut remarquer que $Lin(\mathcal{L}(\mathcal{A})) = L(\mathcal{A})$.*

On peut transférer la borne sur les canaux de communication aux linéarisations, puis aux MSC [LM04]. On dira qu'une linéarisation $u \in \mathcal{T}^\star$ est b-bornée si pour tout préfixe v de u et $p, q \in \mathcal{P}$, $\sum_{a \in \mathcal{C}}(|v|_{p!q(a)} - |v|_{p?q(a)}) \leq b$, c'est à dire qu'il n'y a jamais plus de b messages envoyés et non reçus pour la linéarisation.

On appelle un MSC *universellement-b-borné* si toutes ses linéarisations sont b-bornées, et un ensemble de MSC universellement-b-borné si tous les MSC dans l'ensemble sont b-bornés. De même, on appelle un MSC *existentiellement-b-borné* si au moins une de ses linéarisations est b-bornée et un ensemble de MSC existentiellement-b-borné si tous les MSC dans l'ensemble sont existentiellement-b-bornés. L'ensemble des MSC $\exists$-b-bornés est noté $\mathbb{MSC}^b$. De plus, on notera l'ensemble des linéarisations b-bornés de M par $Lin^b(M)$.

Exemple 24 *Reprenant le MSC M de l'exemple 20 page 48, la linéarisation*

$1!2(x)1!2(x)1!2(x)2?1(x)2?1(x)2?1(x)$ *est* 3*-bornée. De même, la linéarisation* $1!2(x)2?1(x)1!2(x)2?1(x)1!2(x)2?1(x)$ *est* 1*-bornée. Ainsi,* M *est un MSC existentiellement 1-borné et universellement 3-borné.*

Quand on teste l'égalité de deux MSC, on peut abstraire les noms des événements car l'important est la relation d'ordre, donnée par l'étiquetage des événements. On demande toutefois que les MSC aient le même ensemble de processus (au minimum, on renomme au départ les processus pour que ça soit le cas). On peut transposer les algorithmes pour tester l'égalité des traces aux MSC.

Remarque 6 *On peut tester si deux MSC* M, N *sont égaux en testant si leurs ensembles de linéarisations sont égaux. Si on a une linéarisation* u *de* M*, alors on peut tester si* $M = N$ *en testant si* $u \in Lin(N)$*. De même que pour les traces, l'algorithme le plus efficace est de tester si pour tout processus* $p \in \mathcal{P}$*, les projections de* M *et* N *sur* p *correspondent.*

Si on définit un alphabet de trace (infini) sur les événements d'un MSC M, avec pour relation de dépendance aDb si a, b sont sur le même processus ou si b est la réception associée à a, alors on a $[L(M)]_I = L(M)$, c'est à dire que le langage d'un MSC est clos par commutation. Pour la suite de cette thèse, quand on s'intéressera à des processus qui envoient des messages, on ne nommera plus cette relation de commutation, elle sera juste LA relation de commutation. On notera ainsi $/L/$ pour la clôture de L par la relation de commutation. De même, on notera $u \sim v$ si u et v sont deux linéarisations du même MSC. On indexera en revanche d'autres relations de commutation. En fait, de nombreux résultats que nous donnerons seront obtenus en interprétant les MSC en tant que traces. Pour cela, nous aurons besoin de restreindre les canaux pour qu'ils soient existentiellement bornés.

On peut tester l'égalité de manière encore plus simple avec des représentants.

Définition 23 *Un ensemble de linéarisations* $L \subseteq \mathcal{T}^\star$ *est un ensemble de* représentants *pour un ensemble de MSC* $\mathcal{M}$ *si* $[L] = Lin(\mathcal{M})$*.*

Exemple 25 *Si* $\mathcal{M}$ *est existentiellement-b-borné, alors l'ensemble* $Lin^b(\mathcal{M})$ *des linéarisations b-bornées est un ensemble de représentants de* $\mathcal{M}$*.*

Proposition 13 *Soit deux ensembles de MSC* $\mathcal{M}, \mathcal{N}$*, et* L *un ensemble de représentants pour* $\mathcal{M}$*. Alors* $\mathcal{M} \subseteq \mathcal{N}$ *si et seulement si* $L \subseteq Lin(\mathcal{N})$*.*

Par exemple, si $\mathcal{M}$ est existentiellement-b-borné, alors il suffit de tester si $Lin^b(\mathcal{M})$ est inclus dans $Lin(\mathcal{N})$ pour savoir si $\mathcal{M} \subseteq \mathcal{N}$.

Remarque 7 *Dans le cas où on voudrait définir l'ensemble de tous les MSC qui ont au moins une linéarisation dans un ensemble L, on peut utiliser la clôture par commutation $[L]$ de L. Ainsi, on notera l'ensemble des MSC qui ont au moins une linéarisation dans L par $\mathcal{L}([L])$.*

1.4.2 Autres Représentations de la Communication

De même qu'il existe plusieurs types de formalismes pour les langages réguliers, il existent d'autre formalismes que les automates communicants pour décrirent des langages de MSC.

Nous pouvons commencer par donner une définition des langages réguliers de MSC. Soit un automate fini $\mathcal{A}$ générant uniquement des linéarisations sur l'alphabet $\mathcal{T}$. Si $\mathcal{A}$ est clos par commutation, alors on dira que son langage de MSC $\mathcal{L}(\mathcal{A}) = \mathcal{L}(L(\mathcal{A}))$ est régulier.

Exemple 26 *D'après le chapitre sur les automates communicants, on sait que le langage des automates communicants universellement bornés est régulier.*

Dans l'autre direction, si L est un langage de linéarisations reconnu par un automate $\mathcal{A}$, alors chaque boucle de l'automate est étiquetée par un même nombre d'envois que de réceptions sur le même canal (p, q), pour tout canal (p, q). Ainsi, chaque linéarisation reconnue par $\mathcal{A}$ est $|\mathcal{A}|$-bornée. Si de plus $\mathcal{A}$ est clos par commutation, alors les MSC de $\mathcal{L}(L)$ sont universellement $|\mathcal{A}|$-bornés. Ainsi,

Proposition 14 *Soit L un langage régulier de MSC. Alors il existe une borne b telle que L est universellement b-borné.*

En fait, Henriksen et al. [HMNK$^+$04] ont montré que tout langage régulier de MSC était implémentable par un automate communicant (voir théorème 4 page 54).

Nous donnons un formalisme logique, que nous appellerons MSO sur les langages de MSC. Le nom des processus est défini une fois pour toutes avant de définir la formule MSO, en ce que l'alphabet $\mathcal{T}$ contient déjà les noms des processus donnés par les lettres $p!q(a)$ ou $p?q(a)$ ou $p(a)$.

Définition 24 *Une formule MSO de MSC sur l'alphabet $\mathcal{T}$ est de la forme $\varphi := v_a(x) \mid \neg\varphi \mid \varphi \wedge \varphi \mid x < y \mid \exists X\varphi \mid \exists x\varphi \mid x \in X \mid y = \mathrm{msg}(x)$, où a décrit les lettres de $\mathcal{T}$. On demande que φ n'ait pas de variable libre pour être une formule complète.*

L'expression formelle de MSO sur les MSC n'est pas différente de celle de MSO sur les mots. Cependant, il faut bien garder en mémoire que l'écriture $x < y$ parle d'une relation d'ordre partielle et non plus totale comme pour les mots. Ainsi, nous pouvons adapter ce que signifie qu'un MSC satisfait une formule MSO complète φ.

Définition 25 *Un MSC $M = (\mathcal{P}, E, \mathcal{C}, t, m, <)$ satisfait une formule MSO φ sur l'alphabet $\mathcal{T}$ si $M, \emptyset \models \varphi$, où $M, f \models \varphi$ est défini par*

- *Soit Var l'ensemble des symboles de variables et $\mathcal{S}$ l'ensemble des symboles d'ensemble de variables qui apparaissent dans φ.*
- *f est une fonction (partielle) qui associe des symboles $x \in$ Var de variables à un événement de E, et des symboles $X \in \mathcal{S}$ d'ensemble de variables à un sous-ensemble de E.*
- *$M, f \models v_a(x)$ ssi $\lambda(f(x)) = a$.*
- *$M, f \models \neg\varphi$ ssi $M, f \not\models \varphi$.*
- *$M, f \models \varphi \wedge \psi$ ssi $M, f \models \varphi$ et $M, f \models \psi$.*
- *$M, f \models x < y$ ssi $f(x) < f(y)$.*
- *$M, f \models x \in X$ ssi $f(x) \in f(X)$*
- *$M, f \models \exists x\varphi$ ssi $\exists e \in E$ tel que $M, f' \models \varphi$, avec $f'(y) = e$ si $y = x$, $f(y)$ sinon.*
- *$M, f \models \exists X\varphi$ ssi $\exists S \subseteq E$ tel que $M, f' \models \varphi$, avec $f'(y) = S$ si $y = X$, $f(y)$ sinon.*
- *$M, f \models y = msg(x)$ ssi (x, y) est un message.*

L'ensemble des MSC satisfaisant une formule φ représente le langage $\mathcal{L}(\varphi)$.

Si on interprète une formule MSO($\leq$) (sans opérateur msg) sur les linéarisations, on peut définir $Lin(L(\varphi))$, qui est généralement inclus strictement dans $L(\varphi)$, c'est à dire que φ génère a priori des mots qui ne sont pas des linéarisations. De même, une formule MSO φ sur les linéarisations n'est pas close par commutation.

Exemple 27 *Prenons par exemple, $\varphi = \exists x, y(x \neq y \wedge \neg(x < y) \wedge \neg(y < x))$. Aucun mot ne satisfait cette formule, alors que n'importe quel MSC avec*

deux éléments incomparables la satisfait. Un MSC peut donc satisfaire une formule MSO alors qu'aucune de ses linéarisations ne satisfait la formule.

Prenons l'exemple d'un MSC avec une action locale 1(a) sur le processus 1, puis un message de 2 vers 1 : 2 !1(b), 1 ?2(b). Ainsi $1(a)$ *et* $2!1(b)$ *sont incomparables. Ce MSC ne satisfait pas la formule MSO*

$$\exists x, y, z, (v_{1(a)}(x) \wedge v_{2!1(b)}(y) \wedge v_{1?2(b)}(z) \wedge x < z \wedge y < z \wedge (x < y \vee y < x))$$

En revanche, ses deux linéarisations satisfont la formule. Il est donc possible que toutes les linéarisations d'un MSC satisfassent une formule φ*, mais pas le MSC. Ainsi, en général* $\mathcal{L}(L(\varphi)) \neq \mathcal{L}(\varphi)$*.*

Henriksen et al. [HMNK$^+$04] ont montré que les langages de MSC universellement bornés satisfaisant une formule MSO étaient aussi réguliers. On a ainsi un théorème proche du théorème pour les mots :

Théorème 4 *[HMNK$^+$04] Soit un langage* $\mathcal{M}$ *de MSC universellement bornés. Alors les propositions suivantes sont équivalentes :*
- $\mathcal{M}$ *est régulier.*
- $\mathcal{M}$ *est le langage d'un CFM.*
- $\mathcal{M}$ *est le langage d'une formule MSO.*

Nous ne donnons pas la preuve de ce théorème ici, car nous allons en montrer un théorème plus puissant, avec une restriction plus faible sur les canaux. Nous indiquons toutefois que la preuve utilise les automates asynchrones, et le théorème de Zielonka pour distribuer l'alphabet.

Comparé aux résultats dont on dispose pour les mots (théorème 1.1.1), il manque un modèle correspondant aux langages rationnels. De plus, il y a une grosse restriction sur les canaux de communication, qui doivent être universellement borné.

On peut aussi définir des formules LTL pour les MSC. On rappelle qu'une formule LTL sur l'alphabet $\mathcal{T}$ est de la forme $\varphi := a \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathcal{X}\varphi \mid \varphi \mathcal{U} \varphi$, où a décrit des lettres sur l'alphabet $\mathcal{T}$.

Exemple 28 *Prenons l'alphabet* $\mathcal{T} = \{1!2, 2?1, 3!4, 4?3\}$ *concernant quatre processus. On considère la formule* $\varphi = G((\neg 2?1) \vee F(3!4))$ *(on rappelle que* G *et* F *sont des macros usuels définis dans l'exemple 6 page 24). La formule* φ *exprime que si une réception est vue sur le processus 2 depuis le processus 1, alors on doit voir plus tard un envoi du processus* 3 *vers* 4*.*

On peut voir que $2!1\,2?1\,3!4\,4?3 \models \varphi$ *et que* $3!4\,4?3\,2!1\,2?1 \not\models \varphi$*. Pourtant, ces deux linéarisations viennent du même MSC avec deux messages, un du processus 1 vers 2, et un du processus 3 vers 4.*

Ainsi, le langage d'une formule LTL n'est pas clos par commutation[5].

Définition 26 *Un MSC satisfait une formule φ si toutes ses linéarisations satisfont φ. On note l'ensemble des MSC satisfaisant φ par $\mathcal{L}(\varphi)$, c'est à dire $\mathcal{L}(\varphi) = \mathcal{L}(L(\varphi))$*

1.5 MSC-graphes

Comme nous l'avons remarqué précédemment, nous n'avons pas défini de modèle comparable aux langages rationnels pour les mots ou les traces. C'est ce que nous essayons de faire ici. Nous allons donner une définition qui est assez proche des rationnels de mots, sauf qu'au lieu de lettres, on va utiliser des MSC (ce que nous avons déjà fait pour les formules MSO pour les MSC).

On définit tout d'abord une *concaténation de MSC*. Soit M_1, M_2 deux MSC concernant le même ensemble de processus $\mathcal{P}$, avec $M_i = (\mathcal{P}, E_i, \mathcal{C}_i, \lambda_i, m_i, <_i)$. Alors la concaténation de M_1 et M_2, notée $M_1 M_2$, est un MSC $M = (\mathcal{P}, E_1 \uplus E_2, \mathcal{C}_1 \cup \mathcal{C}_2, \lambda_i \uplus \lambda_2, m_i \uplus m_2, <)$, où $<$ est une relation acyclique formée de :

- $a < b$ si $a <_i b$ pour $i \in \{1, 2\}$
- $a < b$ si $P(a) = P(b)$ et $a \in E_1, b \in E_2$

Intuitivement, on concatène M_1 et M_2 en collant les processus communs à M_1 et M_2 ensemble, avec les événements de M_1 avant ceux de M_2.

Exemple 29 *On présente ici la concaténation de M par M. Par définition, on commence par renommer les événements de la deuxième copie de M en des lettres non encore utilisés, et on rassemble les deux copies de M en collant la seconde copie de M en dessous de la première.*

Définition 27 *Les* langages rationnels de MSC *sur les processus $\mathcal{P}$ est la plus petite classe qui contient les singletons MSC sur $\mathcal{P}$, close pas union, concaténation et étoile de Kleene.*

[5]Ce qui montre aussi que les langages de mots de formules MSO ne sont pas clos par commutation.

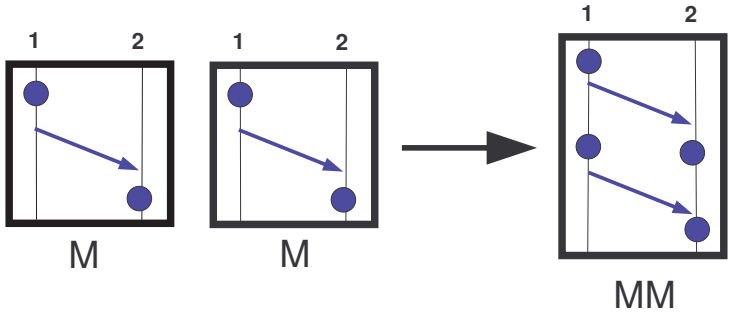


FIG. 1.8 – La composition de M par M.

Evidemment, en vertu de l'égalité des rationnels de mots avec les réguliers, on peut représenter un langage rationnel de MSC par un automate dont les états sont étiquetés par des MSC. On appellera une telle structure un *graphe de diagramme de séquences*, ou encore *MSC-graphe* [MR97].

Définition 28 *Un MSC-graphe est un quintuplet $G = (V, \rightarrow, \lambda, V_0, V_f)$, avec*
- *$(V, \rightarrow)$ est un graphe.*
- *λ est une fonction qui à chaque nœud $v \in V$ associe un MSC.*
- *$V_0 \subseteq V$ est l'ensemble des états initiaux.*
- *$V_f \subseteq V$ est l'ensemble des états finaux.*

Implicitement, un MSC-graphe définit un ensemble de MSC à partir duquel les MSC du langage sont construits, l'ensemble des MSC étiquetant un noeud du MSC-graphe. De même, il définit un ensemble de processus $P(G) = \bigcup_{v \in V} P(\lambda(v))$.

Pour calquer la définition des langages rationnels de MSC, on définit le langage d'un MSC-graphe comme l'ensemble des MSC qui sont des concaténations de MSC lus sur un chemin acceptant du graphe.

Définition 29 *On note par $\mathcal{L}(G) = \{\lambda(v_0) \cdots \lambda(v_n) \mid v_0 \ldots v_n$ est un chemin acceptant de $G\}$ le langage d'un MSC-graphe G .*

Il est toujours plus aisé de comprendre le comportement des linéarisations générées par un MSC-graphe G. Comme il n'est pas toujours possible de calculer toutes les linéarisations, nous expliquons ici comment obtenir un

ensemble $L(G)$ de représentants de $\mathcal{L}(G)$. On commence par choisir une linéarisation de chaque MSC, c'est à dire $\hat{\lambda}$ associe à chaque nœud $v \in V$ du graphe une linéarisation de $\lambda(v)$. On note alors ce langage des mots de G par $L(G)$. Tout mot généré est une linéarisation.

Définition 30 *Soit G un MSC-graphe. Alors $L(G)$ est l'ensemble des linéarisations qui suivent les nœuds de G. C'est à dire $x_1 \cdots x_n \in L(G)$ s'il existe un chemin acceptant $v_0 \cdots v_m$ dans G et une suite d'indices $k_1 < \cdots < k_m$ tels que $x_{k_i} \cdots x_{k_{i+1}-1} = \hat{\lambda}(v_i)$.*

Cependant, en général, $L(G) \subsetneq Lin(\mathcal{L}(G)) = [L(G)]$. En effet, les linéarisations générées par l'automate sous jacent au MSC-graphe G doivent suivre les états du graphe : on ne commence pas une linéarisation d'un nouveau nœud tant qu'on n'a pas fini la linéarisation du nœud courant. Cependant, il est bien possible qu'une linéarisation d'un MSC généré par le MSC-graphe G commence une linéarisation sur un nouveau nœud alors qu'un autre nœud n'est pas fini, parce que rien dans l'ordre nous empêche de le faire. Cependant, cet ensemble de linéarisation $L(G)$ n'est pas inutile parce qu'il représente un ensemble de représentants de $Lin(\mathcal{L}(G)) = [L(G)]$. C'est à dire que si l'on a $L(G)$, on peut théoriquement récupérer tout $Lin(\mathcal{L}(G))$[6].

Exemple 30 *Prenons le MSC-graphe G composé d'un seul nœud initial et final (voir figure 1.9 page suivante), avec une boucle simple sur ce nœud, et étiqueté par un MSC contenant un unique message du processus 1 au processus 2. Ainsi, les MSC qui composent ce langage sont exactement les MSC du protocole asynchrone. En particulier, le MSC M de l'exemple 20 page 48 est engendré par G.*

Alors les linéarisations de M sont par définition des linéarisations engendrées par G. Appelons a pour l'envoi du message et b pour l'action de sa réception. En particulier, abababab est une linéarisation de M. De plus, cette linéarisation appartient à $L(G)$ parce qu'elle suit les états de G. En revanche, la linéarisation $aaabbb$ de M n'est pas dans $L(G)$, parce que on commence un nouveau nœud en lisant une lettre a, alors qu'il reste une lettre b à lire dans le nœud courant.

On peut remarquer qu'il existe dans les normes définies par les organismes internationaux, un norme particulièrement proche des MSC-graphes

[6]Notons cependant qu'en pratique, il se peut que cette opération de clôture soit non calculable.

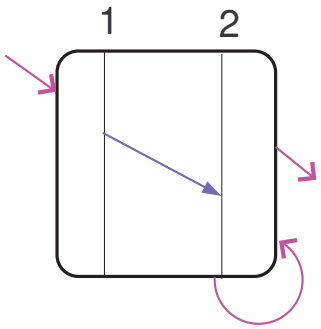


FIG. 1.9 – Un MSC-graphe

[itu96]. En l'occurrence, la norme [itu96] propose un modèle (les *High Level MSC,HMSC*) plus général que les MSC-graphes, en ce qu'elle permet de décrire exactement ceux ci, munis de possibilités additionnelles, comme la hiérarchie et les ports de communication. Cependant, modulo un dépliage des HMSC, on obtient un MSC-graphe équivalent. Nous ne nous occuperons donc pas des HMSC en tant que telles, sauf à la fin de cette thèse, pour parler des résultats qui changent.

On peut alors se demander si les MSC-graphes sont la pièce manquante du théorème, c'est à dire si les MSC-graphes universellement bornés sont expressivement équivalents aux CFM bornés. Nous pouvons commencer par comparer visuellement les deux approches. Les CFM ont comme brique de base les processus. Ils représentent en un tenant le comportement du processus (par un automate), et permettent de composer ces processus en parallèle, récupérant ainsi les messages. On peut donc couper les messages.

De l'autre côté, les MSC-graphes ont comme brique de base les messages (ou plutôt les MSC), et permettent de les composer par une composition séquentielle[7]. En revanche, c'est le comportement des processus qui peut être coupé, et ainsi les choix qui peuvent être faits sont non locaux.

Nous dirons qu'un MSC-graphe est *implémentable* si il est équivalent à un CFM.

[7]Paradoxalement, la composition séquentielle est plus forte que la composition parallèle ici, parce que si on compose séquentiellement des MSC sur des processus différents, alors on obtient la composition parallèle de ces MSC. On peut observer ce phénomène sur l'exemple 1.8 page 56, où les actions locales sur le processus 1 et 2 se composent parallèlement quand on applique la composition séquentielle.

Exemple 31 *Prenons le MSC-graphe G sur deux processus $\{1, 2\}$ de la figure 1.10, avec un seul nœud, initial et final, avec une boucle simple sur lui. Le nœud est étiqueté par le MSC avec deux actions locales, l'action $1(a)$ sur le processus 1 et l'action $2(b)$ sur le processus 2. Alors $Lin(\mathcal{L}(G)) = \{w \mid |w|_{1(a)} = |w|_{2(b)}\}$, qui n'est pas régulier. Il ne peut donc pas être représenté par un CFM.*

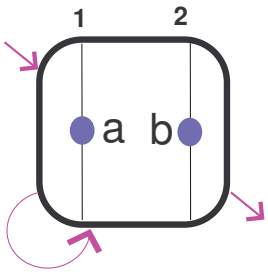


FIG. 1.10 – Un MSC-graphe non implémentable

Ainsi, il existe des MSC-graphes universellement bornés (et même sans communication) qui ne sont pas réguliers. En fait, la raison vient du fait que les MSC-graphe ont un contrôle global. Par là, on veut dire que chaque choix n'est à priori pas pris par un seul processus, comme c'est le cas pour les CFM, mais plutôt par un ensemble de processus.

Exemple 32 *Prenons le CFM $\mathcal{A} = (\mathcal{A}^1, \mathcal{A}^2)$ à deux processus $\{1, 2\}$. L'automate $\mathcal{A}^1$ a un état initial v_0^1 et deux états finaux v_f^1, v_g^1. L'automate commence dans l'état v_0^1, envoie un message au processus 2 et passe dans l'état v_f^1. De là, il peut envoyer un message à 2 en allant dans l'état v_g^1. Il revient de l'état v_g^1 à l'état v_f^1 en recevant un message depuis le processus 2.*

L'automate $\mathcal{A}^2$ a un état initial v_0^2, deux états finaux v_f^2, v_g^2 et un autre état v_1^2. Il commence dans l'état v_0^2, et peut envoyer un message au processus 1 et passer dans l'état v_1^2. De là, il peut recevoir un message de 1 en retournant dans l'état v_0^2. Sinon, il peut choisir depuis v_0^2 de recevoir un message venu du processus 1 en allant dans l'état v_f^2, voir encore un autre pour aller dans v_g^2. Il est à noter que ce protocole est universellement 2-borné et sans blocage.

Cependant, il semble difficile de composer plusieurs MSC pour obtenir un MSC dans le langage $\mathcal{L}(\mathcal{A})$. En effet, comment couper un MSC du langage

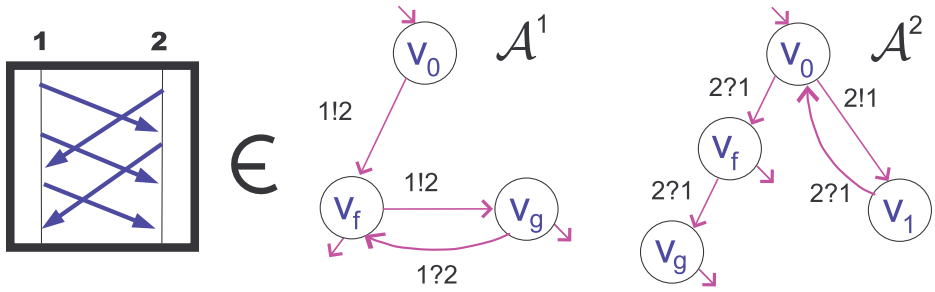


FIG. 1.11 – Un MSC non décomposable issu d'un CFM $(\mathcal{A}^1, \mathcal{A}^2)$ universellement borné et sans blocage.

pour l'exprimer en tant que concaténation de deux MSC (non vides) ? Il faudrait ainsi un nœud dans le MSC-graphe par MSC dans le langage, c'est à dire un nombre infini de nœuds.

Ainsi, il existe des CFM universellement bornés (et même sans blocage) qui génèrent des langages qui ne peuvent pas être reconnu par un MSC-graphe. Nous allons expliquer ici plus formellement d'où provient le problème.

Définition 31 *On appelle un MSC M un* atome *[HM00, HMNK+04](ou encore premier [Mor02]) s'il n'existe pas de MSC $S, T \neq \epsilon$ avec $ST = M$.*

Il est facile de montrer que chaque MSC admet une décomposition unique (à commutation près) en atomes. On notera par $At(M)$ l'ensemble des atomes de la décomposition atomique d'un MSC M. Pour un ensemble de MSC $\mathcal{M}$, on notera $At(\mathcal{M})$ pour l'ensembles des atomes des MSC dans $\mathcal{M}$. On dira que $\mathcal{M}$ est *finiment engendré* si $At(\mathcal{M})$ est fini. On notera $At(G) = \bigcup_{M=\lambda(v),v\in V} At(M)$ pour un MSC-graphe $(V, \rightarrow, \lambda, V_0, V_f)$. On a $At(\mathcal{L}(G)) = At(G)$, donc G est finiment engendré. Le CFM de l'exemple 32 page précédente n'est pas finiment engendré (parce que le MSC de l'exemple est un atome), donc son langage ne peut pas être celui d'un MSC-graphe.

Proposition 15 *Tout langage finiment engendré est existentiellement borné.*

Preuve. La borne existentielle d'un ensemble de MSC $\mathcal{M}$ est exactement la borne maximale des MSC appartenant à $At(\mathcal{M})$. En revanche, on n'a

aucune information a priori concernant une quelconque borne universelle. □

Ainsi, nous avons deux directions qui expliquent que les CFM et les MSC-graphes sont incomparables. D'un côté, les bornes sur les canaux et la génération par un ensemble fini d'atomes, parce que les MSC-graphes sont finiment engendrés et pas les CFM. D'un autre côté, le contrôle, global pour les MSC-graphes alors qu'il est local pour les CFM.

Ces directions sont d'autant plus importantes qu'elles concernent aussi la décidabilité et la complexité des algorithmes, comme l'égalité et la recherche de motifs. En effet, ne pas avoir de bornes sur les canaux rend les problèmes indécidables. De même, avoir un contrôle global rend l'égalité indécidable, même si les canaux sont universellement bornés[8].

On montre que l'égalité de deux MSC-graphes est indécidable. On réduit ce problème à celui de tester si $[L]_I = [K]_I$ pour L, K réguliers, et $(\sigma, I) = (\{a, b, c\}, (a, c), (c, a), (b, c), (c, b))$, qui est indécidable [DR95]. On produit deux MSC-graphes sur deux processus 1,2. Le premier processus est le processus des lettres dépendantes a, b, et le second est le processus de la lettre c. On a ainsi trois actions locales $1(a), 1(b), 1(c)$, et pas de communication. Tester si les deux MSC-graphes obtenus depuis les automates de traces sont égaux revient a tester si les automates sont égaux, ce qui est indécidable.

[8]La recherche de motif pour les MSC-graphes est elle décidable, nous le montrerons dans la seconde partie, voir le théorème 13 page 147.

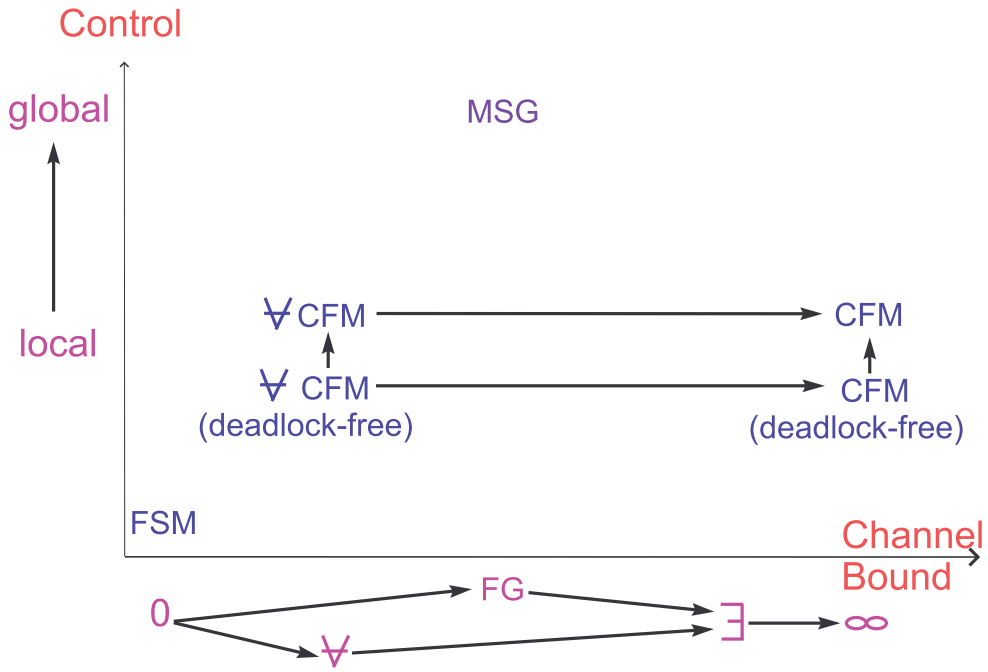


FIG. 1.12 – On peut lire ici le premier diagramme qui compare MSC-graphes (appelé ici MSG) et les CFM. On note FSM pour les automates de mots. Les deux directions considérées sont les bornes sur les canaux (channel bound), qui peuvent être soit pas de communication (0), soit une borne universelle sur les canaux (∀), soir une borne existentielle sur les canaux (∃), soit pas de borne (∞). Enfin, on indique le fait d'être finiment engendré par FG. On compare aussi les structures suivant leur contrôle, qu'il soit global ou local (ou local sans blocage). Généralement, les structures placées exactement en haut d'autres structures sont plus expressives. Néanmoins pour le sens horizontal se pose le problème des structures finiment engendrées et universellement bornées, qui ne sont pas comparables (mais comparables avec toutes les autres). Pour aider le lecteur, on ajoute des flèches simples pour exprimer l'inclusion stricte d'une structure dans une autre. On notera par une égalité le fait que deux structures sont expressivement équivalentes. Enfin, nous différencions les CFM sans blocage (deadlock-free) des CFM avec blocage.

Chapitre 2

One Theorem to Bind Them All

Ash nazg durbatulûk,
Ash nazg gimbatul,
Ash nazg thrakatulûk
agh burzum-ishi krimpatul.
J.R.R. Tolkien

La première section sert à expliquer que la comparaison des MSC-graphes et des CFM est excessivement importante pour les machines communiquant par messages. En particulier, comparer les structures n'est pas juste trouver des restrictions de chaque modèle qui se correspondent. Dans la première section, nous donnons un formalisme expressif, les *CMSC-graphes*, et nous prouvons qu'il contient en même temps les CFM et les MSC-graphes. De plus, il est facile de restreindre ce formalisme structurellement (ce qui n'est pas le cas des CFM). Ainsi, nous donnons une restriction dans la deuxième section, qui correspond à avoir des bornes existentielles sur les canaux, ce qui permet la décidabilité de certains problèmes. Puis nous définissons dans la troisième section la restriction de coopération, que nous prouvons ne pas être très restrictive. Enfin, nous prouvons dans la quatrième section une extension du théorème de Kleene pour les structures communicantes bornées existentiellement sur les canaux.

2.1 Comparer les MSC-graphes avec les Automates Communicants

Nous venons de définir au premier chapitre des structures plus ou moins décidables, plus ou moins expressives, avec ou sans blocages. Il est bien entendu illusoire de trouver un modèle qui regroupe toutes les bonnes propriétés (décidable avec une complexité petite, expressif et implémentable sans blocage).

Essayons de formaliser un peu ce que signifie être implémentable et expressif. Le formalisme de la communication qui cadre le mieux avec les algorithmes distribués est sans doute les CFM, par leur nature distribuée. Ainsi, un formalisme est expressif si il contient les CFM existentiellement bornés. Demander de contenir tous les CFM est sans doute un peu trop restrictif car cela engendrerait l'indécidabilité intrinsèque aux CFM (en fait, aux canaux non bornés des CFM). De même, être implémentable, c'est avoir un algorithme pour construire un CFM équivalent. Si de plus une classe est incluse dans les CFM sans blocage, alors l'implémentation sera sans blocage. Cela montre que au delà de l'aspect théorique, trouver des classes comparables de MSC-graphes et de CFM est important pour connaître l'expressivité et l'implémentabilité des MSC-graphes. Gardant en tête que nous aurons besoin d'algorithmes rapides dans la seconde partie, une classe un peu moins expressive qu'une autre peut également être intéressante si sa complexité se révèle être moins élevée. Ainsi, nous allons nous occuper à remplir le schéma 1.12 page 62.

2.1.1 Les CMSC-Graphes

La première tâche que nous nous assignons est de trouver un formalisme dans lequel nous pouvons exprimer les deux modèles incomparables que sont les CFM et les MSC-graphes. Rappelons nous que les MSC-graphes représentent les messages en un tenant, et ont le droit de couper les processus, alors que les CFM représentent les processus en un tenant, alors qu'ils ont le droit de couper les messages. L'idée est donc de permettre au modèle de couper les messages et les processus. Nous allons ainsi adapter les MSC-graphes en leur permettant de couper les messages, en suivant l'idée de [GMP01].

La première chose à faire est de définir les nouvelles briques de base, que nous appellerons *MSC compositionnels*, ou encore *CMSC*. Intuitivement, les

CMSC sont des MSC où l'on peut spécifier une action d'envoi sans spécifier la réception associée (ou l'inverse). Toutefois, les processus envoyeurs et destinataires sont fixés par le type du message. On appellera *solitaires* de tels événements. Ainsi, la seule différence avec un MSC est que la fonction de message m ne doit plus être une bijection entre l'ensemble des envois S et des réceptions R.

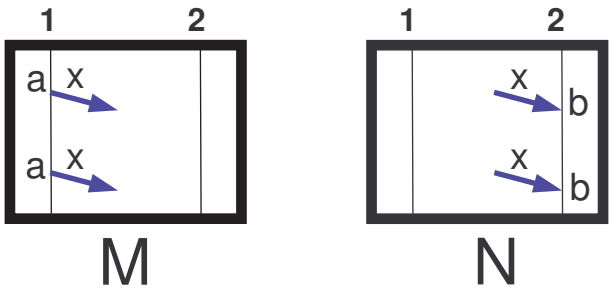


FIG. 2.1 – Un CMSC M représentant deux envois de messages vers le processus 2, et un CMSC N représentant la réception de deux messages sur 2 depuis 1.

On définit ici plus formellement ce qu'est un CMSC.

Définition 32 *Un CMSC M est un sextuplet $M = (\mathcal{P}, E, \mathcal{C}, \lambda, m, <)$ avec*
- *$\mathcal{P}$ est l'ensemble fini des processus*
- *E_p est l'ensemble fini des événements sur le processus p, avec $E = \bigcup_{p\in\mathcal{P}} E_p$.*
- *$\mathcal{C}$ est un ensemble d'étiquettes des messages.*
- *$\lambda : E \to \mathcal{T} = \{p!q(a), p?q(a), p(a) \mid p \neq q \in \mathcal{P}, a \in \mathcal{C}\}$ est la fonction de type, qui a un événement associe son type. On partitionne $E = S \uplus R \uplus L$ en envois $p!q(a)$, réceptions $p?q(a)$ et événements locaux $p(a)$.*
- *$m : S \to R$ une fonction (partielle) de message qui à chaque envoi associe une réception, tel que si $m(s) = r$, alors $\lambda(s) = p!q(a)$ et $\lambda(r) = q?p(a)$ avec des certains $p, q \in \mathcal{P}, a \in \mathcal{C}$.*
- *$< \subseteq E \times E$ est une relation sur les événements, formée de*
  - *un ordre total $<_p$ sur E_p, pour tout processus $p \in \mathcal{P}$*
  - *$s < r$ pour tout message $m(s) = r$.*

Nous appellerons les graphes dont les nœuds sont étiquetés par ces CMSC des CMSC-graphes, par analogie aux MSC-graphes. Comme nous l'avons fait pour les MSC-graphes, il suffit de définir ce qu'est la concaténation des CMSC pour définir les CMSC-graphes. Intuitivement, on va permettre de composer les envois et réceptions solitaires ensemble.

Il y a cependant des problèmes avec les CMSC. Le premier est que l'on veut uniquement des MSC à la fin de la composition des CMSC vus sur un chemin, même si on a besoin lors des calculs intermédiaires de CMSC. Le second problème est qu'il y a a priori plusieurs compositions des mêmes CMSC. Enfin, il est possible qu'aucun CMSC soit concaténation de deux CMSC à cause de la restriction FIFO.

Exemple 33 *Si on reprend l'exemple de la figure 2.1 page précédente, concaténer M avec N peut aboutir à deux messages, mais peut aussi aboutir à une réception solitaire, un message et un envoi solitaire. En effet, il se peut que plus tard, on concatène avant la première réception sur le deuxième processus une émission du message.*

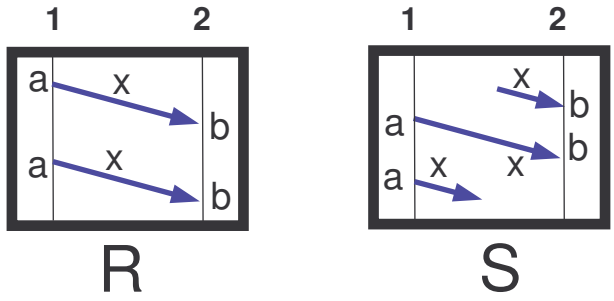


FIG. 2.2 – Les CMSC R, S sont deux concaténations de MN, où M et N sont les CMSC de la figure 2.1 page précédente.

Définition 33 *Soit M_1, M_2 deux CMSC concernant le même ensemble de processus $\mathcal{P}$, avec $M_i = (\mathcal{P}, E_i, \mathcal{C}_i, t_i, m_i, <_i)$. Alors la concaténation de M_1 et M_2, notée $M_1 M_2$, est un ensemble de CMSC $M = (\mathcal{P}, E_1 \uplus E_2, \mathcal{C}_1 \cup \mathcal{C}_2, t_1 \uplus t_2, m, <)$, où m est une fonction (possiblement partielle) de $S \to R$ et $<$ une relation acyclique, avec :*

- *Si* $b = m_i(a)$ $i \in \{1, 2\}$, *alors* $b = m(a)$.
- m *respecte l'ordre FIFO.*
- $a < b$ *si* $a <_i b$ *pour* $i \in \{1, 2\}$.
- $a < b$ *si* $b = m(a)$.
- $a < b$ *si* $P(a) = P(b)$ *et* $a \in E_1, b \in E_2$.

On peut ainsi définir récursivement l'ensemble des compositions d'une suite de CMSC $M_1 \cdots M_n$. Il est bon de remarquer que quelque soit l'ordre dans lequel on compose, on obtient le même ensemble. Bien qu'il y ait un ensemble de CMSC compositions de CMSC, la situation est améliorée par la propriété suivante :

Proposition 16 *Soit une suite de CMSC* $M_1, \ldots, M_n$. *Si il existe un MSC composition de* $M_1 \cdots M_n$, *alors il est unique.*

Preuve. Supposons que M, N sont deux MSC compositions de $M_1 \cdots M_n$. En particulier, la projection de M sur un processus $p \in P$ est entièrement définie par la compositions des projection de $M_1, \ldots M_n$ sur p. Mais cette composition est unique, puisqu'elle ne fait pas intervenir la fonction de message. Ainsi, M et N ont les mêmes projections sur tout processus $p \in P$. Donc ils sont égaux, d'après la remarque 6 page 51.

□

Un *CMSC-graphe* est un graphe dont les nœuds sont étiquetés par un CMSC. Plus formellement,

Définition 34 *Un CMSC-graphe est un quintuplet* $G = (V, \rightarrow, \lambda, V_0, V_f)$, *avec*

- $(V, \rightarrow)$ *est un graphe.*
- λ *est une fonction qui à chaque nœud* $v \in V$ *associe un CMSC.*
- $V_0 \subseteq V$ *est l'ensemble des états initiaux.*
- $V_f \subseteq V$ *est l'ensemble des états finaux.*

Définition 35 *On note le langage d'un CMSC-graphe* G *par* $\mathcal{L}(G)$, *c'est à dire l'ensemble des* **MSC** *compositions de* $\lambda(v_0) \cdots \lambda(v_n)$ *pour* $v_0 \ldots v_n$ *un chemin acceptant de* G.

Exemple 34 *Les CMSC-graphes peuvent exprimer des langages qui ne sont pas (existentiellement) bornés. Ainsi, ils ne sont pas finiment engendrés. Par exemple, le CMSC-graphe décrit sur la figure 2.3 page suivante engendre*

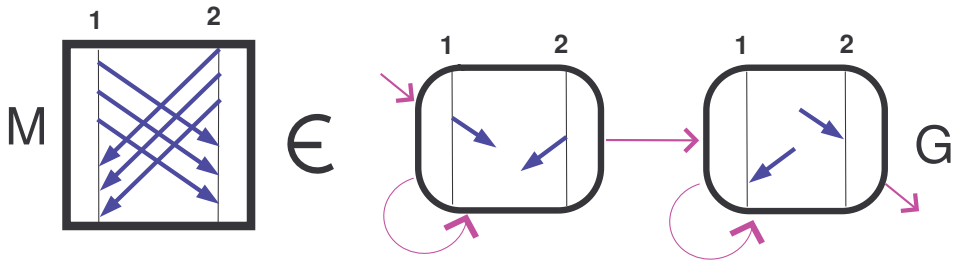


FIG. 2.3 – Un CMSC-graphe G.

les MSC M_n où n messages du processus 1 vers 2 croisent n messages du processus 2 vers 1, pour tout n. Or M_n n'est pas existentiellement $n-1$-borné, donc G n'est pas borné. Les chemins de G qui bouclent un nombre différent de fois autour des deux nœuds ne peuvent pas être composés en MSC, parce qu'il n'y a pas le même nombre d'envois et de réceptions.

Nous avons défini un ensemble de représentants $L(G)$ pour un MSC-graphe G, à savoir l'ensemble des linéarisations qui suivent les nœuds de G. C'est un ensemble de représentants parce que $[L(G)] = Lin(\mathcal{L}(G))$. Par construction, $L(G)$ est un langage régulier, mais il est possible que $[L(G)]$ ne le soit pas. De même, on peut définir $L(G)$ pour un CMSC-graphe G, en tant que l'ensemble des linéarisations qui suivent les nœuds de G. C'est à dire $x_1 \cdots x_n \in L(G)$ si il existe un chemin acceptant $v_0 \cdots v_m$ dans G et une suite d'indices $k_1 \cdots k_m$ avec $\lambda_{k_i} \cdots \lambda_{k_{i+1}-1}$ est une linéarisation de v_i. En revanche, $L(G)$ n'a pas de raison d'être régulier, puisque il est possible que des chemins n'aient pas le même nombre d'envois que de réceptions, donc des chemins de l'automate associé à G ne sont pas forcement étiquetés par des linéarisations.

2.1.2 Un Modèle Général

Nous montrons ici la puissance d'expressivité des CMSC-graphes. Par définition, les CMSC-graphes sont plus expressifs que les MSC-graphes. De plus, nous montrons que nous pouvons transformer explicitement tout CFM en un CMSC-graphe au pire exponentiellement plus grand (dans le nombre de processus).

Proposition 17 *[GMP01] Les CMSC-graphes sont plus expressifs que les CFM et les MSC-graphes.*

Preuve. Intuitivement, on construit l'automate des configurations associé à l'automate communicant, mais sans spécifier les files pour les canaux de communication, qui sont traitées d'une manière implicite par le CMSC-graphe. Il ne reste plus qu'à transformer un automate dont les étiquettes sont sur les transitions à un graphe dont les étiquettes sont sur les nœuds. Ainsi, l'étiquette d'un nœud du CMSC-graphe correspondra exactement à l'étiquette d'une transition du CFM, c'est à dire à une action solitaire. On créé en plus des états initiaux.

Soit $\mathcal{A} = (\mathcal{A}^p, Q_0, Q_f)_{p\in\mathcal{P}}$ un automate communicant avec $\mathcal{A}^p = (Q^p, \rightarrow_p)$. On définit un CMSC-graphe $G = (V, \rightarrow, \lambda, V_0, V_f)$ avec

- $V = (\Sigma \cup \{\epsilon\} \times \prod_{p\in P}(\rightarrow^p) \cup Q_0)$.
- $V_0 = \{\epsilon\} \times Q_0$.
- $V_f = \Sigma \times \prod_{(q_f^p)\in Q_f, p\in P}(q^p \rightarrow^p q_f^p)$
- $\lambda(v) = e$ si $v = (e, T)$ pour $T \subseteq \prod_{p\in P}(\rightarrow^p) \cup Q_0)$.
- $(e, s_1, \ldots, s_\wp) \rightarrow (f, t_1, \ldots, t_\wp)$ pour des transitions $s_1, t_1, \ldots s_\wp, t_\wp$ et des lettres $e, f \in \Sigma$ si il existe k avec $s_p = t_p$ pour $p \neq k$ et $t_k = v \xrightarrow{f}{}^{k} w$, ainsi que $s_k = u \rightarrow v$ ou $s_k = v$.

Supposons que $M \in \mathcal{L}(\mathcal{A})$ soit un MSC dans le langage du CFM. Alors il lui correspond une suite de transitions dans $\mathcal{A}$. Cette suite de transitions correspond exactement à un chemin dans G, donc $M \in \mathcal{L}(G)$ aussi.

Si $M \in \mathcal{L}(G)$, alors il existe un chemin dans G qui correspond à une suite de transitions dans $\mathcal{A}$. Comme M est un MSC, en particulier, tous les canaux sont vides après avoir lu exactement M. Ainsi, le chemin correspondant dans $\mathcal{A}$ est acceptant, donc $M \in \mathcal{L}(\mathcal{A})$.

□

Ainsi, les CMSC-graphes ne sont pas finiment engendrés, ni n'ont une borne sur leur canaux, ni ne sont implémentables (puisque les MSC-graphes ne sont déjà pas implémentables).

Proposition 18 *La recherche de motif et l'égalité sont indécidables pour les CMSC-graphes.*

Preuve. Si la recherche de motif était décidable pour les CMSC-graphes, alors on pourrait aussi la décider pour les CFM puisqu'on a une construction

effective d'un CMSC-Graphe à partir d'un CFM.

□

2.2 Borne Existentielle

D'un point de vue expressivité, les CMSC-graphes semblent parfaits. Cependant, cette expressivité implique également une indécidabilité de toutes les questions non triviales. On peut toutefois tester si un MSC M est généré par un CMSC-graphe G, ie si $M \in \mathcal{L}(G)$. En effet, il suffit de tester si les étiquettes des chemins de longueur au plus $|M|$ se composent en M, et il y en a un nombre fini.

Pour résumer, les structures sans restriction sur les canaux (CMSC-graphes et CFM) sont indécidables. Les structures avec une borne universelle sur les canaux (CFM) ne sont pas assez expressives, tout comme les structures finiment engendrées (MSC-graphes) qui en plus n'ont pas de réelles contrepartie en ce qui concerne les machines. Comme suggéré par le schéma 1.12 page 62, le juste milieu semble être une borne existentielle sur les canaux. On rappelle qu'un MSC est existentiellement b-borné si une de ses linéarisations est b-bornée. Ainsi, un langage de MSC est existentiellement borné si il existe b tel que pour tout MSC M du langage, M est existentiellement b-borné.

Définition 36 *On appellera par* $\exists$-b-CFM *la classe des CFM existentiellement b-bornés, et par* $\exists$-CFM *leur union pour tous les entiers b.*

Proposition 19 *Soit $\mathcal{A}$ un CFM, b un entier.*
Alors savoir si $\mathcal{A} \in \exists$-CFM ou si $\mathcal{A} \in \exists$-b-CFM est indécidable.

Preuve. Il est facile d'adapter les preuves d'indécidabilité pour savoir si un CFM est universellement borné au cas existentiellement borné. De même, on peut étendre le résultat aux CMSC-graphes. On ne sait rien en revanche des CFM sans blocage.

□

Avoir des classes sans test effectif d'appartenance n'est pas très encourageant pour la suite. Pourtant, d'autres résultats plus encourageants militent en faveur de l'utilisation des bornes existentielles.

On rappelle qu'un ensemble de MSC $\mathcal{M}$ admet un *langage de représentants* si il existe un langage de linéarisations K avec $\mathcal{L}(K) = \mathcal{M}$, c'est à dire $Lin(\mathcal{M}) = [K]$.

Proposition 20 *[MM01] Un CFM admet un ensemble régulier de représentants si et seulement si il est existentiellement borné.*

Preuve. Si L est un langage de linéarisations reconnu par un automate $\mathcal{B}$, alors chaque boucle de l'automate est étiquetée par un même nombre d'envois que de réceptions sur le même canal (p, q), pour tout canal (p, q). Ainsi, chaque linéarisation reconnue par A est $|B|$-bornée. Si $M \in \mathcal{L}(\mathcal{A})$, alors il existe une linéarisation dans L, donc M est existentiellement $|B|$-bornée.

Réciproquement, si $\mathcal{A}$ est b-borné, alors l'automate fini des comportements de $\mathcal{A}$ avec des canaux b-bornés génère un langage de représentants de $\mathcal{A}$.

□

Ainsi, une restriction structurelle, qui impliquerait que la classe soit décidable, assurant la borne existentielle sur les canaux serait souhaitable. Une restriction simple sur les CMSC-graphes s'assurant de cette propriété est que les boucles ont le même nombre d'envois que de réceptions sur chaque canal. Ainsi, comme tout chemin assez long a des boucles, le CMSC-Graphe est existentiellement borné.

Définition 37 *Un CMSC-graphe G est appelé* équilibré *si tout CMSC étiquettant une boucle de G a autant d'envois de p à q que de receptions de p à q pour tout canal (p, q).*

En fait, une classe un peu plus restrictive a été définie dans [GMP01], la classe des *CMSC-graphes sûrs* (*safe* CMSC-Graphs [GKM04], encore appelés *realizable* CMSC-graphs [GMP01], ce qui est un très mauvais nom car peut conduire à une confusion avec l'implémentabilité).

Définition 38 *Un CMSC-graphe G est appelé* sûr *si tout chemin acceptant de G est composable en un MSC. Entre autres, cela signifie que tous les chemins acceptants ont un nombre égal d'envois et de réceptions sur chaque canal.*

Ainsi, tout chemin de l'automate associé à un CMSC-graphe sûr G est étiqueté par une linéarisation. En particulier, $L(G)$ est régulier, même si il est possible que $[L(G)]$ ne le soit pas.

Exemple 35 *Le CMSC-graphe de la figure 2.3 page 68 n'est pas sûr, parce que si on boucle n fois autour du premier nœuds puis $m \neq n$ fois autour du second, alors on n'obtient pas un MSC.*

En revanche, tout MSC-graphe est sûr, comme celui de la figure 1.9 page 58.

Proposition 21 *Les CMSC-graphes sûrs et équilibrés sont équivalents. Ils sont inclus dans la classe des CMSC-graphes qui ne génèrent que des MSC existentiellement b-bornés pour un certain b. De plus ils contiennent la classe des CFM ∃-bornés.*

Preuve. Un CMSC-graphe sûr est équilibré, et un CMSC-graphe équilibré G est $\exists$-$|G|$-borné. En effet, si $u \in L(G)$, alors il est composé de boucles, et d'au plus une fois chaque nœud de G. Chaque boucle ayant une différence nulle entre le nombre d'envois et de réceptions, chaque préfixe de u n'a jamais plus de $|G|$ envois solitaires. Ainsi, le langage de représentants $L(G)$ est universellement $|G|$ borné.

Si $G = (V, \rightarrow, \lambda, V_0, V_f)$ est équilibré et $\exists$-b-borné, alors il existe un CMSC-graphe équivalent H qui est sûr, de taille exponentielle en b. L'ensemble des nœuds de H est $V \times \mathcal{S}$, où les éléments $S \in \mathcal{S}$ sont des ensembles d'au plus b envois solitaires. De plus, $\lambda(v, S) = \lambda(v)$ pour $S \in \mathcal{S}$. On a $(v, S) \rightarrow_H (v', S')$ si $v \rightarrow_G v'$ et si S' correspond à S auquel on a retranché les envois reçus dans $\lambda(v)$, et auquel on a rajouté les envois solitaires de $\lambda(v')$. De plus, on s'assure que la propriété FIFO soit conservée en demandant que si v contient un message de p à q, alors $p!q \notin \mathcal{S}$. Un état (v, S) est final si $v \in V_f$ et $S = \emptyset$, ce qui nous assure que H est sûr. Comme G est $\exists$-b-borné, $\mathcal{L}(H) = \mathcal{L}(G)$.

On montrera plus tard que les CMSC-graphes sûrs contiennent les CFM ∃-bornés (voir proposition 26 page 81).

□

Définition 39 *Nous notons la borne universelle de $L(G)$ par b_G.*

De nombreux algorithmes vont dépendre fortement de la borne universelle de $L(G)$, dont l'existence est assurée par la preuve précédente, qui montre que $b_G \leq |G|$. Il est important de constater que dans de nombreux cas, cette borne est bien plus petite que $|G|$.

Exemple 36 *Le CMSC-graphe composé d'un nœud initial étiqueté par un envoi du processus 1 au processus 2, d'un nœud final étiqueté par une réception associée sur le processus 2 depuis le processus 1, avec une transition du nœud initial au nœud final et deux boucles simples sur chaque nœud n'est pas équilibré, donc n'est pas sûr. Il est équivalent au CMSC-graphe sûr (donc*

équilibré) composé d'un nœud initial étiqueté par un envoi du processus 1 au processus 2, d'un nœud final étiqueté par une réception sur le processus 2 depuis le processus 1, et de d'une transition du nœud initial au nœud final, et vice versa. En effet, le second CMSC-graphe n'a qu'une boucle étiquetée par un envoi et une réception sur la même boucle, alors que le premier à une boucle avec un envoi mais pas de réception. En fait, ces deux CMSC-graphes sont équivalents au MSC-graphe avec un seul nœud initial et final composé d'un message du processus 1 au processus 2, avec une boucle simple.

La grande nouveauté de ces classes comparées aux CFM existentiellement bornés, est que non seulement elles sont strictement plus expressives (parce qu'elles contiennent les MSC-graphes), mais en plus elles sont décidables.

Proposition 22 *Savoir si un CMSC-graphe G est sûr ou équilibré est décidable en temps polynomial [GMP01]. En revanche, il est indécidable de savoir si un CMSC-graphe ne génère que des MSC existentiellement b-bornés.*

Preuve. Savoir si un CFM est $\exists$-borné est indécidable, ce qui s'étend facilement aux CMSC-graphes.

Tester si G est équilibré revient à rechercher avec l'algorithme de Bellman-Ford des cycles strictement positifs ou négatifs dans un graphe où le poids est la différence entre le nombre d'envois et de réceptions sur un canal. On fait de même pour tout canal. On peut tester de même si le graphe est sûr. □

La recherche de motifs est décidable pour les CMSC-graphes sûrs (en fait, tous ceux pour lesquels on dispose d'une borne existentielle).

Proposition 23 *Soit M un MSC et G un CMSC-graphe sûr. Savoir s'il existe deux CMSC S, T avec $SMT \in \mathcal{L}(G)$ est décidable.*

Preuve. On donne ici une idée rapide d'une preuve. On note $b = b_G$.

On peut construire (voir section 4.3.2) un automate $\mathcal{A}$ qui reconnaît **toutes** les linéarisations b-bornées de MSC qui contiennent M. Il suffit ensuite de tester si $L(G) \cap L(\mathcal{A})$ est vide ou non. On rappelle que $L(G)$ est régulier, donc cette question est décidable. S'il y a une linéarisation dans $L(G) \cap L(\mathcal{A})$, il ne fait aucun doute qu'il existe deux CMSC S, T avec $SMT \in \mathcal{L}(G)$. Réciproquement, si il existe un MSC $SMT \in \mathcal{L}(G)$, en particulier il a une linéarisation λ dans $L(G)$, donc λ est b-borné. Ainsi, $\lambda \in L(\mathcal{A})$. □

Ainsi, on peut compléter un petit peu notre schéma, en ajoutant un élément nommé CMSG, pour représenter les CMSC-graphes. On ne représente pas les CMSC-graphes équilibrés ou ∃-bornés parce que leur expressivité est la même que celle des CMSC-graphes sûrs, notés safe CMSG. On représente également par ∃-CFM les CFM existentiellement bornés.

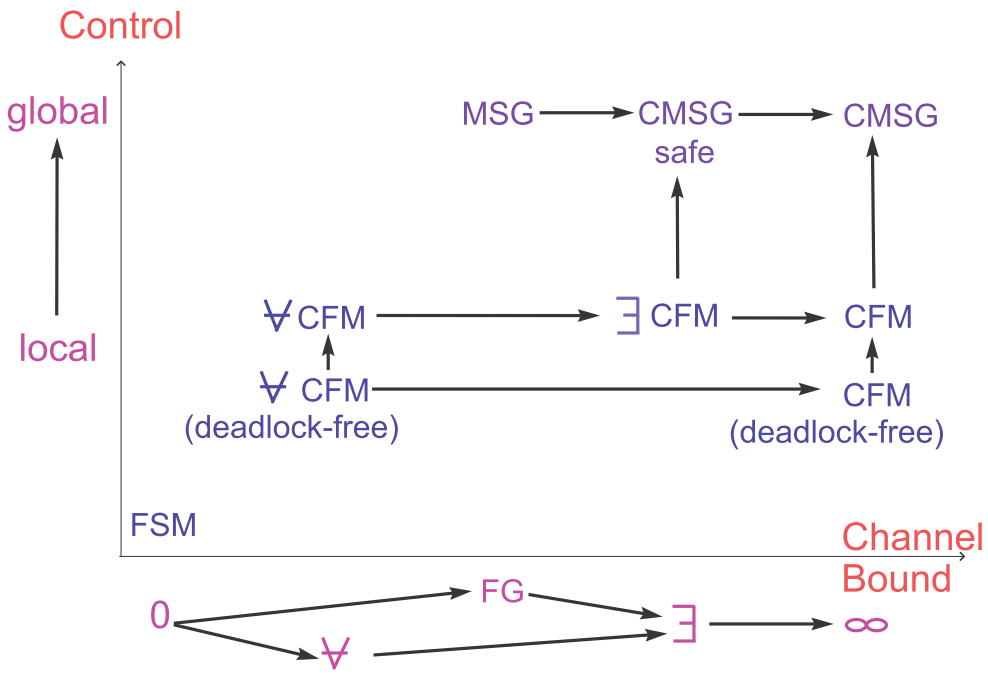


FIG. 2.4 – Diagramme avec les CMSC-graphes existentiellement bornés.

2.3 Coopération des Processus

Comme nous l'avons illustré par la restriction de sûreté, on peut facilement restreindre structurellement les CMSC-graphes pour garantir de bonnes propriétés, contrairement aux CFM où les seules restrictions que l'on connaît, consistant à borner les canaux ou à s'assurer du non blocage, ne sont pas des propriétés décidables. Il reste néanmoins un problème majeur, à savoir quelles restrictions (décidables) peut-on imposer aux CMSC-graphes pour obtenir le même pouvoir expressif que les CFM. Trouver de telles classes et prouver les

résultats seraient l'équivalent du théorème de Kleene pour les mots, puisque les CFM correspondent à un modèle de machines, et les CMSC-graphes aux rationnels.

Nous allons en fait donner un théorème de Kleene-Büchi pour les CFM existentiellement-bornés. La raison pour laquelle on a besoin dans notre preuve d'une borne existentielle est que nous allons nous ramener au cas des traces pour pouvoir utiliser les théorèmes d'Ochmański [Och85] , de Zielonka [Zie87] , et de Thomas [Tho90] ou Ebinger-Muscholl [EM93]. Il semble ainsi assez naturel de définir une restriction sur les processus communicants sur une boucle [GMSZ02], à l'instar de ce qui a été démontré pour les langages rationnels de traces.

Définition 40 *Soit M un CMSC. On dit que M est* connexe *si son* graphe de communication *est (faiblement) connexe. Le graphe de communication de M est $(P(M), \rightarrow)$, où $P(M)$ est l'ensemble des processus ayant une action dans M et $p \rightarrow q$ si M contient un envoi $p!q(b)$ et une réception $q?p(a)$ pour $a, b \in C$.*

Un CMSC-graphe G sera appelé à boucles connexes *(* loop-connected *) si ses boucles (pas forcément simples) sont étiquetées par des CMSC connexes. Si de plus il est sûr, alors il sera appelé* globalement coopératif *(*globally-cooperative*).*

On peut remarquer que le graphe de communication ne dépend que des actions d'un MSC, et pas de leur ordre d'apparition. Ainsi, on définit le graphe de communication d'un sous alphabet de Σ.

Exemple 37 *Le MSC-graphe de la figure 1.9 page 58, qui représente le protocole asynchrone, est à boucles connexes et sûr, donc est globalement coopératif. En effet, son graphe de communication est $1 \rightarrow 2$, donc est (faiblement) connexe.*

En revanche, le CMSC-graphe de la figure 2.3 page 68 n'est pas à boucles connexes (ni sûr) parce que les deux boucles ont pour graphe de communication deux nœuds 1 et 2 déconnectés. Donc il n'est pas globalement coopératif.

On peut tester si un CMSC-graphe est globalement coopératif.

Proposition 24 *[MP99] Savoir si un CMSC-graphe G est globalement coopératif est co-NP-complet.*

Preuve.

On donne ici un algorithme pour tester si G est globalement coopératif. On peut tester si un CMSC-graphe est sûr en temps polynomial (voir la proposition 22 page 73). On devine un sous graphe H de G. On vérifie en temps polynomial qu'il existe une boucle qui passe au moins une fois par chaque sommet de H. De plus, on vérifie que le *graphe de communication* est déconnecté. On a ainsi un algorithme en co-NP.

On montre maintenant que le problème est co-NP-dur en réduisant le problème de 3CNF-SAT à notre problème. Soit un formule $\varphi = C_1 \wedge \cdots \wedge C_m$ avec n variables $x_1, \ldots, x_n$ et $C_i = l_i^1 \vee l_i^2 \vee l_i^3$. La formule φ est satisfiable si et seulement si il existe une valuation telle que pour tout $i \leq m$, les trois littéraux l_i^1, l_i^2, l_i^3 ne sont pas faux dans la valuation.

On va utiliser $2(n+1)$ MSC $V_0, F_0, \ldots, V_n, F_n$, un par variable (positive ou négative), plus $V_0 = F_0$. Le MSC-graphe G a un nœud NX_i par MSC $X_i \in \{V_i, F_i\}$, étiqueté par ce MSC, et des transitions depuis chaque NX_i à chaque NX_{i+1}, où $NX_{n+1} = NX_0$. Ainsi, une boucle simple dans le graphe G correspond à une valuation pour laquelle x_i vrai se traduit par un passage de la boucle par NV_i, et x_i faux par un passage de la boucle par NF_i. On pose NV_i la concaténation des MSCs LV_j^k tels que $l_j^k = x_i$, et des MSCs LF_j^k tels que $l_j^k = \overline{x_i}$. On pose NF_i la concaténation des MSCs LV_j^k tels que $l_j^k = \overline{x_i}$, et des MSCs LF_j^k tels que $l_j^k = x_i$.

Il y a $2m+2$ processus $(P_i^1, P_i^2)_{i \leq m}$, avec P_i^1, P_i^2 correspondant à la clause C_i, plus les processus P_0, P_{m+1} qui correspondent à $V_0 = F_0$.

Le MSC $V_0 = F_0$ est constitué de deux actions locales, une sur P_0 et l'autre sur P_{m+1}. On va définir les LV_j^k, LF_j^k tel que P_0 et P_{m+1} sont connectés si et seulement si il existe une clause dont les trois littéraux sont faux.

Le MSC LF_j^1 est composé d'un message de P_0 à P_j^1. Le MSC LF_j^2 est composé d'un message de P_j^1 à P_j^2. Le MSC LF_j^3 est composé d'un message de P_j^2 à P_{m+1}. Les MSC LV_j^1 et LV_j^2 sont vides, et le MSC LV_j^3 est composé d'un message de P_0 à P_j^2.

Ainsi, P_0 et P_{m+1} sont connectés dans une boucle simple ssi il existe une clause dont les trois littéraux sont faux. De plus, on note que si P_0 et P_{m+1} sont connectés dans une boucle simple, alors l'alphabet de cette boucle est connecté : si LF_i^1, LF_i^3 ou LV_i^3 apparaissent dans la boucle, leurs processus sont connectés à P_0 ou P_{m+1}. Quant à LF_i^2, ses processus sont connectés dans chaque boucle à P_0 grâce à LV_i^3, si l_i^3 est vrai, ou bien à P_{m+1} grâce à LF_i^3, si l_i^3 est faux.

Ainsi, on a que φ est satisfiable si et seulement si il existe une valuation

telle que pour tout $i \leq m$, les trois littéraux l_i^1, l_i^2, l_i^3 ne sont pas faux dans la valuation si et seulement si il existe une boucle simple non connectée.

□

2.3.1 L'Alphabet de Kuske

Pour cette section, on fixe un entier b. On va montrer maintenant que les CMSC-graphes globalement coopératifs sont au moins aussi expressifs que les CFM, c'est à dire que cette restriction n'est pas trop contraignante.

Pour cela, on va se servir du théorème d'Ochmański, donc on va se ramener au cas des traces. Pour se ramener aux traces, on va d'abord se servir d'un alphabet défini dans [Kus03] pour une utilisation avec les MSC universellement bornés. Cependant, on va montrer comment l'utiliser avec un modèle $\exists$-b-borné. Cet alphabet ne dépend pas des contenus des messages, nous les ignorerons donc dans la suite. Cette section reprend le travail préliminaire de [Gen04].

On définit un nouvel alphabet $\Omega = \mathcal{T} \times \{0, \ldots, b-1\}$, avec la relation de dépendance $(e,i)D(f,j)$ si $P(e) = P(f)$ ou si $i = j$ et $\{e,f\} = \{p!q, q?p\}$, pour tout p, q, i, j. Ainsi, $I = \Omega^2 \setminus D$ est symétrique et irréflexive, et donc (Ω, I) est un alphabet de trace.

On définit ici une injection canonique qui à un mot u sur l'alphabet $\mathcal{T}$ associe un mot $\tilde{u}$ sur l'alphabet Ω, en numérotant les événements du même type modulo b. On a $\widetilde{a_1 \cdots a_n} = (a_1, x_1) \cdots (a_n, x_n)$, avec $x_i = |\{j \leq i \mid a_j = a_i\}| \pmod b$, c'est à dire que modulo b, il y a x_i occurrences de la lettre a_i dans le préfixe $a_1 a_2 \ldots a_i$. La fonction inverse $\pi(u)$ consiste à projeter sur la première composante, c'est à dire $\pi((a_1, x_1) \cdots (a_n, x_n)) = a_1 \cdots a_n$. On dira que $u \in \Omega^\star$ est b-borné si $\pi(u)$ est b-borné. On note $\tilde{L} = \{\tilde{u} \mid u \in L\}$.

La relation d'indépendance $I = \Omega^2 \setminus D$ est plus restrictive que la relation de commutation, c'est à dire que pour tout langage X de linéarisations, $\pi([\widetilde{X}]_I) \subseteq [X]$. Intuitivement, D impose que chaque réception $e = q?p$ arrive toujours avant l'envoi $p!q$ associé avec la b-ième réception $q?p$ après e.

Lemme 2 *Soit une linéarisation $u = a_1 \cdots a_n \in \mathcal{T}^\star$ et $\tilde{u} = ((a_1, x_1), \ldots, (a_n, x_n))$. Supposons que (a_i, a_k) soit un message. Alors, on a $(a_i, x_i) = (p!q, n), (a_k, x_k) = (q?p, n)$ pour un certain n. De plus, $a_1 \cdots a_n$ est b-borné si et seulement si il n'y a pas de positions $i < j < k$ telles que (a_i, a_k) soit un message et $(a_i, x_i) = (a_j, x_j)$.*

Preuve. La première propriété découle de la condition FIFO sur les canaux de communication. En ce qui concerne la seconde propriété, supposons d'abord qu'il existe $i < j < k$ tels que décrit dans le lemme. Quand a_j est envoyé, a_i est toujours dans le canal puisque sa réception associée a_k n'est pas encore intervenue. Ainsi, il y a $b+1$ messages dans le canal et u n'est pas b-bornée.

Réciproquement, supposons que $a_1 \cdots a_n$ ne soit pas b-borné, et considérons le plus petit préfixe $a_1 \cdots a_k$ qui n'est pas b-borné. Alors il existe p, q, n avec $(a_k, x_k) = (p!q, n)$ et le dernier envoi $(p!q, n)$ avant (a_k, x_k) n'est pas encore reçu. Ainsi, sa réception associée est a_j avec $j > k$.

□

On peut remarquer que si u est b-borné et (a_i, a_k) est un message, alors il n'existe pas non plus de $i < j < k$ avec $(a_j, x_j) = (a_k, x_k)$. Sinon, prenons a_l l'envoi associé avec a_j. Alors $l < i < j$ entre en contradiction avec le lemme ci dessus.

Lemme 3 *Si $u, v \in \mathcal{T}^\star$, u est une linéarisation b-bornée, et $\tilde{u} \sim_I \tilde{v}$, alors v est aussi une linéarisation b-bornée.*

Preuve. Supposons par l'absurde que $\tilde{v}$ ne soit pas b-bornée. D'après le lemme 2 page précédente il existe des positions $i < j < k$ telles que (a_i, a_k) est un message, et $(a_i, x_i) = (a_j, x_j)$. C'est à dire, $(a_j, x_j) D (a_k, x_k)$. Ainsi, dans $\tilde{u} \in [\tilde{v}]_I$, la position correspondant à (a_k, x_k) arrive après la position de (a_j, x_j), qui est une contradiction puisque u est b-borné.

□

Proposition 25 *Si u est une linéarisation b-bornée d'un MSC M, alors $\pi([\widetilde{u}]_I)$ est l'ensemble des linéarisations b-bornées $Lin^b(M)$ de M.*

Preuve. D'après le lemme 3, chaque mot de $[\widetilde{u}]_I$ est une linéarisation b-bornée.

Pour la réciproque, soit u, v deux linéarisations b-bornées du même MSC M. On veut montrer que $\tilde{u} \sim_I \tilde{v}$. Si on fixe un ordre $\prec$ sur Ω, on peut définir la forme normale lexicographique $s = (s_1, x_1) \cdots (s_n, x_n) = LNF(\tilde{u})$, c'est à dire le plus petit mot pour l'ordre $\prec$ tel que $(s_1, x_1) \cdots (s_n, x_n) \sim_I \tilde{u}$. De même, on note $t = (t_1, y_1) \cdots (t_n, y_n) = LNF(\tilde{v})$. Ces deux mots s, t sont b-bornés car I-équivalents à des mots b-bornés (cf ci-dessus). On va montrer que $s = t$. Ainsi, on montrera que $\tilde{u} \sim_I s = t \sim_I \tilde{v}$.

On sait déjà que $\pi(s)$ et $\pi(t)$ sont des linéarisations de M parce que la relation $\sim_I$ est plus restrictive que la relation de commutation d'un MSC. Supposons par l'absurde que $s \neq t$. Soit k minimal tel que $s_k \neq t_k$. Par symétrie, on peut supposer que $(s_k, x_k) \prec (t_k, y_k)$ pour l'ordre que l'on a choisi sur Ω. Soit $l > k$ l'indice minimal tel que $t_l = s_k$. Comme $\pi(s)$ et $\pi(t)$ sont des linéarisations de M, on a $y_l = x_k$ et $P(s_k) \notin P(t_k \cdots t_{l-1})$. Comme t est en LNF, forcément on doit avoir un $k \leq m < l$ avec $(t_m, y_m)D(s_k, x_k)$. Comme $P(t_m) \neq P(s_k)$, c'est que $x_k = y_m$, $t_m = p!q$ et $s_k = q?p$ ou l'inverse.

Dans le cas $s_k = q?p$, on regarde dans s l'envoi s_i associé à s_k, $i < k$, donc $t_i = s_i$. Ainsi t_i est l'envoi associé à t_l, et t_m est une autre réception sur le même canal avec $i < m < l$ et $y_i = y_m = y_l$.

Dans le cas où $t_m = q?p$, on regarde dans t l'envoi t_i associé à t_m. Comme $(t_i, y_i) = (s_k, x_k)D(s_k, x_k)$, on a $i < k$, donc $t_i = s_i$. On pose s_j l'événement qui correspond dans s à t_m. Ainsi $j > k > i$ et $x_i = x_k = y_m = x_j$, et s_j est la réception associé à s_i.

D'après le lemme 2 page 77, cela signifie que t n'est pas b-bornée, contradiction. Donc $\tilde{u} \sim_I \tilde{v}$, et on a bien toutes les linéarisations b-bornées dans $\pi([\widetilde{X}]_I)$.

□

Corollaire 1 *Si $\mathcal{L}(X)$ est un langage $\forall$-b-borné, alors $\pi([\widetilde{X}]_I) = [X] = Lin(\mathcal{L}(X))$ est l'ensemble de toutes les linéarisations de $\mathcal{L}(X)$.*

Corollaire 2 *Si M est un MSC $\exists$-b-borné, alors on peut lui associer la trace $tr(M)$ sur (Ω, I) composée de ses linéarisations b-bornées.*

Le lemme 2 page 77 se reformule en une réception r associée à un envoi $e = (q?p, n)$ est le premier événement $(q?p, n)$ après e pour $\leq_I$. Pour un ensemble $\mathcal{M}$ de MSC $\exists$-b-bornés, on a $\bigcup_{M \in \mathcal{M}} \pi(\mathrm{tr}(M)) = Lin^b(\mathcal{M})$. On note le pomset associé à la trace $\mathrm{tr}(M)$ par $(E, <_I, \tilde{\lambda})$, où E est l'ensemble des évènements de M. En fait, la relation $<_I$ est la clôture transitive de $m \cup (<_p)_{p \in \mathcal{P}} \cup rev$, où

$$rev(r) = s' \quad \text{ssi} \quad m(s) = r,\ \lambda(s) = \lambda(s'),\ \text{et}$$

$$|\{x \in E \mid s <_p x \leq_p s', \lambda(s) = \lambda(x)\}| = b$$

C'est à dire que rev associe une réception r où $r = m(s)$ avec l'envoi s' qui est le premier événement avec $\tilde{\lambda}(s') = \tilde{\lambda}(s)$ et $s < s'$ (si un tel événement existe).

Lemme 4 *Un MSC M est $\exists$-b-borné si et seulement si la relation $\prec_B = m \cup (<_p)_{p \in cP} \cup \mathrm{rev}$ est acyclique.*

Exemple 38 *Prenons par exemple le MSC de la figure 1.7 page 48. Il est existentiellement 1-borné. Il est donc aussi existentiellement 2-borné. Son alphabet est $\{a, b\}$, où a désigne un envoi de 1 vers 2 et b la réception associée. L'alphabet de Kuske pour $b = 2$ est composé de deux copies de $\{a, b\}$, une noire et une rouge. Ainsi, les premiers et troisième a et b sont noirs, alors que les seconds a et b sont rouges. La relation message et les relations d'ordre sur les processus 1 et 2 sont préservées. De plus, il apparaît une causalité entre la première réception b et le troisième envoi a, qui est la causalité* rev. *Le pomset associé avec cette trace est décrit dans la figure 2.5.*

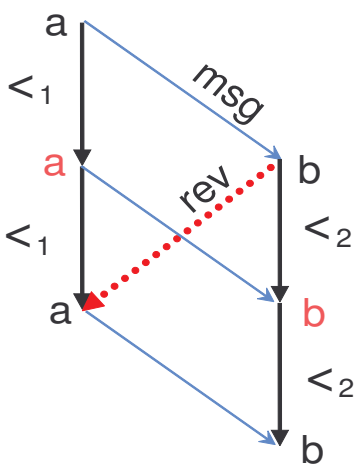


FIG. 2.5 – Exemple de pomset associé à l'alphabet de Kuske.

Prenant un CFM $\exists$-b-borné $\mathcal{A}$, on calcule l'automate $\mathcal{B}$ des comportements avec des canaux b-bornés. On se souvient dans les états de l'automate du nombre modulo b de messages envoyés et reçus par canaux. Cet automate est de taille $O(b|\mathcal{A}|)^{\wp}$. On peut enlever les états qui ne correspondent pas à une linéarisation, c'est à dire que tous les chemins de cet automate seront des MSC (il était possible que ça ne soit pas le cas si le CFM a des blocages). Cet automate est clos par I-commutation d'après la proposition 25 page 78. On applique la construction de la proposition 9 page 38, qui donne un automate

à boucles I-connexes $\mathcal{C}$, exponentiel en la taille de l'alphabet. On interprète cet automate en tant qu'un CMSC-graphe à boucles I-connexes G équivalent au CFM de départ. Ainsi, G est à boucles I-connexes et $\exists$-borné. Comme l'automate ne génère que des linéarisations, G est sûr. Il suffit d'observer que I est plus restrictif que la relation de commutation. Ainsi, en projetant G sur l'alphabet $\mathcal{T}$, on obtient un CMSC-graphe à boucles connexes et sûr, c'est à dire globalement coopératif, et équivalent au CFM de départ. On peut formuler cette construction en le théorème suivant :

Proposition 26 *Les CFM existentiellement bornés sont inclus dans les CMSC-graphes globalement coopératifs.*

Si de plus on a une borne existentielle b sur le CFM $\mathcal{A}$, alors on peut construire un CMSC-graphe globalement coopératif de taille $O((b\wp^2)!(b\mathcal{A})^\wp)$, c'est à dire exponentielle en b et en $\wp$.

Ainsi, les linéarisations b-bornées sont un outil puissant quand on considère des structures existentiellement b-bornées. On notera $Lin^b(G)$ l'ensemble des linéarisations b-bornées d'un CMSC-graphe G. S'il est existentiellement b-borné, alors $Lin^b(G)$ est un ensemble de représentants de G, c'est à dire $[Lin^b(G)] = Lin(\mathcal{L}(G))$ par définition d'une borne existentielle.

Prenant deux CMSC-graphes G, H, alors $\mathcal{L}(G) = \mathcal{L}(H)$ si et seulement si $Lin(\mathcal{L}(G)) = Lin(\mathcal{L}(H))$. Si de plus G, H sont existentiellement b-bornés, alors $Lin(\mathcal{L}(G)) = Lin(\mathcal{L}(H))$ si et seulement si $Lin^b(G) = Lin^b(H)$.

Pour résoudre l'égalité de deux CMSC-graphes équilibrés, on peut donc se servir de $Lin^{b_G}(G)$. Rappelons que b_G est la borne universelle de $L(G)$, voir la définition 39 page 72.

Proposition 27 *Si G est un CMSC-graphe globalement coopératif, alors pour tout $b \geq b_G$, $Lin^b(G)$ est effectivement régulier, reconnu par un automate de taille $|G|^{O(b^2\wp^4|G|)}$, .*

Preuve. Une fois encore, on va se servir de l'alphabet de Kuske (voir section précédente) et du théorème d'Ochmański. On rappelle que $\Omega = \mathcal{T} \times \{0, \ldots, b-1\}$, avec la relation de dépendance $(e,i)D(f,j)$ si $P(e) = P(f)$ ou si $i = j$ et $e = p!q(a)$, $f = q?p(b)$, pour tout p, q, a, b, i, j.

On rappelle aussi qu'on peut interpréter G comme un automate $\mathcal{A}$ sur l'alphabet $\mathcal{T}$. Alors $\mathcal{A}$ reconnaît $L(\mathcal{A}) = L(G)$, qui est un ensemble de représentants de $Lin(\mathcal{L}(G)) = [L(G)]$. On peut le grossir pour qu'il travaille sur l'alphabet Ω, obtenant un automate $\mathcal{B}$ de taille $|G|b^{2\wp^2}$, qui reconnaît

$\widetilde{L(G)}$. On peut ainsi invoquer le théorème d'Ochmański, et plus précisément la proposition 7 page 35.

Soit une suite de mots non vides $x_1y_1 \cdots x_ky_k$ étiquetant un chemin de $\mathcal{B}$ avec $\forall i < j, x_i I y_j$. Il faut donc montrer l'hypothèse de la proposition 7 page 35 qui est $k < N$, pour un certain N indépendant du chemin. On va montrer que $N = \wp|G|(1 + \wp b)$ convient. Tout chemin x_i, y_j de $\mathcal{B}$ correspond à un chemin $\pi(x_i), \pi(y_j)$ de G. Supposons par l'absurde que l'on a un ensemble J de $\wp^2 b + \wp$ indices i tels que tous les $\pi(x_i)$ commencent par le même état s de G. Ainsi, les chemins $\pi(x_j) \cdots \pi(y_{j'-1})$ sont des boucles autour de s, pour j, j' des indices consécutifs dans J. Il y a au plus $\wp$ telles boucles où $\pi(x_j) \cdots \pi(x_{j'-1})$ et $\pi(y_j) \cdots \pi(y_{j'-1})$ ont un processus en commun, puisque $x_i I y_j$ pour $i < j$ (le même processus ne peut être partagé par deux boucles). Comme chaque boucle de G est connexe, soit un processus est partagé, soit il y a un $p!q$ dans un des x_i et un $p?q$ dans un des y_j (ou l'inverse). Comme il y a au moins $\wp^2 b$ boucles de ce second type, il y a au moins b envois sur le même canal $\{p, q\}$, c'est à dire tous les types possible pour ce canal. Donc il y a un envoi s dans un x_i et une réception r dans un y_j, $i < j$ avec sDr, contradiction.

Le propriété 7 page 35 donne un automate simplement exponentiel, de taille $O((2^{2b\wp^2} b^{2\wp^2} |G|)^{\wp|G|(1+\wp b)})$. Cet automate reconnaît $[\widetilde{L(G)}]_I$. Projetant cet automate sur sa première composante, on obtient un automate de même taille qui reconnaît $\pi([\widetilde{L(G)}]_I) = Lin^b(\mathcal{L}(G))$ d'après la proposition 25 page 78. □

Ainsi, on peut décider de l'égalité de deux CMSC-graphes globalement coopératifs. La complexité est EXPSPACE-complète, voir plus loin le théorème 10 page 125.

2.3.2 CMSC-Graphes Réguliers

Nous avons ainsi déjà une inclusion pour un théorème de Kleene assez général concernant les automates communicants. Quid de la réciproque ? Reprenons tout d'abord les langages de MSC réguliers.

Les CMSC-graphes globalement coopératifs sont plus généraux que les langages de MSC réguliers. Par exemple, le protocole asynchrone (voir figure 1.9 page 58 a un ensemble de linéarisations qui n'est pas un langage régulier (c'est un langage de Dyck). En fait, le problème provient du fait que les CMSC-graphes globalement coopératifs sont en général existentiellement

bornés, alors que les langages réguliers sont universellement bornés.

Avec la proposition 26 page 81, on sait déjà que les CFM universellement bornés sont inclus dans les CMSC-graphes globalement coopératifs et universellement bornés. Ainsi, on peut étendre [HMNK$^+$04] à

Proposition 28 *Soit un langage $\mathcal{M}$ de MSC universellement bornés. Alors les propositions suivantes sont équivalentes :*

- *$\mathcal{M}$ est régulier.*
- *$\mathcal{M}$ est le langage d'un CFM.*
- *$\mathcal{M}$ est le langage d'une formule MSO.*
- *$\mathcal{M}$ est le langage d'un CMSC-graphe globalement coopératif.*

Preuve. On a applique le théorème 4 page 54 de [HMNK$^+$04] pour avoir l'équivalence des trois premières classes. La proposition 27 page 81 montre que le langage d'un CMSC-graphe globalement coopératif et universellement borné est régulier, puisque $Lin^b(G)$ représente tout $Lin(\mathcal{L}(G))$. La proposition 26 page 81 montre que les CFM universellement bornés sont transformables en un CMSC-graphe globalement coopératif.

□

Ainsi, on appellera réguliers les CMSC-graphes globalement coopératifs et universellement-bornés.

Historiquement, une restriction syntaxique a été définie par [MP99], [AY99] pour obtenir des MSC-graphes réguliers. Elle correspond à la proposition suivante :

Proposition 29 *G est un CMSC-graphe globalement coopératif et universellement borné si et seulement si le graphe de communication de chaque boucle de G est* **fortement** *connexe.*

Exemple 39 *Par exemple, le CMSC-graphe asynchrone (figure 1.9 page 58) n'est pas régulier parce que son graphe de communication est $1 \rightarrow 2$ non fortement connexe. Pour qu'il soit fortement connexe, il faudrait qu'il y ait une communication de $2 \rightarrow 1$. Ainsi, la figure 39 page suivante présente un CMSC-graphe régulier.*

Pour la régularité, il faut que chaque émission soit acquittée après un nombre borné de messages. Ainsi, tout protocole sans acquittement (pour raison de performance ou parce que certaines liaisons sont unidirectionnelles, comme par exemple avec un satellite) ne pourra pas être régulier. Cela est très restrictif.

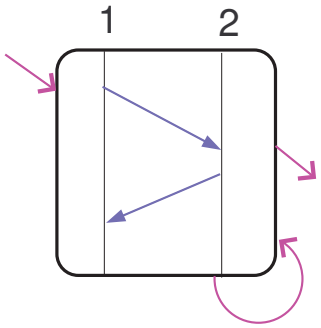


FIG. 2.6 – Un MSC-graphe régulier.

2.4 Extension du théorème de Kleene-Büchi

On va maintenant essayer d'étendre ce théorème au cas où il existe non plus une borne universelle mais une borne existentielle.

Supposons qu'on dispose d'une borne b pour toute la suite de la section. On va montrer le théorème suivant :

Théorème 5 *Soit un langage $\mathcal{M}$ de MSC. Alors les propositions suivantes sont équivalentes :*

1. *$\exists b$ tel que $Lin^b(\mathcal{M})$ est un ensemble régulier de représentants.*
2. *$\mathcal{M}$ est le langage d'un CFM et $< cM$ est $\exists$-borné.*
3. *$\mathcal{M}$ est le langage d'un CMSC-graphe globalement coopératif.*
4. *$\mathcal{M}$ est le langage d'une formule MSO et $< cM$ est $\exists$-borné.*

On va montrer le théorème 5 par une suite de lemmes et de propositions. On a déjà que 2 implique 3 en appliquant la proposition 26 page 81. On peut rappeler aussi que l'on sait que 3 implique 1 en appliquant la proposition 27 page 81. L'implication 2 vers 4 est standard et facile (voir par exemple [BL04]). On va montrer par la suite que 4 implique 1, puis enfin que 1 implique 2. Cette section s'appuie sur le travail accompli dans [GKM04]

2.4.1 De MSO à un ensemble régulier de représentants

On pose (Ω, I) pour l'alphabet de traces de Kuske.

Nous montrons maintenant comment construire un automate reconnaissant $Lin^b(\mathcal{L}(\varphi))$ à partir d'une formule MSO φ. Cette propriété montre que 4 implique 1. Ce type de construction a déjà été utilisé par [MM01], pour faire du model-checking d'un CMSC-graphe sûr contre une formule MSO, mais avec une preuve différente. Ici, nous nous servons de la théorie des traces de Mazurkiewicz, et plus particulièrement d'un résultat permettant de traduire une formule MSO sur les traces en une formule MSO sur les mots [EM93, Tho90].

Proposition 30 *Soit φ une formule MSO sur les MSC. Alors $Lin^b(\mathcal{L}(\varphi))$ est régulier.*

Preuve. On rappelle qu'à un MSC M existentiellement-b-borné, on associe la trace $tr(M) = [u]_I$ sur l'alphabet (Ω, I) où u est une linéarisation b-bornée de M. La trace $tr(M)$ ne dépend pas de la linéarisation b-bornée choisie. On note $<_I$ l'ordre sous jacent de $tr(M)$.

Comme l'ordre visuel $\leq$ d'un MSC est la clôture transitive de $m \cup \bigcup_{p\in\mathcal{P}} <_p$, on peut supposer sans restriction que φ utilise uniquement la relation de message msg et l'ordre sur les processus $<_p$, $p \in \mathcal{P}$.

A la formule MSO φ sur les MSC, on associe une formule $\widetilde{\varphi}$ sur les traces de (Ω, I) comme suit. Chaque symbole $v_a(x)$ est remplacé par $\bigvee_{n\in\{0,\ldots,b-1\}} v_{(a,n)}(x)$.

Chaque symbole $msg(x, y)$ est remplacé par $x \leq_I y$
$\wedge \bigvee_{a=p!q,b=q?p,0\leq n<B} v_{(a,n)}(x) \wedge v_{(b,n)}(y) \wedge \forall z : (v_{(b,n)}(z) \wedge x \leq_I z) \rightarrow y \leq_I z.$
Cette formule exprime que pour x étiqueté par $(p!q, n)$, le nœud y est le plus petit étiqueté par $(q?p, n)$ avec $x \leq_I y$.

Enfin, nous définissons la formule $\widehat{\varphi}$ en tant que la conjonction de $\widetilde{\varphi}$ avec une formule exprimant que la trace sur (Ω, I) est associée avec un MSC $\exists$-b-borné. Il suffit de vérifier que pour tout nœud étiqueté (a, n), le nœud suivant étiqueté par (a, m) satisfait $m = n + 1 \pmod b$, et que la relation msg est une bijection.

Ainsi, $\mathcal{L}(\widehat{\varphi})$ est un ensemble régulier de traces. D'après [EM93], nous avons que $L = \bigcup_{\mathrm{tr}(M)\models\widehat{\varphi}} Lin(\mathrm{tr}(M)) = \widetilde{Lin^b(\mathcal{L}(\varphi))}$ est un ensemble régulier de mots. Sa projection $\pi(L) = Lin^b(\mathcal{L}(\varphi))$ également.

□

2.4.2 Théorème de Zielonka

On arrive à la partie où l'on veut distribuer les choix pour avoir un CFM. La preuve utilise d'une manière cruciale les *applications asynchrones*, et le théorème de Zielonka [Zie87].

On rappelle rapidement ces deux notions (voir également [CMZ89, DR95, DM95]). On note pour une trace $u \in \mathbb{M}(\Sigma, I)$ et un ensemble de lettres $A \in \Sigma$ par $u \downarrow_A$ la plus grande trace préfixe de u qui a ses maximaux étiquetés par des lettres inclues dans A. Par exemple, $u \downarrow_a$ est le plus grand préfixe de u qui a un unique maximum étiqueté par a. Soit un ensemble fini K et un alphabet d'indépendance (Σ, I). Une application asynchrone μ de $\mathbb{M}(\Sigma, I)$ dans K est une application qui vérifie les propriétés suivantes :

- S'il existe une unique maximum z pour deux traces uz, vz avec $\mu(u) = \mu(v)$, alors $\mu(uz) = \mu(vz)$.
- Soit u, v des traces et $A, B \subseteq \Sigma$. Si $\mu(u \downarrow_A) = \mu(v \downarrow_A)$ et $\mu(u \downarrow_B) = \mu(v \downarrow_B)$, alors $\mu(u \downarrow_{A \cup B}) = \mu(v \downarrow_{A \cup B})$

Cela signifie que l'image par une application asynchrone d'une trace u est définie de manière unique par l'image de ses éléments maximaux. C'est à dire les images des traces $(u \downarrow_a)_{a \in A}$ définissent l'image de u, où A est l'ensemble des lettres maximales de u,

Ainsi, pour une application asynchrone μ sur (Ω, I), on notera par $\nu : (\Omega \times K \times K) \to K$ une fonction qui prend en entrée $(e, \mu(u \downarrow_f), \mu(u \downarrow_g))$ et renvoie $\mu(u)$, pour e l'étiquette de l'unique maximum de u, et $\{f, g\}$ contenant l'ensemble des étiquettes des prédécesseurs directes de e. Si $e = (p!q, n)$, f est l'événement qui précède e sur p, et $g = (p?q, n)$). On a alors $\nu(e, \mu(u \downarrow_f), \mu(u \downarrow_g)) = \mu(u)$. Si $e = (q?p, n)$, f est l'événement qui précède e sur q, et $g = (p!q, n)$). On a alors $\nu(e, \mu(u \downarrow_f), \mu(u \downarrow_g)) = \mu(u)$.

On dit qu'un langage de traces L est reconnu par une application asynchrone μ s'il existe un sous-ensemble $\mathrm{Acc} \subseteq K$ avec $L = \mu^{-1}(\mathrm{Acc})$. En fait, les applications asynchrones étendent les morphismes. Comme il suffit d'avoir les valeurs pour les éléments maximaux d'une trace pour avoir les valeurs pour la trace, une mémoire finie (nombre de lettres de l'alphabet Σ) suffit pour calculer la valeur, donc savoir si une trace est reconnue ou non. Les traces reconnaissables par application asynchrone sont des langages réguliers de traces. Le théorème de Zielonka nous donne la réciproque.

Théorème 6 *[Zie87] Un langage de traces est reconnu par application asyn-*

chrone si et seulement s'il est régulier [1].

2.4.3 Un CFM proche d'une application asynchrone

Nous sommes maintenant armés pour prouver que 1 implique 2.

On montre tout d'abord qu'une application asynchrone sur (Ω, I) peut être simulée par un CFM, tel que la simulation est correcte pour les MSC $\exists$-b-bornés. On rappelle que (Ω, I) est l'alphabet de traces de Kuske.

Chaque processeur p se souvient dans sa mémoire *mem* de la valeur associée au dernier événement de chaque Ω_p-type (c'est à dire aux b derniers événements de chaque $\mathcal{T}_p$-type). Une application asynchrone μ étiquette les événements de la trace par une valeur de K. Quand p envoie un message, il joint en donnée de contrôle la valeur par laquelle μ étiquette l'envoi, afin que la réception obtienne cette valeur. Le seul problème est que un envoi s doit aussi connaître la valeur de la réception r avec $s = rev(r)$. A la place, s devine la valeur et la joint en donnée de contrôle. Cette valeur sera alors vérifiée par le processus auquel le message est destiné, qui connaît la valeur correcte grâce à sa mémoire locale *mem*. Pour formaliser ceci, on introduit la notion de *bon étiquetage*.

Exemple 40 *Reprenons l'exemple de la figure 2.5 page 80. Les trois événements a sont étiquetés par une application asynchrone avec k_1, k_3, k_5, et les trois événements b par k_2, k_4, k_6 (voir figure 40 page suivante). L'étiquette k_1 est donnée par l'état initial, donc le CFM la connaît. Il envoie k_1 avec le message, tel que le premier b reçoit la valeur k_1, et peut ainsi calculer sa valeur k_2. On fait de même pour k_3 et k_4, qui se servent des valeurs k_1 et k_2 déjà calculées. Enfin, quand k_5 doit être calculé, le troisième a doit connaître la valeur k_2. Comme il ne peux pas le faire, il va la deviner, calculer son hypothétique k_5, et envoyer k_5 avec la valeur guess devinée au troisième b. Celui ci étant sur le même processus que le premier b, il peut vérifier si la valeur devinée est la bonne, auquel cas il ne bloque pas et peut utiliser k_5 pour calculer k_6, sans danger.*

Soit K un ensemble fini, M un MSC avec ensemble d'événements E, et $update : \Omega \times K \times K \to 2^K$ une application. Pour un événement $t \in E$, on note par t^- le prédécesseur de t sur le même processus. Si t est le premier

[1] En fait, le théorème de Zielonka dit surtout que les langages de traces sont exactement les langages d'automates asynchrones déterministes, que nous n'avons pas définis ici.

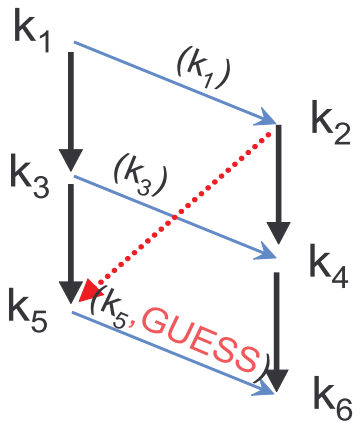


FIG. 2.7 – Simulation d'une application asynchrone par un CFM.

événement sur le processus p, alors on note $t^- = \epsilon_p$. On étend l'ordre $<_p$ à $\epsilon_p <_p e$ pour tout $e \in E_p$. Rappelons que *rev* associe une réception r avec $r = m(s)$ à l'envoi s', qui est le premier événement du même Ω-type que s après s (si il existe). Si il existe s avec $\lambda(s) = p!q$ et qu'il n'y a pas de r avec $rev(r) = s$, alors on écrit $rev(\epsilon_q) = s$.

On dit que $\gamma : E \to K$ est un *bon étiquetage* de M par rapport à l'application *update* et aux valeurs initiales $(k_p)_{p\in\mathcal{P}}$ si $\gamma(\epsilon_p) = k_p$ pour tout $p \in \mathcal{P}$, et $t \in E$

- Si $m(s) = t$, alors $\gamma(t) \in update(\tilde{\lambda}(t), \gamma(t^-), \gamma(s))$,
- Si $rev(r) = t$, alors $\gamma(t) \in update(\tilde{\lambda}(t), \gamma(t^-), \gamma(r))$.

On construit maintenant un CFM $\mathcal{A}$ qui devine un étiquetage, et accepte si et seulement si cet étiquetage est bon par rapport à *update* et $(k_p)_{p\in\mathcal{P}}$. Les données de contrôle sont des éléments de $K \times K$, et l'ensemble S_p des états de $\mathcal{A}_p$ est $S_p = (\Omega_p \uplus \{\epsilon_p\}) \times K^{\Omega_p} \times \{0, \ldots, b-1\}^{\mathcal{T}_p}$. Un état local $(e, mem, cnt) \in S_p$ signifie que :

- e est le Ω-type du dernier événement sur le processus p,
- *mem* associe à chaque Ω_p-type l'étiquetage par K de la dernière lettre étiquetée par ce type.
- *cnt* compte le nombre d'occurrences modulo b de chaque $\mathcal{T}_p$-type.

L'état initial du processus p est $s_p^0 = (\epsilon, (k_a^0)_{a\in\Omega_p}, \overline{0})$, où $\overline{0}$ est la fonction nulle partout.

Nous définissons ensuite la relation de transition : pour deux états (e, mem, cnt) et (e', mem', cnt') de S_p, $a \in \mathcal{T}_p$, et $m \in K \times K$, on a $(e, mem, cnt) \xrightarrow{a,m} (e', mem', cnt')$ si

(a) $cnt'(a) = cnt(a) + 1 \pmod b$, et $cnt'(b) = cnt(b)$ pour tout $b \neq a$,

(b) $e' = (a, cnt'(a))$,

(c) $mem'(f) = mem(f)$ pour tout $f \neq e'$,

(d) $m = (val, guess)$ avec
- Si $a = p!q$, $mem'(e') \in update(e, mem(e), guess)$ et $val = mem'(e')$.
- Si $a = p?q$, $mem'(e') \in update(e, mem(e), val)$ et $guess = mem(e')$.

Informellement, quand un processus envoie un message, il devine une valeur *guess* qui doit correspondre à la valeur de l'événement prédécesseur pour *rev*, calcule $mem'(e') = update(a, mem(e), guess)$ et joint $(mem'(e'), guess)$ en tant que donnée de contrôle. Un processus qui reçoit un message $(val, guess)$ vérifie tout d'abord que $guess = mem(e')$, sinon il bloque. Ensuite, il calcule $mem'(e') = update(a, mem(e), val)$. Ainsi, chaque réception vérifie que la valeur *guess* reçue correspond à la valeur $mem(e')$ de la b-ième réception du même $\mathcal{T}$-type avant lui, c'est à dire la réception immédiatement précédente du même Ω-type e'.

Proposition 31 *Soit ρ une exécution du CFM $\mathcal{A}$ étiquetée par une linéarisation $a_1 \cdots a_n$ d'un MSC M qui a pour ensemble d'événements E. Pour tout $t \in E$, on pose $\gamma(t) = \mathrm{mem}(e)$ pour $(e, \mathrm{mem}, \mathrm{cnt})$ l'état atteint par ρ après $a_1 \cdots a_i$, où la lettre a_i de la linéarisation correspond à l'événement t du MSC. Alors γ est un bon étiquetage par rapport à update et à $(k_p)_{p \in \mathcal{P}}$.*

Preuve. Soit $(s, r) \in E \times E$ un message dans M. Comme ρ est une exécution, on a $(e_s, mem_s, cnt_s) \xrightarrow{a_s,(val,guess)} (e'_s, mem'_s, cnt'_s)$ avec $a_s = \lambda(s)$ et $(e_r, mem_r, cnt_r) \xrightarrow{a_r,(val,guess)} (e'_r, mem'_r, cnt'_r)$ et $a_r = \lambda(r)$. Soit s^-, r^- les prédécesseurs immédiats de s et r. Soit $t \in E$ tel que $rev(t) = s$.

On a $guess = mem_r(e'_r) = \gamma(t)$ par définition de *rev*. Donc $\gamma(s) = mem'_s(e'_s) \in update(a_s, \gamma(s^-), \gamma(t))$ par définition de la fonction de transition. De plus, $val = \gamma(s)$, donc $\gamma(r) = mem'_r(e'_r) \in update(a_r, \gamma(r^-), \gamma(s))$ par définition de la fonction de transition. Ainsi, γ est un bon étiquetage. □

Par analogie avec un CFM, si $update(x, k_1, k_2) = \emptyset$, alors on dira que *update* bloque. Remarquons que pour tout MSC M $\exists$-b-borné, il n'y a pas

de cycle dans la clôture transitive de $m \cup rev \cup (<_a)_{a\in\mathcal{T}}$, donc M admet un bon étiquetage à condition que *update* ne bloque jamais. Ainsi, il existe un chemin acceptant ρ du CFM (qui représente un étiquetage) de $\mathcal{A}$, acceptant M. Cependant, $\mathcal{A}$ peut aussi accepter des MSC non $\exists$-b-bornés.

Soit $\mathcal{M}$ un ensemble de MSC $\exists$-b-bornés avec $Lin^b(\mathcal{M})$ régulier. Puisque $\widetilde{Lin^b(\mathcal{M})}$ est un ensemble régulier de traces pour (Ω, I), on peut appliquer [Zie87] et obtenir une application asynchrone et un ensemble d'acceptation (μ, Acc) reconnaissant $Lin^b(\mathcal{M})$.

Rappelons que pour une application asynchrone μ sur (Ω, I), il existe $\nu : (\Omega \times K \times K) \to K$ une fonction qui prend en entrée $(e, \mu(u \downarrow_f), \mu(u \downarrow_g))$, telle que si e est l'étiquette de l'unique maximum de u, et $\{f, g\}$ contient l'ensemble des étiquettes des prédécesseurs directs de e, alors $\nu(e, \mu(u \downarrow_f), \mu(u \downarrow_g)) = \mu(u)$.

Soit $\mathcal{A}$ le CFM associé à ν et aux valeurs $(k_f^0)_{f\in\Omega}$. De plus, on définit les états finaux du CFM par $(e_p, mem_p, cnt_p)_{p\in\mathcal{P}}$ tels que $(mem_p(e_p))_{p\in\mathcal{P}} \in \mathrm{Acc}$. Alors on a $\mathcal{L}(\mathcal{A}) \cap \mathbb{MSC}^b = \mathcal{M}$. Remarquons qu'il est possible que $\mathcal{A}$ génère aussi des MSC non $\exists$-b-bornés, et donc pas dans $\mathcal{M}$.

Proposition 32 *Soit $\mathcal{M}$ un ensemble de MSC avec $Lin^b(\mathcal{M})$ un ensemble régulier de représentants de $\mathcal{M}$. Il existe un CFM $\mathcal{A}$ avec $\mathcal{L}(\mathcal{A}) \cap \mathbb{MSC}^b = \mathcal{M}$.*

Si $\mathcal{A}''$ est le produit de $\mathcal{A}$ avec le CFM $\mathcal{A}'$ qui accepte $\mathbb{MSC}^b$, on aura bien $\mathcal{L}(\mathcal{A}'') = \mathcal{M}$. La suite va expliquer comment obtenir un tel CFM $\mathcal{A}'$ qui accepte $\mathbb{MSC}^b$.

2.4.4 Les τ-Cycles

Soit $M = (E, \lambda, m, (<_p)_{p\in\mathcal{P}})$ un MSC, $e, f \in E$ et $a \in \mathcal{T}_p$. On définit l'ordre de type $e \lessdot_a f$ si et seulement $e <_p f$, $\lambda(f) = a$ et pour tout g avec $e <_p g <_p f$, on a $\lambda(g) \neq a$. C'est à dire que l'on a $e \lessdot_a f$ quand f est le premier événement après e, sur le même processus p que e, et tel que f soit du type $a \in \mathcal{T}_p$.

On utilise un ensemble de fonctions partielles $Act : E \to E$ sur l'ensemble E des événements d'un MSC, $Act = \{\delta_a \mid a \in \mathcal{T} \uplus \{rev, msg\}\}$ sur E. Pour (e, f) un message, on définit $\delta_{msg}(e) = f$. Pour $rev(f) = e$, on définit $\delta_{rev}(f) = e$. Pour $e \lessdot_a f \in E_p$, on définit $\delta_a(e) = f$. Sinon, on dit que $\delta_a \in Act$ n'est pas défini pour e. Par induction, on définit $\delta_{\sigma\sigma'} = \delta_{\sigma'} \circ \delta_\sigma$ pour $\sigma, \sigma' \in (\mathcal{T} \uplus \{rev, msg\})^\star$. Pour le mot vide, on prend $\delta_\epsilon(e) = e$ pour tout

e. Soit $\tau = a_1 \cdots a_k$. On appelle une suite $e_1 \cdots e_{k+1} \in E^\star$ un *τ-chemin* si $\delta_{a_i}(e_i) = e_{i+1}$ pour tout i. Si $e_{k+1} = e_1$, alors le chemin est appelé un τ-cycle.

Lemme 5 *Soit $\sigma \in (\mathcal{T} \uplus \{\text{rev}, \text{msg}\})^\star$ et $e' \leq e$ avec $\lambda(e) = \lambda(e')$. Supposons que $\delta_\sigma(e)$ soit défini. Alors $\delta_\tau(e')$ est également défini, $\lambda(\delta_\tau(e')) = \lambda(\delta_\tau(e))$ et $\delta_\tau(e') \leq \delta_\tau(e)$ pour tout préfixe τ de σ.*

Preuve. On montre la propriété par induction sur la longueur de τ. L'initialisation $\tau = \epsilon$ est triviale. Soit $\tau = a\tau'$ avec $a \in \mathcal{T} \uplus \{rev, msg\}$.

Si $a = msg$, comme $\delta_a(e)$ est défini, on sait que e est un envoi. Comme $\lambda(e) = \lambda(e')$, e' est aussi un envoi. Puisque M est un MSC, il existe $r \in E$ avec $msg(e) = r$. Donc $\delta_{msg}(e') \leq r$ est défini. Avec FIFO, on a $\delta_{msg}(e') \leq \delta_{msg}(e)$.

Si $a = rev$, comme $\delta_a(e)$ est défini, e est une réception, et il y a $b-1$ réceptions après e du même type, tel que l'envoi s associé avec la $(b-1)$ème réception après e est $\delta_{rev}(e) = s$. Comme $e' \leq e$ est aussi une réception du même type, il existe au moins $b-1$ réceptions après e'. A cause de FIFO, $\delta_{rev}(e') \leq \delta_{rev}(e)$.

Si $a \in \mathcal{T}$, comme $\delta_a(e)$ est défini, il y a un événement de type a après e. Comme $e' \leq e$, il y a aussi un événement de type a après e', et $\delta_a(e') \leq \delta_a(e)$.

On peut ainsi appliquer l'hypothèse d'induction à τ', puisque $\delta_{\tau'}$ est défini pour $\delta_a(e)$, $\lambda(\delta_{\tau'}(e')) = \lambda(\delta_{\tau'}(e))$ et $\delta_a(e') \leq \delta_a(e)$.

□

Soit $\mathcal{R} = m \cup (\lessdot_a)_{a \in \mathcal{T}} \cup rev$. D'après le lemme 4 page 80, il suffit que la relation $\mathcal{R}$ associée à un MSC M soit acyclique pour que M soit existentiellement b-borné. Le lemme suivant montre que il suffit de tester les τ-chemins avec τ borné par une certaine constante pour savoir si il y a un cycle dans $\mathcal{R}$.

Lemme 6 *Soit M un MSC avec un ensemble d'élément E, et $\mathcal{R} = m \cup (\lessdot_a)_{a \in \mathcal{T}} \cup$ rev. Si il y a un cycle dans $(E, \mathcal{R})$, alors il y a un cycle d'au plus $2\wp^2$ événements.*

Preuve. Considérons un cycle de longueur minimale. Soit n sa longueur.

Par l'absurde, supposons que nous avons $\lambda(e_1) = \lambda(e_2)$ pour deux événements e_1, e_2 dans le cycle. On peut supposer par symétrie que $e_2 < e_1$ dans le MSC. Le cycle nous donne un τ-chemin avec $|\tau| < n$ et $\delta_\tau(e_1) = e_2$. En utilisant le lemme 5, on a $e_3 = \delta_\tau(e_2) \leq e_2$ avec $\lambda(e_3) = \lambda(e_2)$. Si $e_3 < e_2$, on peut itérer le procédé pour obtenir $e_4, \ldots$. Comme E est fini, la suite ne peut

pas être infinie, et on va obtenir $e_{i+1} = e_i$ pour un certain i. C'est à dire qu'on a un cycle dans $(E, \mathcal{R})$ de taille strictement inférieure à n, contradiction.

□

Soit $x \in \mathcal{T}$ et $\tau \in (\mathcal{T} \uplus \{rev, msg\})^k$. Un (τ, x)-paquet est un sous ensemble d'événements T dans $\{e \in E \mid \lambda(e) = x\}$, maximal tel que $\delta_\tau(e) = \delta_\tau(e')$ pour tout $e, e' \in T$.

La proposition suivante montre qu'il suffit de tester le dernier événement de chaque (τ, x)-paquet pour savoir si il y a un τ-cycle.

Proposition 33 *Soit M un MSC avec un ensemble d'événements E, $x \in \mathcal{T}_p$, et $\tau \in (\mathcal{T} \uplus \{\mathrm{rev}, \mathrm{msg}\})^k$.*

Alors il existe $e \in E$ avec $\lambda(e) = x$ et $\delta_\tau(e) = e$ si et seulement si il existe un événement f qui est le dernier événement d'un (τ, x)-paquet, tel que $\delta_\tau(f) \leq_p f$.

Preuve. Si il existe $e \in E$ avec $\lambda(e) = x \in \mathcal{T}_p$ et $\delta_\tau(e) = e$, alors on choisit f le dernier événement du (τ, x)-paquet contenant e.

Réciproquement, supposons qu'il existe $f \in E$ avec $\lambda(f) = x$ et $e_1 = \delta_\tau(f) \leq_p f$. Alors on définit une suite $(e_i)_{i \geq 1}$ d'événements sur p avec $e_{i+1} = \delta_\tau(e_i) \leq_p e_i$, en utilisant le lemme 5 page précédente. Puisque la séquence ne peut pas être strictement décroissante par finitude de E, on a $e_i = \delta_\tau(e_i)$ pour un certain i.

□

Nous montrons qu'il existe une borne sur le nombre de paquets en transit dans un MSC $\exists$-b-borné. C'est la raison pour laquelle nous avons introduit les paquets, puisque cette propriété est fausse sur les τ-chemins. Plus précisément, on ne peut pas garder en mémoire finie tous les τ-chemins en transit, mais on peut le faire pour les (τ, x)-paquets, pour tout $x \in \Sigma$.

Lemme 7 *Soit M un MSC, $k > 0$, et $\tau \in (\mathcal{T} \uplus \{\mathrm{rev}, \mathrm{msg}\})^k$. Supposons que $e_0 <_p \cdots <_p e_{kb}$ soient des événements de M avec $\lambda(e_i) = \lambda(e_j)$ pour tout $i, j \leq kb$, que pour tout i, $\delta_\tau(e_i) \in E_p$ est défini,et que $\delta_\tau(e_i) <_p \delta_\tau(e_j)$ pour tout $i < j$.*

Si $e_{kb} <_p \delta_\tau(e_0)$, alors M n'est pas $\exists$-b-borné.

Preuve. On montre d'abord par induction sur la taille $k \geq 0$ de τ que pour tout $i \leq (k-1)b$, si $e_0 <_p \cdots <_p e_{kb}$ et $\delta_\tau(e_0) <_q \cdots <_q \delta_\tau(e_{kb})$, alors il existe $\tau' \in (\mathcal{T} \uplus \{rev, msg\})^*$ avec $\delta_{\tau'}(\delta_\tau(e_i)) \leq e_{i+kb}$. L'initialisation $k = 0$ est triviale en choisissant $\tau' = \tau = \epsilon$.

Soit $\tau a \in (\mathcal{T} \uplus \{rev, msg\})^\star$ avec $|\tau| = k \geq 1$, et $e'_i = \delta_\tau(e_i), e''_i = \delta_a(e'_i)$ pour tout i. Soit $i \leq kb$. Comme $e''_i < e''_{i+j}$, on a $e'_i < e'_{i+j}$ pour $j \leq b$. L'hypothèse de récurrence nous donne un τ' tel que $\delta_{\tau'}(e'_{i+b}) \leq e_{i+b+b(k-1)} = e_{i+bk}$.

Nous prouvons maintenant qu'il existe $a' \in (\mathcal{T} \uplus \{rev, msg\})$ tel que $\delta_{a'}(e''_i) \leq_q e'_{i+B}$. Cela finira l'induction puisque $\delta_{a'\tau'}(e''_i) \leq e_{i+(k+1)B}$, avec le lemme 5 page 91.

Supposons que $a = msg$. Alors, on pose $a' = rev$. On a $msg(e'_{i+j}) = e''_{i+j}$ pour tout $j \leq b$. Par définition de rev, $\delta_{a'}(e''_i) \leq_q e'_{i+b}$.

Supposons que $a = rev$. Alors on pose $a' = msg$. Soit $r = msg(e''_i)$. Alors on a de même $\delta_{a'}(e''_i) = r \leq_q e'_{i+b}$.

Supposons enfin que $a \in \mathcal{T}$. Alors on pose $a' = \lambda(e'_i)$. Comme $e''_i <_q e''_{i+1}$, on a $e'_i <_q e''_i \leq_q e'_{i+1} \leq_q e'_{i+b}$. C'est à dire $\delta_{a'}(e''_i) \leq e'_{i+b}$.

Donc $\delta_{\tau'}(\delta_\tau(e_0)) <_p e_{kb} <_p \delta_\tau(e_0)$. C'est à dire que l'on a un τ'-cycle, ce qui implique que M n'est pas $\exists$-b-borné.

□

La stratégie pour construire un CFM qui accepte un MSC si et seulement si il est $\exists$-b-borné est la suivante. Pour tout type $x \in \mathcal{T}$ et tout τ de taille $k \leq 2\wp^2$, le CFM devine à la volée le dernier événement de chaque (τ, x)-paquet, ce qu'il peut vérifier localement (sinon, il bloque). De plus, il compte le nombre de (τ, x)-paquets en transit. Un MSC est $\exists$-b-borné si et seulement si pour chaque x, τ, le nombre de (τ, x)-paquets en transit ne dépasse jamais kb et est toujours positif ou nul.

2.4.5 Un CFM qui reconnaît $\mathbb{MSC}^b$

Soit $\tau \in (\mathcal{T} \uplus \{rev, msg\})^\star$ avec $\tau = a_1 \dots a_k$ de longueur k qui commence et finit par le même type fixé $x_0 \in \mathcal{T}_p$. Nous définissons maintenant une fonction $update_{(\tau,x_0)}$ telle que $\mathcal{A}$ n'accepte pas de MSC M qui comporte un (τ, x_0)-cycle.

Soit $K = \{0, \heartsuit, \dagger\}^k$ un ensemble de fonctions. On pose $\tau_i = a_1 \cdots a_i$ pour tout $i \leq k$. Une fonction de K associe chaque i à 0 si il n'y a pas de (τ_i, x_0)-chemin arrivant à l'événement courant, à $\heartsuit$ (vivant) si un (τ, x_0)-chemin qui est le dernier d'un paquet passe par l'événement courant, et sinon à $\dagger$ (mort) si il y a un (τ_i, x_0)-chemin qui arrive sur l'événement courant, mais soit le (τ, x_0)-chemin associé n'est pas défini, soit il n'est pas le dernier d'un paquet. On appellera un événement s un *marqueur* si il est le dernier dans son (τ, x_0)-paquet.

Soit $M = (E, \lambda, m, (<_p))$ un MSC. On peut définir facilement une fonction $update_{(\tau,x_0)}$ telle que $\gamma : E \to K$ est un bon étiquetage par rapport à $update_{(\tau,x_0)}$ et les valeurs initiales $\bar{0}$, si pour chaque $t \in E_q$,

– $\gamma(t)(i) = \heartsuit$ si il existe $s \in E$ avec $\lambda(s) = x_0$ et

(1) s est un marqueur et

(2) $\delta_{\tau_i}(s) = t$ ou $\delta_{\tau_i}(s) \leq_q t <_q \delta_{\tau_{i+1}}(s)$,[2]

– Sinon, $\gamma(t)(i) = \dagger$ si il existe $s \in E$ avec $\lambda(s) = x_0$ et (2).

– $\gamma(t)(i) = 0$ si il n'y a pas de $s \in E$ satisfaisant la condition (2).

On pose $\mathcal{A}_{(\tau,x_0)}$ pour le CFM associé à $update_{(\tau,x_0)}$. Il respecte les conditions ci-dessus. Supposons que $x_0 \in \mathcal{T}_p$.

Nous ajoutons au processus p un compteur *transit* tel que, quand un événement t apparaît, *transit* est d'abord décrémenté si $t = \delta_\tau(s)$ pour un marqueur s, et ensuite incrémenté si t est un marqueur. Le CFM bloque dès que *transit* devient strictement négatif ou dépasse kB, ce qui assure la bornitude de *transit*. Informellement, *transit* compte le nombre de (τ, x_0)-paquets en transit à chaque instant.

Proposition 34 *Tout M avec un (τ, x_0)-cycle est rejeté par $\mathcal{A}_{(\tau,x_0)}$.*

Preuve. Supposons par l'absurde que M a un (τ, x_0)-cycle et est accepté par le CFM. Soit $e_1 <_p \cdots <_p e_n$ les marqueurs de M pour x_0, τ. Soit e_i le premier marqueur tel que $\delta_\tau(e_i) \leq_p e_i$ (il existe d'après la proposition 33 page 92). Par définition d'un paquet et d'après le lemme 5 page 91, nous savons que $\delta_\tau(e_j) <_p \delta_\tau(e_i)$ pour tout $j < i$. Ainsi, quand $\delta_\tau(e_j) <_p \delta_\tau(e_i)$ survient, exactement $i - 1$ paquets ont été envoyés et $i - 1$ ont été reçus, c'est à dire $transit = 0$. Comme $\delta_\tau(e_i)$ est défini et est un marqueur, quand $\delta_\tau(e_i)$ arrive, on a $transit = 0$ et un marquer prend fin. Nous avons une contradiction puisque *transit* devient strictement négatif.

□

Proposition 35 *Tout MSC $M \in \mathbb{MSC}^b$ est accepté par $\mathcal{A}_{\tau,x_0}$.*

Preuve. Supposons que M est $\exists$-b-borné. Soit $e_1, \ldots, e_n$ ses marqueurs pour τ, x_0.

Puisque M est $\exists$-b-borné, *transit* est toujours positif. En appliquant la proposition 7 page 92, on sait que *transit* ne dépasse jamais kB.

□

[2]Ce cas est possible quand la dernière action de τ_{i+1} est un type de $\mathcal{T}$

En utilisant le lemme 5 page 91, nous savons que $\mathcal{L}(\mathcal{A}) = \mathbb{MSC}^b$, pour $\mathcal{A} = \prod_{x\in\mathcal{T},|\tau|\leq 2\wp^2} \mathcal{A}_{\tau,x}$. Avec la proposition 32 page 90, nous obtenons

Proposition 36 *Soit $\mathcal{M}$ un ensemble de MSCs tel que $Lin^b(\mathcal{M})$ est un ensemble régulier de représentants de $\mathcal{M}$. Alors il existe un CFM $\mathcal{A}$ tel que $\mathcal{L}(\mathcal{A}) = \mathcal{M}$.*

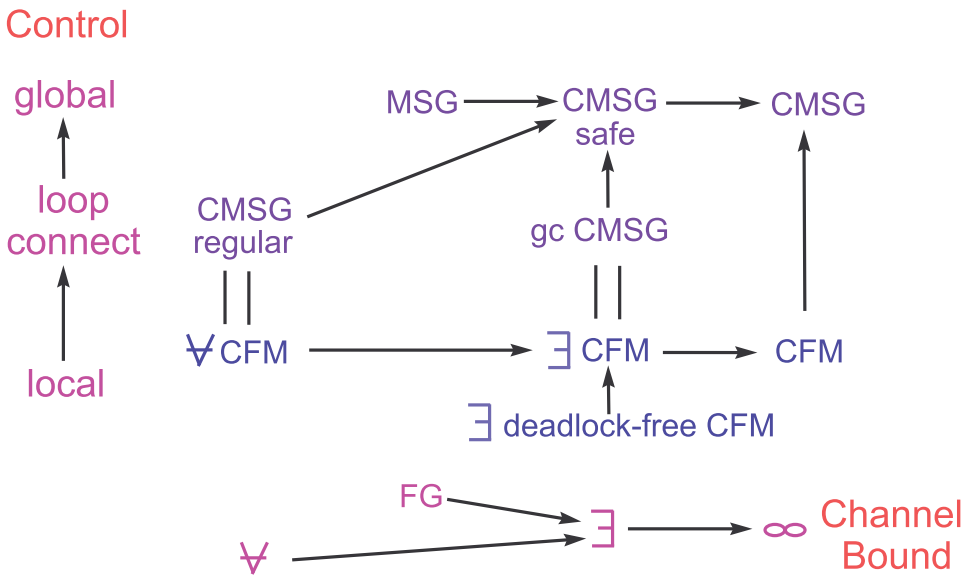


FIG. 2.8 – Diagramme de comparaison des MSC-graphes avec les CFM.

2.5 Voir plus loin ?

Cette extension du théorème de Kleene montre donc que les CMSC-graphes globalement coopératifs sont très puissants. Ils sont expressifs parce qu'ils contiennent les CFM existentiellement bornés. Ils sont implémentables parce qu'ils sont inclus dans les CFM existentiellement bornés. On peut décider si un CMSC-graphe est globalement coopératif, et on peut décider de l'égalité de deux tels CMSC-graphes. Un résultat semblable a été donné par [BL04], prouvant que l'expressivité des CFM (sans restriction) était la même

que celle de la logique EMSO sur les MSC, mais où les ordres employés sont celui de message et l'ordre immédiat sur les processus. Ces ordres sont à degrés entrants et sortant bornés, et il est probable que le resultat ne soit pas généralisable à la logique EMSO usuelle utilisée dans cette thèse et dans [HMNK$^+$04, MM01, Kus03, GKM04].

Cependant, la complexité pour décider de l'égalité de deux CMSC-graphes semble trop grande. De plus, l'implémentation se fait avec énormément de blocages. Ainsi, il serait très utopique de croire que l'on peut se servir de cette implémentation pour développer des protocoles.

Néanmoins, on sait que ces blocages sont inévitables, parce que les CFM sans blocage sont strictement moins expressifs que les CFM. Ainsi, l'idée serait de regarder la classe des CMSC-graphes qui correspond aux CFM sans blocage (éventuellement existentiellement bornés).

Les CFM sans blocage ont d'autres avantages que juste l'implémentabilité. En effet, certains problèmes, indécidables pour les CFM, deviennent décidables dans le cas où il n'y a pas de blocage. Par exemple, nous avons montré dans la proposition 12 page 47 que savoir si un CFM sans blocage est universellement-b-borné est décidable, alors que le problème est indécidable pour les CFM généraux.

Nous avons utilisé des techniques de langages réguliers pour nos preuves. Une autre technique pour obtenir des résultats consiste à utiliser des bons ordres, voir [FS01]. Par exemple, des bons ordres sont donnés par les lemmes de Dickson et Higman. Le lemme de Higman permet d'obtenir la décidabilité des automates communicants Lossy (qui peuvent perdre des messages) [AJ96]. Malheureusement, la complexité de l'accessibilité dans les automates communicants Lossy est non primitive récursive [Sch02].

De même, le lemme de Dickson a permis de prouver la décidabilité de l'accessibilité dans les réseaux de Petri [May84]. Ces lemmes permettent également d'avoir des heuristiques pour calculer certaines constantes. Dans [LMW04], on trouve une heuristique pour savoir si un CFM $\mathcal{A} = (\mathcal{A}^p)_{p\in\mathcal{P}}$ est universellement borné (mais on ne connaît pas la borne). Pour ça, on va permettre plus de comportements, en allégeant la propriété FIFO à FIFO pour deux messages qui ont le même contenu de message. Ainsi, cela revient à avoir un canal par type de message, chaque canal peut être assimilé à un compteur, et on peut utiliser le lemme de Dickson.

Nous allons maintenant illustrer ces méthodes en donnant un algorithme pour calculer une borne universelle (si elle existe) pour un CFM sans blocage. Il est possible que cet algorithme retourne qu'il n'y a pas de borne alors qu'il

y en a une, mais si il renvoie une borne, alors elle est correcte.

On construit maintenant un automate fini $\mathcal{B}$ à partir de $\mathcal{A}$. On arrête la construction de cet automate dés que l'on se rend compte que $\mathcal{A}$ n'est pas borné. Si on ne s'en rend pas compte, c'est que $\mathcal{A}$ est borné, et alors $\mathcal{B}$ représente les comportements de $\mathcal{A}$.

Les états de $\mathcal{B}$ sont représentés par $(\mathcal{F}, f, C^0)$, où $\mathcal{F}$ est un ensemble d'états globaux $((v_p), (b_{p,q})_{p,q\in\mathcal{P}})$ de $\mathcal{A}$, avec $v_p \in \mathcal{A}^p$ pour $p \in \mathcal{P}$, et $b_{p,q}$ le nombre de messages en transit dans le canal (p, q). La fonction f associe à un état global de $\mathcal{F}$ un ensemble de lettres $f(V, B) \subseteq \mathcal{T}$. Enfin, $C^0 \in \mathcal{F}$ est la configuration courante.

Intuitivement, l'ensemble $\mathcal{F}$ va garder les contenus minimaux des canaux pour chaque $\mathcal{P}$-uplet $(v_p)_{p\in\mathcal{P}}$, où minimal s'entend pour l'ordre partiel $(b_{p,q}) \leq (b'_{p,q})$ si $\forall p, q, b_{p,q} \leq b'_{p,q}$. Le lemme de Dickson [Dic13] nous assure que les contenus minimaux des canaux sont en nombre borné dépendant de $\mathcal{P}$, donc que $\mathcal{F}$ est fini. Quant à $f((v_p), (b_{p,q}))$, il va conserver l'alphabet vu depuis la première occurrence de l'état global $(v_p), (b_{p,q})$.

Voilà la fonction de transitions de $\mathcal{B}$.

$(\mathcal{F}, f, (v_p)_{p\in\mathcal{P}}, C^0) \xrightarrow{e} (\mathcal{F}', f', C^1)$, avec $C^0 = ((v_p)_{p\in\mathcal{P}} b_{p,q})_{p,q\in\mathcal{P}}$, $C^1 = (w_p)_{p\in\mathcal{P}}, (c_{p,q})_{p,q\in\mathcal{P}}$, $e \in \mathcal{T}^p$, $v_p \xrightarrow{e,m\ p} w_p$, $v_q = w_q$ pour tout $q \neq p$ et les propositions suivantes sont satisfaites :

- Si $e \in L^p$, alors $b_{r,s} = c_{r,s}$ pour tout $r, s \in \mathcal{P}$,
- Si $e = p!q(a)$, alors $c_{p,q} = b_{p,q}(a, m)$, et $b_{r,s} = c_{r,s}$ pour tout $(r, s) \neq (p, q)$
- Si $e = p?q(a)$, alors $b_{q,p} = (a, m)c_{q,p}$, et $b_{r,s} = c_{r,s}$ pour tout $(r, s) \neq (q, p)$.
- Pour tout $C \in \mathcal{F}$, $f'(C) = f(C) \cup \{e\}$.
- Si $C^1 \in \mathcal{F}$, alors si une composante connexe de $f(C^1)$ n'est pas fortement connexe, l'algorithme s'arrête et renvoie que $\mathcal{A}$ n'est pas borné.
- Si $\exists C \in \mathcal{F}\ C < C^1$, alors l'algorithme s'arrête et renvoie que $\mathcal{A}$ n'est pas borné[3].
- Sinon, on rajoute C^1 à $\mathcal{F}$, on enlève les $C > C^1$ de $\mathcal{F}$, et on note $f(C^1) = \emptyset$.

Proposition 37 *Si $\mathcal{A}$ est universellement borné, alors $\mathcal{B}$ est fini et équivalent à $\mathcal{A}$. Sinon, l'algorithme renvoi que $\mathcal{A}$ n'est pas universellement borné.*

Preuve.

[3]Cette propriété n'est pas nécessaire, mais permet de se rendre compte plus vite si $\mathcal{A}$ n'est pas universellement borné.

Soit $\mathcal{C}$ l'automate (a priori infini) des configurations de $\mathcal{A}$. L'automate $\mathcal{C}$ est infini si et seulement si $\mathcal{A}$ n'est pas universellement borné. On note pour deux configurations globales $C = ((v_p)_{p\in\mathcal{P}} b_{p,q})_{p,q\in\mathcal{P}}$, $C' = (w_p)_{p\in\mathcal{P}}, (c_{p,q})_{p,q\in\mathcal{P}}$ de $\mathcal{C}$ que $C < C'$ si $v_p = w_p$ pour tout p et $b_{p,q} \leq c_{p,q}$ pour tout $p, q \in \mathcal{P}$ et il existe $p_0, q_0 \in \mathcal{P}$ avec $b_{p_0,q_0} < c_{p_0,q_0}$. Si on trouve un chemin entre C et C', alors on peut itérer ce chemin (puisque les états des automates $(\mathcal{A}^p)$ sont les mêmes et qu'il y a assez d'envoi en transit pour tirer cette transition. Comme $\mathcal{A}$ est sans blocage, si $\mathcal{C}$ est accessible, on a un MSC dont une linéarisation a pour préfixe ce mot, donc ce MSC généré par $\mathcal{A}$ n'est pas b_{p_0,q_0} borné. Ainsi, on fait grossir b_{p_0,q_0} aussi gros que l'on veut. Donc $\mathcal{A}$ n'est pas universellement borné.

Supposons que l'algorithme renvoie que $\mathcal{A}$ n'est pas universellement borné. Supposons qu'il existe un chemin voyant un état global C puis plus tard un état global C^1 avec $C < C^1$. Ce chemin correspond à un chemin dans $\mathcal{C}$, qui contient un chemin entre C et C^1. Le CFM $\mathcal{A}$ n'est donc pas b-borné. Sinon, c'est qu'il existe une boucle autour d'une configuration globale accessible C, étiquetée par une linéarisation λ dont le graphe de communication a une composante connexe non fortement connexe. On note par $CC_1, \ldots, CC_k$ les composantes fortement connexes du graphe de communication de λ, ordonnées telles que CC_i ne reçoit jamais de message de cc_j, $j > i$. La propriété nous dit qu'il y a un message entre CC_i et CC_j. On sait que $\lambda \sim \lambda_1, \lambda_2, \lambda_3$, avec λ_2 les événements étiquetés par une lettre de $CC_1 \cdots CC_i$. En particulier, λ_2 contient un envoi mais pas de réception associée (l'envoi de CC_i à CC_j). Puisqu'il n'y a pas d'envoi de λ_1 ou λ_3 vers λ_2, on a $\lambda^k \sim (\lambda_1\lambda_2\lambda_3)^k \sim \lambda_1\lambda_2^k(\lambda_3\lambda_1)^{k-1}\lambda_3$. Ainsi, pour un certain μ, $\mu\lambda_1\lambda_2^k$ est un préfixe d'une linéarisation d'un MSC reconnu par $\mathcal{A}$, qui n'est pas k borné. Donc $\mathcal{A}$ n'est pas universellement borné.

Si $\mathcal{C}$ est infini, d'après le lemme de Koenig, on a un chemin initial qui passe par une infinité de configurations différentes de $\mathcal{C}$. D'après le lemme de Dickson, il existe deux configurations C, C', avec C vu avant C' et $C < C'$. Par construction, on a une configuration $D \leq C$ dans $\mathcal{F}$. Donc quand on voit C', il est comparé à D, et l'algorithme renvoie que $\mathcal{A}$ n'est pas universellement borné.

□

Les mêmes propriétés ont permis de tester si un MSC-graphe est implémentable par un CFM sans blocage et sans donnée additionnelle dans le cas où la propriété FIFO n'est pas requise [Mor02].

Deuxième partie

Vérification des Graphes de MSC

Chapitre 3

Implémentation

L'union même de la médiocrité fait la force
Homère

Les MSC-graphes sont un modèle théorique intéressant parce qu'ils correspondent aux langages rationnels pour la communication par messages. Néanmoins, ils ont été définis assez tôt par la communauté d'informatique pratique puis normalisés par [itu96]. En effet, l'idée est qu'il est difficile de donner directement un algorithme distribué (ou de manière analogue un automate communicant) répondant à un cahier des charges précis. De plus, on a déjà vu que les automates communicants sont indécidables, donc on ne peut pas directement tester s'il suit le cahier des charges. Les CMSC-graphes peuvent servir de cahiers des charges, en donnant un CMSC-graphe pour décrire les scénarios qui doivent être permis, et un autre CMSC-graphe pour décrire les scénarios qui doivent être évités. En effet, étant visuels, ils sont plus faciles à comprendre et manipuler que les automates communicants. L'idée serait donc de partir du CMSC-graphe décrivant les scénarios permis, et de le modifier jusqu'à ce qu'il soit implémentable (on sait que ce n'est pas toujours possible d'implémenter un CMSC-graphe en un automate communicant), tout en testant si les modifications ne permettent pas de mauvais scénarios. On peut évidemment décrire le cahier des charges par d'autres formalismes que les MSC-graphes, par exemple avec des formalismes logiques. On verra dans la suite que si le théorème 5 page 84 nous donne des informations précieuses, il ne fournit pas une réponse totalement satisfaisante.

3.1 Choix Local et Implémentation

La première question de vérification à laquelle nous nous intéressons est très particulière aux MSC-graphes : nous voulons tester si un CMSC-graphe est implémentable en un CFM, et si oui, connaître la taille du CFM associé.

3.1.1 Définition

Comme nous l'avons vu dans l'exemple 31 page 59, il existe des MSC-graphes très simples non implémentables (voir figure 1.10 page 59). La raison est que deux composantes qui n'ont peut-être aucune information l'une sur l'autre peuvent être forcées de faire les mêmes choix (choix global) dans un MSC-graphe, alors que cela est impossible dans un CFM. Ainsi, savoir si un MSC-graphe est implémentable ou non est une question sensée.

Au premier abord, le théorème 5 page 84 semble donner la réponse, c'est à dire que les CMSC-graphes qui sont implémentables sont exactement ceux équivalents aux CMSC-graphes globalement coopératifs. Néanmoins, cette réponse ne clôt pas la question. En effet, il est possible qu'un CMSC-graphe ne soit pas donné sous une forme globalement coopérative, mais qu'il existe un CMSC-graphe globalement coopératif équivalent.

Exemple 41 *Prenons le MSC-graphe de la figure 3.1 page ci-contre à un nœud initial v_0 et un nœud final v_f. Le nœud v_0 contient uniquement l'événement $1(a)$ local sur le processus 1. Le nœud v_f contient uniquement l'événement $2(b)$ local sur le processus 2. Il y a une boucle sur chacun des nœuds, plus une transition du nœud v_0 au nœud v_f. Il existe deux boucles, qui sont chacune connectée, donc le graphe est connecté. Si on considère le même graphe avec en plus une transition du nœud v_f au nœud v_0, alors on crée une boucle non connectée (le graphe de communication est composé des processus 1 et 2). Néanmoins, ces deux MSC-graphes sont équivalents, produisant le langage composé d'au moins un événement $1(a)$ et d'un événement $2(b)$.*

Ainsi, le théorème 5 page 84 permet juste de reformuler la question d'implémentabilité en la question de savoir si un CMSC-graphe est équivalent à un CMSC-graphe globalement coopératif. Malheureusement, savoir si un MSC-graphe (même universellement borné) est équivalent à un CMSC-graphe globalement coopératif est déjà une question indécidable [HMNK$^+$04].

Bien sûr, si le graphe est par chance déjà globalement coopératif, alors nous savons qu'il est implémentable. Cependant, même dans ce cas, il reste

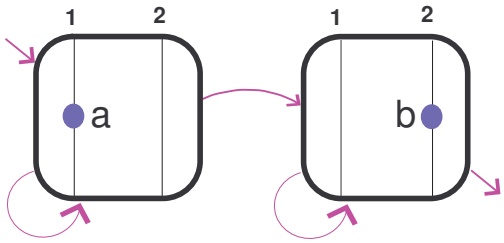


FIG. 3.1 – Un MSC-graphe globalement coopératif.

des problèmes. Premièrement, la construction fait appel au théorème de Zielonka (coûteux), puis à deux constructions non moins coûteuses, qui vont donner des implémentations gigantesques. Deuxièmement, l'implémentation produit d'innombrables blocages, ce qui est bien évidemment à proscrire pour un usage concret. Le théorème 5 page 84 n'est donc pas utilisable dans la pratique, même s'il apporte des informations intéressantes.

A la place, une approche plus pratique a été apportée sous la forme d'une sous-classe stricte des CMSC-graphes globalement coopératifs [HJ00]. Puisque les choix globaux expliquent les problèmes d'implémentation, nous allons demander que chaque scénario dans le graphe soit local, c'est à dire qu'aucun scénario d'un nœud ne doit être initié par deux processus. De plus, on demande que le processus qui initie un nœud soit présent dans chacun de ses prédécesseurs, ce afin qu'il soit informé qu'il devra initier un nouveau scénario. Cette restriction est appelée *local-choice* [HJ00], et cette définition particulière a été définie dans [GM03, GMSZ02] (elle porte dans ce papier le nom de local, mais il y est démontré qu'elle est aussi expressive que la définition originelle).

Définition 41 *Soit G un CMSC-graphe. Alors G est appelé* à choix local *(*local-choice*) si*

- *il est sûr*
- *pour toute transition $v \longrightarrow w$, w a un unique événement minimal* $\min(w)$. *De plus, le processus minimal de w, noté $pmin(w)$, est présent dans le nœud v.*
- *Il existe un processus p_0 tel que chaque nœud initial de G a un unique événement minimal situé sur p_0.*

Exemple 42 *Le MSC-graphe de la figure 1.9 page 58 est à choix local, parce qu'il n'y a qu'un nœud, étiqueté par un MSC avec un unique élément minimal. De plus, ce processus minimal appartient évidemment à l'unique nœud prédécesseur.*

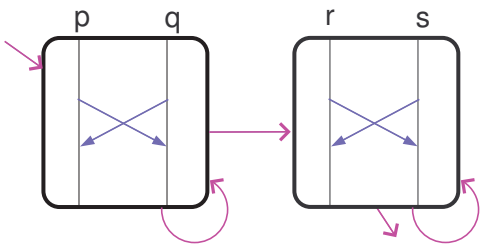


FIG. 3.2 – Un MSC-graphe régulier mais pas à choix local.

Le MSC-graphe de la figure 3.2 n'est pas à choix local, parce que les nœuds possèdent deux processus minimaux. Ces deux exemples montrent que CMSC-graphes réguliers et à choix local sont incomparables.

On peut également vérifier que le MSC-graphe de la figure 3.1 page précédente n'est pas à choix local. Même si chaque nœud a un unique événement minimal, le processus minimal du nœud v_f n'apparaît pas dans v_0.

Remarque 8 *Chaque boucle d'un CMSC-graphe à choix local est connectée puisque tous les processus d'un nœud sont connectés à l'unique processus minimal, et que ce processus minimal est présent dans chaque nœud antérieur. De plus, un CMSC-graphe à choix local est sûr par définition, donc c'est une sous-classe des CMSC-graphes globalement coopératifs.*

En particulier, d'après le théorème 5 page 84, chaque CMSC-graphe à choix local est implémentable. Nous allons montrer ici qu'en fait, un algorithme d'implémentation très simple existe, peu coûteux et surtout sans blocage. Ce résultat a été publié dans [GMSZ02].

Proposition 38 *Soit G un CMSC-graphe à choix local. Alors G est implémentable sans blocage. De plus, la taille du CFM équivalent est linéaire dans la taille du CMSC-graphe.*

Preuve. L'idée pour implémenter un CMSC-graphe à choix local est que le processus minimal courant choisisse le nœud *node* à exécuter. Le processus

minimal courant choisit également le prochain processus minimal *pmin* parmi les processus minimaux des successeurs de *node*.

Ces deux informations sont transmises par chaque processus, de telle manière qu'elles soient connues de tous les processus du scénario (ce qui est le cas par minimalité du processus *pmin*). Ainsi, chaque processus exécute le même scénario. De plus jamais deux processus ne choisissent de débuter un scénario.

Voici l'algorithme d'implémentation pour chaque processus p.

```
void main(){
  int pmin, node;
  initialisation();
  while (true) {
    if (p ≠ pmin) then polling();
       else {node=guess(node); pmin=guess_proc(node);}
    execute_path(node,pmin); }
  }


void polling(){
   while (true) {
      if p receives a message containing v, q
            { node=v; pmin=q; return;} }
   }
```

L'initialisation se fait en choisissant $pmin = p_0$.

L'algorithme $execute\_path(w, q)$ signifie que le processus p exécute les événements de w, (s'il y en a), et envoie avec chaque message l'information de contrôle (w, q).

L'algorithme $guess(node)$ tire au hasard un nœud parmi les nœuds successeurs de *node* pour lesquels le processus courant p est minimal. L'algorithme $guess\_proc(node)$ tire au hasard un processus minimal parmi les processus minimaux dans les nœuds successeurs de *node*. Il peut ne deviner aucun processus, si le nœud actuel est final.

La justesse de l'algorithme est facile à montrer. En effet, chaque nœud est connecté (sinon, il y aurait au moins deux processus minimaux). Chaque processus sauf *pmin* commence son scénario en recevant un message. Ainsi,

il est informé du nœud à exécuter *node* et du prochain processus minimal. Comme pour chaque nœud successeur, son processus minimal apparaît dans le nœud courant, il sera informé qu'il devra commencer le nœud suivant, et les autres processus apprendront qu'ils ne devront pas commencer de nœud. □

Il est bon de noter que l'hypothèse de sûreté est inutile pour cette proposition.

La restriction de choix local apparaît alors être une heuristique pour l'implémentation sans blocage. En effet, si le CMSC-graphe dont on dispose est à choix local ou bien équivalent à un CMSC-graphe à choix local, alors il sera implémentable sans blocage. Néanmoins, s'il n'est pas équivalent à un CMSC-graphe à choix local, cela ne signifie pas qu'il n'est pas implémentable sans blocage.

Exemple 43 *Prenons deux CMSC-graphes non vides à choix local sur des ensembles de processus $\mathcal{P}_1, \mathcal{P}_2$ disjoints. On pose G le CMSC-graphe construit à partir de ces deux CMSC-graphes, avec une transition depuis les états finaux du premier vers les états initiaux du second. Comme les ensembles de processus sont disjoints, il y aura 2 processus minimaux et non un seul, donc ce CMSC-graphe ne peut pas être équivalent à un CMSC-graphe à choix local. Néanmoins, en mettant côte à côte les implémentations de chaque CMSC-graphes, nous obtenons une implémentation sans blocage de G.*

Nous pouvons ainsi terminer le diagramme de comparaison que nous avions commencé, en ajoutant les CMSC-graphes à choix local.

3.1.2 Des Protocoles Concrets

Nous décrivons maintenant deux protocoles concrets, qui serviront à imager les algorithmes que nous allons décrire. Nous allons décrire tout d'abord le protocole WRS (pour Writer-Reader-Server), qui a été donné en partie dans [SC02]. Ensuite, nous décrirons le protocole USB 1.1, qui nous servira à illustrer l'implémentation par les CMSC-graphes à choix local.

Le protocole *WRS* est un protocole qui met en scène trois processus, un écrivain W, un lecteur R et un serveur S. L'écrivain peut mettre à jour une donnée x sur le serveur en envoyant un message $write(x)$. Il a le droit d'annuler son écriture (action locale $fail$), et alors il envoie un message $abort(x)$ au serveur. Sinon, il doit confirmer sa valeur (action locale ok) et envoyer le

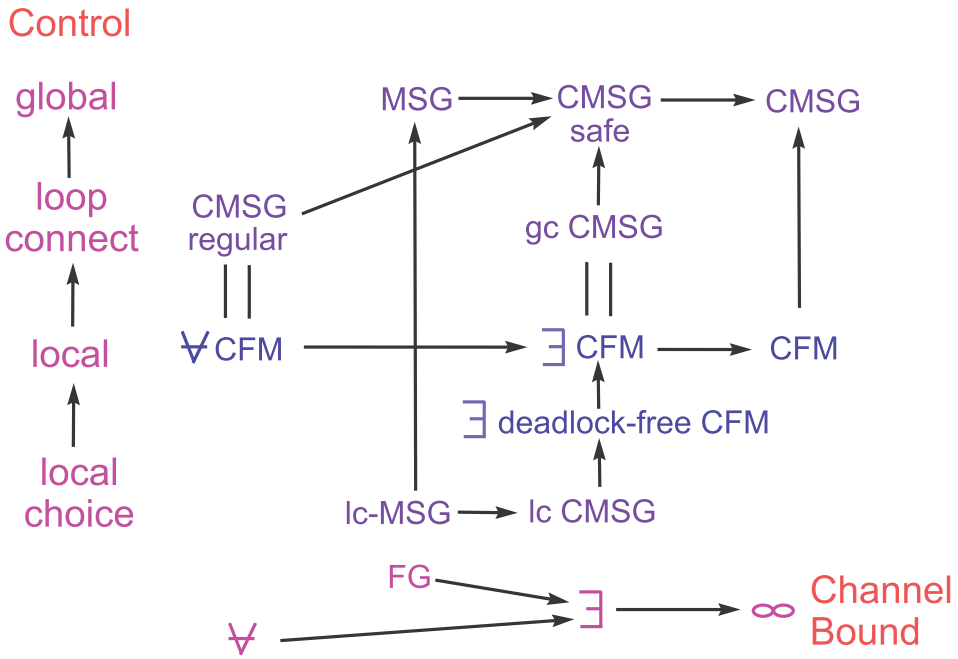


FIG. 3.3 – Diagramme de comparaison des MSC-graphes avec les CFM.

message $commit(x)$ au serveur. Alors, le serveur revient à la donnée précédente par l'action locale rb (rollback). Concernant le lecteur, il peut demander une valeur en envoyant $read(x)$ au serveur. Celui ci lui renvoie la valeur de x courante qu'il possède. Au cas où la valeur ait été par la suite annulée par l'écrivain, le serveur doit envoyer un message $abort(x)$ au lecteur. Dès que le serveur est sûr de la valeur envoyée, il confirme la valeur en envoyant $commit(x)$ au lecteur. Evidemment, on peut faire varier facilement le nombre d'écrivains et de lecteurs pour ce protocole. Ce protocole ne peut pas être décrit par un CMSC-graphe à choix local puisque le lecteur et l'écrivain peuvent choisir de manière concurrente d'envoyer un message.

Le protocole *USB* (Universal Serial Bus, ou bus serie universel) décrit plusieurs modes de communication entre deux processus, un maître (appelé host) et un esclave (appelé function dans la norme) [usb96]. Toutes les commandes sont données par le processus maître (généralement un ordinateur, rarement un PDA), qui a des contraintes plus fortes que l'esclave, lequel ne fait qu'obéir aux ordres du processus maître (au contraire de la norme firewire qui est plus symétrique).

Trois grandes interactions peuvent être effectuées entre le maître et l'esclave. Tout d'abord, le mode de transfert le plus souvent utilisé est le mode isochrone. Le processus maître envoie un premier message pour expliquer lequel des deux processus doit envoyer tel ou tel fichier à l'autre processus. Puis la communication se fait directement par une suite de paquets, grâce au protocole asynchrone (voir figure 1.9 page 58). Cette partie n'est pas universellement bornée, mais peut être décrite à l'aide d'un MSC-graphe (elle est finiment engendrée, donc existentiellement bornée). La seconde interaction est très proche du transfert isochrone. Le maître demande à l'esclave de lui envoyer un compte rendu de son état (protocole setup). Un suite d'au plus 1024 messages est alors envoyée de l'esclave au maître. S'il y a plus d'informations à transmettre, le maître doit faire une nouvelle requête setup. Ainsi, cette partie du protocole est universellement 1024 bornée et existentiellement 1 bornée (et toujours finiment engendrée). De plus, les MSC-graphes obtenus sont à choix local.

Enfin, le transfert bulk est l'analogue du protocole de bit alterné. Là, tous les messages reçus doivent être acquittés avec la parité du message, afin que l'envoyeur soit sûr que son message a été reçu. Afin que tous les canaux restent bornés, une limite est imposée aux envois en transit. Ainsi, cette partie du protocole est universellement bornée. Nous avons représenté une partie du mode de transfert bulk dans la figure 3.4 page ci-contre.

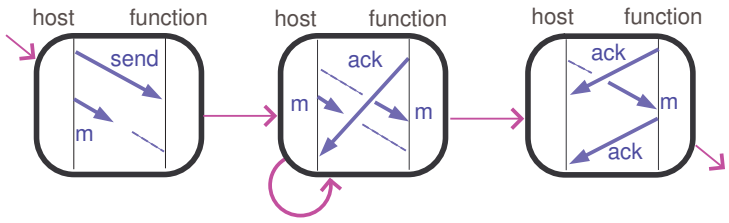


FIG. 3.4 – CMSC-Graphe spécifiant les transactions bulk de usb 1.1.

Exemple 44 *Le CMSC-graphe de la figure 3.4 n'est pas à choix local. En effet, les processus* function *et* host *commencent tous les deux à envoyer dans le nœud qui boucle. Cependant, nous pouvons transformer ce CMSC-graphe en un CMSC-graphe à choix local équivalent, voir la figure 3.5. Nous voulons donner un algorithme pour construire un CMSC-graphe à choix local équivalent à un CMSC-graphe, quand c'est possible. On peut ainsi décrire le protocole* usb 1.1*[usb96] complètement avec un CMSC-graphe à choix local.*

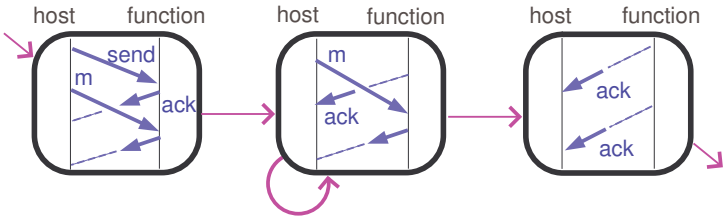


FIG. 3.5 – CMSC-Graphe à choix local spécifiant les transactions bulk de usb 1.1.

3.2 Expressivité

3.2.1 Caractérisation des MSC-graphes à Choix Local

Puisque les CMSC-graphes à choix locaux sont implémentables aisément sans blocage, il est important de connaître leur expressivité. De plus, si nous arrivons à caractériser les langages générés par cette classe, nous obtiendrons

peut être un moyen de tester si un CMSC-graphe est équivalent à un CMSC-graphe à choix local.

En fait, nous avons obtenu une caractérisation sémantique des MSC-graphes à choix local dans [GM03]. Pour cela, une notion cruciale est celle des triangles. On dit qu'un CMSC T est un *triangle* s'il a un unique élément minimal $min(T)$ pour l'ordre visuel. On pose T_n l'ensemble des triangles de taille bornée par n. On définit d'abord un CMSC-graphe H_n^T générique. Le CMSC-graphe H_n^T possède, pour chaque triangle $T \in T_n$, un nœud v_T étiqueté par T. Il y a une transition $v_T \to v_{T'}$ dès que $P(\min(T')) \in P(T)$. On peut remarquer que H_n^T est un CMSC-graphe à choix local.

Proposition 39 *Un CMSC-graphe globalement coopératif G est équivalent à un CMSC-graphe à choix local ssi il existe un entier n tel que $\mathcal{L}(G) \subseteq \mathcal{L}(H_n^T)$. Si c'est le cas, on peut obtenir un CMSC-graphe équivalent à G, de taille exponentielle en $|G|$ et n.*

Preuve. Si $\mathcal{L}(G) \subseteq \mathcal{L}(H)$ pour un certain CMSC-graphe à choix local H, alors $\mathcal{L}(G) \subseteq \mathcal{L}(H_n^T)$ pour $n = |H|$.

Inversement, si $\mathcal{L}(G) \subseteq \mathcal{L}(H_n^T)$, alors on calcule un automate $\mathcal{A}$ acceptant $Lin^{b_G}(G)$ en utilisant le théorème 5 page 84, de taille exponentielle en $|G|$. En synchronisant $\mathcal{A}$ avec H_n^T, on obtient un CMSC-graphe H de taille $|\mathcal{A}||H_n^T|$. On rappelle que H_n^T est exponentiel uniquement en n.

Voici comment est obtenue la synchronisation. Nous avons un nœud pour chaque paire (T, s), où $T \in T_n$ et s est un état de $\mathcal{A}$. Le nœud (T, s) est étiqueté par T. Les transitions sont les couples $(T, s) \to (T', s')$ si $T \to T'$ dans H_n^T et $s \xrightarrow{x} s'$, pour une linéarisation x de T. Les nœuds initiaux sont de la forme (T, s_0), où s_0 est le nœud initial de $\mathcal{A}$. Les nœuds finaux sont ceux de la forme (T, s), où $s \xrightarrow{x} f$ est une suite de transitions étiquetées par une linéarisation x de T amenant à un état final f de $\mathcal{A}$.

□

Cette propriété pour savoir si un CMSC-graphe est à choix local concerne le langage de G plutôt que sa forme. Néanmoins, elle n'apporte que peu d'information en l'état sur les CMSC-graphes à choix local. Un algorithme de test pour savoir si un CMSC-graphe globalement coopératif est équivalent à un CMSC-graphe à choix local ne peut pas non plus être dérivé, parce que nous n'avons pas de borne sur n pour savoir pour quel H_n^T tester si $\mathcal{L}(G) \subseteq \mathcal{L}(H_n^T)$.

Remarque 9 *Nous pouvons définir de la même manière le MSC-graphe à*

choix local générique H_n^M, construits sur l'ensemble des MSC triangles. De plus, un CMSC-graphe globalement coopératif G est équivalent à un MSC-graphe à choix local ssi il existe un entier n tel que $\mathcal{L}(G) \subseteq \mathcal{L}(H_n^M)$.

Nous allons tout d'abord exclure des CMSC-graphes qui sont assurés de ne pas être équivalents à des CMSC-graphes à choix local.

Proposition 40 *Soit G un CMSC-graphe sûr mais non globalement coopératif. Alors G ne peut pas être équivalent à un CMSC-graphe à choix local.*

Preuve. Supposons par l'absurde que $\mathcal{L}(G) \subseteq \mathcal{L}(H_n^T)$ (voir la proposition 39).

Soit une boucle de G non connectée, étiquetée par $M = RS$ avec $P(S) \cap P(R) = \emptyset$. Il est possible que le CMSC M ne soit pas un MSC. Comme G est sûr, il existe L, N deux CMSC avec $LM^kN \in \mathcal{L}(G) \subseteq \mathcal{L}(H_n^T)$ pour tout k.

On choisit $k = n * (|L| + |N| + 2)$, et on prend ρ un chemin de H_n^T étiqueté par LM^kN. Chaque nœud de ρ qui contient un élément de L ou de N contient au plus n éléments de M. Ainsi, il y a des nœuds de ρ qui contiennent uniquement des éléments de M. Chacun doit être étiqueté uniquement par des éléments de S ou de R, ce qui est une contradiction car les éléments minimaux des nœuds sont totalement ordonnés dans un chemin d'un CMSC-graphe à choix local.

□

On va ensuite éliminer les CMSC-graphes avec deux longues composantes en parallèle. Pour cela, nous avons besoin d'un lemme portant sur la structures des triangles.

Lemme 8 *Soit $LMN = B_0C_0B_1 \cdots B_{2\wp}C_{2\wp}B_{2\wp+1}$ avec M un* **MSC** *et L, N des CMSCs. De plus, on suppose que pour tout i, chaque C_i est un triangle avec $C_i \cap M \neq \emptyset$. Alors il existe k avec $C_k \subseteq M$.*

Preuve. Pour tout i, on décompose $C_i = D_iE_iF_i$ avec E_i le CMSC contenant les événements de M, c'est à dire $E_i = C_i \cap M$, et D_i les événements de L, c'est à dire $D_i = C_i \cap L$. On montre que pour au plus $\wp$ indices i, on a $D_i \neq \epsilon$, et pour au plus $\wp$ indices i, on a $F_i \neq \epsilon$, d'où la preuve du lemme.

On a $D_0E_0F_0 \ldots D_{\wp}E_{2\wp}F_{2\wp} = D_0 \ldots D_{\wp}E_0 \cdots E_{2\wp}F_0 \cdots F_{2\wp}$ parce que L est un préfixe de LMN et N un suffixe de LMN.

Pour tout i, $C_i = D_iE_iF_i$ est un triangle. Ainsi, D_i et E_i partagent un processus p_i, ou bien E_i et F_i en partagent un. Sinon, C_i contiendrait un

envoi et sa réception séparés, un dans D_i ou F_i et l'autre dans E_i. Cela n'est pas possible parce que si E_i contient un des deux, alors M aussi. Or M est un MSC, donc il contient aussi l'autre, c'est à dire que E_i contient aussi l'autre.

Soit J l'ensemble des indices pour lesquels D_i et E_i partagent un processus p_i. Comme E_i commute avec D_j, $i > j$, on a $p_i \neq p_j$ pour tout i, j tels que $i \neq j$. Ainsi, on a au plus $\wp$ MSC D_i non vides, ce qui prouve le lemme.

□

Cela nous permet de montrer que si un langage contient une exécution avec deux MSC en parallèle de taille non bornée, alors il ne peut pas être un langage de CMSC-graphe à choix local :

On appelle deux MSC R, S un *couple parallèle dans* G s'il existe deux CMSC L, N avec $LRSN \in \mathcal{L}(G)$ et $P(R) \cap P(S) = \emptyset$.

Proposition 41 *Soit G un CMSC-graphe à choix local. Soit R, S un couple parallèle dans G associé aux CMSC L, N. On pose M le MSC $M = RS$.*

Alors soit $|R| \leq 2\wp|G|$, soit $|S| \leq 2\wp|G|$.

Preuve. Supposons par l'absurde que $\lambda(\rho) = LMN$ pour un chemin ρ de G. De plus, supposons que $M = R \parallel S$ soit un MSC, avec $|R| > 2\wp|G|$ et $|S| > 2\wp|G|$.

Au moins $2\wp$ nœuds de ρ s'intersectent avec R, et chacun de ces nœuds est un triangle. Ainsi, d'après le lemme 8 page précédente, R contient un nœud de ρ. De même, S contient un autre nœud de ρ. Comme les éléments minimaux des nœuds sont totalement ordonnés dans un chemin d'un CMSC-graphe à choix local, on obtient une contradiction.

□

La restriction de ne pas avoir un grand couple parallèle est sans doute la restriction la plus importante des CMSC-graphes à choix local. Elle n'est de plus pas du tout nécessaire pour l'implémentation sans blocage. En effet, les automates totalement en parallèle (qui ne communiquent pas) acceptent une implémentation triviale sans blocage.

On va maintenant montrer une deuxième propriété qui concerne uniquement les MSC-graphes à choix local, et non plus les CMSC-graphes à choix local. On verra que cette restriction ne s'applique pas à tous les CMSC-graphes à choix local, ce qui renforce leur intérêt.

Soit un CMSC M et un événement e de M. Alors e est une *pointe* de M si le futur $\{f \in M \mid e \leq f\}$ de e pour l'ordre visuel est un MSC (en particulier, il ne doit pas contenir de réception sans envoi associé). Dans un **MSC**-graphe

à choix local, chaque élément qui débute un nœud est une pointe. Cela n'est pas toujours le cas dans un CMSC-graphe à choix local.

Soit G un CMSC-graphe. On dit qu'un MSC-triangle M est un *MSC-triangle sans G-pointe* s'il existe un MSC-triangle M et un MSC N avec $LMN \in \mathcal{L}(G)$ et LMN possède une unique pointe à l'intérieur de M (i.e., min_M). Il est bon de noter que LMN peut avoir d'autres pointes que min_M, tant qu'elles ne sont pas dans M. De plus, LM peut avoir plusieurs pointes dans M, mais ces pointes ne le seront plus dans LMN.

Proposition 42 *Soit G un MSC-graphe à choix local. Considérons un MSC-triangle M sans G-pointe. Alors $|M| \leq 2\wp|G|$.*

Preuve. On prend un chemin pour lequel G est un facteur sans pointe. Supposons par l'absurde que $|M| > 2\wp|G|$. Alors $|M|$ intersecte plus de $2\wp$ nœuds de ce chemin. En appliquant le lemme 8 page 111, on obtient un nœud v_i entièrement dans M, c'est à dire une pointe (l'élément minimal de v_i), ce qui est une contradiction.

□

Exemple 45 *Le MSC R constitué de n actions locales $1(a)$, et le MSC S constitué de n actions locales $2(b)$ sont un couple en parallèle pour le MSC-graphe de la figure 3.1 page 103.*

De plus, prenant le MSC-graphe G de la figure 3.7 page 117, le MSC étiquetant le chemin qui prend une fois le noeud initial et n fois le noeud final est un triangle sans G-pointe.

Une conséquence immédiate des propositions précédentes est qu'un CMSC-graphe globalement coopératif avec des couples parallèles non bornés ou avec des MSC-triangles sans pointe non bornés ne peut être équivalent à un MSC-graphe à choix local. On montre maintenant la réciproque, caractérisant exactement les CMSC-graphes qui sont équivalents à des MSC-graphes à choix local. Pour cela, on a besoin d'un lemme supplémentaire.

Lemme 9 *Soit M un MSC et e, e' deux pointes incomparables de M. Alors $Futur(e) \cap Futur(e') = \emptyset$.*

Preuve. Soit e, e' deux pointes incomparables de M. Supposons par l'absurde qu'il existe $r \in \text{Futur}(e) \cap \text{Futur}(e')$ et choisissons r minimal. Alors r est une réception (il doit avoir deux prédécesseurs immédiats, or un envoi n'en

a qu'un). Soit s l'envoi associé. Soit $s \in \text{Futur}(e)$, soit $s \in \text{Futur}(e')$, mais il ne peut pas être dans les deux (sinon, r ne serait pas minimal). Cela signifie que e ou e' n'est pas une pointe, puisque son futur contient une réception sans l'envoi associé, contradiction.

□

On peut alors montrer la caractérisation des **MSC**-graphes à choix local.

Théorème 7 *Soit G un CMSC-graphe sûr. Supposons qu'il existe un entier b tel que :*

1. *G est globalement coopératif.*
2. *Chaque $M \in \mathcal{L}(G)$ est un triangle.*
3. *Chaque MSC-triangle sans G-pointe M satisfait $|M| \leq b$.*
4. *Chaque MSC couple parallèle M, N pour G satisfait soit $|M| \leq b$, soit $|N| \leq b$.*

Alors $\mathcal{L}(G) \subseteq \mathcal{L}(H^M_{\wp b})$.

Preuve. Soit $M \in \mathcal{L}(G)$. On montre comment décomposer M en un chemin ρ de $H^M_{\wp b}$.

Soit E l'ensemble des pointes de M, excepté min_M, et F l'ensemble des minima de E. Comme les pointes de F sont incomparables, leur futurs sont en parallèle d'après le lemme 9 page précédente.

On applique la quatrième hypothèse, qui nous dit qu'il existe $e \in F$ tel que pour tout $f \neq e$, $|\text{Futur}(f)| \leq b$. En utilisant la troisième hypothèse, on obtient $|M \setminus \text{Futur}(e)| \leq b + \sum_{f \neq e, f \in F} |\text{Futur}(f)| \leq \wp b$. Ainsi, nous pouvons définir le premier nœud de ρ comme le nœud de $H^M_{\wp b}$ étiqueté par $M \setminus \text{Futur}(e)$. On itère récursivement le procédé à $\text{Futur}(e)$ pour obtenir la solution.

□

3.2.2 Deux Algorithmes de Test

Le théorème 7 est utile parce qu'elle nous donne un moyen de tester si un CMSC-graphe sûr est équivalent à un MSC-graphe à choix local, à l'aide de quatre tests. Le premier test est co-NP-complet (voir la proposition 24 page 75). Une fois que l'on a un CMSC-graphe globalement coopératif, on peut faire les tests suivants. Le second test est LOGSPACE (voir la proposition 43 page suivante). Il a été également montré sans trop de difficulté dans [GM03] que le troisième test est PSPACE (mais sans borne inférieure),

alors que le quatrième test est co-NP. Ces tests, s'ils sont positifs, donnent de plus des valeurs à b. Le problème est que le troisième test donne une valeur à b exponentielle en la taille de G (alors que le quatrième test donne une valeur polynomiale). Nous ne donnerons pas ici ces constructions, parce qu'elles sont inutilisables en pratique (à cause de l'explosion exponentielle de b, à cause de la troisième propriété). Néanmoins, on peut se souvenir que la troisième propriété n'est pas nécessaire pour être équivalent à un CMSC-graphe à choix local. En fait, nous montrons dans [Gen04] que dans un tel cas, on peut faire mieux.

Proposition 43 *Soit G un CMSC-graphe sûr. On peut tester en LOG-SPACE si tout $M \in \mathcal{L}(G)$ est un triangle.*

Preuve.

Pour chaque processus p, on efface les nœuds de G où le processus p reçoit un message avant d'en envoyer un. Alors on veut savoir si les états où le processus p envoie un message avant d'en recevoir un est accessible depuis l'état initial dans ce nouveau graphe. S'il existe un tel p, alors il existe un MSC $M \in \mathcal{L}(G)$ avec deux éléments minimaux.

□

Ainsi, on peut résumer par le théorème suivant :

Théorème 8 *Soit G un CMSC-graphe sûr. Alors on peut tester en PSPACE si G est équivalent à un MSC-graphe à choix local.*

Si c'est le cas, on obtient un MSC-graphe à choix local de taille au plus doublement exponentielle en $|G|$.

L'exemple 46 montre qu'il y a des cas de MSC-graphes à choix local exponentiels dans la taille des MSC-graphes globalement coopératifs auxquels ils sont équivalents.

Exemple 46 *La famille de MSC-graphes globalement coopératifs ci dessous (figure 3.6 page suivante) est trivialement équivalente à une famille de MSC-graphes à choix local de taille exponentielle en la taille des MSC-graphes globalement coopératifs. Ces graphes ont tous un unique nœud initial et final, sans transition, et il n'est pas possible de faire mieux puisqu'on ne peut pas trouver de pointes autres que l'unique minimum de chaque MSC.*

Comme nous l'avons remarqué plus tôt, la deuxième explosion exponentielle vient du seul test PSPACE, celui pour les pointes, qui doivent être sou-

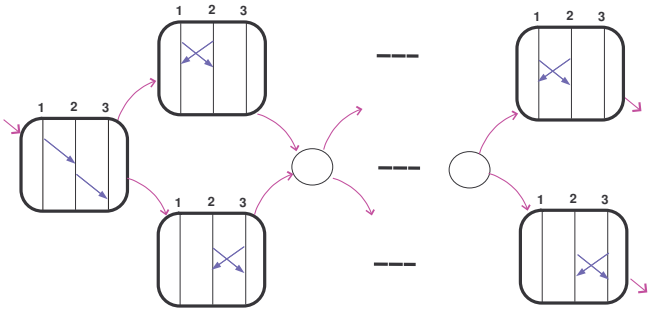


FIG. 3.6 – Une famille de MSC-graphe difficile à exprimer avec des choix locaux.

vent présentes pour que le CMSC-graphe soit équivalent à un MSC-graphe à choix local. Cependant, on sait tout aussi bien implémenter un CMSC-graphe à choix local, pour lesquels la présence de pointes n'est pas nécessaire.

Nous allons montrer qu'un test pour savoir qu'un CMSC-graphe sûr est équivalent à un **CMSC**-graphe à choix local en co-NP, et si la réponse du test est oui, alors on peut construire un CMSC-graphe équivalent exponentiel en la taille du CMSC-graphe de départ. Ainsi, nonobstant le fait de couvrir plus de protocoles implémentables, on va également réduire d'une exponentielle la taille de l'implémentation obtenue !

Exemple 47 *La famille de MSC-graphes globalement coopératifs de la figure 3.7 page suivante n'est pas équivalente à une famille de MSC-graphes à choix local, parce qu'il il y a des MSC de taille infinie avec une unique pointe. Cependant, on peut facilement définir un CMSC-graphe à choix local équivalent.*

On veut décomposer un triangle T en deux sous triangles $T = T_1T_2$. On appelle les événements minimaux $\min(T) = \min(T_1) = e$ et $\min(T_2) = f$. Dans un CMSC, il y a au plus deux événements g, h immédiatement après e (l'événement g successeur immédiat de e sur le même processus, et la réception h de e si e est un envoi). Evidemment, soit $f \geq g$ ou $f \geq h$. Si on veut minimiser la taille de T_1, un choix optimal est donc de prendre soit $f = h$, soit $f = g$. Le triangle T_2 est défini comme étant l'ensemble des événements $\text{Futur}(f)$ dans le futur de f, et T_1 est l'ensemble des événements qui ne sont pas dans $\text{Futur}(f)$ mais sont dans $\text{Futur}(e)$, c'est à dire

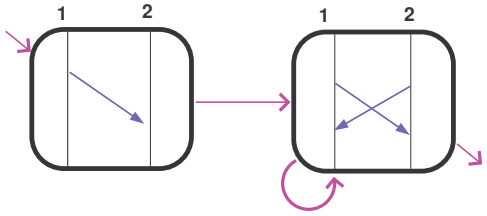


FIG. 3.7 – Les CMSC-graphes à choix local sont plus expressifs que les MSC-graphes à choix local.

$F(f) = \text{Futur}(e) \setminus \text{Futur}(f) = T_1$. Nous montrons que si $F(g)$ et $F(h)$ sont assez grands, alors nous serons capable de trouver deux boucles que nous pourrons itérer, de telle manière que $F(g)$ et $F(h)$ deviennent aussi grands que nous voulons. Rappelons que $H^T_{b_0}$ est le CMSC-graphe à choix local générique avec des noeuds qui sont des triangles de taille au plus b_0.

Proposition 44 *Un CMSC-graphe G globalement coopératif est équivalent à un CMSC-graphe à choix local si et seulement si $\mathcal{L}(G) \subseteq \mathcal{L}(H^T_{b_0})$, avec $b_0 = 4b_G\wp^2|G| + 1$. Ce test peut être effectué en co-NP.*

Preuve. Supposons que $\mathcal{L}(G) \not\subseteq \mathcal{L}(H^T_{b_0})$. Cela signifie qu'il existe un MSC $M \in \mathcal{L}(G)$ et un envoi $e \in M$ tels que pour tout $f \in \{g, h\}$ successeur immédiat de e, nous avons $|F(f)| > b_0$. Sinon, nous pourrions décomposer tout MSC M en des triangles de taille au plus b_0. Notons que e doit être un envoi parce qu'une reception n'a qu'un successeur direct, et donc ne peut poser problème. Ce test peut être effectué en co-NP.

Nous allons d'abord montrer comment grossir $F(g)$, puis par symétrie nous pourrons également grossir $F(h)$. Le MSC M étiquette un chemin de G. Comme il y a b_0 événements dans $F(g)$, il y a au moins $4b_G\wp^2+1$ occurrences du même événement $e_g \in F(g)$. C'est à dire que nous pouvons décomposer M en une séquence $M = BB_1 \cdots B_nB'$ avec B_i étiquetant une boucle de G. De plus, B_i commence et termine par e_g, et $n = 4b_G\wp^2 + 1$.

Parmi ces n boucles, au plus $2b_G\wp^2$ peuvent changer les Ω-types de $\text{Futur}(g)$. Plus précisément, on rappelle que l'alphabet de Kuske (voir section 2.3.1) est $\Omega = \mathcal{T} \times \{0, \ldots, b_G - 1\}$. On pose $Type_i(g)$ l'ensemble des Ω-types dans le futur de g à la fin de B_i, c'est à dire dans $(BB_1 \cdots B_i) \cap \text{Futur}(g)$. Il y a au plus $2b_G\wp^2$ boucles B_i avec $Type_{i-1}(f) \neq Type_i(f)$, puisque $Type_i(f)$ est une

suite croissante bornée. De la même manière, il y a au plus $2b_G\wp^2$ boucles qui peuvent changer les Ω-types de Futur(h). Ainsi, il y a au moins une boucle qui ne change ni les Ω-types de Futur(g), ni ceux de Futur(h). Nous pouvons donc itérer cette boucle B_k sans retirer d'événements à $F(g)$ ou Futur(h) à cause d'une causalité.

Plus formellement, soit $M' = BB_1B_2 \cdots B_kB_k \cdots B_nB'$ le même MSC que M, mais où la boucle B_k a été itérée une deuxième fois. On note Futur$'(h)$, $F'(h)$, et Futur$'(g), F'(g)$ les futurs et non futurs de h, g dans M'. On montre que $F(h) \subseteq F(h)$ et que $F(g) \subsetneq F'(g)$. On pose $f \in \{g, h\}$. Supposons par l'absurde que Futur$'(f) \cap F(f) \neq \emptyset$. On note $d_1 \prec d_2 \prec \cdots \prec d_m$ une chaîne d'ordre avec $d_1 = f, d_m \in F(f)$, et où $d_i \prec d_{i+1}$ si $m(d_i) = d_{i+1}$ ou bien si $d_i <_p d_{i+1}$ pour un certain processus p.

Nous allons montrer que $d_m \in$ Futur(f), une contradiction avec $d_m \in F(f)$. Supposons qu'il y ait un i avec d_i dans le premier B_k et d_{i+1} dans le second. Nous allons supprimer le premier B_k pour obtenir le MSC M. Nous avons dans M, $d_{i+1} <_M d_m$. Comme B_k conserve les Ω-types de Futur(f), nous avons un $d'_i \in$ Futur(f) dans $BB_1 \cdots B_{k-1}$ du même Ω-type que d_i. Si $d_i <_p d_{i+1}$, alors dans M, nous avons aussi $d'_i <_p d_{i+1}$. Donc $d_m \in$ Futur(f).

Sinon, c'est que $m(d_i) = d_{i+1}$. Prenons d le premier élément de B_k du même $\mathcal{T}$-type que d_i (c'est à dire que la seconde composante du Ω-type peut être différente). On a $d'_i <_p d <_p d_i$. Ainsi $d \in$ Futur(f) et nous avons un $d' \in$ Futur(f) avant le premier B_k du même Ω-type que d. Ainsi, nous avons au moins b_G envois du même $\mathcal{T}$-type que d_i dans $[d', d[$, c'est à dire dans Futur(f) et dans $BB_1 \cdots B_{k-1}$, donc dans M. Comme $BB_1 \cdots B_{k-1}$ étiquette un chemin du CMSC-graphe sûr G, il a au plus b_G envois solitaires par canal : ainsi, en supprimant le premier B_k, nous avons un $d'' \in$ Futur(f) avec $m_M(d'') = d_{i+1}$. Donc $d_m \in$ Futur(f).

Sinon, c'est qu'il y a une des deux occurrences de B_k qui ne contiennent aucun $(d_i)_{i \leq m}$. On va enlever cette occurrence et obtenir le MSC M. Nous considérons la nouvelle relation d'ordre dans M. Si $d_i <_p d_{i+1}$ dans M', c'est encore vrai dans M. Si $m_{M'}(d_i) = d_{i+1}$, supposons que $m_M(d_i) \neq d_{i+1}$. Sinon, $d_m \in$ Futur(f). On a donc d_i avant le B_k effacé et d_{i+1} après le B_k effacé. Il doit aussi exister dansle B_k effacé un envoi d du même $\mathcal{T}$-type que d_{i+1} . Nous pouvons alors raisonner comme ci-dessus pour prouver que $d_m \in$ Futur(f).

Concernant $F'(g)$, la boucle B_k contient au moins un événement de $F'(g)$, donc $F(g) \subsetneq F'(g)$. On peut itérer cette boucle, qui fait grossir $F(g)$ sans faire retrécir $F(h)$.

De même, on peut décomposer M en fonction des boucles pour $F(h)$. Ainsi, il y a une boucle qu'on peut itérer, qui contient au moins un événement de $F(h)$, et qui ne diminue pas la taille de $F(g)$. Comme G est sûr, on obtient un MSC de $\mathcal{L}(G)$ en itérant les deux boucles. Ainsi, on obtient pour tout k un MSC $M_k \in \mathcal{L}(G) \setminus \mathcal{L}(H_k^T)$. Ainsi, G n'est équivalent à aucun CMSC-graphe à choix local.

□

En utilisant la propriété 39 page 110 et la propriété précédente, on obtient

Théorème 9 *Soit G un CMSC-graphe sûr. Alors il est décidable en co-NP si G est équivalent à un CMSC-graphe à choix local. Si la réponse est positive, alors un CMSC-graphe à choix local équivalent à G peut être construit de taille exponentielle en $|G|$.*

Nous pouvons également montrer qu'il existe des cas où cette exponentielle doit être payée, ce qui implique l'optimalité du théorème précédent.

3.3 Voir plus loin ?

Nous avons à peu près supprimé la restriction d'avoir souvent des pointes avec les CMSC-graphes à choix local en comparaison avec les MSC-graphes à choix local, tout en diminuant la complexité de transformer un CMSC-graphe en un CMSC-graphe à choix local. Cependant, le manque de parallélisme, qui est l'autre grande restriction des CMSC-graphes à choix local, empêche de représenter certains protocoles implémentables, tout en empêchant une description concise d'autres protocoles.

Comme a priori, le parallélisme ne nuit pas à l'implémentation, l'idée est de rajouter un opérateur de parallélisme au niveau des CMSC-graphes. En donnant une restriction de choix local de telle manière à ce que le CMSC-graphe soit facilement implémentable sans blocage, on résoudrait en partie le problème du manque de parallélisme. Cependant, l'expressivité, les algorithmes de test d'un CMSC-graphe à choix local équivalent et de model-checking seraient plus compliqués. On pourrait même étendre l'opérateur de parallélisme entre deux MSCs n'ayant aucun processus en commun à un opérateur de mélange (shuffle), faisant le mélange des actions sur les processus commun. Les papiers de [Für80, MS94] seraient certainement d'une grande aide pour résoudre ces problèmes.

La question de l'implémentabilité sans blocage est une question difficile. D'autres approches ont été menées. Elles consistent à tester si un MSC-graphe est implémentable par un CFM sans donnée ajoutée (c'est à dire sans donnée de contrôle) et sans blocage [AEY00, AEY01, Loh02]. Le résultat est qu'on ne peut tester une telle implémentation que pour les MSC-graphes globalement coopératifs, en complexité EXPSPACE-complet. Cependant, ce résultat ne prend pas en compte certains MSC-graphes à choix local (qui sont donc implémentables sans blocage si on permet d'ajouter des données). Le papier de [BM03] donne un résultat théorique intéressant, en ce sens qu'il caractérise les MSC-graphes globalement coopératifs implémentables sans blocage et avec données additionnelles : ce sont ceux qui, à renommage des messages près, sont 'cohérents'. Le renommage des messages est analogue à ajouter des données de contrôle. En testant en EXPSPACE si le langage d'un MSC-graphe est cohérent, il s'assure de pouvoir implémenter tous les MSC-graphes implémentables sans donnée additionnelle ni blocage. Toutefois, nous ne savons pas borner le renommage : certains MSC-graphes à choix local ne sont toujours pas pris en compte par ce test (ceux dont le langage n'est cohérent que si les messages sont renommés). Cette approche est également plus coûteuse que celle décrite dans cette thèse.

Chapitre 4

Model-Checking

Felix qui potuit rerum cognoscere causas [1]
Virgile in les Géorgiques, Chant II, vers 489

La deuxième grande question de vérification à laquelle nous nous intéressons est plus standard : il s'agit de tester si un modèle satisfait une certaine propriété. C'est ce que l'on appelle le *model-checking*. Nous donnerons des résultats pour plusieurs types de formalismes. Tout d'abord, nous expliquerons en quoi les logiques usuelles (MSO, LTL) sur les mots sont inadaptées aux problèmes de tester des modèles concurrents. Ensuite, nous présenterons les résultats quand la propriété à satisfaire est exprimée par un MSC-graphe. Cette formulation a l'avantage d'être visuelle, et elle généralise la formulation par ensembles finis de MSC, utilisée par les ingénieurs. Enfin, nous introduirons un nouveau formalisme, logique concurrente analogue de la logique temporelle linéaire (LTL). Cette dernière logique a l'avantage de pouvoir exprimer facilement des propriétés intéressantes. De plus, les formules sont souvent concises, ce qui rend la vérification efficace en pratique. Un implantation est d'ailleurs en cours.

Les modèles peuvent également être exprimés à l'aide de plusieurs formalismes. Les modèles de protocoles intéressants sont les CFM et les CMSC-graphes. On rappelle qu'on peut souvent supposer que le modèle $\mathcal{M}$ est donné sous la forme d'un automate acceptant des représentants de $\mathcal{M}$. En effet, un automate de représentants peut être obtenu de taille linéaire en un CMSC-graphe sûr G. De plus, on peut obtenir un automate acceptant $Lin^{b_G}(G)$

[1] Heureux celui qui a pu pénétrer les causes secrètes des choses.

(voir le théorème 5 page 84) si $\mathcal{M}$ est donné sous la forme d'un CMSC-graphe globalement coopératif. L'automate est alors exponentiel en la taille du graphe $\mathcal{M}$. De plus, si $\mathcal{M}$ est donné sous la forme d'un CFM, on peut également calculer un automate de taille exponentielle en $\wp$ reconnaissant $Lin^b(\mathcal{M})$. Les CFMs ont deux inconvénients comparés aux CMSC-graphe sûrs. Premièrement, on ne connaît pas d'automate de taille polynomiale en $\mathcal{M}$ qui représente un ensemble de représentants[2] de $\mathcal{M}$. D'autre part, même si un CFM est $\exists$-borné, il est indécidable de calculer la borne b. On ne peut même pas tester si un CFM est $\exists$-b-borné.

Dans les théorèmes, quand on aura besoin de la borne existentielle d'un CFM $\mathcal{A}$ pour assurer la décidabilité d'un algorithme, on prendra pour hypothèse que $\mathcal{A}$ est un CFM $\exists$-b-borné.

4.1 Logiques

Une des logiques les plus populaires pour les mots est probablement la logique temporelle linéaire (*LTL*). Elle permet d'exprimer de manière concise des propriétés intéressantes. On peut obtenir un automate acceptant exactement les exécutions satisfaisant une formule LTL, de taille exponentielle en la taille de la formule. Si le modèle est donné par un automate de mots, alors le model-checking est PSPACE dans la taille de la formule (et linéaire dans la taille de l'automate, qui est généralement beaucoup plus gros que la formule).

Définition 42 *Un ensemble de MSC $\mathcal{M}$ satisfait une formule LTL φ si $\mathcal{M} \subseteq \mathcal{L}(\varphi)$. On rappelle que $\mathcal{L}(\varphi)$ contient les MSC dont toutes les linéarisations satisfont φ, ie $Lin(\mathcal{M}) \subseteq L(\varphi)$.*

Comme nous l'avons rappelé précédemment, le langage d'un formule LTL n'est pas clos par commutation.

Ainsi, on a facilement

Proposition 45 *[AY99] Savoir si un CFM ou un MSC-graphe satisfait une formule LTL est indécidable.*

Savoir si un CFM $\mathcal{A}$ universellement b-borné satisfait une formule LTL φ est PSPACE en $|\varphi|_\wp$ et temps linéaire en $b + |\mathcal{A}|$.

[2] C'est le but des réductions d'ordre partiel [Pel98], dont nous reparlerons plus tard.

Savoir si un CMSC-graphe régulier G satisfait une formule LTL φ est PSPACE en $|\varphi||G|$.

Pour les résultats de décidabilité, on calcule pour le CFM $\mathcal{A}$ ou le CMSC-graphe G l'automate acceptant l'ensemble des linéarisations du système, c'est à dire $L(\mathcal{A})$ ou $[L(G)]$. Comme une formule LTL φ ne parle pas de MSC, on ne sait pas utiliser une représentation régulière de $Lin^b(\mathcal{L}(\varphi))$, et plus généralement un ensemble de représentants. Ainsi, la logique LTL dans cette forme n'a pas les mêmes propriétés séduisantes pour les modèles concurrents qu'elle a pour les mots. Pour les traces, des logiques LTL plus adaptées ont été définies [TW02, DG02, DG04]. Nous verrons également plus loin (section 4.3 page 133) une logique adaptée pour les MSC, proche de LTL

La logique *MSO* sur les MSC a quant à elle des propriétés identiques pour les mots et pour les modèles concurrents [Mad01].

Proposition 46 *[MM01] Savoir si un CFM satisfait une formule MSO sur les MSC est indécidable.*

Savoir si un CMSC-graphe sûr ou un CFM existentiellement b-borné satisfait une formule MSO φ est Non élémentaire en $|\varphi|$.

A part pour les cas où on ne connaît pas de borne existentielle concernant les canaux de communication, on peut décider si un modèle concurrent satisfait une formule MSO sur les MSC. Evidemment, il ne peut y avoir de miracle concernant la complexité, puisque le model-checking d'une formule MSO est déjà non élémentaire dans le cas des mots. Ainsi, ces deux formalismes ne sont pas très adaptés au model-checking de structures concurrentes.

4.2 MSC-Graphes

4.2.1 Appartenance (Membership)

Usuellement, la spécification est donnée sous la forme d'un ensemble (généralement fini) de MSC à proscrire, et d'un ensemble de MSC qui doivent être possibles.

Dans le cas d'un ensemble fini, il suffit de tester un par un les MSC pour savoir si ils sont possibles dans le système. C'est ce qu'on appelle le problème d'*appartenance* (*membership*).

Proposition 47 *Savoir si un MSC appartient au langage d'un CFM peut être testé en temps linéaire.*

Savoir si un MSC M appartient au langage d'un CMSC-graphe est NP-complet en le nombre $\wp$ de processus, et NLOGSPACE en $|M|$ [AEY01].

Cependant, la borne inférieure ne tient plus dans le cas où le CMSC-graphe est à choix local. Dans ce cas là, on peut prouver que la question est NLOGSPACE-complète, avec un algorithme en temps quadratique.

Soit un événement e d'un MSC M. On rappelle que l'ensemble des événements $\{f \in M \mid e \leq f\}$ dans le *futur* de e est noté Futur(e).

Proposition 48 *Savoir si un MSC appartient au langage d'un CMSC-graphe à choix local est NLOGSPACE-complet et en temps quadratique.*

Preuve. Soit M un MSC et G un CMSC-graphe à choix local. Supposons que $M \in \mathcal{L}(G)$. Soit $v_1 \cdots v_n$ un chemin dans G étiqueté par $M_1 \cdots M_n = M$. Alors pour tout i, $M_i \dots M_n$ est un triangle. Ainsi, on peut repérer la position actuelle dans le chemin du CMSC-graphe par (l'événement minimal d') un nœud, et la position dans le MSC par l'événement minimal parmi ceux qui n'ont pas encore été vus dans le chemin. Plus précisément, si on sait que $M_1 \cdots M_i$ est un préfixe de M, alors on vérifie que le suffixe $M \setminus M_1 \cdots M_i$ de M n'a qu'un seul minimum e_i. En effet, s'il possède deux minimaux, alors il est impossible de trouver un sous chemin de G étiqueté par Futur$(e_i) = M_{i+1} \cdots M_n$. On vérifie alors que $M_1 \dots M_{i+1}$ est aussi un préfixe de M, en testant si M_{i+1} est un préfixe de Futur(e_i). Nous avons donc un algorithme NLOGSPACE.

Concernant un algorithme en temps quadratique, il suffit de construire un graphe G' composé de $|M|$ copies de chaque nœud. Pour tout nœud v de G et événement e de M, il y a un nœud (v, e) dans G'. Il y a une transition dans G' de (v, e) à (w, f) si Futur$(e) \setminus$ Futur$(f) = \lambda(e)$. Nous rappelons que la complexité de l'égalité de deux MSCs est en temps linéaire. Notons que (v, e) définit au plus un seul f, donc le nombre de transitions est $O(|M||G|)$. Il ne suffit plus que de demander si ϵ est accessible dans le graphe G'. La complexité de l'accessibilité étant linéaire en la taille du graphe, nous obtenons un complexité totale de $O(|M||G|)$.

□

Notons que pour ce résultat, l'hypothèse de sûreté (ou d'équilibre), contenue dans la définition de choix local, est inutile.

4.2.2 Model-Checking

Une généralisation simple de ce problème est de considérer que la propriété est donnée sous la forme d'un graphe de MSC. Comme les MSC-graphes ne sont pas facilement complémentables, nous sommes obligés de séparer la vérification en deux questions distinctes :

- Le model-checking négatif, qui consiste à demander si le modèle, un ensemble de MSC $\mathcal{M}$, produit un MSC d'un ensemble de MSC à proscrire $\mathcal{M}'$: c'est à dire est ce que $\mathcal{M} \cap \mathcal{M}' = \emptyset$.
- Le model-checking positif, qui consiste à demander si le modèle, un ensemble de MSC $\mathcal{M}$, est contenu dans l'ensemble des MSC qui sont possibles $\mathcal{M}'$: c'est à dire est ce que $\mathcal{M} \subseteq \mathcal{M}'$.

On peut alors appliquer le théorème 5 page 84 (et plus particulièrement la proposition 27 page 81) pour obtenir les propriétés suivantes.

Théorème 10 *Savoir si un MSC-graphe contient tous les MSC est indécidable [MP99].*

Le model-checking négatif de deux MSC-graphes est indécidable [MP99].

Soit G un CMSC-graphe sûr, H un CMSC-graphe globalement coopératif et $\mathcal{A}$ un CFM. Alors on peut décider les problèmes suivants :

- $\mathcal{L}(G) \cap \mathcal{L}(H) = \emptyset$ *est PSPACE-complet en* $|H|, b_G$.
- $\mathcal{L}(G) \subseteq \mathcal{L}(H)$ *est EXPSPACE-complet en* $|H|, b_G$.
- $\mathcal{L}(G) \cap \mathcal{L}(\mathcal{A}) = \emptyset$ *est PSPACE en le nombre de processus* $\wp$.
- $\mathcal{L}(G) \subseteq \mathcal{L}(\mathcal{A})$ *est EXPSPACE en le nombre de processus* $\wp$.

On rappelle que b_G est la borne universelle sur $L(G)$, un ensemble de représentant d'un CMSC-graphe sûr G (elle est au plus $|G|$). Il est bon de noter que même en restreignant le problème aux MSC-graphes réguliers, on ne peut pas gagner en complexité (les bornes inférieures proviennent d'ailleurs de ce cas [MP99]).

On peut citer qu'une sous classe des CMSC-graphes globalement coopératifs a été définie dans [GMSZ02]. Il s'agit de la sous-classe des CMSC-graphes localement coopératifs (voir la définition dans [GMSZ02]), pour lesquels une représentation régulière de $Lin^{b_G}(G)$ n'est exponentielle qu'en le nombre des processus $\wp$. Ainsi, on a

Proposition 49 *[GMSZ02] Soit G un CMSC-graphe sûr, H un CMSC-graphe localement coopératif.*

- $\mathcal{L}(G) \cap \mathcal{L}(H) = \emptyset$ *est PSPACE-complet en* $\wp$.

– $\mathcal{L}(G) \subseteq \mathcal{L}(H)$ *est EXPSPACE-complet en* $\wp$.

Ainsi, si le nombre de processus $\wp$ est fixé, alors la complexité du model-checking devient la même que celle des automates.

4.2.3 MSC-Graphes à Choix Local

Il se trouve que les CMSC-graphes à choix local forment une sous-classe des CMSC-graphes localement coopératifs. Ainsi, il se pourrait que la complexité du model-checking soit encore meilleure. C'est effectivement le cas, puisque nous avons démontré dans [GMSZ02] que la complexité du model-checking entre deux MSC-graphes à choix local était la même que celle des automates de mots. Il est bon de noter que dans le cas des **CMSC**-graphes à choix local, on ne connaît pas de meilleur algorithme que celui donné pour les CMSC-graphes localement coopératifs, c'est à dire exponentiellement plus compliqué en le nombre des processus que le model-checking des automates de mots. Dans ce cas, la démonstration est simple, en transformant le CMSC-graphes à choix local G en un CFM $\mathcal{A}$ de taille linéaire en G. Puis on calcule une représentation régulière pour $Lin^{b_G}(\mathcal{A})$, exponentielle uniquement dans le nombre de processus. Si le CMSC-graphe est à choix local, les bornes inférieures ne s'appliquent plus, donc la complexité exacte est encore ouverte.

Nous avons tout d'abord besoin de la notion de (non) décomposition d'un triangle pour donner un lemme qui nous permettra de résoudre le model-checking des MSC-graphes à choix local d'une manière très efficace.

Nous rappelons qu'un triangle n'a qu'un seul événement minimal. Un événement dont le futur est un MSC est appelé une *pointe*. En l'occurrence, l'événement minimal d'un triangle MSC est une pointe. Intuitivement, nous raffinons un triangle en une suite de triangles aussi petits que possible, tout en conservant la propriété de choix local. Nous assurons ainsi une sorte d'unicité (voir le lemme 10 page 128) quand seront comparés l'un à l'autre deux chemins de CMSC-graphes à choix local raffinés.

Quitte à modifier légèrement un MSC-graphe à choix local, on peut supposer que tous les successeurs d'un nœud v commencent par le même processus, que nous appellerons $\mathrm{root}(v)$ (sinon, il suffit de faire une copie de chaque nœud par processus qui commence un nœud successeur, ce qui multiplie la taille au plus par $\wp$). Nous appellerons un tel MSC-graphe *local*. Il est bon de noter que $\mathrm{root}(v)$ n'est pas le processus minimal de v, c'est le processus minimal de w avec $v \rightarrow w$.

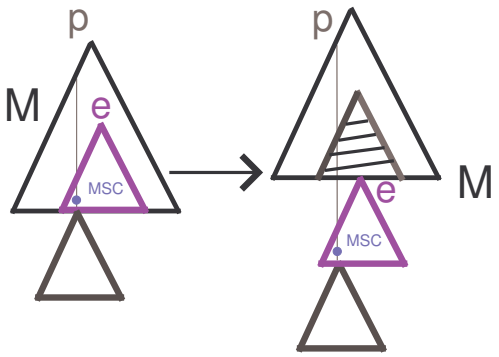


FIG. 4.1 – Une p-décomposition d'un MSC.

Définition 43 *Soit M un MSC et p un processus. On dit que M est p-décomposable si il existe un événement e de M tel que Futur(e) est un MSC et contient un événement sur le processus p, et $M \neq$ Futur(e). Notons que Futur(e) doit contenir l'envoi ou la réception associée par la fonction de message avec chaque événement de Futur(e). En d'autres termes, M est p-décomposable s'il existe un MSC triangle T contenant un événement sur p avec $M = NT$ pour un certain N.*

Nous dirons qu'un nœud v d'un MSC-graphe est p-décomposable si le MSC étiquetant v est p-décomposable. Un MSC-graphe local est non décomposable *si aucun de ses nœuds v n'est p-décomposable, avec $p = root(v)$.*

Proposition 50 *Soit H un MSC-graphe à choix local.*

Alors on peut construire en temps quadratique un MSC-graphe G' local et non décomposable, équivalent à H. De plus, G' est de taille $\wp|H|$.

Preuve. Nous avons déjà expliqué comment obtenir un MSC-graphe G local depuis un MSC-graphe H à choix local. Le MSC-graphe G est de taille $\wp|H|$.

Soit v un nœud de $G = (V, \rightarrow, \lambda, V_0, V_f)$, et $M_1 = \lambda(v)$. On montre comment décomposer récursivement le nœud v. La décomposition est appliquée à un MSC triangle M par rapport à un processus p. Nous commençons par prendre $M = M_1$ et $p = p_1 = root(v)$ (si v n'a pas de successeur, nous choisissons arbitrairement $p_1 \in P(M_1)$). Si M_1 est p_1-décomposable, alors on pose $M_1 = M_2 N_1$ avec N_1 le triangle de taille minimale contenant un événement

sur p_1. Donc N_1 n'est pas p_1-décomposable. Nous répétons le procédé avec $M = M_2$ et p_2 le processus minimal de N_1. La décomposition s'arrête lorsque M n'est plus p-décomposable.

Nous obtenons ainsi une suite $N_l, \ldots, N_1$ de MSC triangles et une suite $p_l, \ldots, p_1$ de processus avec $M_1 = N_l \cdots N_1$, et $pmin(N_i) = \text{root}(N_{i+1}) = p_{i+1}$. De plus, $p_i \in P(N_i)$ pour tout i. Nous remplaçons ainsi v par un nouveau chemin $w_l, \ldots, w_1$, où w_i est étiqueté par le triangle N_i. Notons que ce nouveau chemin $w_l, \ldots, w_1$ satisfait la propriété de localité. De plus, les CMSC étiquetant les nœuds de ce nouveau chemin sont tous non décomposables par rapport au processus minimal du successeur.

Nous appliquons cet algorithme à chaque nœud v de G, en obtenant un CMSC-graphe local et non décomposable G'. Comme chaque événement de G appartient à un et un seul nœud de G', G' est de taille $|G'| \leq \wp|H|$, et équivalent à H. Comme chaque pas de décomposition demande au plus un temps $O(|G|)$, l'algorithme est quadratique.

□

Le lemme suivant signifie que pour décider si deux chemins $(v_1, \ldots, v_k)$ et $(w_1, \ldots, w_l)$ de MSC-graphes non décomposables et locaux sont étiquetés par les mêmes MSC, on peut agir de la façon suivante. Supposons que v_i soit étiqueté par M_i et w_j par N_j. Nous montrons qu'il doit exister un indice i tel que $M_j = N_j$ pour tout $j \leq i$, mais $M_{i+1} \neq N_{i+1}$. De plus, $N_{i+2} \cdots N_l$ est un suffixe de M_{i+1}, et $M_{i+2} \cdots M_k$ est un suffixe de N_{i+1}. Notons que pour v_{i+1} et w_{i+1} fixés, nous pouvons décider facilement si de telles factorisations existent. Si un tel indice i n'existe pas, alors les deux chemins ne sont pas étiquetés par le même MSC. Il suffit pour avoir ce résultat d'appliquer le lemme donné ci dessous de manière récursive.

Lemme 10 *Soit $(v_1, \ldots, v_k)$, $(w_1, \ldots, w_l)$ deux chemins de MSC-graphes locaux et non décomposables.*

Supposons que v_i soit étiqueté par M_i et w_j soit étiqueté par N_j. Supposons de plus que $M_1 \cdots M_k = N_1 \cdots N_l$.

Alors une de ces conditions est satisfaite :

1. *$M_1 = N_1$ ou bien*
2. *$M_1 = XN_2 \cdots N_l$ et $N_1 = XM_2 \cdots M_k$, pour un certain MSC X. De plus, $P(M_2 \cdots M_k) \cap P(N_2 \cdots N_l) = \emptyset$.*

Preuve. Si $l = 1$ ou $k = 1$, alors le second cas est trivialement satisfait. Sinon, on pose $X = M_1 \cap N_1$ l'intersection de M_1, N_1, c'est à dire le plus

grand préfixe commun de M_1, N_1. On pose $M_1 = XY$, $N_1 = XY'$, où $P(Y) \cap P(Y') = \emptyset$ et X est un MSC triangle. On pose également un MSC Z tel que $N_2 \cdots N_l = YZ$. Alors, $M_2 \cdots M_k = Y'Z$.

Nous montrons tout d'abord qu'au moins un des MSC Y, Y', Z doit être vide. Par l'absurde, supposons que Y, Y', Z sont non vides. On choisit e un événement minimal de Z. Comme $YZ = N_2 \cdots N_l$ est un MSC triangle, nous devons avoir $P(e) \in P(Y)$[3]. Par symétrie, $P(e) \in P(Y')$. Mais nous savons que $P(Y) \cap P(Y') = \emptyset$, contradiction.

Si Z est vide, le second cas du lemme est satisfait.

Par symétrie, on peut supposer que Y est vide, c'est à dire que M_1 est un préfixe de N_1. Nous voulons montrer que Y' est aussi vide, c'est à dire que $M_1 = N_1$. Nous montrons d'abord que $N_1 = M_1 \cdots M_j$ pour un certain j, et finalement que $j = 1$.

Soit j le premier indice tel que $N_1 \subseteq M_1 \cdots M_j$. Par l'absurde, nous supposons que l'événement minimal de N_2 appartient à un certain M_i, avec $i \leq j$. Soit f l'événement minimal de M_i. Supposons tout d'abord que $f = e$. Comme G est local, $M_i \cdots M_j$ est un préfixe de $N_2 \cdots N_l$. Ainsi, $N_1 \subseteq M_1 \cdots M_{i-1}$, ce qui contredit la minimalité de j.

Ainsi, $f < e$, et l'événement f appartient à N_1. Comme e est un envoi (sinon, Futur(e) ne serait pas un MSC), et que $f < e$ dans $N_1 N_2$, il existe un événement $g \geq f$ dans N_1 avec $P(g) = P(e) = p$. De plus, la restriction de Futur(f) à N_1 est un MSC puisque f est l'événement minimal de M_i. Cela rentre en contradiction avec la non décomposabilité de N_1. Ainsi $N_1 = M_1 \cdots M_j$.

Nous avons $l \geq 2$, c'est à dire $j < k$. Ainsi, $root(M_j) = \min(M_{j+1}) = \min(N_2) = root(N_1)$. Il existe donc un événement f' de M_j avec $p = P(f') = \text{root}(M_j)$. Donc $e' = \min(M_j) \leq f'$ et les événements e', f' appartiennent aussi à N_1. Mais cela signifie que N_1 est p-décomposable, puisque Futur(e') restreint à N_1 est un MSC. Ainsi, nous avons $e' = \min(N_1)$, c'est à dire $j = 1$, donc Y' vide.

□

Nous pouvons ainsi montrer un résultat très utile :

Théorème 11 *Soient G et G' deux MSC-graphes à choix local.*

[3]Nous aurions pu définir et donner les mêmes résultats en prenant des CMSC-graphes à la place de MSC-graphes. Jusqu'ici. En effet, si nous avons un CMSC à la place d'un MSC, il est possible que Y contienne l'envoi associé à la réception qui est un élément minimal de Z. Or sans ce résultat, toute la démarche est vouée à l'échec.

1. *Décider si* $\mathcal{L}(G) \cap \mathcal{L}(G') \neq \emptyset$ *est NLOGSPACE-complet. De plus, cette question peut être résolue en temps déterministe* $O(\wp^2(|G| + |G'|)^2)$.
2. *Décider si* $\mathcal{L}(G) \subseteq \mathcal{L}(G')$ *est PSPACE-complet en* $|G'|$.

Preuve. Les bornes inférieures proviennent des résultats classiques sur les automates de mots.

Problème d'intersection Nous montrons que le problème d'intersection peut être résolu en temps $O((|G| + |G'|)^2)$. Quitte à appliquer la proposition 50 page 127 à G et G', on peut supposer que G et G' sont locaux et non décomposables.

Supposons que $\mathcal{L}(G) \cap \mathcal{L}(G') \neq \emptyset$. Il existe alors deux chemins acceptants de G, G' étiquetés par $M = M_1 \cdots M_k$ et $N = N_1 \cdots N_l$ avec $M = N$. C'est à dire que ces chemins satisfont les hypothèses du lemme 10 page 128.

Nous définissons un MSC-graphe $G \times G'$ avec $V \times V'$ pour ensemble de nœuds, où V est l'ensemble des nœuds de G et V' celui de G'. Il y a une transition $(v, v') \longrightarrow (w, w')$ dans $G \times G'$ si $\lambda(v) = \lambda(v')$, $v \longrightarrow w$ dans G et $v' \longrightarrow w'$ dans G'. Un nœud (v, v') de $G \times G'$ est ainsi étiqueté par $\lambda(v, v') = \lambda(v) = \lambda(v')$.

Nous testerons le second cas du lemme 10 page 128 en utilisant un ensemble T de nœuds cibles de $G \times G'$. On pose $(v, v') \in T$ ssi :

1. $\lambda(v) = XY$ et $\lambda(v') = XY'$, où $P(Y) \cap P(Y') = \emptyset$.
2. Y' étiquette un chemin depuis un successeur de v jusqu'à un état final de G (si Y' est le MSC vide, alors v doit être un état final de G). Symétriquement, Y étiquette un chemin depuis un successeur de v' jusqu'à un état final de G'.

Nous pouvons calculer si (v, v') est un nœud cible en effectuant une recherche en profondeur dans G, G', pour trouver un chemin final étiqueté par Y depuis un successeur de v' (si v' n'est pas final), et symétriquement pour Y'. Ainsi, l'ensemble cible T peut être calculé en temps quadratique ou espace logarithmique d'après la proposition 48 page 124.

D'après le lemme 10 page 128, $\mathcal{L}(G) \cap \mathcal{L}(G') \neq \emptyset$ si et seulement si T n'est pas accessible dans $G \times G'$ depuis (v^0, v'^0), où v^0 et v'^0 sont respectivement des nœuds initiaux de G et G'. Ainsi, ce problème peut être résolu en temps linéaire dans le nombre de nœuds de $G \times G'$, c'est à dire en temps $O(|G| \cdot |G'|)$. Cela donne ainsi la borne $O(\wp^2(|G| + |G'|)^2)$ en tenant compte de la transformation en un MSC-graphe local et non-décomposable.

La borne supérieure NLOGSPACE est aisée à montrer, nous donnons juste les idées de la preuve. Nous ne transformons pas G, G' en des MSC-graphes locaux et non-décomposables, ni ne construisons le MSC-graphe produit $G \times G'$. Comme d'habitude avec les algorithmes d'accessibilité, il suffit d'une mémoire stockant un seul nœud de $G \times G'$ à chaque instant. Etant donnés un nœud (v, v') de $G \times G'$, le MSC triangle $\lambda(v, v')$ étiquetant (v, v') est identifié de manière unique par deux paires d'événements, (e, f) de G et (e', f') de G'. Plus précisément, $\lambda(v, v') = \text{Futur}(e) \setminus \text{Futur}(f) = \text{Futur}(e') \setminus \text{Futur}(f')$. Nous pouvons tester de manière analogue si $(v, v') \to (w, w')$ est une transition de $G \times G'$ en NLOGSPACE. Pour l'ensemble des nœuds cibles, on peut remarquer que chaque MSC X, Y, Y' apparaissant dans la définition de T est un triangle. Ainsi, nous pouvons raisonner comme ci dessus pour décider en NLOGSPACE si un nœud de $G \times G'$ appartient à T.

Problème d'inclusion

Nous montrons maintenant que le problème d'inclusion se résout en PSPACE. On peut supposer que G, G' sont locaux et non décomposables. Nous allons tester si il existe un chemin acceptant dans G étiqueté par M qui ne correspond à aucune étiquette d'un chemin acceptant dans G'. Pour cela, nous allons comme d'habitude deviner un chemin ρ de G et considérer tous les chemins correspondants possibles dans G'. Pour cela, nous construisons un graphe $\widehat{G \times G'}$ à partir $G \times G'$, qui correspond à peu près au produit de G par le graphe des parties de G'.

Le chemin ρ deviné détermine d'une manière unique tous les chemins correspondants dans G'. Chacun de ces chemins de G' peut rentrer dans l'un ou l'autre cas du lemme 10 page 128. A chaque instant, nous considérons un nœud v de G et un ensemble W' de nœuds de G' qui correspondent au premier cas du lemme 10 page 128. A chaque fois que $(v, v') \in T$ (voir la définition de T pour le problème d'intersection) avec $v' \in W'$, l'algorithme stocke le suffixe de v' qui reste à vérifier dans G, comme indiqué dans le second cas du lemme 10 page 128. Ainsi, un état de $\widehat{G \times G'}$ est un triplet (v, W', S) où v est un nœud de G, W' est un sous ensemble de nœuds de G' et S est un ensemble de suffixes de MSC étiquetant des nœuds de G'.

Plus formellement, soit Suff l'ensemble des suffixes des MSC étiquetant un nœud de G'. Les états de $\widehat{G \times G'}$ sont $V \times 2^{V'} \times 2^{\text{Suff}}$. Les transitions sont définies par $(v_1, W'_1, S_1) \longrightarrow (v_2, W'_2, S_2)$, avec

– $v_1 \to v_2$

– $W'_2 = \{v'_2 \mid \exists v'_1 \in W'_1, (v_1, v'_1) \longrightarrow_{G\times G'} (v_2, v'_2)$[4]$\}$
– S_2 est défini par les deux conditions suivantes :
  (A) Si $M = \lambda(v_1)Y$ pour un certain $M \in S_1$, alors $Y \in S_2$,
  (B) Si $(v_2, v'_2) \in T$ pour un certain $v'_2 \in W'_2$, et $\lambda(v'_2) = XY'$ avec $X = \lambda(v_2) \cap \lambda'(v'_2)$, alors $Y' \in S_2$.

Remarquons que dans le cas B ci-dessus, si $\lambda(v_2) = \lambda(v'_2)$, alors le MSC vide $\epsilon \in S_2$. On pose $\hat{T}$ l'ensemble de nœuds finaux de $\widehat{G \times G'}$: on a $(v, W', S) \in \hat{T}$ si v est final dans G, et que le MSC vide n'appartient pas à S. Les nœuds initiaux sont $(v^0, \{v'^0\}, \{Y'\})$, pour v^0, v'^0 nœuds initiaux de G, G' et $\lambda(v^0) = XY, \lambda'(v'^0) = XY'$, avec $P(Y) \cap P(Y') = \emptyset$. Si X n'éxiste pas, alors nous avons facilement $\mathcal{L}(G) \not\subseteq \mathcal{L}(G')$.

Nous montrons qu'un état de $\hat{T}$ est atteignable si et seulement si $\mathcal{L}(G) \setminus \mathcal{L}(G') = \emptyset$. En tant que problème d'accessibilité dans un graphe de taille exponentiel, un algorithme PSPACE en découle facilement.

Supposons qu'il existe un chemin acceptant $\rho = v_0 \cdots v_l$ dans G, étiqueté par M, tel que $M \notin \mathcal{L}(G')$. Le chemin ρ défini de manière unique un chemin $\hat{\rho} = \hat{v}_0 \cdots \hat{v}_l$ de $\widehat{G \times G'}$ avec $\hat{v}_i = (v_i, W'_i, S_i)$, pour tout i. Nous montrons que $\hat{v}_l \in \hat{T}$. Par l'absurde, supposons que le MSC vide appartienne à S_l. Nous considérons la transition $\hat{v}_k \to \hat{v}_{k+1}$ qui a généré un suffixe Y' dans S_{k+1} (cas B), qui a ensuite amené (par répétition du cas A) au MSC vide dans S_l. Ainsi, il existe une transition $v'_k \to v'_{k+1}$ dans G' telle que la paire $(v_{k+1}, v'_{k+1}) \in T$. De plus, $\lambda(v'_{k+1}) = (\lambda(v_{k+1}) \cap \lambda'(v'_{k+1}))Y'$. Par définition des transitions (plus précisément de la composante W'), nous avons $\lambda(v_i) = \lambda'(v'_i)$ pour tout $i \leq k$. Par choix de k, $\lambda'(v'_{k+1} \cdots v'_m) = \lambda(v_{k+1} \cdots v_l)$, donc $\lambda'(v'_0 \cdots v'_m) = M$ avec v'_m final dans G' par définition de l'ensemble T. C'est une contradiction.

Réciproquement, supposons qu'il existe un chemin acceptant $(v_0, W'_0, S_0), \ldots, (v_l, W'_l, S_l)$ dans $\widehat{G \times G'}$ et $(v_l, W'_l, S_l) \in \hat{T}$. On pose $M = \lambda(v_0 \cdots v_l)$. Par l'absurde, supposons qu'il existe un chemin acceptant $v'_0 \cdots v'_m$ de G' étiqueté par M. Alors, nous pouvons appliquer le lemme 10 page 128. Il existe un indice k tel que $\lambda(v_i) = \lambda'(v'_i)$ pour tout $i \leq k$, et $(v_{k+1}, v'_{k+1}) \in T$. Alors, $v'_{k+1} \in W'_{k+1}$, et $Y' \in S_{k+1}$, où $\lambda'(v'_{k+1}) = (\lambda'(v'_{k+1}) \cap \lambda(v_{k+1}))Y'$. Par définition de T, le MSC vide appartient à S_l, contradiction.

□

On peut remarquer que l'hypothèse de sûreté pour un CMSC-graphe à choix local est très importante pour ce résultat. En effet, il est déjà NP-complet de savoir si le langage des CMSC-graphes à choix local équilibré

[4]C'est à dire, $\lambda(v_1) = \lambda'(v'_1)$ et $v'_1 \to v'_2$.

(au lieu de sûr) est non vide. La preuve, facile (réduction à 1-in-3 SAT), est laissée au lecteur.

Proposition 51 *On peut tester en PSPACE si un MSC-graphe G' est inclus dans un MSC-graphe à choix local G.*

Preuve. Remarquons tout d'abord qu'en vertu de la propriété 39 page 110, $G' \subseteq G \subseteq H^M_{|G|}$ implique que G' est équivalent à un MSC-graphe à choix local G'', de taille exponentielle en G' et G. On teste donc tout d'abord cette propriété en temps PSPACE.

On teste ensuite si $G'' \subseteq G'$, ce qui est PSPACE en G' et en temps linéaire en G'' d'après le théorème 11 page 129. Ainsi, nous avons un test PSPACE en G, G'.

□

Il est bon de remarquer qu'on ne peut pas appliquer la même astuce pour tester le model-checking négatif d'un MSC-graphe globalement coopératif contre un MSC-graphe à choix local. En effet, étant donné qu'il est possible d'interpréter un MSC en tant que MSC-graphe à choix local (quitte à rajouter un unique événement minimal), la complexité du model-checking négatif est au moins aussi haute que celle de l'appartenance pour un MSC-graphe globalement coopératif, complexité qui est déjà NP-complet.

Nous pouvons ainsi colorer le diagramme 4.2 page suivante de comparaison des MSC-graphes avec les CFM par une couleur relative au coût du model-checking négatif entre deux (C)MSC-graphes de cette classe. La complexité va de temps polynomial à indécidable, en passant par PSPACE(pr) en le nombre de processus, et PSPACE(G) en la taille du graphe G.

4.3 Template MSC

Ainsi, à part pour tester le vide de l'intersection de deux MSC-graphes à choix local, le model-checking semble trop onéreux. Nous allons ici proposer un nouveau formalisme logique, dont la complexité dans certains cas semble raisonnable, puisqu'il a les mêmes propriétés que LTL.

Le *template MSC* N dans la partie droite de la figure 4.3 page 136 décrit l'ensemble des CMSC contenant le message a.

L'idée est d'utiliser des boîtes (ou trous), qui peuvent être remplacé par n'importe quel CMSC. Ainsi, nous retrouvons un peu la même idée que [MPS98], sauf que les trous sont définis explicitement.

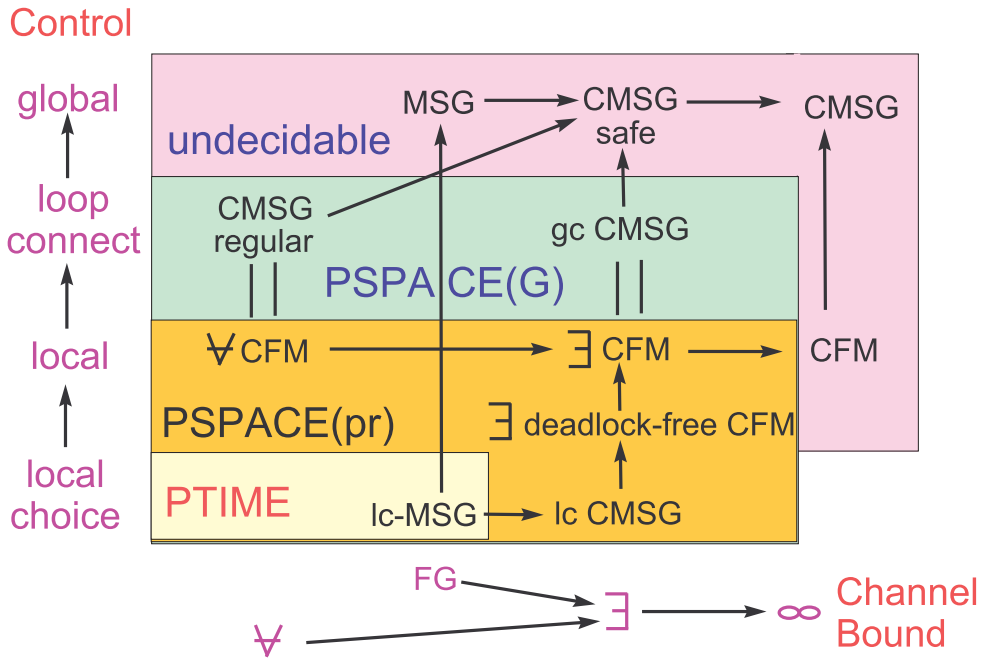


FIG. 4.2 – Coût du vide de l'intersection de deux (C)MSC-graphes.

Définition 44 *Un template MSC est un nuplet* $(\mathcal{P}, E, \Gamma, \mathcal{C}, \lambda, m, <)$*, où* $M = (\mathcal{P}, E, \mathcal{C}, \lambda, m, <)$ *est un CMSC, et :*

- Γ *est un ensemble fini de boites,*
- $\lambda : E \cup \Gamma \to \mathcal{T} \cup 2^{\mathcal{T}}$*, avec* $\lambda : E \to \mathcal{T}$ *la fonction de type de* M *et* $\lambda(\gamma) \subseteq \mathcal{T}$ *les types de messages permis dans la boite* $\gamma \in \Gamma$*. On pose* $\Gamma_p \subseteq \Gamma$ *l'ensemble des boîtes* γ *telles que* $\lambda(\gamma)$ *permet un événement sur le processus* p*.*
- $\leq \subseteq (E \cup \Gamma)^2$ *étend l'ordre* $<$ *du CMSC tel que pour chaque processus* $p \in \mathcal{P}$*, la restriction de* $<$ *à* $E_p \cup \Gamma_p$ *est un ordre total.*

L'ordre entre les boîtes et les événements assure qu'un template MSC peut être effectivement représenté par un diagramme. La sémantique d'un template MSC est l'ensemble (infini) des CMSC obtenus en remplaçant les boîtes par des CMSC dont les événements sont de types permis. Plus formellement, on définit :

Définition 45 *Un template MSC* $M = (\mathcal{P}, E, \Gamma, \mathcal{C}, \lambda, m, <)$ *définit un ensemble de CMSC, que l'on note* $\mathcal{L}(M)$*.*

Un CMSC $(\mathcal{P}, E' = E \cup \bigcup_{\gamma \in \Gamma} E^\gamma, \mathcal{C}, \lambda', m', <')$ *appartient à* $\mathcal{L}(M)$ *ssi il est obtenu en remplaçant chaque boîte* $\gamma \in \Gamma$ *par un CMSC* M^γ*, qui peut être vide, avec un ensemble d'événements* E^γ *tel que :*

- *La fonction de type* λ' *est l'union de* λ *et de la fonction de type de chaque* M^γ*. Il est demandé que* $\lambda'(e) \in \lambda(\gamma)$ *pour tout événement* $e \in E^\gamma$ *(c'est à dire que* M^γ *contient uniquement des événements de type autorisé).*
- *La fonction de message* m' *étend* m *ainsi que la fonction de message de chaque* M^γ*. Il est en outre demandé que* m' *préserve la restriction FIFO sur les événements faisant parti d'un message.*
- *L'ordre visuel* $<'$ *est l'union de* $<$*, de l'ordre visuel de chaque* M^γ*, et de l'ensemble de toutes les paires* (e, f) *satisfaisant* $m'(e) = f$*, ou bien* $P(e) = P(f)$ *plus l'une des conditions suivantes :*
  - $e \in E^\gamma, f \in E^\kappa$ *avec* $\gamma < \kappa$ *(*e, f *sont tous les deux dans des boîtes différentes),*
  - $e \in E^\gamma, f \in E$ *avec* $\gamma < f$ *(*e *est dans une boîte avant* f*),*
  - $e \in E, f \in E^\gamma$ *avec* $e < \gamma$ *(*f *est dans une boite après* e*).*

Remarque 10 *Ainsi, l'ordre (requis dans la définition des templates) entre un événement* e *et une boite* γ *qui partagent un processus, n'implique pas un ordre entre* e *et* **tous les événements** *de* γ *(dans la sémantique). C'est à*

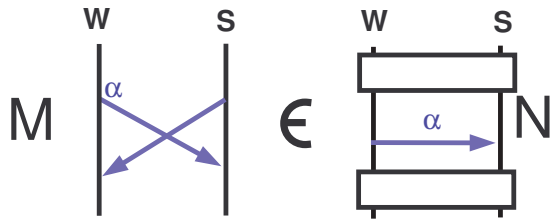


FIG. 4.3 – Template MSC N représentant tous les (C)MSC contenant le message a.

dire que nous pouvons avoir deux événements non ordonnés e, f dans M', avec f venant d'une boite γ, et $e < \gamma$ (cependant, $P(e) \neq P(f)$ dans ce cas). De plus, nous pouvons avoir $e < \gamma < f$ dans M avec $e, f \in E$, $\gamma \in \Gamma$, mais en remplaçant γ par le MSC vide (par exemple), e, f ne sont pas ordonné dans M'.

Remarquons aussi que dans notre définition, il est possible d'obtenir plusieurs fonctions de message m' pour la même instanciation $(M^\gamma)_{\gamma\in\Gamma}$ des boîtes. Cela est requis parce que nous allons concaténer deux template MSC M_1, M_2, tel que le résultat est un MSC. Ainsi, la fonction de message de M_2 dépendra aussi de M_1.

Remarquons enfin que la compositionalité dans les boîtes est requise si l'on veut par exemple décrire l'ensemble de tous les *MSC* contenant un message donné. Supposons par exemple que dans la figure 4.3, les boîtes de N soient instanciées par des MSC plutôt que par des CMSC. Alors le MSC M de la partie gauche n'appartiendrait pas au langage $\mathcal{L}(N)$ du template MSC.

Pour un template MSC M, nous définissons $Lin(M) = Lin(\mathcal{L}(M))$, c'est à dire en tant que l'union des linéarisations de tous les CMSC de $\mathcal{L}(M)$.

Les templates MSC ne peuvent décrire que des motifs simples de communication. Pour augmenter leur expressivité, nous allons utiliser un opérateur temporel de précondition/postcondition, qui permet en particulier d'exprimer des propriétés de sûreté et de vivacité (assume / guarantee), voir la sous section 4.3.1 page 138.

Définition 46 *Soit M_a, M_g deux templates MSC. Alors $M_a \rightarrowtail M_g$ et $M_a \rightarrowtail \neg M_g$ sont des templates MSC temporels, qui définissent des ensembles de MSC, notés par $\mathcal{L}(M_a \rightarrowtail M_g), \mathcal{L}(M_a \rightarrowtail \neg M_g)$:*

- $\mathcal{L}(M_a \rightarrowtail M_g) = \{N \in MSC \mid$ *pour chaque décomposition* $N = ST$, *soit* $S \notin \mathcal{L}(M_a)$ *soit* $T \in \mathcal{L}(M_g)\}$.
- $\mathcal{L}(M_a \rightarrowtail \neg M_g) = \{N \in MSC \mid$ *pour chaque décomposition* $N = ST$, *soit* $S \notin \mathcal{L}(M_a)$ *soit* $T \notin \mathcal{L}(M_g)\}$.

Remarquons que S, T peuvent être des CMSC, mais $ST = N$ doit être un MSC. Remarquons également que les template MSC temporels généralisent les MSC, puisque tout MSC M peut être représenté comme $\epsilon \rightarrowtail M$, où ϵ est le MSC vide.

Un exemple de template MSC temporel est donné par la figure 4.4, dans le cadre du protocole WSR, voir section 3.1.2 page 106. Il exprime que si aucune action n'a été effectuée sur l'écrivain W avant l'envoi d'une valeur du serveur S au lecteur R, alors le serveur peut valider la valeur sans danger.

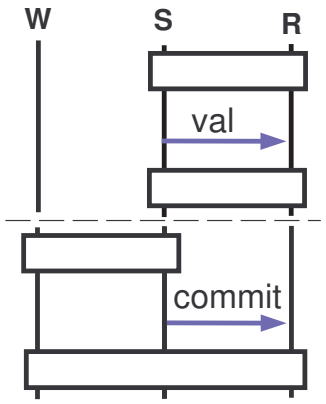


FIG. 4.4 – Un template MSC temporel exprimant une propriété du protocole Writer-Server-Reader, voir section 3.1.2 page 106.

Une *formule de templates MSC* est une conjonction $\bigwedge_i (M_a^i \rightarrowtail (\bigvee_j \pm M_g^{ij}))$, où $\pm$ signifie que la postcondition M_g peut apparaître sous forme soit positive, soit négative. C'est à dire, pour chaque spécification temporelle, nous avons une précondition sous forme positive, plus une disjonction de postconditions sous forme positive ou négative (scénarios interdits). Ainsi, un MSC N appartient à $\mathcal{L}(M_a \rightarrowtail \bigvee_j \pm M_g^j)$ si pour chaque décomposition $N = ST$, dés que $S \in \mathcal{L}(M_a)$, soit il existe un M_g^j sous forme positive avec $T \in \mathcal{L}(M_g^j)$, ou bien il existe un M_g^j sous forme négative avec $T \notin \mathcal{L}(M_g^j)$. Entre autres, il

est permis d'utiliser la postcondition **false**, avec $\mathcal{L}(\mathbf{false}) = \emptyset$. Par exemple, un MSC N satisfait $M \rightarrowtail \mathbf{false}$ ssi aucun préfixe de N n'appartient à $\mathcal{L}(M)$. La formule $\epsilon \rightarrowtail \neg M$ décrit quand à elle le complémentaire de $\{M\}$. Notons que nous ne savons pas comment exprimé le complémentaire de $\{M\}$ en utilisant des CMSC-graphes.

4.3.1 Expressivité

Les formules de templates MSC peuvent décrire facilement et de manière concise des propriétés intéressantes. De plus, on peut tester si un modèle satisfait une formule de templates MSC avec des conditions assez faibles sur le modèle (voir la sous section suivante). Nous pouvons les utiliser pour décrire des propriétés globales sur les configurations de MSC, et utiliser les boîtes en temps que filtres, en restreignant le type des événements permis. Dans les exemples ci dessous, on note par γ une boîte dont le type n'a pas été restreint, et par $\gamma_{\neg a}$ une boîte qui peut générer tous les événements de tous les types, sauf a.

- $(\gamma A) \rightarrowtail \text{false} = \epsilon \rightarrowtail \neg(\gamma A \gamma)$: Aucune exécution ne contient le MSC A.
- $\gamma \rightarrowtail \gamma A \gamma$: Toutes les exécutions contiennent infiniment souvent le MSC A.
- $\gamma A \rightarrowtail \gamma B \gamma$: A chaque fois que A apparaît, le MSC B doit apparaître plus tard. C'est à dire qu'il y a un B tel que tous les CMSC A sont strictement avant ce B.
- $(\gamma A \rightarrowtail \gamma a \gamma) \wedge [\epsilon \rightarrowtail (\gamma_{\neg a} \vee \gamma_{\neg a} A \gamma)]$: Le MSC A peut apparaître. Si c'est le cas, alors l'événement a doit arriver plus tard. De plus, l'événement a ne peut pas apparaître avant que A ne soit apparu. On peut interpréter a comme une alarme, qui retentit si le scénario A se produit.

Le théorème ci dessous montre que l'expressivité des boîtes compositionnelles a cet inconvénient que la satisfaisabilité d'une formule est indécidable. Cependant, nous pouvons tester pour toute borne b si il existe un MSC $\exists$-b borné qui satisfait une formule de templates MSC Φ.

Proposition 52 *Soit un entier positif b et une formule de templates MSC Φ.*

1. *Savoir si il existe un MSC $\exists$-b-borné dans $\mathcal{L}(\Phi)$ est décidable.*
2. *Il est indécidable de savoir si $\mathcal{L}(\Phi) \neq \emptyset$.*

La preuve du premier point est un corollaire du model-checking (voir la prochaine sous section). Pour le second point, nous pouvons réduire le problème de correspondance de Post, en utilisant une communication par canaux non bornés.

Preuve. Nous réduisons une variante du problème de correspondance de Post (PCP), voir exemple 11 page 28, au problème de satisfaisabilité. Soit $(v_i, w_i)_{1\leq i\leq n}$ une suite de couples de lettres.

La réduction de PCP est décrite en utilisant le système WRL définit ci dessous. Le système WRL est composé de trois processus : un écrivain W, un lecteur R et un journal (log) L. Le comportement voulu est que l'écrivain envoie au lecteur une suite de messages parmi n messages possibles $(V_i)_{1\leq i\leq n}$. Quand l'écrivain a fini d'envoyer un message, il envoie un message m_1 au processus L. Quand le lecteur a fini de recevoir, il envoie un message m_2 au processus L. Après cela, l'écrivain peut recommencer à envoyer un message, ou s'arrêter.

Le symbole γ décrit une boîte sur trois processus, sans aucune restriction de type. Le système est décrit par $(\epsilon \rightarrowtail \bigvee_{1\leq i\leq n} N_i) \wedge (M_1 \rightarrowtail \bigvee_{0\leq i\leq n} N_i)$, où

- $M_1 = \gamma m_1 m_2$, signifiant que m_1 suivi immédiatement par m_2 apparaîtra à la fin de M_1.
- $N_0 = \epsilon$ signifiant que le protocole s'arrête.
- $N_i = V_i m_1 m_2 \gamma$ signifiant que V_i doit arriver, et puis m_1, m_2.

En utilisant le même motif $m_1 m_2$ dans la précondition et la postcondition, on initie une récurrence.

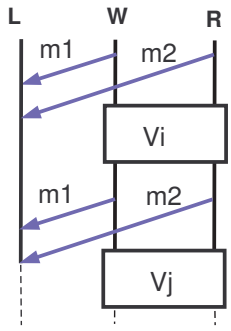


FIG. 4.5 – MSC généré par le protocole WRL.

Rappelons que m_1 est un message de W à L, et m_2 est un message de R à L. Nous utilisons des CMSC V_i, dans lequel $|v_i|$ envois solitaires partent de l'écrivain vers le lecteur, le j-ème envoi étant étiqueté par la j-ème lettre de v_i, et dans lequel le lecteur reçoit $|w_i|$ réceptions solitaires depuis l'écrivain, chacune étiquetée par la lettre correspondante de w_i.

Nous rappelons que dans une instanciation avec des boîtes compositionnelles, le premier envoi solitaire d'une boîte est reçu par la première réception solitaire. Si l'envoi et la réception associée ne sont pas étiquetés par la même lettre, alors le MSC obtenu n'est pas légal. Ainsi, nous sommes assurés qu'un MSC généré par cette formule de templates MSC satisfait $v_{I_1} \cdots v_{I_k} = w_{I_1} \cdots w_{I_k}$. De plus, les envois doivent précéder les réceptions associées, ce qui assure que $|v_{I_1} \cdots v_{I_j}| \geq |w_{I_1} \cdots w_{I_j}|$, pour tout j. Alors, il y a un MSC dans le langage WRL ssi le problème PCP a une solution.

Il reste cependant à montrer comment on peut imposer que les boîtes soient instanciées par les CMSC V_i. Nous montrons ici comment simuler les CMSC V_i avec des boîtes, en utilisant des messages additionnels m_3, m_4. Nous remplaçons un envoi solitaire a par deux messages m_3 de W à L qui entourent une boîte, dont le seul type permis est $W!R(a)$. Pour interdire à une boîte d'être vide, nous ajoutons une règle : deux m_3 consécutifs sont interdits, c'est à dire que nous rajoutons la règle $(\gamma m_3 m_3 \mapsto m_4 \gamma)$, où m_4 est un message de R à W. Le message m_4 n'est pas permis dans le protocole WRL puisque il n'y a aucun envoi de R à W.

De la même manière, nous pouvons nous assurer qu'une boîte ne comporte qu'un seul envoi solitaire. Pour cela, nous utilisons la règle $\gamma\gamma_1\gamma_1 \mapsto m_4\gamma$ où γ_1 est une boîte concernant uniquement le processus W avec comme restriction de ne pas utiliser les types m_1 ou m_2. Nous faisons de même pour les réceptions solitaires.

□

4.3.2 Model-checking

Nous considérons maintenant la vérification d'un protocole de communication S par rapport à une formule de templates MSC Φ.

Définition 47 *Soit S un automate (de linéarisations), et $M_a \rightarrowtail \pm M_g$ un template MSC temporel. Nous avons $S \models (M_a \rightarrowtail \pm M_g)$ si $\mathcal{L}([S]) \subseteq \mathcal{L}(M_a \rightarrowtail \pm M_g)$. La satisfaction d'une formule de templates MSC est défini de manière usuelle à partir de la sémantique de $\wedge, \vee$.*

Une adaptation triviale de la proposition 52 page 138 montre que

Proposition 53 *Savoir si un CFM satisfait une formule de templates MSC est indécidable.*

Pour la décidabilité, le système S doit disposer d'une borne existentielle, noté b_S. Comme nous nous intéressons à $\mathcal{L}([S])$, $L(S)$ peut ne générer que des représentants de $\mathcal{L}(S)$. Rappelons qu'on peut obtenir un tel automate de taille linéaire en la taille d'un CMSC-graphe sûr, ou exponentielle en le nombre de processus $\wp$ à partir d'un CFM $\exists$-b_S-borné.

Dans la suite, nous donnons des résultats pour tester si $S \models \Phi$ pour plusieurs classes de formules de templates MSC Φ. Alors que S est généralement très grand, une formule Φ et la borne b_S sont généralement beaucoup plus petites. Ainsi, nous nous efforcerons de garder la complexité linéaire en la taille du modèle S. Nous allons transformer la formule en un automate. Ainsi, nous pourrons appliquer les algorithmes usuels concernant les automates. Remarquons enfin que l'on peut décomposer le test de $S \models \bigwedge_i \Phi_i$ en plusieurs sous tests $S \models \Phi_i$.

Proposition 54 *Soit un template MSC M_g et un automate S avec une borne existentielle b_S. Nous pouvons tester si $S \models \epsilon \rightarrowtail M_g$ en EXPSPACE en $(b_S|M_g|)$ et LOGSPACE en $|S|$.*

Preuve. On pose M le MSC obtenu à partir de M_g en remplaçant chaque boîte par le MSC vide. Soit E l'ensemble des événements d'un template MSC M_g. On fixe une linéarisation $x = x_1 \cdots x_n$ de M. Nous montrons comment construire un automate non déterministe $\mathcal{A}_x$ acceptant **chaque** linéarisation de $Lin(M_g)$ dont les événements apparaissent dans l'ordre donné par x.

Pour chaque boîte γ de M_g et chaque processus p permis par γ, nous utilisons un symbole γ^p. Nous fixons tout d'abord le début et la fin de la boîte γ sur le processus p en choisissant deux positions $i \leq j$ dans x, et en ajoutant une occurrence de γ^p entre chaque événements x_k, x_{k+1}, pour $i \leq k < j$. Le choix de i, j doit être en accord avec la position de γ sur le processus p. Par exemple, les événements sur p avant γ, ainsi que les symboles κ^p avec $\kappa < \gamma$ doivent apparaître avant x_i. Soit y un mot obtenu de cette façon. Il reste à remplacer chaque symbole γ^p par $X^*_{\gamma,p}$, où $X_{\gamma,p} \subseteq \lambda(\gamma)$ est l'ensemble des types du processus p permis par γ.

Afin d'obtenir toutes les linéarisations de $Lin(M_g)$, l'automate non déterministe génère toutes les permutations possibles d'événements préservant la

suite de boîtes $X^*_{\gamma,p}$ sur le processus p entre chaque événements x_i, x_{i+1}. Par exemple, supposons que nous ayons $X^*_{\gamma_1,p}X^*_{\gamma_3,r}X^*_{\gamma_2,p}X^*_{\gamma_1,q}$ entre deux événements consécutifs. Alors l'automate générera $(X_{\gamma_1,p} \cup X_{\gamma_3,r} \cup X_{\gamma_1,q})^*(X_{\gamma_3,r} \cup X_{\gamma_2,p} \cup X_{\gamma_1,q})^*$.

Nous devons nous assurer que les messages de M sont préservés. Pour cela, nous utilisons un compteur (de valeur maximale b_S) par message dans M. De plus, pour s'assurer qu'un message n'est jamais reçu avant d'être envoyé, nous utilisons un autre compteur par canaux de communication (de valeur maximale b_S). Quand un message m décrit dans M (donc pas dans une boîte) du processus p à q est envoyé, le compteur C_m est initialisé à 0. Nous l'incrémentons quand un message de p à q est envoyé, et le décrémentons quand un message de p à q est reçu. Quand le message m est finalement reçu, nous bloquons le compteur C_m. De la même manière, nous incrémentons et décrémentons le compteur correspondant à un canal, et le bloquons si sa valeur devient -1 ou $b_S + 1$. L'automate accepte si tous les compteurs sont à 0.

Cet automate non déterministe est exponentiel en $b_S|M_g|$ et accepte **toutes** les linéarisations b_S-bornées de $Lin(M_g)$. Si M_g est donné sous forme positive, alors nous pouvons tester $\mathcal{L}(S) \subseteq Lin(M_g)$ en espace exponentiel en $b_S, |M_g|$.

□

Si la postcondition est donnée sous forme négative $\neg M_g$, nous pouvons tester $\mathcal{L}(S) \cap Lin(M_g) = \emptyset$ en espace polynomial en $b_S, |M_g|$.

En ce qui concerne la précondition M_a, nous construisons le même automate. Nous calculons ainsi en espace polynomial les états de S qui peuvent être atteints depuis un état initial par une exécution étiquetée par un CMSC de $\mathcal{L}(M_a)$. Nous obtenons ainsi :

Théorème 12 *Tester si* $S \models M_a \rightarrowtail \bigvee_i (\pm) M^i_g$ *est un problème EXPSPACE en* $b_S|M_g|$*, et PSPACE en* $|M_a|$*.*

Tester si $S \models M_a \rightarrowtail \bigvee_i \neg M^i_g$ *est un problème PSPACE en* $b_S(|M_g| + |M_a|)$*.*

Ainsi, l'expressivité des formules de templates MSC a un coût en terme de complexité. Néanmoins, les logiques d'ordre partiel sont généralement encore plus coûteuses (par exemple LTrL [TW02, Wal98, DG04] est non élémentaire). Un fragment naturel de LTrL, qui a des rapports avec les formules de template MSC, est de n'utiliser comme opérateur temporel que F (un jour)

et G (toujours). Ce fragment est aussi en EXPSPACE, [AMP98, Wal98] (il est même EXPSPACE complet).

Une idée est de restreindre un peu les formules de template MSC pour baisser la complexité. La restriction est que la postcondition ne doit contenir que deux boîtes au maximum. Nous allons décrire uniquement le cas qui pose problème, c'est à dire celui où M_g, donné sous forme positive, contient un MSC entouré de deux boîtes sans restriction de type. Ce cas est intéressant également parce qu'il correspond exactement à la *recherche de motif* (*pattern matching*).

Définition 48 *Un MSC M est un* motif *d'un MSC N si et seulement si il existe deux CMSC S, T tels que $N = SMT$* [5].

Nous pensons que cette restriction est sensée puisque même la propriété d'alarme donnée dans la sous section 4.3.1 page 138 utilise uniquement deux boîtes. La complexité du model-checking est alors PSPACE en la taille de la formule, et NLOG dans la taille du système. Cela est exactement la complexité du model-checking de LTL.

Proposition 55 *Soit un automate S, et un template MSC M avec deux boîtes. Tester si $S \models \epsilon \rightarrowtail M$ est dans PSPACE($b_S|M|$).*

Supposons que $M = \gamma P \gamma$, où γ est une boîte sans restriction de type, et P un MSC fini. Nous adaptons l'algorithme de recherche de motif décrit dans [LWZ90] pour les traces de Mazurkiewicz. Intuitivement, nous effectuons une recherche de motif de la projection $P|_p$ sur chaque processus p. Nous supprimons ensuite les occurrences de $P|_p$ qui ne correspondent pas à un MSC motif complet P.

Plus formellement, pour un MSC N et un processus p, nous notons par $N|_p$ la projection des événements de N sur p. L'idée est de calculer pour chaque processus p les positions x_p de $M|_p$ pour lesquelles $P|_p$ est un suffixe, et de vérifier si il existe un nuplet de positions $(x_p)_{p\in\mathcal{P}}$ qui correspond à un MSC motif P. Un motif $P|_p$ est localisé immédiatement **après** son dernier événement (par exemple, dans la figure 4.6, nous avons noté l'endroit où est localisé un motif d'un processus par des boîtes A, B, C, D). De plus, si P n'a pas d'événement sur p, nous considérons qu'il y a un motif $P|_p$ après chaque événement de M sur p.

[5] Notons que la fonction de message de $N = SMT$ restreinte aux événements de M est totale est bijective.

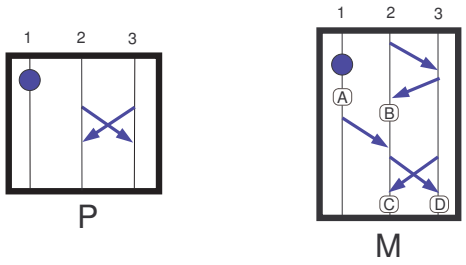


FIG. 4.6 – Le motif P ne se trouve pas dans M.

Définition 49 *Soit x_p, x_q deux occurrences de $P|_p$ et $P|_q$ dans un MSC M. On appelle x_p, x_q compatibles si il n'existe pas de message (s, r) dans M entre p et q avec $x_p < s < r < x_q$ ou entre q et p avec $x_q < s < r < x_p$.*

De plus, si il y a un message envoyé par p et reçu par q dans P, alors il ne doit pas non plus y avoir de message (s, r) dans M entre p et q avec $s < x_p$ et $x_q < r$.

Exemple 48 *Dans la figure 4.6, il n'y a pas d'occurrence de P dans M.*

La boîte A est compatible avec les boîtes B et D. Par contre la boîte A n'est pas compatible avec la boîte C parce qu'il y a un message après A qui est reçu avant C. Notons que A est compatible avec D même si il y a une chaîne de messages entre A et D. Cependant, les boîtes A, B, D ne sont pas deux à deux compatibles parce que B et D ne le sont pas (il y a un message après B et avant D).

La prochaine proposition montre que cette relation de compatibilité des projections deux à deux suffit pour avoir la compatibilité de toutes les projections. Notons qu'il est ici primordial que P concerne tous les processus de M. C'est pour cela que si P n'a pas d'événement sur p, alors nous considérons qu'il apparaît à chaque instant sur p).

Proposition 56 *Un $\wp$-uplet $(x_p)_{p\in\mathcal{P}}$ est une occurrence de P dans un MSC M ssi x_p, x_q sont compatibles pour tout p, q.*

Preuve. L'implication de gauche à droite est triviale. Pour la réciproque, nous raisonnons par l'absurde. Supposons que $(x_p)_{p\in\mathcal{P}}$ n'est pas un motif de P parce que les envois de p à q ne correspondent pas aux réceptions sur q depuis p.

Il y a plusieurs cas à considérer. Soit x_p s'arrete avant que le dernier message m de p à q reçu par x_q ne soit envoyé. Alors le message m est après x_p et avant x_q, ce qui n'est pas possible. Sinon, x_q finit avant la réception du dernier message m de p à q venant de x_p. Alors, ce message m est avant x_p et après x_q, ce qui n'est pas possible d'après la règle additionnelle.

Le dernier cas pour lequel $(x_p)_{p\in\mathcal{P}}$ n'est pas un motif est du à une chaine de messages $(s_k, r_k)_{1\leq k\leq m}$ avec $p_k = P(s_{k+1}) = P(r_k)$, $r_k < s_{k+1}$ pour tout k, tels que $p_1 = p$, $p_{m+1} = q$, $x_p < s_1$, $r_m < x_q$. Ainsi, il existe un k avec $x_{p_k} < s_k$ et $r_k < x_{p_{k+1}}$. Mais cela signifie que $(x_{p_k}, x_{p_{k+1}})$ ne sont pas compatibles, ce qui est également une contradiction, puisque P concerne tous les processus. □

L'idée générale est de générer à la volée les occurrences x_p des motifs $P|_p$ pour tout p, et de calculer le dernier motif sur q avec lequel x_p est compatible. S'il existe q tel que x_p n'est compatible avec aucun x_q, nous oublions x_p. Par exemple, supposons qu'il y ait un message (s, r) entre p et q. De plus, supposons que X_q soit la dernière occurrence sur q avant la réception r. Pour toute occurrence $x_p < s$ sur le processus p, l'occurrence X_q est le dernier motif avec lequel x_p peut être compatible, puisqu'une occurrence $x_q > r$ sur q satisferait $x_p < s < r < x_q$. Dans le cas où X_q n'existe pas, x_p ne peut être compatible avec personne, et nous devons le détruire.

Soit $\mathcal{A}$ l'automate correspondant au produit des automates déterministes recherchant les motifs $P|_p$. Chaque automate est obtenu en appliquant l'algorithme de Knuth-Morris-Pratt.

A partir de $\mathcal{A}$, nous construisons un automate $\mathcal{B}$ qui reconnaît les linéarisations qui ne contiennent pas le motif P. Les états de $\mathcal{B}$ sont de la forme $(a, S, \text{Pattern}, c)$, où

- a est un état de $\mathcal{A}$.
- S est l'ensemble des envois solitaires qui restent à recevoir.
- $\text{Pattern} = \bigcup_p \text{Pattern}_p$, où Pattern_p est l'ensemble des occurrences (non détruites) de $P|_p$ sur le processus p.
- c est la fonction de compatibilité. Pour $x \in \text{Pattern}, p \in \mathcal{P}, c(x, p) \in \text{Pattern}_p \cup \{+\infty\}$ est la dernière occurrence avec laquelle x est compatible. Sa valeur est $+\infty$ si x peut être compatible avec n'importe quelle occurrence sur p.

Nous construisons alors les transitions. Soit e un événement. Alors on a $(a, S, \text{Pattern}, c) \rightarrow_e (a', S', \text{Pattern}', c')$ ssi

1. $a \rightarrow_e a'$ dans $\mathcal{A}$,

2. si $e = p!q$ alors on appelle Create_new_send(e),
3. si $e = q?p$, alors soit $s \in S$ l'envoi associé à e.
   On appelle Update_dependencies(s, e),
4. si $P(e) = p$ et $\mathcal{A}$ reconnaît un motif $P|_p$, alors Create_new_pattern(p).

Create_new_send(e) met à jour S en lui ajoutant e. Nous décrivons la procédure Update_dependencies(s, e). Soit X_q la dernière occurrence sur q avant e. Pour chaque motif x_p sur p avant s, $c'(x_p, q) = \min(c(x_p, q), X_q)$. Si une telle occurrence X_q n'existe pas, alors Pattern$'$ = Pattern $\setminus \{x_p\}$ pour toutes les occurrences x_p sur p avant s. Nous effectuons également l'inverse (p prend la place de q) si il y a un message de p à q dans P. De plus, dans le cas où on détruit une occurrence x_r, on pose X_r la dernière occurrence sur r avant x_r. Nous mettons à jour chaque occurrence x_t telle que $c(x_t, r) = x_r$ avec $c'(x_t, r) = X_r$. Si un tel X_r n'existe pas, nous détruisons toutes les occurrences x_t avec $c(x_t, r) = x_r$. Create_new_pattern(p) rajoute un motif sur p à S.

Proposition 57 *On fixe les états finaux de $\mathcal{B}$ comme étant de la forme $(a, S = \emptyset, Pattern, c)$, avec $Pattern_p = \emptyset$ pour au moins un processus p. Alors $M \in \mathcal{L}(\mathcal{B})$ ssi P n'apparaît pas dans M.*

Preuve. Une induction triviale prouve qu'aucune occurrence détruite ne peut faire partie d'une occurrence de P dans M.

Ainsi, si $(x_p)_{p\in\mathcal{P}}$ est une occurrence de P dans M, alors aucun des x_p ne sera jamais détruit, et ainsi $\mathcal{B}$ n'acceptera pas. Remarquons que dès qu'une occurrence P^0 de $P = (x_p)_{p\in\mathcal{P}}$ est vu, plus aucune occurrence de $P|_p$ après P ne sera détruite, puisqu'elles seront toujours compatible avec au moins P^0. Ainsi, l'algorithme génère le premier P de M, et non toutes les occurrences dans M.

Nous montrons par récurrence que si il n'y a pas d'occurrence de P dans M, alors l'algorithme va trouver un processus avec $\text{Pattern}_p = \emptyset$, et donc accepter l'exécution. Soit Pattern les occurrences non détruites à la fin de l'algorithme. Supposons par l'absurde que pour tout processus p, il existe une occurrence $x_p \in \text{Pattern}_p$. Choisissons x_p minimale dans Pattern_p. Comme $(x_p)_{p\in\mathcal{P}}$ n'est pas une occurrence de P dans M, en appliquant la proposition 56 page 144, nous savons qu'il y a un message (s, r) entre p et q avec $x_p < s < r < x_q$ (les autres cas se traitent de la même manière). Alors dès que r est reçu, nous avons $c(x_p, q) < x_q$. D'après la définition de c, $c(x_p, q)$ ne

grandira jamais. Ainsi, quand $c(x_p, q) < x_q$ sera détruit (ce qui doit arriver puisque x_q est minimale dans Pattern_q à la fin), x_p sera également détruite, ce qui est une contradiction.

□

Nous pouvons facilement restreindre Pattern_p tel qu'il n'ait qu'une seule occurrence entre chaque envoi solitaire de p. En effet, toutes les occurrences qui ne sont pas séparées par des envois solitaires se comporteront de la même manière dans le futur. Ainsi, pour tout couple d'occurrences $x_q < y_q$ sur q non séparées par un envoi solitaire, et pour toute occurrence x_p sur p avec $c(x_p, q) = x_q$, nous mettons à jour $c'(x_p, q) = y_q$, et pour tout processus $r \neq q, c'(x_p, r) = \min(c(x_p, r), c(x_q, r))$. Nous pouvons alors détruire x_q.

C'est à dire que l'algorithme n'a besoin de mémoriser qu'au plus b_S envois solitaires, $b_S + \wp$ occurrences, chacune avec une mémoire de $\wp$ occurrences, et les états de l'automate $\mathcal{A}$. Alors $\mathcal{B}$ est au plus de taille $2^{O(\wp(b_S+\wp+\log(|P|))}$.

Théorème 13 *Soit S un automate, $M_g = \bigvee_i (\pm) M_g^i$ une postcondition avec au plus deux boîtes, et soit M_a un template MSC.*

Tester si $S \models M_a \mapsto \bigvee_i (\pm) M_g^i$ est en PSPACE($b_S(|M_g| + |M_a|)$).

Nous montrons que le problème est PSPSACE-complète très rapidement, excluant la possibilité d'une restriction intéressante de complexité polynomial. C'es presque une constante quand on s'intéresse à des systèmes concurrents : une explosion exponentielle est très rarement évitable, du moins en théorie.

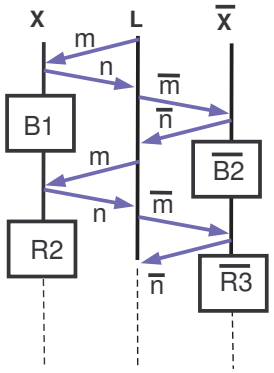
Théorème 14 *Soit S un automate et M_g^i des templates MSC avec au plus deux boîtes. Tester si $S \models \epsilon \mapsto \bigvee_i M_g^i$ est PSPACE-dur. Ce résultat est vrai également si les boîtes sont instanciées par des MSC (à la place des CMSC), si il y a trois processus uniquement, si S accepte toutes les linéarisations plutôt que juste des représentants, et si $b_S = 4$ est une borne universelle.*

Preuve. Nous réduisons $S \models \epsilon \mapsto \bigvee_i M_g^i$ au problème de l'acceptance d'une machine de Turing T d'espace linéaire sur le mot w. Soit N l'espace nécessaire à T sur le calcul de w. La preuve utilise l'idée classique d'un codage d'une machine de Turing d'espace linéaire, comme utilisé dans [MP99, Wal98]. Dans notre cas, nous avons besoin de compteurs de taille logarithmique pour décrire la position sur le ruban de travail, parce que des templates MSC temporels de tailles polynomiales ne peuvent pas servir pour tester l'égalité de deux chaînes de tailles polynomiales.

Soit Σ l'alphabet du ruban de T. Nous définissons un automate S sur trois processus $X, L, \overline{X}$. Il y a des événements locaux $\Sigma_X = \Sigma \cup \{(1,b),(1,r),\ldots,(N,b),(N,r)\}$ sur X et des événements locaux $\Sigma_{\overline{X}} = \overline{\Sigma} \cup \{\overline{(1,b)},\overline{(1,r)},\ldots,\overline{(N,b)},\overline{(N,r)}\}$ sur $\overline{X}$. De plus, il y a un message m de L à X et un message n de X à L. Nous définissons enfin un message $\overline{m}$ de L à $\overline{X}$ et un message $\overline{n}$ de $\overline{X}$ à L.

L'automate génére $(mn(\Sigma \cup \Sigma_{\overline{X}})^* \overline{mn}(\Sigma_X \cup \Sigma_{\overline{X}})^*)^*$. Il est facile de générer toutes les linéarisations de ce système. De plus, la borne universelle sur les canaux de communication est 4.

Considérons une éxécution $\rho = \alpha_0, \alpha_1 \cdots \alpha_k$ de T, dans laquelle chaque α_i est une configuration de taille N. Nous codons ρ avec $mn\overline{mn}B_1\overline{B_2}mn\overline{mn}R_2\overline{R_3}$ $mn\overline{mn}B_3\overline{B_4}\cdots$ comme dans la figure ci dessous. Remarquons que $\overline{B_i}$ et R_i ne sont pas ordonnés.



Nous définissons $B_i = (0,b)\alpha_{i,0}(1,b)\cdots(N,b)\alpha_{i,N}$, ainsi que R_j de la même manière, mais avec r remplaçant b. Nous définissons $\overline{B_j}$ de la même manière que B_j, sauf que $\overline{x}$ remplace x pour tout lettre x.

Nous définissons enfin Φ telle que $S \models \Phi$ ssi il n'y a pas d'exécution acceptante de T sur w. C'est à dire que nous voulons que $\mathcal{L}(\Phi)$ soit exactement l'ensemble des cas qui ne codent pas des exécutions de la machine de Turing, plus les codages des chemins qui ne sont pas acceptants.

Un MSC M ne correspond pas à un chemin acceptant sur w ssi il appartient à l'un des ensembles suivant :

1. S_{shape} est l'ensemble des MSC qui ne sont pas de la forme requise pour $B_i, \overline{B_i}, R_i, \overline{R_i}$.
2. S_{eq} est l'ensemble des MSC pour lesquels il existe i tel que $R_i \neq \overline{B_i}$.

3. S_{succ} est l'ensemble des MSC pour lesquels il existe i tel que $\overline{R_{i+1}}$ ne soit pas une configuration successeur de R_i.
4. S_{init} décrit les MSC pour lesquels B_1 n'est pas la configuration initiale.
5. S_{fin} décrit les MSC pour lesquels B_n n'est pas la configuration finale.

Nous expliquons brièvement comment définir les formules de templates MSC qui décrivent les ensembles précédents.

Il est facile d'exprimer que M n'a pas la bonne forme, puisque c'est un test local. En ce qui concerne les ensembles décrits par S_{eq} et S_{succ}, nous utilisons les compteurs pour identifier les positions. Par exemple, pour S_{eq}, nous définissons :

– γ une boîte sans restriction.
– $M_{eq} = \vee_{i \leq n} \vee_{a \neq a'} \gamma\,(i,b)a(\overline{i},r)\overline{a'}\,\gamma$.

On constate que $\mathcal{L}(\epsilon \mapsto (M_{eq})) = S_{eq}$. Nous pouvons alors définir $\Phi = \epsilon \mapsto (M_{shape} \vee M_{eq} \vee M_{succ} \vee M_{init} \vee M_{final})$.

□

Nous donnons ici un tableau pour résumer la complexité du model-checking des formules de templates MSC. Nous ne l'avons pas décrit, mais une restriction très contraignante permet le model-checking en temps polynomial (voir [GMMP04]).

$S \models M_a \rightarrowtail M_g$	M_a	M_g
PTIME	closed gaps and S complete or one closed gap	± one closed gap, no disjunction
PSPACE	no restriction	negative or ± two gaps
EXPSPACE	no restriction	no restriction

FIG. 4.7 – Complexité du Model-Checking des templates

4.4 Voir plus loin ?

Les détails concernant le model-checking sont assez bien compris désormais avec les résultats que nous venons de décrire. Néanmoins, deux directions restent à développer. La première est de connaître la complexité exacte du model-checking des CMSC-graphes à choix local, qui navigue entre NLOGSPACE et PSPACE en le nombre de processus.

Ensuite, les templates MSC sont un formalisme encore jeune. Ainsi, leur expressivité exacte n'est pas connue. Ils sont strictement inclus dans la logique du premier ordre. En fait, en terme de logique FO, nous pourrions décrire une formule de templates MSC avec une suite d'opérateurs G 'toujours' suivie d'une suite d'opérateurs 'un jour' F. De plus l'opérateur $\mathcal{U}$ d'until n'est permis que très affaibli via les boîtes qui symbolisent des filtres sur les types. Une relaxation de ces contraintes pour connaître la complexité dans le cas où plus d'expressivité est possible serait très intéressante.

De plus, les templates MSC permettent de spécifier des protocoles complets (voir [GMMP04]). Ils seraient alors à comparer avec les triggered MSC de [SC02], avec qui ils partagent de nombreuses possibilités, tout en rajoutant toute la puissance des boîtes. De même, les Live Sequence Charts (LSCs) de [DH99] sont un autre formalisme. Cependant, la sémantique est très différente : Alors que les LSCs permettent de très nombreuses possibilités, ils sont très restreints par leur sémantique synchrone.

Chapitre 5

Heuristique pour réduire la taille des modèles

Hmmmm, donuts !
Homer

Comme nous l'avons remarqué dans le chapitre précédent, la complexité du model-checking des modèles concurrents est souvent très coûteuse [HKV97]. Obtenir une complexité polynomiale se fait très souvent au détriment de l'expressivité du modèle ou de la spécification (MSC-graphes à choix local ou template MSC à une seule boîte).

Plutôt que de s'attaquer à baisser la complexité en imposant des contraintes aux modèles ou spécifications, nous allons nous intéresser dans cette section à réduire la taille des modèles. Les deux approches proposées ici sont d'abord d'abstraire une partie du modèle. Il est en effet judicieux d'oublier les parties du modèle qui n'apparaissent pas explicitement dans la spécification à satisfaire. Néanmoins, ces parties à oublier peuvent tout de même avoir un impact sur le reste du modèle (et en particulier les parties apparentes dans la spécification), en ce qu'elles peuvent créer des dépendances ou non entre les parties appartenant à la spécification. Cette approche est souvent utilisée en vérification, et les erreurs qu'elle peut entraîner sont souvent considérées comme négligeables, parce qu'elles peuvent être repérées. Ainsi, une nouvelle abstraction peut être fabriquée puis vérifiée, sans les erreurs d'approximations trouvées précédemment. Si cette méthode peut très bien être appliquée aux systèmes concurrents (comme à d'autres), nous nous intéressons ici plutôt à une reconstruction complète du modèle sans les parties superflues pour

la spécification. Evidemment, l'utilisation simultanée de ces deux techniques (oublier des parties vraiment superflues et reconstruire un modèle sans les parties plus problématiques) donneraient sans doute les meilleurs résultats en pratique.

La seconde approche est de considérer des modèles compressés. Plutôt que de les compresser nous mêmes (comme dans [ACE+03]), nous allons considérer que le modèle est déjà donné sous forme compressée. Si cela parait étrange, c'est le cas de beaucoup de gros systèmes, en ce qu'ils réutilisent souvent les mêmes briques de base (description par macros, ou encore hiérarchique). Ne pas redéfinir plusieurs fois les mêmes macros est ainsi une sorte de compression (très proche en fait de LZW) puisque cela permet de diminuer la taille de la description du système. Plutôt que de 'décompresser' le modèle avant de le traiter, nous montrons qu'il est souvent plus judicieux de conserver l'information de redondance. En effet, déplier le modèle avant de le traiter n'est pas très malin parce que les mêmes calculs risquent d'être effectués plusieurs fois. Ainsi, l'information de redondance sert souvent à économiser (beaucoup) de temps de calcul.

5.1 Projection

Considérons un MSC $M = (\mathcal{P}, E, \mathcal{C}, \lambda, m, <)$ et un sous ensemble d'événements $E' \subseteq E$. Nous définissons la *projection* de M sur E', notée $\pi_{E'}(M)$, en tant qu'un ordre partiel $(E', \leq', \lambda')$, avec $\leq' = \leq |_{E'}$, et $\lambda' = \lambda|_{E'}$. C'est à dire que l'on oublie les événements de $E \setminus E'$, et que l'ordre sur les événements restants est hérité de celui de M. Par exemple, on pourra choisir E' comme l'ensemble des événements situés sur un sous ensemble $\mathcal{P}' \subseteq \mathcal{P}$ de processus. Les événements de E' seront appelé événements **restants**, et les événements de $E \setminus E'$ **effacés**.

Nous définissons une fonction m' sur E', analogue de la fonction de message, avec $m'(e) = f$ si $e, f \in E'$ et $m(e) = f$, ou bien si $e \lessdot' f$ et $P(e) \neq P(f)$. C'est à dire que si e et f sont des événements sur des processus différents, si $e < f$ sont ordonnés dans M et s'il n'y a pas d'événement restant entre e et f (mais il peut tout de même y avoir des événements effacés), alors $m'(e) = f$. En définissant une fonction de type λ' sur E', on obtient un MSC $(\mathcal{P}, E', \mathcal{C}, \lambda', m', < |_{E'})$, que nous appelons une projection d'un MSC, ou *pMSC*, voir figure 5.1.

Remarquons qu'il se peut que $\pi_{E'}(M)$ ne corresponde pas au pomset

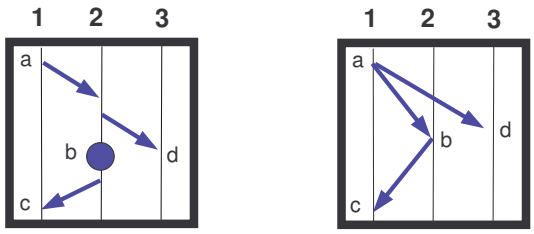


FIG. 5.1 – a) un MSC b) projection sur $\{a, b, c, d\}$

associé à un MSC. En effet, un événement peut être ordonné par rapport à deux événements sur des processus différents (voir l'élément b sur la figure 5.1), et ainsi tenir le rôle de plusieurs envois et receptions. Nous appellerons de tels événements des événements multitypes. Les événements multitypes ne sont pas un réel problème, car nous pourrions aussi bien définir les MSC avec ce genre de possibilités. Nous supposerons par simplicité que ce genre de cas n'apparaît pas.

Nous définissons de manière analogue la projection d'un langage de CMSC-graphe $G = (V, \rightarrow, v^0, v^F, \lambda)$ sur E, notée $\pi_{E'}(\mathcal{L}(G)$, et définie comme l'ensemble des projection de chaque MSC de $\mathcal{L}(G)$ sur toutes les occurrences des événements de E'. Nous considérons que tous les noeuds ont des noms d'événements disjoints, et on note E leur ensemble. Soit $E' \subseteq E$. Alors on peut définir aisément $\pi_{E'}(\mathcal{L}(G))$ comme étant $\{\pi_{E'}(\lambda(v_1) \cdots \lambda(v_k)) \mid v_1 \rightarrow \cdots \rightarrow v_k$ est un chemin acceptant de $G\}$.

5.1.1 Projections et Sûreté

Si nous voulons avoir des algorithmes sur les projections des langages, ou bien même juste comprendre les dépendances dans un tel langage, alors nous devons représenter ce langage sous une forme de CMSC-graphe. En fait, nous avons montré dans [GHM03] que les projections de langages de CMSC-graphes sûrs sont des langages de CMSC-graphes sûrs. Nous rappelons que nous considérons qu'il n'y a pas d'événements multitypes. Notons que si il y avait de tels événements, alors nous pourrions aisément adapter les algorithmes à une extension des CMSC-graphes sûrs.

Nous savons que les projections de langages de MSC-graphes sont plus ex-

pressives que les MSC-graphes. Par exemple, nous pouvons décrire le CMSC-graphe à gauche de la figure 5.2 par la projection sur les processus $\{A, B\}$ du MSC-graphe au milieu de la figure 5.2.

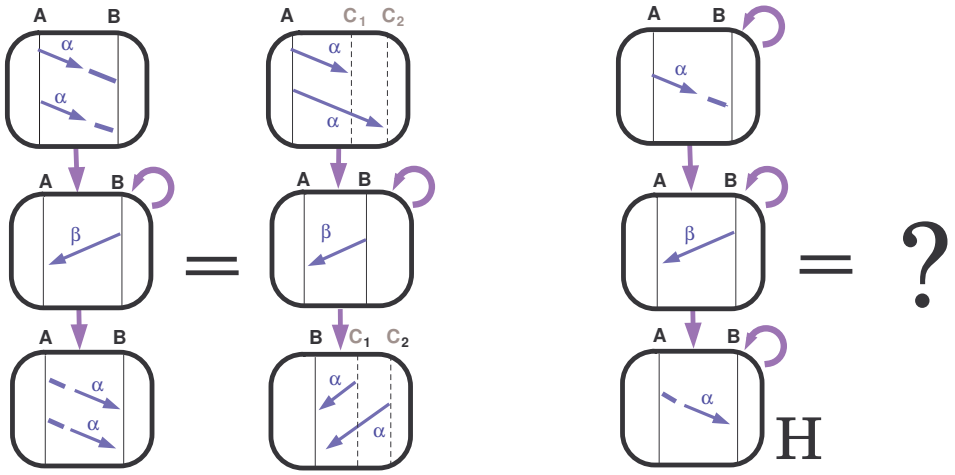


FIG. 5.2 – Comment décrire le CMSC-graphe de droite avec une projection d'un MSC-graphe ?

La partie droite de la figure 5.2 montre un CMSC-graphe non sûr (car non existentiellement borné) qui ne peut pas être décrit par la projection d'un MSC-graphe (ni d'un CMSC-graphe sûr d'ailleurs). Intuitivement, prenons le CMSC-graphe qui décrit des envois solitaires α, qui sont reçus après un nombre arbitraire de messages β. Pour décrire cela avec une projection de MSC-graphes, il faut pour chaque envoi un processus effacé différent (voir le dessin au milieu de la figure 5.2). Ainsi, si il y a un nombre arbitraire d'envois solitaires, alors il semble impossible d'avoir une projection de MSC-graphe associée.

Nous montrons maintenant que nous pouvons exprimer la projection d'un langage de MSC-graphe par un CMSC-graphe sûr.

Soit M un MSC avec un ensemble d'événements E. On pose $M' = \pi_{E'}(M)$ pour un $E' \subseteq E$ fixé. Nous devons deviner le nouveau type des événements $e \in M'$ (envoi vers quel processus, réception depuis quel processus). Par exemple, pour le MSC-graphe projeté de la figure 5.2, nous devons deviner que le premier événement α doit être un envoi vers B. Afin de vérifier que

tout a été deviné correctement, nous devons nous souvenir des processus apparaissant dans le futur de tout événement restant e. Parmi ces processus, nous devons connaître les processus vus par un événement restant $e' > e$. Nous appellerons de tels processus **morts** pour e.

Si $p \in \text{DeadF}(e)$, alors nous savons que e ne sera jamais un envoi vers p. Par contre, pour $p \in \text{LiveF}(e)$, nous n'avons aucune information.

Ainsi, nous définissons pour chaque élément restant $e \in E'$:

$$\begin{aligned} F(e) &= \{P(f) \mid e \leq f \in E\} \quad \text{(processus dans le futur de } e\text{)} \\ \text{DeadF}(e) &= \bigcup_{e<e' \in E'} F(e') \quad \text{(processus morts pour } e\text{)} \\ \text{LiveF}(e) &= F(e) \setminus \text{DeadF}(e) \quad \text{(processus vivants pour } e\text{)} \end{aligned}$$

Un événement restant $e \in E'$ est appelé **à finir** si $\text{LiveF}(e) \neq \emptyset$. Sinon, il sera appelé **testé**. Intuitivement, quand un événement e est créé, il est à finir, et son type est deviné. Il ne devient testé que lorsque nous avons la preuve qu'il ne peut plus être prédécesseur immédiat d'un événement f avec $P(e) \neq P(f)$. Dès qu'un événement est testé, on peut donc l'oublier. Si on s'aperçoit que le type d'un événement a été mal deviné, alors on arrête le chemin actuel. Supposons avoir déjà vu deux événements restants $e < e'$ avec $p \in F(e')$. Alors e ne peut plus être immédiatement avant un événement restant à venir sur p, donc nous connaissons son type vis à vis du processus p (il ne peut plus être un envoi vers une réception à venir sur p).

L'ensemble des événements à finir après avoir lu le CMSC M' est noté $\text{ToCheck}(M')$. Le lemme suivant borne la taille de $\text{ToCheck}(M')$ en fonction du nombre de processus, et indépendamment de la taille de M'.

Lemme 11 *Soit ρ un chemin initial d'un MSC-graphe H, et M' le pMSC défini en projetant le MSC étiquetant ρ. Alors $|\mathit{ToCheck}(M')| \leq \wp^2$.*

Preuve. Pour tout $e < e'$ de M' avec $P(e) = P(e')$, nous avons $\emptyset \neq F(e') \subseteq F(e) \subseteq \mathcal{P}$. Ainsi, il y a au plus $\wp$ événements sur le même processus avec des $F(e)$ différents.

Supposons par l'absurde qu'il y ait strictement plus de $\wp^2$ événements à finir. Alors il y a deux événements e, e' de E' sur le même processus $P(e) = P(e')$, tel que $F(e) = F(e')$. Par symétrie, on peut supposer que $e < e'$, et donc $\text{LiveF}(e) = \emptyset$. Une contradiction. □

Soit G un MSC-graphe et E' un sous ensemble des événements des noeuds

de G. Nous construisons maintenant un CMSC-graphe sûr G' dont le langage est $\mathcal{L}(G') = \pi(\mathcal{L}(G))$.

A l'arrivée de chaque événement e sur le processus p, on le place dans $e \in U$, c'est à dire qu'il est à finir. Nous savons d'après les événements du passé si c'est une réception ou non. Nous allons deviner si c'est un événement local ou bien un envoi vers un certain processus q. Nous conserverons cette information jusqu'à ce que e soit testé, grâce à une fonction to : $U \to \mathcal{P}$. Nous définissons $\text{to}(e) = p$ si e est deviné être un événement local, et $\text{to}(e) = q$ si e est deviné être un envoi vers le processus q.

Les noeuds de G' sont associés à un chemin initial $\rho = (v_0 \to v_1 \to \cdots \to v_k)$ de H étiqueté par M, et on pose $M' = \pi(M)$. Le lemme 11 page précédente impliquera que $2^{\wp^3}$ noeuds sont suffisants. Pour chaque chemin ρ, nous créons un noeud $[v, U, \text{to}, \text{LiveF}, \text{DeadF}]$ pour chaque étiquetage to : $U \to \mathcal{P}$, avec la sémantique suivante :

1. $v = v_k$ est le dernier noeud de ρ,
2. $U = \text{ToCheck}(M')$ est l'ensemble des événements à finir de M',
3. LiveF (DeadF, resp.) est une application qui à $e \in U$ associe l'ensemble des processus vivants et morts pour e respectivement.
4. Si $\text{to}(e) \neq P(e)$, alors e est supposé être un envoi solitaire vers le processus $\text{to}(e)$. Sinon $\text{to}(e) = P(e)$, et e est supposé être un événement local, ou e est un envoi déjà reçu.

Un noeud $x = [v, U, \text{to}, \text{LiveF}, \text{DeadF}]$ est étiqueté par un CMSC M' défini sur les événements restants étiquetant le noeud v. Il reste à définir le type des événements de M'. Nous déterminons toutes les paires d'événements restants $(e, f) \in E' \times E'$ telles que $m(e) = f$ et les définissons comme messages. Chaque événement $f \in E'$ tel que $e \lessdot f$ pour un événement restant e de ρ est une réception solitaire depuis le processus $P(e)$. Enfin, chaque événement à finir $e \in U$ est soit un envoi solitaire vers $to(e)$ (si $\text{to}(e) \neq P(e)$) ou un événement local.

Les noeuds initiaux de G' sont ceux correspondant à un chemin initial $\rho = (v_0)$ composé d'un seul état. Les états finaux $x = [v, U, \text{to}, \text{LiveF}, \text{DeadF}]$ sont ceux associés avec un chemin acceptant de G et tels que $to(e) = P(e)$ pour tout e, c'est à dire que tous les événements à finir doivent être des événements locaux.

Il reste à définir les transitions entre les noeuds de G'. Soit x, y des noeuds de G'. Alors $x \to y$ si

1. Il existe un chemin initial ρ étiqueté par un certain M et un noeud u de G tels que x est associé à ρ et y à ρu.
2. On pose $M' = \pi(M)$.
3. Pour tout $e \in \text{ToCheck}(M')$ avec $\text{to}(e) = P(e)$ (e est supposé être un événement local ou a déjà été reçu), aucun événement f restant de u ne satisfait $m'(e) = f$.
4. Pour tout $e \in \text{ToCheck}(M')$ avec $\text{to}(e) \neq P(e)$ (e est supposé être un envoi vers $\text{to}(e)$), aucun événement f restant de u ne satisfait $m'(e) = f$ avec $P(f) \neq \text{to}(e)$. De plus, on demande que $to(e) \notin \text{DeadF}'(e)$.

Si une de ces conditions est fausse, alors un type a été mal deviné, et le noeud ne pourra pas atteindre un état final. Nous détruisons alors cet état pour obtenir un CMSC-graphe sûr.

La construction précédente donne :

Théorème 15 *Soit G un CMSC-graphe sûr, $b = b_G$, et considérons une projection π. Alors nous pouvons construire un MSC-graphe sûr G' avec $\mathcal{L}(G') = \pi(\mathcal{L}(G))$, et de taille $|G|(2^{O(b+\wp^3)})$. De plus, G' est existentiellement $b + \wp^2$ borné.*

Preuve.

Si G est un MSC-graphe, le lemme 11 page 155 implique que G' est existentiellement $\wp^2$ borné.

Comme il y a au plus $\wp^2$ événements dans U, il y a au plus $2^{O(\wp^2)}$ fonctions to différentes, et $2^{O(\wp^3)}$ fonctions $\text{LiveF}(e)$ différentes.

Si G est un CMSC-graphe, nous devons en plus nous souvenir des envois restants qui sont encore solitaires. Si une réception solitaire (restante ou effacée) reçoit un envoi solitaire restant, alors cet envoi n'est plus solitaire). Il y a au plus b tels envois solitaires, qui augmente d'autant la borne existentielle. De plus, il y a au plus 2^b ensembles d'envois solitaires différents.

Notons que si nous autorisons des événements multitypes, alors to associe à e un sous ensemble de $\mathcal{P}$ et non un élément de $\mathcal{P}$, et alors le nombre de fonction to différentes devient $(2^{O(\wp^3)})$, le reste de l'algorithme restant inchangé.

Vu la construction de G', tous les noeuds qui mènent à un état final sont étiquetés par des MSC, puisqu'un envoi deviné solitaire vers q doit avoir été reçu dans un autre noeud. De plus, les noeuds ne menant pas à des noeuds finaux (car les types ont été mal devinés) ont été supprimés. Ainsi G' est sûr. □

En fait, la réciproque est aussi vraie.

Théorème 16 *Les projections de langages de MSC-graphes (ou de CMSC -graphes sûrs) ont le même pouvoir expressif que les CMSC-graphes sûrs.*

Preuve. Soit G un CMSC-graphe sûr et $\exists$-b-borné. Nous définissons un MSC-graphe H sur les processus $\mathcal{P} \cup \{(p,q,k) \mid p,q \in \mathcal{P}, 1 \leq k \leq b\}$, et une projection π telle que $\pi(\mathcal{L}(H)) = \mathcal{L}(G)$.

Comme G est sûr, on peut associer chaque envoi solitaire e de p à q à un entier $b(e) \leq b$ tel que chaque CMSC préfixe d'un MSC dans $\mathcal{L}(G)$ ayant e pour élément maximal contienne $b(e)$ envoi solitaires de p à q (en comptant e).

La projection efface les événements sur les processus auxiliaires (p,q,k). Il y a un noeud v' dans H par noeud v de G. Soit M étiquetant v. On définit l'étiquette M' de v comme suit. Les messages de M sont conservés dans M'. Chaque envoi solitaire $e \in M$ de p à q est remplacé par un message de p au processus additionnel $(p,q,b(e))$. Chaque réception solitaire sur q de p est remplacée par une suite de message de $(p,q,1)$ à q, puis de $(p,q,2)$ à $(p,q,1)$, et ainsi de suite jusqu'à un message de $(p,q,b(e))$ à $(p,q,b(e)-1)$. Notons que dans chaque cas, un événement solitaire e correspond à un événement sur le processus original de $\mathcal{P}$. Nous devons uniquement montrer que les fonctions de messages m, m' sont les mêmes. Soit e un envoi solitaire de G. Après avoir reçu exactement $b(e)-1$ message dans G sur q depuis p, nous avons $(p,q,1) \in F(e)$. Alors quand une autre réception f sur q de p arrivera, nous aurons dans H un message de $(p,q,1)$ à la réception f, et nous aurons bien $e \lessdot f \in \pi(H)$, donc $m'(e) = f = m(e)$.

□

Grâce au théorème 15 page précédente, nous savons que nous pouvons tester si la projection d'un CMSC-graphe sûr G :

- satisfait une formule MSO (complexité non élémentaire),
- satisfait une formule φ de template MSC à deux boites. La complexité est PSPACE en $\wp + b_G + \varphi$ et linéaire dans la taille du système.
- a une intersection vide avec un CMSC-graphe H globalement coopératif. La complexité est PSPACE en $|H| + b_G$.
- est inclus dans un CMSC-graphe H globalement coopératif. La complexité est EXPSPACE en $|H| + b_G$.

5.1.2 Caractérisation des MSC-graphes

Dans [HZJ03], les auteurs montrent que projeter des MSC-graphes peut permettre de faire apparaître des informations cachées, telles des canaux cachés de communication. Afin de comprendre mieux ces structures, une représentation sous forme de MSC-graphe est préférable, même si elle n'est pas toujours possible.

Ainsi, nous allons résoudre la question de savoir tester si un CMSC-graphe sûr est équivalent à un MSC-graphe, et construire ce MSC-graphe quand c'est le cas, comme nous l'avons décrit dans [GHM03].

De plus, la question s'est déjà posée de savoir caractériser les automates communicants qui sont équivalents à des MSC-graphes [MP01]. Comme ces automates communicants doivent être existentiellement bornés (puisque les MSC-graphes le sont), la question est incluse dans la question de savoir caractériser les CMSC-graphe sûrs qui sont équivalents à des MSC-graphes.

Quand on travaille avec les MSC-graphes, une notion cruciale est celle des atomes, qui définissent une brique de base de la construction des MSC-graphes, comme les triangles sont la brique de base des CMSC-graphes à choix local. De ce point de vue, les algorithmes vont être assez similaires à ceux pour les CMSC-graphes à choix local.

Pour l'instant, nous n'avons pas trop parlé des atomes car toutes nos preuves parlent de CMSC-graphes et non de MSC-graphes, structures pour lesquelle les atomes n'ont que peu de rôle. Historiquement, les atomes étaient très utilisés parce que les théorèmes étaient restreint aux MSC-graphes [HM00, MP01, HMNK$^+$04, Mor02, GMSZ02].

Nous allons tout d'abord énoncer quelques résultats nécessaires concernant les atomes, que l'on peut trouver dans les papiers qui les ont introduits, [HM00, HMNKT00a]. On rappelle tout d'abord qu'un MSC M est un *atome* si il n'existe pas de MSC $S, T \neq \epsilon$ avec $ST = M$.

Bien évidemment, cette définition donne implicitement un algorithme sous optimal pour tester si un MSC est un atome. Voici l'algorithme connu le plus performant pour tester si un MSC est un atome, et pour le décomposer en MSC atomiques.

Définition 50 *[HM00] Soit M un MSC. Le* graphe atomique $G^a = (V, \rightarrow)$ *associé à M est défini par V est l'ensemble des événement de M et $v_1 \rightarrow v_2$ ssi $v_1 < v_2$ ou si $m(v_2) = v_1$.*

C'est à dire que le graphe est composé de l'ordre du MSC, plus des arcs

retour de chaque message. Avec un tel graphe, on peut montrer la propriété suivante.

Proposition 58 *[HM00] La décomposition d'un MSC M en atomes s'obtient en calculant les composantes fortement connexes du graphe atomique G^a associé à M. Chaque atome de la décomposition de M correspond à un élément de la composante fortement connexe, et ses événements sont les événements de la composante fortement connexe.*

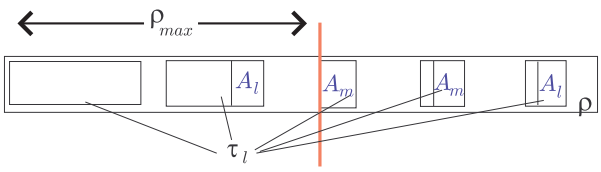
Notons que la décomposition d'un MSC en MSC atomiques est unique à commutation des atomes près. En fait, la décomposition atomique des MSC a servi de base à un alphabet de trace dont les éléments sont les atomes, et la relation de commutation est $(A_1, A_2) \in I$ ssi $P(At_1) \cap P(At_2) = \emptyset$, voir [HMNK$^+$04, GMSZ02, Mor02].

Dans la suite, et comme pour les CMSC-graphes à choix local, nous allons montrer qu'un CMSC-graphe sûr G est équivalent à un MSC-graphe si et seulement si il existe un n tel que $G \subseteq H_n$ pour le MSC-graphe générique H_n que nous définissons maintenant. On pose At_n l'ensemble des atomes de taille au plus n. On pose également H_n le MSC-graphe qui possède, pour chaque atome $A \in At_n$, un noeud v_A étiqueté par A. De plus, il y a une transition entre chaque paire de noeuds (le graphe est une clique).

Proposition 59 *Un CMSC-graphe sûr G est équivalent à un MSC-graphe ssi il existe un entier n tel que $\mathcal{L}(G) \subseteq \mathcal{L}(H_n)$. Si c'est le cas, on peut obtenir un MSC-graphe équivalent à G, de taille exponentielle en b_G et n.*

On pose $b = b_G$. Remarquons que si G est globalement coopératif, alors nous pouvons utiliser la même preuve que la proposition 39 page 110 pour les CMSC-graphes à choix local. Ainsi, il suffirait de calculer un automate acceptant $Lin^b(G)$ puis faire le produit avec H_n pour obtenir un MSC-graphe équivalent à G. Nous allons démontrer le résultat même si $Lin^b(G)$ n'est pas régulier, c'est à dire même si G n'est pas globalement coopératif. Nous pouvons calculer une sorte de clôture atomique de G, plutôt que la clôture par commutation. Nous utilisons en fait la même stratégie que la preuve de [MP99] du théorème d'Ochmanski (voir propriété 6 page 33). Nous utilisons l'automate $\mathcal{B}$ qui calcule les *chemins à trous* de G (voir propriété 7 page 35).

Supposons que G est $\exists$-b-borné et $G \subseteq H_n$. Nous construisons le produit synchronisé de l'automate $\mathcal{B}$ qui calcule les chemins à $(b+1)n$ trous de G et de H_n, obtenant un MSC-graphe H.



Par définition de $\mathcal{B}$, nous avons $\mathcal{L}(H) \subseteq \mathcal{L}(G)$. De plus, si ρ est un chemin acceptant de G étiqueté par M, alors on pose $M = A_1 \cdots A_m$ la décomposition de M en atomes de At_n. De plus, nous réordonnons $A_1 \cdots A_n$ tels que $i \leq j$ s'il existe $e \in A_i$, $f \in A_j$ avec $e < f$ (nous n'avons pas le choix pour cette règle), et sinon si le premier événement de A_i est apparu avant le premier événement de A_j dans ρ.

Lemme 12 *Soit $l \leq m$ et τ_l le sous chemin à trous de ρ étiqueté par $A_1 \cdots A_l$. Alors τ_l a au plus $(b+1)n$ trous.*[1]

Preuve.

On choisit $max \leq l$ tel que le chemin (sans trou) ρ_{max} préfixe de ρ qui s'arrête avant le premier événement de A_{max} soit le plus long.

Le CMSC étiquetant ρ_{max} a au plus b envois solitaires puisque c'est un chemin de G, parmi lesquels b' appartiennent à ρ_l. Ainsi, $b'' = b - b'$ est le nombre d'envois solitaires dans $\rho_{max} \setminus \tau_l$ (la différence de deux chemins est obtenus en enlevant du premier chemin les noeuds du second).

Nous affirmons d'abord qu'il n'y a pas de MSC atomique A_p complètement dans $\rho_{max} \setminus \tau_l$. Sinon, A_p satisferait $p > l \geq max$ puisqu'il n'est pas dans τ_l. De plus A_p apparaît entièrement dans ρ avant qu'un seul événement de A_{max} ne soit apparu (par choix de max), ce qui contredit le choix de l'ordre sur les A_i.

Comme tout atome incomplet de $\rho_{max} \setminus \tau_l$ contribue d'une unité au moins pour le nombre d'envois solitaires de ρ_{max}, il y a au plus b'' atomes incomplets dans $\rho_{max} \setminus \tau_l$. Comme chaque atome incomplet a une taille d'au plus $n-1$, il y a au plus $(n-1)b''$ événements dans $\rho_{max} \setminus \tau_l$, c'est à dire au plus $(n-1)b''+1$ trous dans $\tau_l \cap \rho_{max}$.

De plus, par définition de max il y a exactement un atome complet, qui est A_{max}, dans $\tau_l \setminus \rho_m$. Les autres atomes de $\tau_l \setminus \rho_{max}$ sont incomplets, et

[1] Remarquons qu'un sous chemin correspondant à une autre permutation de $A_1 \cdots A_n$ pourrait avoir plus de trous, voir un nombre non borné de trous.

contribuent d'une unité au moins pour le nombre de réceptions solitaires de $\rho \setminus \rho_{max}$, qui est égal à b'. Ainsi, il y a au plus $b'+1$ différents atomes dans $\tau_l \setminus \rho_{max}$. Cela donne au plus $(b'+1)n$ événements dans $\tau_l \setminus \rho_{max}$, et donc $(b'+1)n$ trous. Ainsi, nous concluons que τ_l contient au plus $(b'+b''+1)\cdot n = (b+1)\cdot n$ trous.

□

Ainsi il existe un chemin dans $\mathcal{B}$ étiqueté par $M = A_1 \cdots A_n$. Nous avons trivialement un chemin dans H_n étiqueté aussi par $A_1 \cdots A_n$, et donc $M \in \mathcal{L}(H)$ montre que H est équivalent à G.

Exemple : Soit G un CMSC-graphe a 4 noeuds v_1, v_2, v_3, v_4, étiquetés par s, s, r, r, où s est un envoi du processus 1 au processus 2, et r la réception associée.

Il y a des transitions de v_1 à v_2, de v_2 à v_3, de v_3 à v_2, et de v_3 à v_4. Soit A le seul atome de G, constitué du message (s, r). Ainsi chaque noeud de H est étiqueté par A (ou par ϵ). Nous décrivons les états utiles de H :

- s_1 est l'état $[v_1, \{1\}]; [v_3, \{2\}]$, constitué de deux chemins (ou un chemin à un trou), un de v_1 à v_1 avec le processus 1, et l'autre de v_3 à v_3 avec le processus 2,
- s_2 est l'état $[v_1, \{1\}]; [v_4, \{2\}])$,
- s_3 est l'état $([v_1, v_3, \{1, 2\}]; [v_3, \{2\}])$ constitué de deux chemins, un de v_1 à v_3, l'autre de v_3 à v_3,
- s_4 est l'état final $([v_1, v_4, \{1, 2\}])$.

Nous avons par exemple une transition de $(\epsilon, \emptyset)$ vers s_1 et une vers s_2. Cependant, s_2 correspond à un chemin à trou qui n'arrivera jamais à l'état final s_4. La raison est que nous ne pouvons pas l'étendre par A : le segment $[v_4, \{2\}]$ interdit d'utiliser le noeud v_3, puisqu'il contient le processus 2. Il y a aussi des transitions de s_1 à s_3, de s_3 à s_4, de s_1 à s_4 et une boucle sur s_3. La boucle sur s_3 correspond au chemin $[v_2, \{1\}]; [v_3, \{2\}]$ étiqueté par A, et à la connexion par $[v_2, \{1\}]$ entre $([v_1, v_3, \{1, 2\}]$ et $[v_3, \{2\}])$, alors que $[v_3, \{2\}]$ est disjoint.

5.1.3 Les Atomes sont Réguliers

Néanmoins, comme pour le cas des CMSC-graphes à choix local, la propriété 59 page 160 ne nous donne pas d'algorithme pour tester si un CMSC-graphe sûr est équivalent à un MSC-graphe, puisque nous ne connaissons pas à priori de borne sur le n tel que $\mathcal{L}(G) \subseteq \mathcal{L}(H_n)$.

Pour G un CMSC-graphe, on pose Gen(G) l'ensemble des atomes N_1 tels qu'il existe N_0, N_2 avec $N_0N_1N_2 \in \mathcal{L}(G)$. Notons que nous ne demandons pas que N_0, N_2 soient des MSC. Il est possible ainsi que N_1 ne fasse jamais partie de la décomposition atomique d'un MSC de $\mathcal{L}(G)$. Néanmoins, N_1 est inclus dans un atome de la décomposition atomique d'un MSC de $\mathcal{L}(G)$. De plus, tous les atomes de la décomposition atomique des MSC de $\mathcal{L}(G)$ appartienne à Gen(G). Ainsi, les atomes de la décomposition atomique des MSC de $\mathcal{L}(G)$ sont de taille bornée par b si et seulement si les atomes de Gen5g) sont de taille b bornée.

Nous allons montrer que si G est sûr, nous pouvons donner un automate acceptant des représentants de Gen(G). Nous allons donc pouvoir tester facilement si Gen(G) est fini ou non, c'est à dire s'il existe une borne n tel que $\mathcal{L}(G) \subseteq \mathcal{L}(H_n)$.

L'idée est simple : nous allons définir un automate non déterministe $\mathcal{A}(G)$ qui va, pour tout MSC $M \in \mathcal{L}(G)$, deviner un facteur N_1 de $M = N_0N_1N_2$ et vérifier que N_1 est un atome. Pour s'assurer que l'automate devine un facteur, chaque événement e de M est associé à un nombre $k \in \{0, 1, 2\}$, signifiant si $e \in N_0$ ($k = 0$), $e \in N_1$ ($k = 1$) ou $e \in N_2$ ($k = 2$) De plus, nous testons si N_i est bien un MSC et non un CMSC. L'automate $\mathcal{A}(G)$ générera uniquement les événements qui ont été devinés être dans N_1.

Le partie la plus dure est que $\mathcal{A}(G)$ doit tester si M est un atome. La proposition 58 page 160 nous donne un algorithme pour tester si N_1 est atomique en calculant si son graphe atomique $G^a(M)$ est fortement connexe.

Cependant, nous devons tester cette propriété à la volée avec un automate fini. Or le graphe $G^a(M)$ n'a aucun raison d'être borné, ni son nombre de composantes connexes intermédiaires. Nous changeons donc cette propriété légèrement :

Nous disons qu'un événement e de M **voit** un autre événement f si et seulement si il existe un chemin de e à f dans $G^a(M)$. La proposition 58 page 160 se traduit donc par M est un atome si et seulement si tout les événements de M voit tous les événements de M. Cependant, nous savons d'office que tout élément voit les éléments de son futur sur le même processus. Ainsi, nous pouvons affaiblir un peu la proposition 58 page 160.

Proposition 60 *Soit un MSC M. Alors M est un atome ssi pour chaque paire de processus p, q, le dernier événement de p voit le premier événement de q.*

Ainsi, pour tout processus p, nous allons conserver en mémoire son dernier événement last_p de type $k = 1$ (c'est à dire dans N_1). De plus, nous devons conserver les envois solitaires pour reconstruire l'ordre associé à $G^a(M)$. Nous savons qu'il existe des représentants de G tels que l'ensemble $\mathcal{S}$ des envois solitaires est borné puisque G est sûr.

Soit $X = \{\text{last}_p \mid 1 \leq p \leq \wp\} \cup \mathcal{S}$. A chaque paire $(x, p) \in X \times \{1, \ldots, \wp\}$, nous associons un entier $T(x, p) \in \{0, 1, 2\}$ qui indique

- $T(x, p) = 2 : x$ voit le premier événement de N_1 sur p.
- $T(x, p) = 1 : x$ voit un événement de N_1 sur p (mais pas le premier).
- $T(x, p) = 0 : x$ ne voit aucun événement de N_1 sur p.

Ainsi, $\mathcal{A}(G)$ utilise une fonction de vision $T : x, p \in X \times \mathcal{P} \mapsto T(x, p)$, et une fonction $S : x \in X \mapsto S(x)$ où $S(x) \subseteq \mathcal{S}$ est l'ensemble des envois solitaires vus par x.

D'après la proposition 60 page précédente, calculer la fonction de vision de chaque événement last_p suffit pour décider si N_1 est atomique.

Proposition 61 *Soit G un CMSC-graphe sûr. Alors nous pouvons construire un automate $\mathcal{A}(G)$ acceptant des représentants de Gen(G).*

Preuve. Soit $\mathcal{B} = (V, \rightarrow, I, F)$ un automate acceptant $L(G)$, de taille linéaire en G. Nous définissons maintenant l'automate $\mathcal{A}(G)$. Chaque état de $\mathcal{A}(G)$ est de la forme $[v, T, S, k, X]$, où v est un état de $\mathcal{B}$, $X = \{\text{last}_p \mid 1 \leq p \leq \wp\} \cup \mathcal{S}$ est l'ensemble défini plus haut (il est au plus de taille $\wp + b_G$), et T, S, k sont les fonctions de vision, d'envois solitaires et de type. Quand x voit pour la première fois y, nous mettons à jour T', S' en appelant Update(x, y), qui fixe $S'(x) = S'(x) \cup S'(y)$ et $T'(x, p) = \max((T'(y, p), T'(x, p))$ pour tout p.

Nous avons $[v, T, S, k, X] \xrightarrow{\ell} [v', T', S', k', X']$, si il existe e avec $v \xrightarrow{e} v'$ et $p = P(e)$, et que X', S', k', T' sont obtenus par l'algorithme suivant.

1. Recopier T dans T', X dans X', k dans k' et S dans S',
2. Si e est un envoi, le rajouter à $\mathcal{S}'$.
3. Si e est une réception, enlever l'envoi correspondant de $\mathcal{S}'$.
4. Nous devinons une valeur $k'(\text{last}_p) \geq k(\text{last}_p)$. Si $k'(\text{last}_p) = 1$, alors $e \in N_1$, et donc la transition est étiquetée par $\ell = e$. Sinon $\ell = \epsilon$ et l'algorithme s'achève pour cette transition.
5. $\text{last}_p = x$.

6. Nous définissons $T'(\text{last}_p, q) = 0$ pour $q \neq p$, et $S'(\text{last}_p) = \emptyset$. Alors, on pose $T'(\text{last}_p, p) = 2$ si $k(\text{last}_p) = 0$ (e est le premier événement dans N_1 sur le processus p). Sinon $T'(\text{last}_p, p) = 1$.
7. Si e est une réception, on pose $s \in \mathcal{S}$ l'envoi solitaire associé.
   (a) Tester si $k'(\text{last}_p) = k(s)$.
   (b) Update(e, s).
   (c) Pour chaque $x \in X$ tel que soit $T'(x, p) \neq 0$ soit $s \in S'(x)$ (c'est à dire que x voit e ou s), Update(x, e) .

Un état $[v, T, S, k, X]$ est initial si $v \in I$, $k(x) = T(x, p) = 0$ pour tout $x \in X$, $p \in \mathcal{P}$ et si $\mathcal{S} = \emptyset$. Un état $[v, T, S, k, X]$ est final si $\mathcal{S} = \emptyset$ (il ne reste plus d'envoi solitaire) et $T(\text{last}_p, q) = 2$ pour tout $p, q \in \mathcal{P}$ (le dernier événement sur p voit le premier élément sur q pour tout p, q).

□

On pose $M = N_0 N_1 N_2$ un MSC et ρ un chemin d'étiquette M. Montrons par induction sur le nombre d'événements de M que

Lemme 13 *$T(x, i) = 1$ ssi x peut atteindre $last_i$, $T(x, i) = 2$ ssi x peut atteindre le premier événement sur le processus i dans $G^a(M)$, et $s \in S(x)$ ssi x peut atteindre l'envoi solitaire s dans $G^a(M)$.*

Preuve. L'initialisation est triviale. Supposons que la propriété soit vraie pour M, et montrons la pour Mx.

Supposons que x peut atteindre last_p dans $G^a(Me)$ après n événements lus du MSC. Si c'est un élément de p, alors la propriété est vérifiée. Sinon, si x est un envoi, il est solitaire puisque la réception arrivera plus tard, et l'événement ne voit rien, ce qui n'est pas vrai. Donc x doit être une réception. La seule transition qui sort de x dans $G^a(Me)$ mène à l'envoi associé $s \in \mathcal{S}$. Dans $G^a(Me)$, s doit voir last_p (qui n'est pas x). Donc il doit aussi le voir dans $G^a(M)$. Nous appliquons ainsi l'hypothèse de récurrence pour avoir $T'(x, p) = 1$.

Soit un événement $e \neq x$. Si e voit last_p dans $G^a(M)$, nous appliquons l'hypothèse de récurrence, et obtenons $T(x, p) = 1$. Sinon, c'est que x est utile à e pour voir last_p. Si x est un envoi, rien ne se passe. Donc x est une réception, d'envoi associé s. Soit e voit s dans $G^a(M)$, et alors quand x arrive, e voit juste x en plus de ce qu'il voyait avant. Donc x est sur le processus p, et $T'(e, p) = 1$ d'après la règle 5c) et la définition de update. Sinon, x ne voyait pas s dans $G^a(M)$, et s voyait un événement sur p dans $G^a(M)$. Par

hypothèse d'induction, $T(s,p) = 1$. De plus, comme e voit x dans $G^a(Me)$ et qu'il ne voyait pas s dans $G^a(M)$, c'est qu'il voyait un événement sur $P(x)$ dans $G^a(M)$. Par hypothèse d'induction, $T(e, P(x)) \neq 0$. D'après la règle 5c), et Update(e,x), $'T(e,p) = max(T(e,p), max(T(s,p), T'(x,p)) = 1$.

Nous laissons en exercice pour le lecteur les autres cas (premier événement et envoi solitaire).

Réciproquement, supposons que $s \in S'(x)$, avec $P(s) = p$. Si x est l'événement s, alors x voit l'envoi s. Sinon, x est une réception associée à u', satisfaisant $s \in S(u)$. Remarquons que $s \neq u$ puisque $u \notin \mathcal{S}'$. En appliquant l'hypothèse de récurrence à M, comme $s \in S(u)$, l'événement u voit s. Donc sa réception associée x aussi, et la propriété est vérifiée.

Supposons que $s \in S'(e)$, avec $P(s) = p$ et $e \neq x$. Si $s \in S(e)$, alors l'hypothèse de récurrence dit que e voyait s dans $G^a(M)$, et donc toujours dans $G^a(Mx)$. Sinon, x sert à e pour que $s \in S'(e)$. La seule solution est que x soit une réception associée à un envoi u avec $s \in S(u)$ et $T(e, P(x)) \neq 0$. C'est à dire par hypothèse de récurrence que e voit x, qui voit u qui voit s, donc e voit s. □

Ainsi, si $A \in \text{Gen}(G)$, alors il existe une linéarisation x de A telle que $x \in L(\mathcal{A}(G))$. De même, si $x \in L(\mathcal{A}(G))$, alors x est une linéarisation d'un atome, et il existe y, z avec $yxz \sim v$ et $v \in L(G)$. Ainsi, x est la linéarisation d'un atome $A \in \text{Gen}(G)$. C'est le cas si et seulement si les atomes de $\text{Gen}(G)$ sont de taille non bornée. Il n'y a pas de boucle accessible et coaccessible si et seulement si les atomes sont de taille bornée. Si c'est le cas, alors leur taille est bornée par $|\mathcal{A}(G)|$.

Remarque 11 *Cette construction n'est applicable que si nous avons une borne existentielle.*

Ainsi, en utilisant l'automate construit dans la proposition 61 page 164, nous pouvons tester si un CMSC-graphe sûr est équivalent à un MSC-graphe en PSPACE.

Proposition 62 *Soit G un CMSC-graphe sûr $\exists$-b-borné. On peut tester en PSPACE en b et $\wp$ si Gen(G) est fini. De plus, ce problème est co-NP-dur. Si Gen(G) est fini, alors la taille du plus grand atome est au plus exponentielle en b et $\wp$.*

Preuve. Pour la preuve de co-NP-dureté, nous donnons une réduction du complémentaire du problème du chemin hamiltonien. Soit $G = (V, E)$

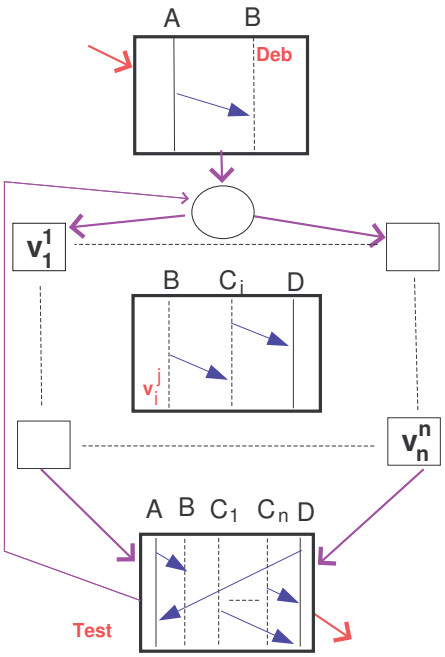


FIG. 5.3 – Une projection d'un MSC-graphe difficile à décrire en MSC-graphe.

un graphe à n noeuds. Le graphe G n'a pas de chemin hamiltonien ssi tout chemin de longueur n contient un noeud deux fois.

Pour générer les chemins de longueur n, nous utilisons n copies de G, avec n^2 noeuds (v_i^j). La fonction de transition est $v_i^j \rightarrow v_{i'}^{j'}$ ssi $v_i \rightarrow_G v_{i'}$ et $j' = j + 1$.

Nous construisons le MSC-graphe H comme sur la figure 5.3, avec des noeuds v_i^j étiquetés par des MSC M_i décrit dans la figure 5.3. Nous projetterons H sur les processeurs $\{A, D\}$. Nous pourrions également coder le même système, mais avec un CMSC-graphe sûr (non projeté).

Nous avons besoin d'un noeud initial Deb et d'un noeud final Test, avec $v_i^j \rightarrow$ Test ssi $j = n$ et Test $\rightarrow v_i^j$ ssi $j = 1$. De plus, Deb $\rightarrow v_i^j$ ssi $j = 1$

La figure 5.3 décrit les MSC étiquetant les noeuds de H. L'idée est de créer un ordre entre les processus A et D seulement si un noeud v_i a été vu deux fois, comme le montre les figures 5.4 page suivante, 5.5 page suivante ci-dessous (v_1 y est vu deux fois).

Ainsi, s'il y a un chemin hamiltonien dans G, en itérant la boucle correspondant à ce chemin, nous obtenons un atome de $\text{Gen}(\pi(\mathcal{L}(H)))$ arbitraire-

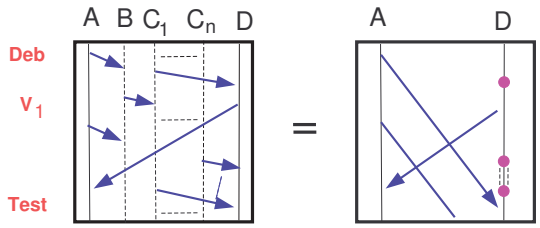


FIG. 5.4 – Aucun noeud n'est vu deux fois.

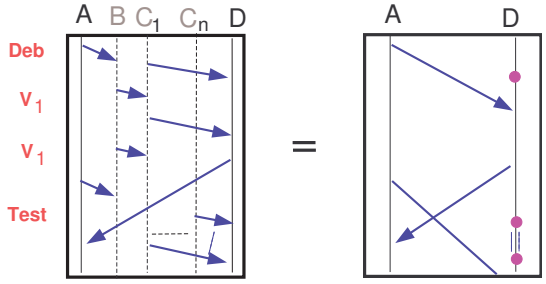


FIG. 5.5 – v_1 est vu deux fois.

ment grand (voir figure 5.4). Si il n'y a pas de chemin hamiltonien dans G, les atomes de $\pi(\mathcal{L}(H))$ sont tous des messages entre A to D, ou deux messages se croisant entre A et D (voir figure 5.5). C'est à dire que $\text{Gen}(\pi(\mathcal{L}(H)))$ est fini.

□

Théorème 17 *Soit G un CMSC-graphe sûr et $\exists$-b-borné. G est équivalent à un MSC-graphe si et seulement si $Gen(G)$ est fini. On peut faire le test en PSPACE en b et $\wp$. Si le test est positif, alors on peut construire un MSC-graphe équivalent de taille au plus exponentielle en la taille de G et du plus grand atome de $Gen(G)$, c'est à dire au plus doublement exponentiel dans $\wp$ et b.*

Ainsi, nous pouvons décrire dans le diagramme 5.6 page suivante par des flèches rouges les algorithmes de test décidables, pour savoir si un CMSC-graphe d'une classe possède une écriture en un CMSC-graphe équivalent d'une autre classe (nous utilisons l'algorithme que nous venons de présenter, plus les algorithmes pour les (C)MSC-graphes à choix local). En fait, le dia-

gramme 5.6 est complet, car nous savons que les autres tests non triviaux sont indécidables.

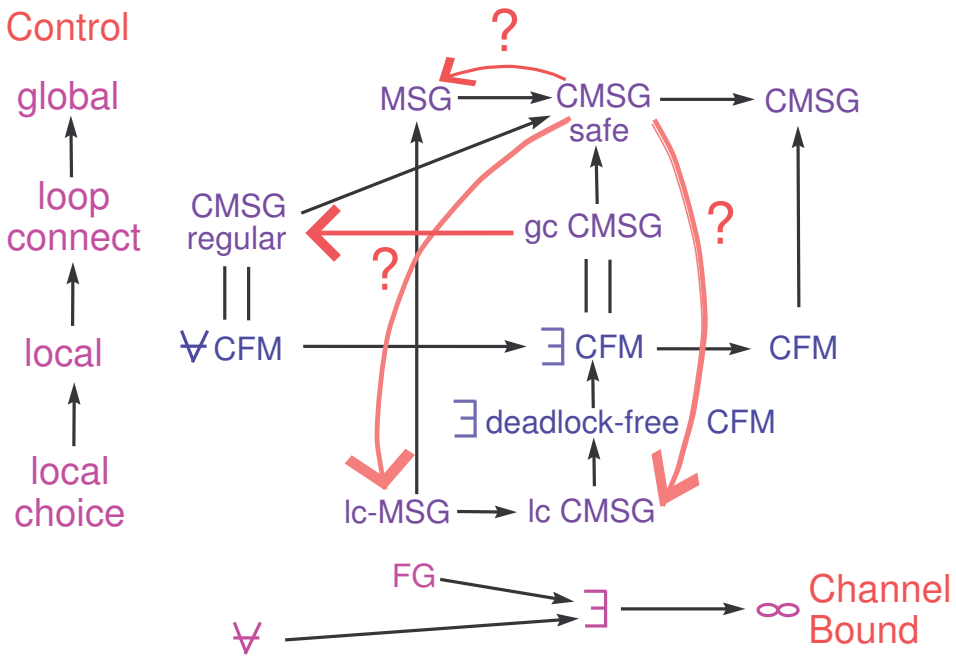


FIG. 5.6 – Tests d'appartenance décidables.

5.2 Compression et Hiérarchie

Plusieurs formalismes utilisent la hiérarchie dans la formulation des systèmes, comme les statecharts et les diagrammes en UML. La norme Z.120 [itu96] inclue elle aussi les MSC-graphes sous forme hiérarchique (appelés alors HMSC). Le fait est que les algorithmes oublient souvent ce formalisme pour ne s'appliquer qu'aux MSC-graphes. Si la décidabilité n'est pas remise en question car il est toujours possible de déplier un HMSC en un MSC-graphe plus gros, la complexité peut changer (le dépliage peut être exponentiel). S'il est parfois impossible de faire mieux que de déplier la structure hiérarchique (problème d'appartenance), il existe de nombreux cas intéressants où cela apporte un vrai gain en terme de complexité (reconnaissance

de motif). Notons que de nombreux autres algorithmes concernant les MSC hiérarchiques ont été étudiés dans [GM02].

5.2.1 Syntaxe et Sémantique des nested MSC.

Puisque les HMSC désignent des graphes, nous préférerons l'appellation *nMSC* (pour *nested MSC*) pour désigner les MSC décrits de manière hiérarchique, alors que les nMSC-graphes désigneront les graphes de MSC décrits de manière hiérarchique.

La norme ITU Z120 [itu96] permet la réutilisation d'un MSC déjà décrit sans avoir besoin de le redéfinir. La définition que nous donnons est faite de telle sorte qu'elle préserve le caractère visuel des MSC (voir la figure 5.7 page suivante). En fait, la définition est la même que celle des templates MSC, sauf qu'ici, le contenu de chaque boîte est entièrement défini par la donnée d'un MSC hiérarchique.

Définition 51 *Un nMSC* $M = (M_q)_{q=1}^n$ *est une suite finie de macros de la forme* $M_q = (\mathcal{P}_q, E_q, B_q, \varphi_q\mathcal{C}, \lambda_q, m_q, <_q)$

Chaque macro M_q *est composé de :*

- *Un ensemble fini d'événements* E_q*.*
- *Un ensemble fini de processus* $\mathcal{P}_q$*.*
- *Un ensemble fini de références* B_q *(boîtes) utilisées par* M_q*.*
- *Une fonction* φ_q *qui associe chaque référence* $b \in B_q$ *avec un indice* $1 \leq \varphi_q(b) \leq q$*.*
  Ainsi, une référence b *sera constitué du MSC représenté par* $M_{\varphi_q(b)}$*. Notons que* $B_1 = \emptyset$*. Nous demandons de plus que* $P_{\varphi_q(b)} \subseteq P_q$*.*
- *Une fonction de type* $\lambda_q : E_q \longrightarrow \mathcal{T}$*, qui associe chaque événement avec un type* $i!j, i?j$ *ou* $i(a)$*, avec* $i, j \in \mathcal{P}_q$*,* $i \neq j$ *et* $a \in \mathcal{C}$*.*
- *La fonction de message* $m_q : S_q \longrightarrow R_q$ *qui associe chaque envoie de type* $i!j$ *avec une réception* $j?i$*, pour tout* $i \neq j$*.*
- *La relation acyclique* $<_q$ *sur l'ensemble des événements et références* $E_q \cup B_q$*, formée de :*
  - *Pour chaque processus* $k \in \mathcal{P}_q$*, la relation* $<_q$ *est un ordre total sur l'ensemble* $E_{q,k}$ *des événements de* k *et l'ensemble des références* $b \in B_q$ *avec* $k \in \mathcal{P}_{\varphi_q(b)}$*.*
  - $e <_q f$ *quand* $m_q(e) = f$ *dans* M_q*.*

La profondeur de M *est le plus grand* d *tel qu'il existe une suite* $q_1 > \cdots > q_{d+1}$ *avec* $\varphi_{q_j}(b) = q_{j+1}$ *pour un certain* $b \in B_{q_j}$*, pour tout* $1 \leq j \leq d$*.*

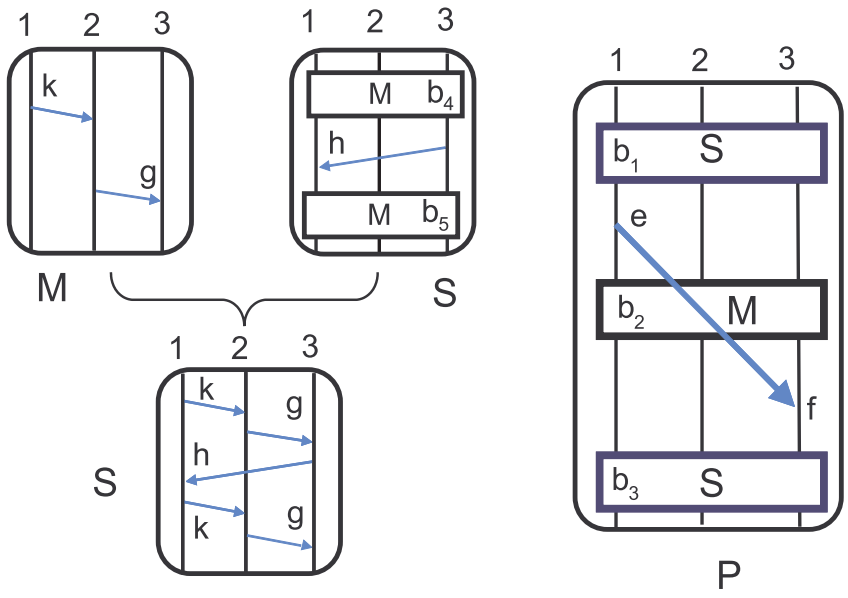


FIG. 5.7 – Un nMSC P utilisant deux références, S et M.

Exemple 49 *Considérons le nMSC P de la Figure 5.7. Il utilise trois références, $B_P = \{b_1, b_2, b_3\}$ qui correspondent à $\varphi_P(b_1) = \varphi_P(b_3) = S$ et $\varphi_P(b_2) = M$. La profondeur de P est 2. L'ordre visuel $<_P$ de P impose sur le processeur 1 l'ordre $b_1 <_P e <_P b_2 <_P b_3$.*

La sémantique d'un nMSC est le MSC défini en remplaçant chaque référence de M par son MSC correspondant. Récursivement, il suffit de définir la sémantique des nMSC de profondeur 1. Soit $M = (M_q)_{q=1}^n$ un nMSC de profondeur 1, avec $M_q = (\mathcal{P}_q, E_q, B_q, \varphi_q, \mathcal{C}, \lambda_q, m_q, <_q)$. Pous simplifier les notations, nous notons ci dessous b à la place de $\varphi_n(b)$.

Le MSC $\langle \mathcal{P}, E, \mathcal{C}, \lambda, m, < \rangle$ associé à $M = (M_q)_{q=1}^n$ est donné par $\mathcal{P} = \mathcal{P}_n$, $E = \uplus_{b \in B_n} E_b \uplus E_n$, $\lambda = \cup_{q=1}^n \lambda_q$ et $m = \cup_{q=1}^n m_q$. L'ordre visuel $<$ est défini par $e < f$ si et seulement si soit $m(e) = f$, soit $P(e) = P(f)$ et une des conditions suivantes :

- $e, f \in E_n$ et $e <_n f$,
- $e, f \in E_b$ et $e <_b f$,
- $e \in E_1, f \in b$ et $e <_n b$,
- $e \in E_b, f \in E_1$ et $b <_n f$,

– $e \in E_b, f \in E_{b'}$ et $b <_n b'$,

où $b, b' \in B_n$. Pour simplifier la notation, nous notons le MSC associé à $M = (M_q)_{q=1}^n$ par M.

Exemple 50 *Dans la figure 5.7 page précédente, la partie inférieure droite représente le MSC associé au nMSC S de profondeur 1. Notons que l'événement $g \in E_M$ apparaît deux fois dans S. Si nous voulons distinguer ces deux instances, nous pouvons les indexer par la boîte dont elles sont issues.*

Notons aussi que la syntaxe des nMSC impose que $b_1 <_1 e$, mais cela ne signifie pas que tous les événements de $S = \varphi_P(b_1)$ doivent arriver avant $e \in E_P$. Par exemple, la première occurrence de g dans S précède e dans P, mais pas la seconde occurrence de g.

Il est également possible qu'un nMSC M syntaxiquement correct ne définisse pas un MSC à cause de l'ordre FIFO. Par exemple, le message (e, f) de P violerait la condition FIFO si M contenait un message du processus 1 au processus 3. Néanmoins, il peut être vérifié en temps polynomial si un nMSC satisfait la condition FIFO.

Taille d'un nMSC. La taille d'une référence est égale au nombre de ses processus. La taille d'un nMSC $M = (M_q) = (\mathcal{P}_q, E_q, B_q, \varphi_q\mathcal{C}, \lambda_q, m_q, <_q)$ est $|M| = \sum_p(|E_p| + \sum_{b \in B_q} |b|)$. La longueur d'un nMSC M est le nombre d'événements du MSC qu'il représente. La longueur est notée $||M||$. Notons qu'on peut calculer la longueur d'un nMSC en temps linéaire en la taille du nMSC. Ce simple calcul constitue notre premier algorithme qui bénéficie de la hiérarchie.

Définition 52 *Un* nMSC-graphe *est une suite finie $G = (G_q)_{q=1}^n$, où chaque G_q est soit un graphe étiqueté soit un nMSC. Un graphe étiqueté G_q est un quintuplet $(V_q, \rightarrow_q, \varphi_q, V_q^0, V_q^f)$ avec :*

- *Un graphe dirigé $(V_q, \rightarrow_q)$,*
- *Un ensemble de noeuds initiaux V_q^0 et finaux V_q^f,*
- *Une fonction φ_q qui associe chaque noeud v avec une référence $1 \leq \varphi_q(v) < q$, représentant $G_{\varphi_q(v)}$.*

Ainsi, un noeud d'un nMSC-graphe peut être associé soit avec un graphe, soit avec un nMSC. Cette définition étend les automates hiérarchiques [AY98] et les nMSC. En fait, les automates hiérarchiques de [AY98] correspondent exactement au cas où il n'y a qu'un seul processus.

La sémantique d'un nMSC-graphe $G = (G_q)_{q=1}^n$ est l'ensemble (possiblement infini) des MSC $\mathcal{L}(G)$ défini récursivement comme suit. Si G_q est un nMSC, alors $\mathcal{L}(G_q)$ est le MSC singleton défini par G_q. Si G_q est un graphe étiqueté, alors $\mathcal{L}(G_q)$ est l'ensemble des MSC associé à un chemin acceptant de G_q. A un chemin $v_1, \ldots, v_n$ de G_q, nous associons l'ensemble des MSC $M_1 \cdots M_n$, où $M_i \in L(G_{\varphi_q(v_i)})$ pour tout $1 \leq i \leq n$. Enfin, $\mathcal{L}(G) = \mathcal{L}(G_n)$.

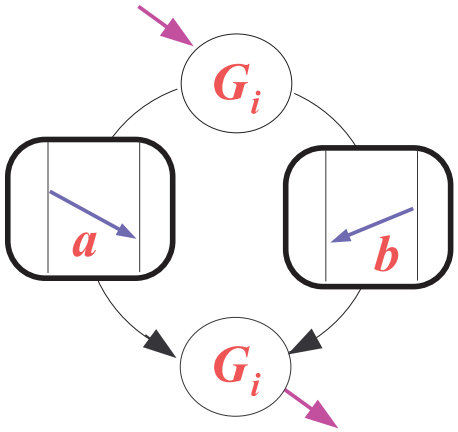


FIG. 5.8 – Un nMSC-graphe G_{i+1} générant $(a+b)^{2^i}$ avec $G_1 = \epsilon$.

5.2.2 Reconnaissance de Motifs

Nous commençons en donnant un algorithme assez représentatif de la manière d'utiliser la hiérarchie pour ne pas recalculer plusieurs fois les mêmes choses, et donc pour baisser le temps de calcul. Notons qu'il n'était pas donné dans [GM02]. En fait, une idée qui fonctionne bien avec la hiérarchie est qu'il faut généralement utiliser un algorithme de programmation dynamique. Parfois, il faut également avoir recours à des arguments arithmétiques [GM02].

Définition 53 *Un nMSC M est un* motif *d'un nMSC N si et seulement si il existe deux CMSC S, T tels que $N = SMT$.*

Nous commençons par montrer que la reconnaissance d'un nMSC motif M dans un nMSC N est en temps polynomial dans le cas particulier où $||M|| = ||N||$. Dans ce cas, la reconnaissance de motif se réduit à tester

l'égalité de deux nMSC. Souvenons nous que tout comme les traces, l'égalité de MSC se teste facilement en testant si les projection sur chaque processeur p se correspondent, c'est à dire si $M|_p = N|_p$ (voir la remarque 6 page 51). Nous allons procéder de cette manière. La projection d'un nMSC sur un processus correspond exactement à un mot compressé par *straight line program* (*SLP*).

Définition 54 *Un SLP sur l'alphabet A est une grammaire hors contexte avec des non terminaux $V = \{X_1, \ldots, X_n\}$, un non terminal initial X_n et des règles de $V \times (V \cup A)^+$. Pour tout non terminal il y a exactement une règle ayant ce non terminal pour règle gauche. De plus, pour toute règle $X_i \longrightarrow \alpha$, tous les X_j de α satisfont $i < j$.*

Les contraintes sur les règles imposent que chaque variable X_i génère un unique mot. Par simplicité, nous exprimons le mot généré par un non terminal X_i aussi par X_i. La longueur d'un non terminal X_i est la longueur du mot généré par X_i et est noté par $||X_i||$. Notons que $||X_i||$ peut être exponentiel dans le nombre de règles. La taille d'un SLP est la somme des tailles des règles.

Un nMSC M peut être identifié à $\wp$ SLP $L^1, \ldots, L^\wp$. Le SLP L^p génère la projection $M|_p$ de M sur le processus $p \in \mathcal{P}$. Nous notons les variables utilisées par L^p par $X|_p$, où X est la référence utilisé par le nMSC $M = (M_q)_{q=1}^n$. La variable initiale de chaque L^p est $M_n|_p$.

Exemple 51 *Dans la figure 5.7 page 171, la projection sur le processus 1 du nMSC P est représentée par $P|_1 \to S|_1 e M|_1 S|_1$, $S|_1 \to M|_1 h M|_1$ et $M|_1 \to k$.*

Nous pouvons ainsi utiliser le théorème suivant qui prouve que l'égalité des SLP (donc des nMSC) est décidable en temps polynomial.

Proposition 63 ([Pla94]) *Soit P un SLP, et A, B deux non terminaux de P. Nous pouvons déterminer si A et B génèrent le même mot en temps $O(|P|^5 \log(|P|))$.*

Ainsi, nous pouvons tester en temps $O((|M| + |N|)^5 \log(|M| + |N|))$ si deux nMSC N, M sont égaux. Nous pouvons néanmoins utiliser un meilleur algorithme, qui résout la reconnaissant d'un SLP motif dans un SLP.

Contrairement à l'égalité, la recherche d'un MSC motif dans un MSC ne peut pas se réduire à la recherche de chaque projection. Bien entendu, il est nécessaire que pour tout p, $M|_p$ soir un motif de $N|_p$.

Proposition 64 *Soit M, N deux MSC. Nous pouvons tester en temps linéaire si M est un motif de N.*

Preuve. L'idée vient de la reconnaissance de motif pour les traces de Mazurkiewicz [LWZ90]. Nous appliquons l'algorithme de Knuth-Morris-Pratt pour déterminer en temps linéaire toutes les occurrences de $M|_p$ dans $N|_p$, pour tout $p \in \mathcal{P}$. Alors nous cherchons des $\wp$-uplets d'occurrences $(M|_p^0)_{p\in\mathcal{P}}$ qui forment un facteur M. Ainsi, nous recherchons une configuration de N qui a $M|_p$ comme suffixe pour tout processus p. Cela est fait en se remémorant l'ensemble J des processus i satisfaisant cette condition. Si $J = \mathcal{P}$, alors nous avons trouvé un motif. Sinon, nous progressons sur un processus $i \notin J$ jusqu'à la prochaine occurrence du motif $M|_i$. Pour chaque processus $p \in \mathcal{P}$, nous calculons l'événement f_p maximal sur le processus p telle que $f_p < e$. Nous mettons à jour la configuration courante sur chaque processus avec le maximum entre f_p et la configuration précédente. Comme chaque élément de N n'est traité qu'une seule fois, la complexité est linéaire. □

Exemple 52 *Reprenons la figure 4.6. L'algorithme de recherche de motif de P dans M commence au début de M. Puis il avance à la première occurrence A, et il est toujours au début sur les processus 2 et 3. Nous avons $J = \{1\}$.*

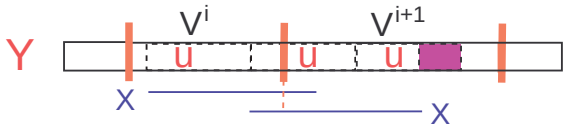
Puis il avance sur le processus 2 jusqu'à l'occurrence B. Il doit donc avancer sur le processus 3 d'une réception et d'un envoi. Alors $J = \{1, 2\}$. On n'avance sur le processus 3 jusqu'à D. Ca oblige d'avancer sur le processus 2 jusqu'à C, et sur le processus 1 jusqu'à la fin. Comme un processus a été fini sans qu'il n'y ait d'occurrence sur lui, nous savons qu'il n'y a pas de motif P dans M.

De la même manière que la reconnaissance de motif pour les MSC, la stratégie pour la recherche de motif des nMSC utilise dans un premier temps une représentation implicite de toutes les positions où le SLP $M|_p$ apparaît en tant que motif du SLP $N|_p$ [MST97] (voir aussi [Pla94]). Ensuite, nous calculons les positions où un $\wp$-uplet de projections $(M|_p)_{p\in\mathcal{P}}$ forme un facteur de M.

Nous notons une occurrence particulière d'une projection $M|_p$ par $M|_p^0, M|_p^1$ etc. Nous dirons qu'un $\wp$-uplet d'occurrences $(M|_p^0)_{p\in\mathcal{P}}$ forme un facteur de N si il existe une occurrence M^0 de M dans N telle que pour tout $p \in \mathcal{P}$, $M^0|_p = M|_p^0$.

Proposition 65 ([MST97]) *Soit P un SLP et A, B deux non terminaux*

de P. Il est possible de calculer une représentation implicite de toutes les occurrences de A dans B en temps $O(|A|^2|B|^2)$.



progression arithmétique $occ(X, Y, V^i)$

L'idée derrière l'algorithme de [MST97] vient de [Pla94] et de la combinatoire des mots. Soit X un non terminal de A et supposons que le mot défini par X est un facteur du mot défini par B. Supposons que X n'est facteur d'aucun non terminal Y sur le plus bas niveau de hiérarchie, c'est à dire dont la règle associée est $Y \to \alpha \in A^*$. On choisit alors une règle $Y \longrightarrow V^1 \cdots V^k$, et $i \leq k-1$ tels que X apparaît dans Y (on appelle X^0 cette occurrence de X) et V^i est le premier non terminal parmi $V^1 \cdots V^k$ que X^0 rencontre sans être totalement inclus dans V^i (voir la figure ci dessus). En particulier, Y est le plus petit non terminal qui contient X^0. On pose $occ(X, Y, V^i)$ l'ensemble des positions de Y à partir desquelles une occurrence X^0 de X commence, commence à l'intérieur de V^i et finit strictement après V^i. On pose également $occ(X, Y) = \bigcup_i occ(X, Y, V^i)$.

En utilisant un argument combinatoire (le lemme de Fine et Wilf), il peut être montré que $occ(X, Y, V^i)$ est une progression arithmétique qui peut être calculée par un algorithme de programmation dynamique en temps polynomial. Par exemple, considérons les mots $Y = aaababababbabb$ et $X = ababab$. Il y a trois occurrences de X dans Y, la première commençant à partir de la troisième lettre de Y, puis la cinquième lettre, puis la septième lettre.

Remarque 12 *En utilisant l'algorithme de [MST97], nous obtenons immédiatement un algorithme pour l'égalité de deux SLPs M, N. Cet algorithme est en temps $O(|M|^2|N|^2)$.*

Nous allons expliquer ici comment rechercher un nMSC motif M dans un nMSC N lorsqu'il n'y a pas de communication à l'intérieur de M. Pour le cas où les processus communiquent entre eux, il faut utiliser un algorithme très différent utilisant de manière cruciale l'arithmétique, comme décrit dans [GM02]. En réunissant les deux cas, nous obtenons :

Théorème 18 *Soit M, N deux nMSC. Savoir si le MSC décrit par M est un motif du MSC décrit pas N peut se résoudre en temps $O(|\mathcal{C}_M|^2|M|^2|N|^2) \leq$*

$O(\wp^2|M|^2|N|^2)$, où $\mathcal{C}_M$ est le nombre de composantes connexes dans le graphe de communication de M.

Tout d'abord, nous allons supposer que tous les processus de N apparaissent dans M. Si ça n'est pas le cas, nous pouvons modifier légèrement M et N en ajoutant à chaque référence Y de N un événement local loc_p entre chaque événement ou boîte de chaque processus $p \in P(N) \setminus P(M)$.

Soit M, N deux nMSC. Pour chaque référence X de M ou N, nous savons calculer en temps $O(|M|^2|N|^2)$ une représentation compacte de toutes les occurrence de $M|_p$ dans N pour tous les processus $p \in \mathcal{P}$. On rappelle maintenant le sous cas du lemme qui a servit pour les algorithmes sur les templates MSC avec deux boîtes (qui correspondent aux MSC contenant un certain motif). Ce sous-cas correspond au fait que les processus de M ne communiquent pas.

Définition 55 *Soit $a \in occ(M|_p, Y), b \in occ(M|_q, Y)$ deux occurrences de projections de M sur les processus $p \neq q$. Alors a, b sont compatibles si et seulement s'il n'y a pas de message depuis le processus p vers le processus q envoyé après a et reçu avant b (ou l'inverse).*

Lemme 14 *Soit $a_p \in occ(M|_p, Y)$, pour tout $p \in \mathcal{P}$. Alors $(a_p)_{p\in\mathcal{P}}$ est une occurrence de M dans Y ssi a_p, a_q sont compatibles pour tout couple de processus $p \neq q$.*

Ainsi, on voit facilement que b est compatible avec une occurrence de $occ(M|_p, Y, V^i)$ si et seulement si il est compatible avec la première occurrence de $occ(M|_p, Y, V^i)$, puisque qu'il ne peut pas y avoir de mauvais message séparant les occurrences de $occ(M|_p, Y, V^i)$ (toutes les occurrences se rencontrent en le dernier événement de V^i, et n'envoient pas de message). Ainsi, nous pouvons supposer sans restriction que $occ(M|_p, Y, V^i)$ est un singleton. Soit $p \in \mathcal{P}$. Notons que les occurrences de $M|_p$ dans Y sont totalement ordonnées par l'ordre total sur le processus p de Y. Nous comprendrons les fonctions min et max par rapport à ces ordres totaux.

Proposition 66 *Soit $a = (a_p)_{p\in\mathcal{P}}, b = (b_p)_{p\in\mathcal{P}} \in (occ(M|_p, Y))_{p\in\mathcal{P}}$ deux occurrences de M dans Y. Alors $(\min(a_p, b_p))_{p\in\mathcal{P}}$ et $(\max(a_p, b_p))_{p\in\mathcal{P}}$ sont aussi des occurrences de M dans Y.*

Preuve. En appliquant le lemme 14, il suffit de vérifier que $\min(a_p, b_p)$, $\min(a_q, b_q)$ sont compatibles pour tout $p \neq q$. Le seul cas à vérifier est quand

$\min(a_p, b_p) = a_p < b_p$ et $\min(a_q, b_q) = b_q < a_q$. Supposons par l'absurde qu'il y ait un message de p à q envoyé après a_p et reçu avant b_q. Alors a_p et $a_q > b_q$ ne seraient pas compatibles. Le cas du message d'en l'autre direction est symétrique. Nous avons donc une contradiction.

□

Si M apparaît en tant que motif de N, la proposition 66 page précédente assure qu'il y a une unique occurrence minimale de M dans N, où minimale se comprend en prenant l'ordre partiel par composante sur les $\wp$-uplets de $(occ(M|_p, N))_{p\in\mathcal{P}}$. Afin de trouver l'occurrence minimale de M dans une référence X de N, nous allons construire l'occurrence minimale compatible dans chaque référence $Y \in X$. Si Y ne contient pas complètement le MSC représenté par M, alors nous aurons besoin de plus d'information concernant les composantes à l'extérieur de Y qui sont compatibles avec un composant à l'intérieur de Y. Comme il peut y avoir de nombreuses références X qui contiennent dans un certain niveau de hiérarchie le non terminal Y, nous codons cette information par des occurrences imaginaires notées $\downarrow_p$ et $\uparrow_p$, pour chaque processus $p \in \mathcal{P}$. L'occurrence $\downarrow_p$ signifie une occurrence de $M|_C$ après Y, alors que $\uparrow_p$ signifie une occurrence de $M|_p$ avant Y. Ainsi, nous étendons l'ordre à $\uparrow_p < a_p < \downarrow_p$ pour tout $a_p \in occ(M|_p, Y)$. Pour $p \neq q$, nous disons que $\uparrow_p, a_q \in occ(M|_q, Y)$ sont compatibles si il n'y a pas de message de p à q avant a_q dans Y (et symétriquement pour $\downarrow$).

Définition 56 *Soit Y une référence de N. Soit $E \subseteq \{\neq\uparrow_C, =\downarrow_C \mid C \in \mathcal{C}_M\}$ un ensemble de contraintes. Nous définissons $Up_E^Y = (a_p)_{p\in\mathcal{P}}$ le $\wp$-uplet minimal qui satisfait :*

1. *Pour tout $p \in \mathcal{P}$, $a_p \in occ(M|_p, Y) \cup \{\uparrow_p, \downarrow_p\}$.*
2. *Les occurrences $(a_p)_{p\in\mathcal{P}}$ sont compatibles deux à deux.*
3. *$(a_p)_{p\in\mathcal{P}}$ satisfait les contraintes de E, c'est à dire que $(a_p)_{p\in\mathcal{P}}$ satisfait $(\neq\uparrow_q) \in E$ si $a_q \neq\uparrow_q$ et $(=\downarrow_q) \in E$ si $a_q =\downarrow_q$.*

Notons que l'existence d'une occurrence dans la définition précédente est assurée par le $\wp$-uplet $(a_p)_{p\in\mathcal{P}}$ avec $a_p =\downarrow_p$ pour tout processus p. En d'autres termes, nous pouvons toujours espérer trouver une occurrence de M après Y.

Exemple 53 *Les deux conditions extrêmes correspondent à $Up_\emptyset = (\uparrow_p)_{p\in\mathcal{P}}$ et $Up_{(=\downarrow_p)_{p\in\mathcal{P}}} = (\downarrow_p)_{p\in\mathcal{P}}$.*

Dans la figure 5.9 page ci-contre, nous avons :

- $Up_{\{\neq\uparrow_1\}} = (a, \uparrow_2, e, \uparrow_4, \uparrow_5) = Up_{\{\neq\uparrow_1,\neq\uparrow_3\}}$.
- $Up_{\{=\downarrow_2\}} = (b, \downarrow_2, e, \uparrow_4, \uparrow_5)$.
- $Up_{\{\neq\uparrow_4,=\downarrow_5\}} = (\uparrow_1, \uparrow_2, \uparrow_3, g, \downarrow_5)$.

Le lemme suivant montre qu'il suffit de calculer récursivement les $\wp$-uplets Up^Y_E pour des contraintes E bien choisies.

Lemme 15 *Soit* $(b_p)_{p\in\mathcal{P}} = Up^N_{(\neq\uparrow_p)_{p\in\mathcal{P}}}$. *Alors* M *est un motif de* N *ssi* $b_p \neq\downarrow_p$, *pour tout* $p \in \mathcal{P}$.

Le problème est que nous pouvons avoir besoin de $\wp$-uplets Up^Y_E pour des ensembles E de contraintes arbitraires. Or il y a un nombre exponentiel d'ensembles E. Heureusement, nous pouvons éviter également de calculer tous ces ensembles, en ne calculant Up^Y_E que pour les singletons $E = \{\neq\uparrow_p\}$ et $E = \{\downarrow_p\}$, $p \in \mathcal{P}$. Nous montrons tout d'abord que ces $\wp$-uplets suffisent à calculer en temps polynomial Up^Y_E pour n'importe quel E.

Lemme 16 *Soit* $E, F \subseteq \{\neq\uparrow_C, =\downarrow_C |\ C \in \mathcal{C}_M\}$ *deux ensembles de contraintes. Alors* $Up^Y_{E\cup F} = \max(Up^Y_E, Up^Y_F)$.

Preuve. Soit $b = (b_p)_p = \max(Up^Y_E, Up^Y_F)$. Nous avons trivialement $Up^Y_{E\cup F} \geq Up^Y_E$ et $Up^Y_{E\cup F} \geq Up^Y_F$, donc $Up^Y_{E\cup F} \geq b$. De plus, $Up^Y_{E\cup F}$ est le plus petit ensemble qui vérifie les trois propriétés que b vérifie aussi : le $\wp$-uplet $(b_p)_{p\in\mathcal{P}}$ est deux à deux compatibles et il satisfait les contraintes de $E \cup F$. Ainsi, $b = Up^Y_{E\cup F}$. □

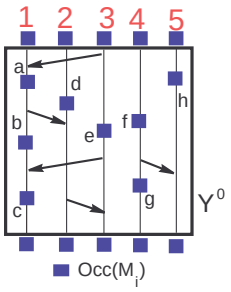
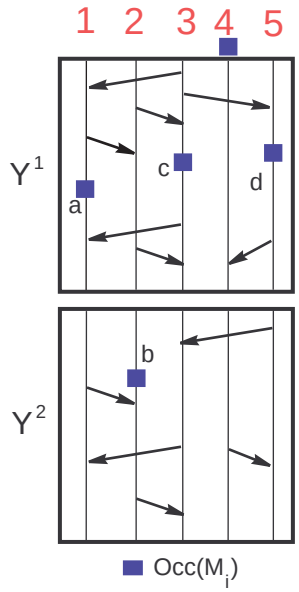


FIG. 5.9 – Exemple de contraintes et d'occurrences minimales.



$$Up^{Y^1Y^2}_{\{\neq\uparrow_5\}} = (a, b, c, \uparrow_4, d).$$
$$Up^{Y^1}_{\{\neq\uparrow_5\}} = (a, \downarrow_2, c, \uparrow_4, d) = Up^{Y^1}_{\{=\downarrow_2,\neq\uparrow_5\}}.$$
$$Up^{Y^2}_{\{\neq\uparrow_2\}} = (\uparrow_1, b, \uparrow_3, \uparrow_4, \uparrow_5).$$

Proposition 67 *Considérons une référence Y de N et un processus $p \in \mathcal{P}$. Alors $Up^Y_{\{\neq\uparrow_p\}}$ et $Up^Y_{\{=\downarrow_p\}}$ peuvent être calculés en temps $O(|Y|_\wp{}^2)$ à partir des $\wp$-uplet $(Up^Z_{\{\neq\uparrow_p\}})_{p\in\mathcal{P}}$ et $(Up^Z_{\{=\downarrow_p\}})_{p\in\mathcal{P}}$, où Z apparaît à droite dans la règle associée à Y.*

Preuve. Si Y est au plus bas niveau de la hiérarchie, alors nous appliquons la proposition 18 page 176 pour avoir la réponse en temps linéaire.

Supposons que $Y = Y^1Y^2$.

Nous voulons calculer l'ensemble $E_\downarrow \subseteq \mathcal{P}$ composé de tous les processus p tel que $M|_p$ n'a pas d'occurrence dans Y^1 compatible avec les contraintes. C'est à dire que $M|_p$ doit apparaître soit dans Y^2, soit après Y. Nous commençons en posant $E_\downarrow = \emptyset$ et en augmentant $E_\downarrow$ jusqu'a ce qu'il existe une paire d'occurrences a, b telle que :

– $(a_p)_{p\in\mathcal{P}}$ est une occurrence de M dans Y^1 avec $a_p =\downarrow_p$ ssi $p \in E_\downarrow$,
– $(b_p)_{p\in\mathcal{P}}$ est une occurrence de M dans Y^2 avec $b_p =\uparrow_p$ ssi $p \notin E_\downarrow$.

L'algorithme pour calculer $Up^Y_{\{\neq\uparrow_q\}}$ est décrit ci dessous (pour $Up^Y_{\{=\downarrow_q\}}$, le raisonement est similaire) :

(1) Soit $E_\downarrow = \emptyset$

(2) Calculer $(a_p) = Up_E^{Y^1}$, avec $E = \{\neq\uparrow_q\} \cup \{=\downarrow_p \mid p \in E_\downarrow\}$
(3) Mettre à jour $E_\downarrow = \{ p \mid a_p = \downarrow_p\}$

(4) Calculer $(b_p) = Up^{Y^2}_{(\neq\uparrow_p)_{p\in E_\downarrow}}$
(5) Mettre à jour $E_\downarrow = \{ p \mid b_p \neq \uparrow_p\}$.
Si $E_\downarrow$ a changé, goto (2).

(6) Soit $d_p = b_p$ si $p \in E_\downarrow$, et $d_p = a_p$ sinon.
(7) Return $(d_p)_{p\in\mathcal{P}}$.

Notons que chaque fois que l'ensemble $E_\downarrow$ change à l'étape (5), il grossit strictement. Ainsi, l'algorithme retourne à l'étape (2) au plus $O(\wp)$ fois.

Notant par $E_\downarrow^t$ la valeur de $E_\downarrow$ après t itérations, la t-ième itération a besoin d'un temps $\wp(|E_\downarrow^t| - |E_\downarrow^{t-1}|)$. Ainsi, la compléxité totale de l'algorithme est au plus de $O(\wp^2)$.

Si Y a plus de deux références Y^i, alors nous procédons de la même manière, mais avec plusieurs ensembles $E_{\downarrow,i}$, un par référence Y^i, $i > 1$. Nous avons $p \in E_{\downarrow,i}$ si $M^0|_p$ doit apparaître après Y^i. A chaque pas de la boucle, au moins un des ensembles a changé, et donc il y a au plus $\wp|Y|$ pas de calcul (Y a au plus $|Y|$ références), qui donne une complexité totale d'au plus $O(|Y|\wp^2)$.

□

En appliquant cet algorithme pour toute référence de Y et toute contrainte $\downarrow_p$ ou $\neq\uparrow_q$, nous obtenons un algorithme en temps $O(\wp^3|N|) = O(|M|^2|N|^2)$.

5.2.3 Le Problème d'Appartenance

Néanmoins, il existe des cas où la hiérarchie ne peut pas être utilisée efficacement. Nous présentons ici le problème de l'appartenance d'un mot compressé dans un automate hiérarchique, qui montre en particulier qu'une description hiérarchique ne peut pas être bénéfique en ce qui concerne l'appartenance d'un nMSC dans un nMSC-graphe. Néanmoins, nous savons que si l'automate ou le mot n'est pas compressé, alors il existe un algorithme polynomiale pour résoudre le problème d'appartenance [GM02], voir aussi [Ryt99].

Nous rappelons la proposition 47 page 123 : savoir si un MSC M appartient au langage d'un CMSC-graphe G est NP-complet en le nombre $\wp$ de

processus, et NLOGSPACE en $|M|, |G|$. Ainsi, si M est un nMSC et G un nMSC-graphe, alors la taille du MSC décrit par M est $O(2^{|M|})$ et la taille du MSC-graphe décrit par G est $O(2^{|G|})$. Cela nous donne donc une complexité de PSPACE par rapport à $|M|$ et $|G|$. Si on n'y prend pas garde, le saut en complexité semble assez faible, de NP-complet à PSPACE. Néanmoins, si nous analysons la complexité de plus près, la hiérarchie ne permet pas de réduire la représentation du facteur limitant, le nombre de processus $\wp$. Ainsi, la hiérarchie ne peut pas servir à réduire la complexité de tous les problèmes.

Pour que cette affirmation soit correcte, il faut démontrer qu'il n'est pas possible d'avoir un algorithme intrinsèquement meilleur. Nous allons en fait démontrer qu'un problème plus simple, celui de l'appartenance d'un SLP à un automate hiérarchique [AY98], est déjà PSPACE-complet (voir le théorème 19). La définition d'un automate hiérarchique découle de celle des nMSC-graphes, puisqu'un nMSC-graphe avec un seul processus correspond exactement à un automate hiérarchique.

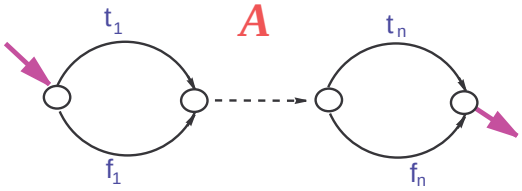
Théorème 19 *Soit W un mot défini par un SLP et $\mathcal{A}$ un automate hiérarchique. Alors le problème d'appartenance $W \in L(\mathcal{A})$ est PSPACE-complet. Si l'alphabet est unaire, alors le problème d'appartenance est NP-complet.*

Remarque 13 *La NP-dureté du cas unaire a été prouvée aussi dans [Ryt99].*

Preuve. Nous prouvons la réduction du problème *(1-in-3) SAT* au problème d'appartenance unaire. Nous aurons besoin de ces idées pour le cas non unaire.

Soit $\varphi = \wedge_{j=1}^{m} C(\alpha_j, \beta_j, \gamma_j)$ une instance de (1-in-3) SAT sur n variables $(x_i)_{i=1,n}$. Ici, une clause $C(\alpha_j, \beta_j, \gamma_j)$ est vraie ssi exactement un des littéraux $\alpha_j, \beta_j, \gamma_j$ est vrai (voir l'exemple 9 page 27). Nous utilisons l'alphabet unaire $\{a\}$. Remarquons que chaque mot $x \in a^*$ est uniquement défini par sa longueur.

Nous associons à chaque clause $C_j = C(\alpha_j, \beta_j, \gamma_j)$ le mot $w_j \in a^*$ de longueur 4^j. Ce mot peut être défini par un SLP de taille polynomial. On pose $W = w_1 \cdots w_m \in a^*$, le mot de longueur $\sum_{j=1}^{m} 4^j$. L'automate $\mathcal{A}$ est constitué d'une suite de n choix entre des transitions étiquetées par t_i et par f_i, où $t_i = \sum_{j \in R_i} 4^j$ et $R_i = \{j \mid x_i \in \{a_j, b_j, c_j\}\}$. De la même manière, $f_i = \sum_{j \in S_i} 4^j$ et $S_i = \{j \mid (\neg x_i) \in \{a_j, b_j, c_j\}\}$.



Chaque chemin de $\mathcal{A}$ correspond à une valuation σ où chaque variable x_i est vraie si le chemin passe par t_i, et fausse si le chemin passe par f_i. On pose n_j le nombre de littéraux de C_j qui sont vrai dans σ. Souvenons nous que σ satisfait la formule φ ssi $n_j = 1$ pour tout j. Il est facile de voir que ρ est étiqueté par le mot $L \in a^*$ de longueur $\sum_{j=1}^{m} n_j 4^j$. Remarquons que comme chaque clause possède trois littéraux, $n_j \in \{0, 1, 2, 3\}$ pour tout j. La longueur de L en base 4 est alors $(n_m n_{m-1} \dots n_1 0)_4$. Nous avons donc $W = L$ ssi $(11 \dots 10)_4 = (n_m n_{m-1} \dots n_1 0)_4$, c'est à dire ssi $n_j = 1$ pour tout j. Ainsi, il y a un chemin de $\mathcal{A}$ étiqueté par W ssi il existe une valuation satisfaisant φ.

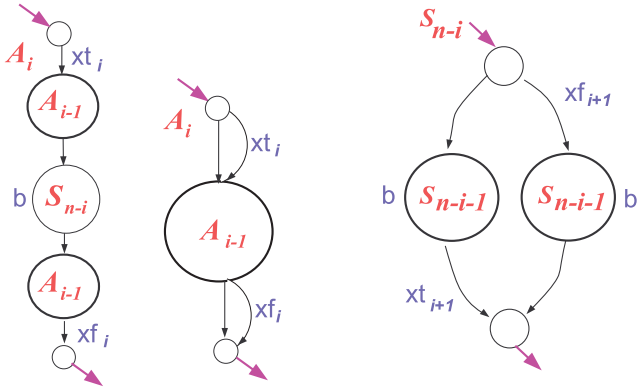
Dans un second temps, nous réduisons le problème *(1-in-3) QBF* (one-in-three quantified boolean formula) au problème d'appartenance hiérarchique. Soit φ une instance de (1-in-3) QBF de la forme $\varphi = Q_n x_n \cdots Q_1 x_1 \psi$, où $Q_i \in \{\exists, \forall\}$ et la formule ψ est de la forme $\wedge_{j=1}^{m} C(\alpha_j, \beta_j, \gamma_j)$. Comme précédemment, une clause $C(\alpha_j, \beta_j, \gamma_j)$ est vraie ssi un littéral au moins est vrai.

L'idée est de faire correspondre un arbre de valuation des variables aux chemins d'un automate hiérarchique $(\mathcal{A}_i)_{i=0,n}$, et de valider ces choix par un SLP $(W_i)_{i=0,n}$. Nous définissons l'automate $\mathcal{A}_i$ et les SLP W_i par récurrence sur $i = 0, \dots, n$. Ici, nous utilisons un alphabet binaire $\{a, b\}$. La lettre a aura la même signification que dans le cas unaire, et la lettre b sera utilisée comme symbole délimiteur.

Nous définissons les mots $w_j, t_i, f_i \in a^*$ par rapport à ψ comme précédemment. C'est à dire que chaque w_j est associé à la clause C_j et t_i, f_i sont associés aux variables x_i. De plus, nous associons chaque variable x_i à un mot $w_{i+m} \in a^*$ de longueur 4^{i+m}. On pose $W_0 = w_1 \cdots w_{n+m}$ le mot de a^* de longueur $\sum_{j=1}^{n+m} 4^j$, et $\mathcal{A}_0$ l'automate constitué d'une ϵ-transition de son état initial à son état final. On pose aussi S_0 l'automate constitué d'une transition étiquetée par b (c'est même le seul automate où b apparaît explicitement). Les mots $(W_i)_{i=1,n}$ compressés par SLP sont définis par :

– $W_i \longrightarrow W_{i-1}$, si $Q_i = \exists$,

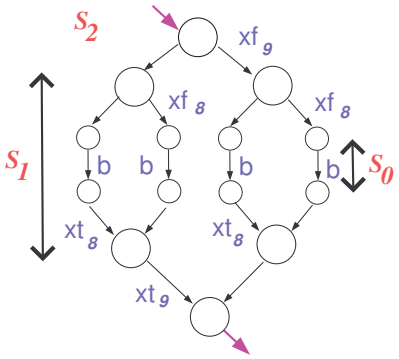
– $W_i \longrightarrow W_{i-1}\, b\, W_{i-1}$, si $Q_i = \forall$.



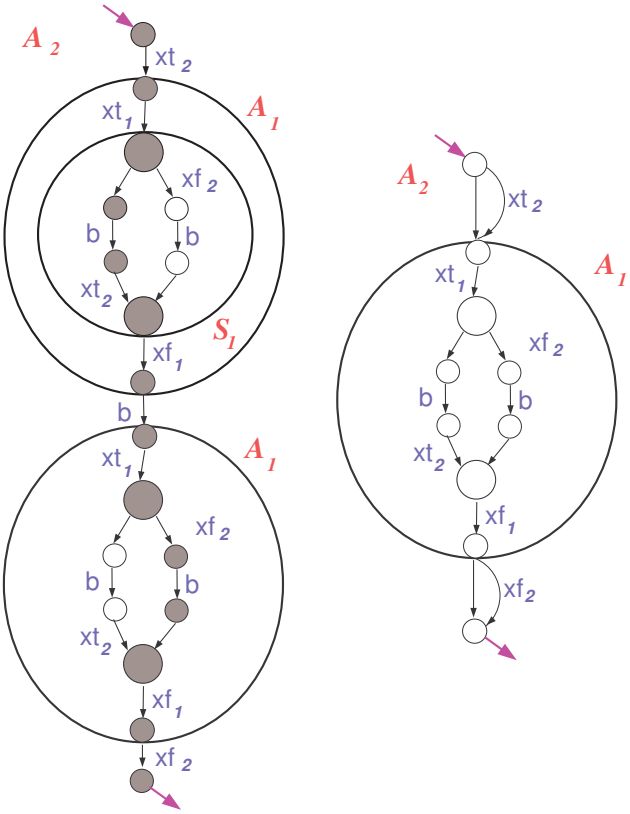
La définition récursive des automates $(\mathcal{A}_i)_{i=1,n}$ et $(S_i)_{i=0,n-1}$ est illustrée par la figure ci dessous. Les transitions sont soit étiquetées par ϵ, soit par $xt_i = t_i\, w_{i+m}$, soit par $xf_i = f_i\, w_{i+m}$. L'automate de gauche définit $\mathcal{A}_i$ lorsque $Q_i = \forall$. L'automate du milieu définit $\mathcal{A}_i$ lorsque $Q_i = \exists$. Enfin, l'automate de droite défini S_i. Dans la figure, nous avons rappelé la position des délimiteurs b.

L'idée est la suivante : les valeurs de $x_{i+1}, \ldots, x_n$ sont déjà choisies (par les niveaux de hiérarchie >i) lorsque l'automate appelle $\mathcal{A}_i$. L'automate $\mathcal{A}_i$ à gauche fixe x_i à vrai, puis utilise S_{n-i} pour régénérer les valeurs fixées de $x_{i+1}, \ldots x_n$, et finalement fixe x_i à faux. L'automate $\mathcal{A}_i$ au milieu devine si x_i est vrai (en prenant ou non la transition étiquetée par xt_i) ou faux (en prenant la transition étiquetée par xf_i). Si il choisit les deux transitions, étiquetées par xt_i et xf_i ou aucune d'elles, alors le mot étiquetant le chemin ne pourra pas égaler W_n, parce que W_n contient exactement une occurrence de w_{i+m} entre chaque occurrence de b.

Pour illustrer la manière dont S_{n-i} régénère les valeurs de $x_{i+1}, \ldots, x_n$, nous montrons S_{n-i} pour $n = 9, i = 7$ dans la figure ci dessous.



Nous illustrons la manière dont $\mathcal{A}_i$ travaille dans deux exemples ci dessous, qui montre le dépliage de l'automate $\mathcal{A}_2$ pour $\varphi = \forall x_2 \forall x_1 \psi$ à gauche et pour $\varphi = \exists x_2 \forall x_1 \psi$ à droite.



Les automates $\mathcal{A}_i$ et S_i sont définis de telle manière que chaque chemin de $\mathcal{A}_i$ est étiqueté par au plus un xt_i et au plus un xf_i entre chaque b, et pour chaque i (nous supposons que chaque automate commence et fini par

une occurrence fictive de b). C'est à dire que le chemin peut être étiqueté par xt_i et xf_i, mais jamais par deux xf_i ou deux xt_i. Par l'absurde, supposons qu'il y ait deux occurrences de b dans $\mathcal{A}_i$ entre lesquelles il y a un chemin de l'un à l'autre étiqueté par deux xt_i (le cas xf_i est symétrique). Nous prenons l'automate $\mathcal{A}_i$ minimal qui a cette propriété. Par minimalité de $\mathcal{A}_i$, Soit la première transition xt_i de $\mathcal{A}_i$ est prise, ou bien cela arrive entre S_{n-i} et une des deux occurrences de $\mathcal{A}_{i-1}$, inclues. Examinons les définitions de (A_i) et (S_i) de plus près :

1. Avant le b de S_{n-i}, aucun xt_j ne peut être vus.
2. Après le b de S_{n-i}, un xt_j au plus peut être vu si $j > i$, et aucun xt_j si $j \leq i$.
3. Avant le premier b de A_i, un xt_j peut être vu si $j \leq i$, mais aucun xt_j si $j > i$.
4. Après le dernier b de A_i, un xt_j au plus peut être vus.

Il suffit de combiner ces remarques pour chaque cas. Entre le début de A_i et le premier b, nous appliquons 3. Entre le dernier b de A_{i-1} et le b de S_{n-i}, nous appliquons 4 et 2. Les deux autres cas sont symétriques, prouvés en appliquant 1 et 3 et 4.

Prouvons que $W_n \in L(\mathcal{A}_n)$ ssi il existe un arbre de valuation VT satisfaisant φ. En utilisant la propriété que nous venons juste de montrer, nous savons qu'entre chaque b de tout chemin de $\mathcal{A}_n$, il y a au plus trois w_j et deux w_{i+m} pour tout $1 \leq j \leq m, 1 \leq i \leq n$. Ainsi, notre codage en base 4 est toujours valable. Ainsi, un chemin ρ de $\mathcal{A}_n$ est étiqueté par W_n ssi pour tout $1 \leq k \leq n+m$, il y a exactement un w_k entre chaque b.

Supposons que VT est un arbre de valuation satisfaisant φ. La racine de VT est étiquetée par x_n et tous les nœuds du niveau l sont étiquetés par x_{n-l}. Les feuilles (sur le niveu $n+1$) ne sont pas étiquetées. Un nœud v étiqueté par x_i correspond à une valuation $\sigma(v)$ des variables $x_{i+1}, \ldots, x_n$. Une valuation $\sigma(v)$ définit deux mots $T(v), F(v)$ comme suit : le mot $T(v)$ est la concaténation de tous les xt_j pour lesquels $j > i$ et x_j est vrai dans $\sigma(v)$. Le mot $F(v)$ est la concaténation de tous les xf_j pour lesquels $j > i$ et x_j est faux dans $\sigma(v)$. Soit v un nœud de VT étiqueté par x_i. Nous définissons le mot $\rho(v) = T^{-1}(v)W_iF^{-1}(v)$. Rappelons que $T(v), F(v)$ sont des mots de a^*. Ainsi, $\rho(v) = T^{-1}(v)W_iF^{-1}(v)$ est obtenu après avoir effacé $|T(v)|$ fois la lettre a à gauche de W_i, et avoir effacé $|F(v)|$ fois la lettre a à droite de W_i.

Nous montrons par récurrence sur le niveau i que $\rho(v)$ est dans $L(\mathcal{A}_i)$ pour chaque nœud v de VT sur le niveau i.

Si v est une feuille de VT, alors elle définit une valuation acceptante pour ψ, et donc $T(v)F(v) = W_0$ en utilisant le même argumentaire que dans le cas unaire. Ainsi, $\rho(v) = W_0 W_0^{-1} = \epsilon \in L(\mathcal{A}_0)$.

Considérons un nœud interne v de VT, étiqueté par x_i avec $Q_i = \forall$. Soit v_1, v_2 les fils de v, avec v_1 correspondant à x_i vrai, et v_2 à x_i faux. Par hypothèse d'induction, $\rho(v_1)$ et $\rho(v_2)$ sont dans $L(\mathcal{A}_{i-1})$. Ainsi,

$$\begin{aligned}\rho(v) = T^{-1}(v)W_i F^{-1}(v) &= T^{-1}(v)W_{i-1} b W_{i-1} F^{-1}(v) \\ &= T^{-1}(v)T(v_1)\rho(v_1)F(v_1)bT(v_2)\rho(v_2)F(v_2)F^{-1}(v) \\ &= xt_i\rho(v_1)F(v_1)bT(v_2)\rho(v_2)xf_i\end{aligned}$$

Nous avons utilisé les équations $T^{-1}(v)T(v_1) = xt_i$ pour le fils 'positif' v_1 de v et $F^{-1}(v)F(v_2) = xf_i$ pour le fils 'négatif' v_2 de v. De plus, $F(v_1)bT(v_2) = F(v)bT(v) \in L(S_{n-i})$ puisque les indices des variables fausse de $\sigma(v_1)$ et des variables vrais de $\sigma(v_2)$ forment une partition de $\{i+1, \ldots, n\}$. Cela montre que $\rho(v) \in L(\mathcal{A}_i)$.

Considérons un nœud interne v étiqueté par x_i avec $Q_i = \exists$. Supposons par symétrie que v_1 est le fils de v dans VT (c'est à dire que x_i est vrai). Par hypothèse de récurrence, nous savons que $\rho(v_1)$ est dans $L(\mathcal{A}_{i-1})$. Il est facile de montrer que $\rho(v) \in L(\mathcal{A}_i)$ en utilisant :

$$\begin{aligned}\rho(v) = T^{-1}(v)W_i F^{-1}(v) &= T^{-1}(v)W_{i-1} F^{-1}(v) \\ &= T^{-1}(v)T(v_1)\rho(v_1)F(v_1)F^{-1}(v) \\ &= xt_i\rho(v_1)\end{aligned}$$

Cela termine la récurrence.

Pour la réciproque, les arguments sont les mêmes Un mot $W = W_n$ de $\mathcal{A} = \mathcal{A}_n$ défini des sous mots $\rho(v)$ de $L(\mathcal{A}_i)$ comme ci dessus, étiquetés par $T^{-1}(v)W_i F^{-1}(v)$. Pour chaque feuille v, cela signifie que $\sigma(v)$ satisfait exactement un littéral par clause. □

Le théorème 19 page 182 montre facilement que le problème est également PSPACE-complet dans le cas de nMSC et de nMSC-graphe. Des arguments similaires peuvent être utilisés pour le cas où G est un MSC-graphe et non un nMSC-graphe, en utilisant un nombre polynomial de processus [GM02]. Le tableau complet de la complexité de l'appartenance est donné ci dessous. Notons que les algorithmes polynomiaux proviennent d'algorithmes de programmation dynamique.

M	G	mot	MSC
Plat	Nested	P	NP-complet
Plat	Nested	P	NP-complet
Nested	Plat	P	PSPACE-complet
Nested	Nested	PSPACE-complet	PSPACE-complet

FIG. 5.10 – Complexité du problème d'acceptance.

5.3 Voir plus loin ?

Ainsi, nous avons expliqué comment réduire la taille des MSC-graphes avec deux techniques générales, et donné des algorithmes de vérification pour ces cas, que ce soit la projection ou la compression.

Néanmoins, comme nous l'avons remarqué plus tôt, il faudrait également réduire la représentation régulière pour les CFM, qui ont pourtant un fort potentiel d'avoir une représentation plus compacte que l'automate des exécutions. En effet, cet automate contient toutes les linéarisations b-bornées acceptées par le CFM, alors que des représentants suffiraient. C'est ainsi vers les réductions d'ordre partiel [Pel98] qu'il faudrait se tourner pour calculer des représentations plus compactes. Cumulé avec une borne existentielle sur les canaux (pour laquelle il faudrait disposer d'un algorithme de calcul approché), cela pourrait donner de bons résultats.

Conclusion et Perspectives

Nous concluons cette thèse en simplifiant le diagramme de comparaison des MSC-graphes avec les automates communicants aux classes vraiment intéressantes (MSG et CFM n'étant là qu'en tant que bornes), voir la figure 5.11.

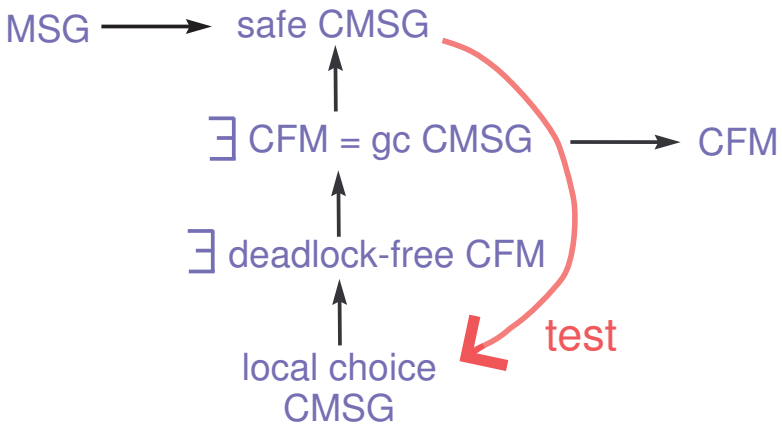


FIG. 5.11 – Les systèmes existentiellement bornés

Si le théorème 5 page 84 donne une information théorique très importante -l'equivalence de formalismes de communication par messages-, il est difficile à utiliser en pratique dû à la complexité des algorithmes sous-jacents. En pratique, pour avoir des algorithmes d'une complexité raisonnable, il faut travailler avec une représentation abstraite du système (voir Chapitre 5). L'abstraction via projection semble à ce propos beaucoup plus utilisable que

la compression, qui peut mener à des algorithmes rapides, mais nettement plus compliqués à écrire. Ainsi, le théorème 18 page 176 -un algorithme quadratique de recherche de motif pour les MSCs compressés- possède une preuve aussi compliquée à obtenir que celle du théorème 5, mais le résultat est nettement moins joli, mathématiquement parlant. Enfin, l'utilisation d'un formalisme de logique temporelle (template MSC) pourrait amener à des outils aussi répandus que ceux de la logique LTL pour les mots.

Le théorème 11 page 129 -algorithme quadratique de model-checking négatif des MSC-graphes à choix local- est la troisième grosse difficulté de cette thèse, et contrairement au théorème 18, donne un résultat très pratique. Cela assure les MSC-graphes à choix local de la praticabilité des algorithmes, qui s'approchent beaucoup de ceux pour les automates finis. De plus, la projection et la compression sont orthogonales et peuvent être combinées.

Concernant les perspectives, nous avons déjà mentionnées celles en rapport avec chaque chaptitre dans les sections 'voir plus loin ?'. Nous donnons ici des perspectives plus générales.

Le plus gros travail à faire est d'implémenter les résultats les plus prometteurs, pour expérimenter la complexité réelle de ces algorithmes (c'est ce qui commence à être fait avec les templates MSCs). De plus, il semble clair que les automates communicants sans blocage sont plus proches de la pratique que les automates communicants généraux. A ce titre, ils sont très loin de nous avoir révéler tous leurs secrets. Par exemple, nous ne savons toujours par quelle classe de CMSC-graphes correspond à cette classe d'automate communicant existentiellement borné, malgré de nombreuses tentatives. En regardant encore plus loin, nous connaissons encore moins de résultats concernant les systèmes ouverts, systèmes généralement analysables sous forme de jeux.

Index

Table des figures

Bibliographie

[ACE+03] Rajeev Alur, Swarat Chaudhuri, Kousha Etessami, Sudipto Guha, and Mihalis Yannakakis. Compression of partially ordered strings. In *CONCUR 2003 - Concurrency Theory, 14th International Conference, Marseille, France, September 3-5, 2003, Proceedings*, volume 2761 of *Lecture Notes in Computer Science*, pages 42–56. Springer, 2003.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2) :183–235, 1994.

[AEY00] Rajeev Alur, Kousha Etessami, and Mihalis Yannakakis. Inference of message sequence charts. In *Proceedings of the 22nd International Conference on Software Engineering, Limerick (Ireland)*, pages 304–313. ACM, 2000.

—— A characterization is given for finite sets of MSCs which are implementable into CFM (without control data), with or without deadlocks.

[AEY01] Rajeev Alur, Kousha Etessami, and Mihalis Yannakakis. Realizability and verification of MSC graphs. In *Proceedings of the 28th International colloquium on Automata, Languages and Programming (ICALP'01), Crete (Greece) 2001*, number 2076 in Lecture Notes in Computer Science, pages 797–808. Springer, 2001.

—— This paper considers the problem of implementing an MSC-Graph without adding control data. This implementation gives a communicating finite state machine, either with and without deadlocks. It generalizes [AEY00].

[AHP96] Rajeev Alur, Gerard H. Holzmann, and Doron A. Peled. An analyzer for message sequence charts. *Software Concepts and Tools*, 17(2) :70–77, 1996.

[AJ96] Parosh Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2) :91–101, 1996.

[AKY99] Rajeev Alur, Sampath Kannan, and Mihalis Yannakakis. Communicating hierarchical state machines. In *Automata, Languages and Programming, 26th International Colloquium, ICALP'99, Prague, Czech Republic*, number 1644 in Lecture Notes in Computer Science, pages 169–178. Springer, 1999.

—— An extension of [AY98] considering automata able to communicate through FIFO message queues.

[AMP98] Rajeev Alur, Kenneth L. McMillan, and Doron Peled. Deciding global partial-order properties. In *Automata, Languages and Programming, 25th International Colloquium, ICALP'98, Aalborg, Denmark*, volume 1443 of *Lecture Notes in Computer Science*, pages 41–52. Springer, 1998.

[APP95] Rajeev Alur, Doron Peled, and Wojciech Penczek. Model-checking of causality properties. In *Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science (LICS '95)*, pages 90–100. IEEE, 1995.

[AY98] Rajeev Alur and Mihalis Yannakakis. Model checking of hierarchical state machines. In ACM, editor, *SIGSOFT '98, Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering, Lake Buena Vista, Florida, USA*, pages 175–188, 1998.

—— The authors consider a model of automaton where (SLP) compression is allowed. In other words, one can reuse macros already defined. They show that the model checking against an LTL formula is still polynomial in the size of the compressed automaton.

[AY99] Rajeev Alur and Mihalis Yannakakis. Model checking of message sequence charts. In *Proceedings of the 10th International Conference on Concurrency Theory CONCUR'99,*

Eindhoven (The Netherlands), number 1664 in Lecture Notes in Computer Science, pages 114–129. Springer, 1999.

—— Same result as [MP99] with the same class of regular MSC-Graph considered (called bounded MSC-graph here).

[BBF+01] B. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and P. Schnoebelen. *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer, 2001.

—— The LSV book for Model-Checking.

[BH99] Ahmed Bouajjani and Peter Habermehl. Symbolic reachability analysis of fifo-channel systems with non regular sets of configurations. *Theoretical Computer Science*, 221(1-2) :211–250, 1999.

[BJR97] G. Booch, I. Jacobson, and J. Rumbaugh. *Unified Modeling Language User Guide*. Addison-Wesley, 1997.

[BL04] Benedikt Bollig and Martin Leucker. Message-passing automata are expressively equivalent to EMSO logic. In *Proceedings of the 15th International Conference on Concurrency Theory (CONCUR'04)*, number 3170 in Lecture Notes in Computer Science, pages 146–160, London, UK, 2004. Springer.

—— Here, EMSO means a weaker version of EMSO, where only the message order and immediate order on processes is allowed. It is unlikely that the same result holds with the (not only immediate) process order.

[BM03] Nicolas Baudru and Rémi Morin. Safe implementability of regular message sequence chart specifications. In *Proceedings of the ACIS Fourth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD'03), October 16-18, 2003, Lübeck, Germany*, pages 210–217. ACIS, 2003.

[Büc60] J.R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. Internat. Congr. on Logic, Methodology*

and Philosophy of Science, pages 1–11. Stanford Univ. Press, Stanford, CA, 1960.

[BZ83] D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2) :323–342, 1983.

—— This paper considers automata communicating through unbounded FIFO buffers. This paper does not show the equivalence with Turing machines, which was only shown in an IBM internal report of 1982.

[CDHL00] Benoît Caillaud, Philippe Darondeau, Loïc Hélouët, and Gilles Lesventes. HMSCs as partial specifications... with PNs as completions. In *MOVEP*, 2000.

—— The authors assume that an MSC-graph specification is not complete. Hence, in order to realize it, a bigger model is allowed. Here, they consider Petri Net as the bigger model.

[CGP00] Edmund Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 2000.

[CLR99] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to algorithms*. MIT Press, Cambridge, Massachusetts, 1999.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2001.

[CMZ89] Robert Cori, Yves Métivier, and Wiesław Zielonka. Asynchronous mappings and asynchronous cellular automata. Tech. rep. 89-97, LABRI, Univ. Bordeaux, 1989.

[Dal00] Víctor Dalmau. *Computational Complexity of Problems over Generalized Formulas*. PhD thesis, Universitat politécnica de Catalunya (UPC), 2000.

—— PhD thesis considering the dichotomy theorem and claim of Schaefer [Sch78].

[DG02] Volker Diekert and Paul Gastin. LTL is expressively complete for Mazurkiewicz traces. *Journal of Computer and System Sciences*, 64 :396–418, 2002.

[DG04] Volker Diekert and Paul Gastin. Pure future local temporal logics are expressively complete for Mazurkiewicz traces. In *Proceedings of the 6th Latin American theoretical INformatics Symposium (LATIN'04)*, number 2976 in Lecture Notes in Computer Science, pages 232–241. Springer, 2004.

——— The paper settles a long standing problem for Mazurkiewicz traces : the pure future local temporal logic defined with the basic modalities exists-next and until is expressively complete. The analogous result with a global interpretation was solved some years ago by Thiagarajan and Walukiewicz (1997) and in its final form without any reference to past tense constants by Diekert and Gastin (2000). Each, the (previously known) global or the (new) local result generalizes Kamp's Theorem for words, because for sequences local and global viewpoints coincide. But traces are labelled partial orders and then the difference between an interpretation globally over cuts (configurations) or locally at points (events) is significant. For global temporal logics the satisfiability problem is non-elementary (Walukiewicz 1998), whereas for local temporal logics both the satisfiability problem and the model checking problem are solvable in PSPACE (Gastin and Kuske 2003) as in the case of words. This makes local temporal logics much more attractive.

[DH99] Werner Damm and David Harel. LSCs : Breathing Life into Message Sequence Charts. In *Proc. 3rd IFIP Int. Conf. on Formal Methods for Open Object-Based Distributed Systems (FMOODS'99)*, pages 293–312. Kluwer Academic Publishers, 1999.

[Dic13] E. Dickson. Perfect numbers. *American Journal of Mathematics*, 35 :413–423, 1913.

[DM95] Volker Diekert and Anca Muscholl. Construction of asynchronous automata. In V. Diekert and G. Rozenberg, editors, *Book of Traces*, pages 249–267. World Scientific, Singapore, 1995.

[DR95] V. Diekert and G. Rozenberg. *Book of Traces*. World Scientific, Singapore, 1995.

[EM93] Werner Ebinger and Anca Muscholl. Logical definability on infinite traces. In *Proceedings of the ICALP 1993*, number 700 in Lecture Notes in Computer Science, pages 335–346. Springer, 1993. Journal version in Theoretical Computer Science 154, pp. 67-84, 1996.

[Für80] Martin Fürer. The complexity of the inequivalence problem for regular expressions with intersection. In *Proceedings of the ICALP 1980*, volume 85 of *Lecture Notes in Computer Science*, pages 234–245. Springer, 1980.

[FS01] Alain Finkel and Philippe Schnoebelen. Well-structured transition systems everywhere ! *Theoretical Computer Science*, 256(1-2) :63–92, 2001.

[Gen04] Blaise Genest. Compositionality everywhere, the end of MSC-graphs ?, 2004.

—— Internal LIAFA report, prior to the paper [GKM04].

[GHM03] Blaise Genest, Loïc Hélouët, and Anca Muscholl. High-level message sequence charts and projections. In *Proceedings of the CONCUR 2003 : Marseilles, France*, number 2761 in Lecture Notes in Computer Science, pages 308–322. Springer, 2003.

—— This paper considers abstraction for MSC-graphs, by means of projection of events. More formally, one can delete some sends without deleting the corresponding receives (or vice versa), hence yielding a complicated partial-order relation. It does not always corresponds to an MSC-graph, but it corresponds to a safe CMSC-graph. An algorithm is given to reconstruct an MSC-graph when it is possible, together with model checking against an MSC-graph.

[GJ78] M. Garey and D. Johnson. *Computers and Intractability : A Guide to the Theory of NP-completeness*. Freeman, San Francisco, 1978.

[GK03] P. Gastin and D. Kuske. Satisfiability and model checking for MSO-definable temporal logics are in PSPACE. In *Procee-*

dings of the 14th International Conference on Concurrency Theory (CONCUR'03), number 2761 in Lecture Notes in Computer Science, pages 222–236. Springer, 2003.

[GKM04] Blaise Genest, Dietrich Kuske, and Anca Muscholl. A kleene theorem for a class of communicating automata with effective algorithms, 2004.

—— Work in progress, follow up of [Gen04].The restriction considered here is an existential bound on the channel buffer. With that restriction, it is shown that CFMs, globally cooperative CMSC-graphs and MSO are expressively equivalent.

[GM02] Blaise Genest and Anca Muscholl. Pattern matching and membership for hierarchical message sequence charts. In *Proceedings of the LATIN 2002 : Theoretical Informatics, 5th Latin American Symposium, Cancun, Mexico*, number 2286 in Lecture Notes in Computer Science, pages 326–340. Springer, 2002.

—— The only paper considering hierarchy for MSCs/MSC-graphs (HMSCs for hierarchical MSCs). Using hierarchy by means of macro can be seen as using (SLP) compression. pattern matching and membership are considered.

[GM03] Blaise Genest and Anca Muscholl. Don't choose globally, implement easily, 2003.

—— Internal report of LIAFA, giving a characterization for local choice MSC-graphs.

[GMMP04] Blaise Genest, Marius Minea, Anca Muscholl, and Doron Peled. Specifying and verifying partial order properties using templates MSCs. In *Proceedings of the Fossacs 2004, Barcelona , Spain*, number 2987 in Lecture Notes in Computer Science, pages 195–210. Springer, 2004.

—— This paper introduces a weak form of global temporal logic on MSCs using explicit gaps. Model Checking against an MSC-graph and satisfiability are considered.

[GMP01] Elsa Gunter, Anca Muscholl, and Doron Peled. Compositional message sequence charts. In *Proceedings of the Tools and Algorithms for the Construction and Analysis of Systems, 7th International Workshop (TACAS'01)*, number 2031 in Lecture Notes in Computer Science, pages 496–511. Springer, 2001. Journal version to appear in the International Journal on Software Tools for Technology Transfer.

——— This paper introduces the Compositional MSCs/MSC-graphs (CMSC/CMSC-Graph), together with safe CMSC-graph (they call them realizable CMSC-graphs, which is a really nasty name). These notions proved to be really strong, and were reused for restriction/generalization later.

[GMP04] Blaise Genest, Anca Muscholl, and Doron Peled. Message sequence charts. In *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 537–558. Springer, 2004.

——— This survey gathers results contained into [MP99, GMP01, GMSZ02].

[GMSZ02] Blaise Genest, Anca Muscholl, Helmut Seidl, and Marc Zeitoun. Infinite-state high-level MSCs : Model-checking and realizability. In *Proceedings of the 29th International colloquium on Automata, Languages and Programming (ICALP'02), Malaga (Spain), 2002*, number 2380 in Lecture Notes in Computer Science, pages 657–668. Springer, 2002. journal version to appear in JCSS.

——— This paper introduces a notion generalizing regular MSC-graph, namely global-cooperativity (gc-MSC-graphs). Subclasses of gc-MSC-graphs are also studied, namely local-choice and local-cooperativity. These three classes impose no universal bound on the message channels, hence are not regular. The complexity for model checking is given, together with a deadlock free implementation of any local-choice MSC-graphs.

[Har87] David Harel. Statecharts : A visual formulation for complex

systems. Science of Computer Programming, 8(3) :231–274, 1987.

[HJ00] Loïc Hélouët and Claude Jard. Conditions for synthesis of communicating automata from HMSCs. In *5th International Workshop on Formal Methods for Industrial Critical Systems*, Berlin, 2000.

—— This paper focuses on giving sufficient conditions to be able to implement an MSC-graph. They define local-choice MSC-Graph, and show that it is not sufficient for implementability. The reason is that control data is not allowed in the CFM implementation.

[HK99] D. Harel and H. Kugler. Synthesizing state-based object systems from lsc specifications. Technical report, Weizmann Institute of Science, Rehovot, Israel, Octobre 1999.

[HKV97] David Harel, Orna Kupferman, and Moshe Vardi. On the complexity of verifying concurrent transition systems. In *CONCUR '97 : Concurrency Theory, 8th International Conference, Warsaw, Poland*, number 1243 in Lecture Notes in Computer Science, pages 258–272. Springer, 1997.

[HM00] Loïc Hélouët and Pierre Le Maigat. Decomposition of Message Sequence Charts. In *Proceedings of the 2nd Workshop on SDL and MSC (SAM2000), Col de Porte, Grenoble*, pages 46–60, 2000.

—— One of the first definition of atoms. Moreover, a nice algorithm for checking atomicity is given by testing the strong connexion of a graph (connection graph), which was reused later.

[HMNK+04] Jesper G. Henriksen, Madhavan Mukund, K. Narayan Kumar, Milind Sohoni, and P.S. Thiagarajan. A Theory of Regular MSC Languages. *To Appear in : Information and Computation.*, pages –, 2004.

—— Survey paper gathering works done in [HMNKT00a, HMNKT00b, MNKS00]. More precisely,

it defines regular MSC languages, CFM languages, MSO-definable language and shows that their restrictions to universally bounded languages are equivalent. Moreover, regular MSG languages equals their restrictions on b-bounded and finitely generated languages.

[HMNKT00a] Jesper G. Henriksen, Madhavan Mukund, K. Narayan Kumar, and P.S. Thiagarajan. On message sequence graphs and finitely generated regular MSC languages. In *Proceedings of the 27th International colloquium on Automata, Languages and Programming (ICALP'00), Geneva (Switzerland), 2000*, number 1853 in Lecture Notes in Computer Science, pages 675–686. Springer, 2000.

—— The authors show that regular MSC-graphs are equivalent to finitely generated regular sets of MSCs. Moreover, it is undecidable to check whether an MSC-Graph is equivalent to some regular MSC-Graph. One of the first definition of atoms.

[HMNKT00b] Jesper G. Henriksen, Madhavan Mukund, K. Narayan Kumar, and P.S. Thiagarajan. Regular collections of message sequence charts. In *Proceedings of the 25th Symposium on Mathematical Foundations of Computer Science (MFCS'00), Bratislava (Slovakia), 2000*, number 1893 in Lecture Notes in Computer Science, pages 405–414. Springer, 2000.

—— This paper defines regular MSC languages, CFM languages, MSO-definable language and shows that their restrictions to b-bounded languages are equivalent.

[HU79] J. E. H. Hopcroft and Jeffrey D. Ullman. *Introduction to Automate Theory, Languages, and Computation.* Reading, Menlo Park ,London. Addison-Wesley Publishing Company, 1979.

[HZJ03] Loïc Hélouët, Marc Zeitoun, and Claude Jard. Covert channels detection in protocols using scenarios. In *SPV'03 : Security Protocols Verification, Marseilles*, 2003.

[itu96] Itu-ts. itu-ts recommendation z.120 : Message sequence chart (msc). itu-ts, geneva, september 1996., 1996.

—— The standard for MSCs and HMSCs.

[Kle56] Stephen Kleene. Representation of events in nerve nets and finite automata. In Shannon and McCarthy, editors, *Automata Studies*, pages 3–42. Princeton University Press, Princeton, N.J., 1956.

[Kus03] Dietrich Kuske. Regular sets of infinite message sequence charts. *Information and Computation*, 187 :80–109, 2003.

—— This paper features an encoding of universally bounded communication with traces.

[LL90] Leslie Lamport and Nancy Lynch. Distributed computing : models and methods. In *Handbook of Theoretical Computer Science*, volume B, pages 1157–1200. Elsevier, Amsterdam, 1990.

[LL95] P. Ladkin and S. Leue. Comments on a proposed semantics for basic Message Sequence Charts. *The Computer Journal*, 37(9), Janvier 1995.

[LM04] Markus Lohrey and Anca Muscholl. Bounded MSC communication. *Information and Computation*, 189 :135–263, 2004.

—— Definition of the universal and existential bounds, as well as algorithms to calculate the bounds.

[LMW04] Stefan Leue, Richard Mayr, and Wei Wei. A scalable incomplete test for the boundedness of uml rt models. In *Proceedings of the 28th Tools and Algorithms for the Construction and Analysis of Systems (TACAS'04), Barcelona (Spain) 2004*, number 2988 in Lecture Notes in Computer Science, pages 327–341. Springer, 2004.

[Loh02] Markus Lohrey. Safe realizability of high-level message sequence charts. In *Proceedings of the Concurrency Theory, 13th International Conference (CONCUR'02)*, number 2421 in Lecture Notes in Computer Science, pages 177–192. Springer, 2002.

—— This paper closes the open gap of [AEY01] between PSPACE-hard and EXPSPACE for safe realizability

of regular MSC-Graphs. It shows that this problem is EXPSPACE-complete. Moreover, it extends the result to gc-MSC-graphs.

[LWZ90] Hai-Ning Liu, Celia Wrathall, and Kenneth Zeger. Efficient solution to some problems in free partially commutative monoids. *Information and Computation*, 89(2) :180–198, 1990.

[Lyn96] Nancy Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, 1996.

[Mad01] P. Madhusudan. Reasoning about sequential and branching behaviours of message sequence graphs. In *Proceedings of the 28th International colloquium on Automata, Languages and Programming (ICALP'01), Crete (Greece) 2001*, number 2076 in Lecture Notes in Computer Science, pages 809–820. Springer, 2001.

[Mau95] S. Mauw. The formalization of Message Sequence Charts. Technical report, Eindhoven University of Technology, 1995.

—— One of the first paper formalizing MSCs.

[May84] E. Mayr. An algorithm for the general Petri net reachability problem. *SIAM Journal of Computing*, 13 :441–459, 1984.

[Maz77] A. Mazurkiewicz. Concurrent program schemes and their interpretations. In Aarhus University, editor, *DAIMI Rep. PB78*, 1977.

[MM01] P. Madhusudan and B. Meenakshi. Beyond message sequence graphs. In *Proceedings of the FST TCS 2001 : Foundations of Software Technology and Theoretical Computer Science, 21st Conference, Bangalore (India)*, number 2245 in Lecture Notes in Computer Science, pages 256–267. Springer, 2001.

—— The Authors show that one can model-check MSO against safe CMSG (they define the term 'CMSG' with the definition of safe CMSG). The basic idea is that safe CMSGs give an existential bound b for representatives. One can construct an automaton closed under commutation recognizing b-bounded executions which satisfy the MSO formula. It generalizes [Mad01].

[MNKS00] Madhavan Mukund, K. Narayan Kumar, and Milind Sohoni. Synthesizing distributed finite-state systems from MSCs. In *Proceedings of the 11th International Conference on Concurrency Theory CONCUR'00, University Park, USA*, number 1877 in Lecture Notes in Computer Science, pages 521–535. Springer, 2000.

—— Implementation of regular MSC-graphs into deterministic CFMs (MPA).

[MNKT03] Madhavan Mukund, K. Narayan Kumar, and P.S. Thiagarajan. Netcharts : Bridging the gap between HMSCs and executable specifications. In *Proceedings of the 27th International Conference on Concurrency Theory (CONCUR'03), Marseilles (France), 2003*, Lecture Notes in Computer Science, pages 293–307. Springer, 2003.

—— The authors want to get rid of finite generation of MSC-Graphs. They define MSC-languages with safe Petri Nets as composition. Moreover, they get implementability for free. Hence, Netcharts are a subclass of CMSG.

[Mor02] Rémi Morin. Recognizability vs realizability and mso-definability in unbounded high level message sequence charts. In *Proceedings of the 19th Symposium on Theoretical Aspects of Computer Science (STACS'02, Antibes, France)*, number 2285 in Lecture Notes in Computer Science, pages 523–534. Springer, 2002.

—— An algebraic look into MSC languages. More precisely, it compares recognizability, regularity, rationality and MSO-definability of infinite MSC languages, extending results of [HMNK$^+$04]. One result is that one can check whether a gc-MSC-Graph (c-HMSC) is implementable into a non-FIFO CFM.

[MP91] Zohar Manna and Amir Pnueli. *The temporal logic of reactive and concurrent systems*. Springer, Berlin-Heidelberg-New York, 1991.

[MP99] Anca Muscholl and Doron Peled. Message sequence graphs and decision problems on Mazurkiewicz traces. In *Proceedings of the 24thSymposium on Mathematical Foundations of Computer Science (MFCS'99), Szklarska Poreba (Poland) 1999*, number 1672 in Lecture Notes in Computer Science, pages 81–91. Springer, 1999.

—— Definition of strong loop-connectedness, which led to the first decidability result for Model Checking between MSC-Graphs.

[MP01] Anca Muscholl and Doron Peled. From finite state communication protocols to high-level message sequence charts. In *Proceedings of the 28th International colloquium on Automata, Languages and Programming (ICALP'01), Crete (Greece) 2001*, number 2076 in Lecture Notes in Computer Science, pages 720–731. Springer, 2001.

—— The authors give a syntactic characterization of the CFMs that one can express by MSC-graphs.

[MPS98] Anca Muscholl, Doron Peled, and Zhendong Su. Deciding properties of message sequence charts. In *Proceedings of the 1st International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'98), Lisbon, Portugal, 1998*, number 1378 in Lecture Notes in Computer Science, pages 226–242. Springer, 1998.

—— The authors consider a variant of Model Checking between MSC-Graphs with a different (gap) semantics. More precisely, the property is considered to be incomplete, that is, the system can contain events that are not specified by the property. In this case, model checking is decidable, unlike the case with usual semantics.

[MR97] S. Mauw and M.A. Reniers. High-level message sequence charts. In *SDL'97 : Time for Testing - SDL, MSC and Trends. Proceedings of the 8th SDL Forum, Evry (France) 1997*, pages 291–306, 1997.

—— Definition of the HMSCs.

[MS94] Alain J. Mayer and Larry J. Stockmeyer. Word problems-this time with interleaving. *Information and Computation*, 115(2) :293–311, 1994.

[MST97] Masamiki Miyazaki, Ayumi Shinohara, and Masayuki Takeda. An improved pattern matching algorithm for strings in terms of straight-line programs. In *Combinatorial Pattern Matching, 8th Annual Symposium, CPM '97, Aarhus, Denmark*, number 1264 in Lecture Notes in Computer Science, pages 1–11. Springer, 1997.

—— The authors improve the pattern matching for words compressed in SLPs to quadratic time in size of text and pattern.

[Mul63] D.E. Muller. Infinite sequences and finite machines. In *Proc. of the 4th Ann. IEEE Symposium on Switching Circuit Theory and Logical Design*, pages 3–16, 1963.

[Och85] Edward Ochmański. Regular behaviour of concurrent systems. *Bulletin of EATCS*, 27 :56–67, 1985.

[Pap94] C.H Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

—— A well known book.

[Pel93] Doron Peled. All from One, One for All : on Model Checking Using Representatives. In *Proceedings of Computer Aided Verification, 5th International Conference, CAV '93, Elounda, Greece*, number 697 in Lecture Notes in Computer Science, pages 409–423. Springer, 1993.

—— First paper with the idea of using representatives instead of whole executions.

[Pel98] Doron Peled. Ten years of partial order reduction. In *Computer Aided Verification, 10th International Conference, CAV'98, Vancouver, BC, Canada, June 28 - July 2, 1998, Proceedings*, volume 1427 of *Lecture Notes in Computer Science*, pages 17–28. Springer, 1998.

[Pel00] Doron Peled. Specification and verification of message sequence charts. In *Proceedings of Formal Techniques for Distributed System Development, FORTE/PSTV 2000, Pisa, Italy*, pages 139–154, 2000.

—— Survey about Model-Checking against Partial Order Logic. Idea that if the Logic is closed under commutation, then we can choose representatives for the MSG.

[Pla94] Wojciech Plandowski. Testing equivalence of morphisms on context-free languages. In *Algorithms - ESA '94, Second Annual European Symposium*, pages 460–470, 1994.

—— Plandowsky shows how to test equality of two compressed words in SLP in PTIME. The idea is that either patterns of the words are isolated, then there is few patterns and one can check them one by one, or they form an arithmetic progression, and then one can use arithmetics to compute the solution more efficiently.

[Pnu77] Amir Pnueli. The temporal logic of programs. In IEEE Computer Society Press, editor, *Proceedings of the 18th International IEEE symposium on the foundation of Computer Science (FOCS'77)*, pages 46–57, 1977.

[Pos46] E. Post. A variant of a recursively unsolvable problem. *Bulletin of AMS*, 52 :264–268, 1946.

[PP92] Wuxu Peng and S. Purushotaman. Analysis of a class of communicating finite state machines. *Acta Informatica*, 29 :499–522, 1992.

[PP02] Dominique Perrin and Jean-Eric Pin. *Infinite Words*. Academic Press, 2002.

[Ryt99] Wojciech Rytter. Algorithms on compressed strings and arrays. In *SOFSEM '99, Theory and Practice of Informatics, 26th Conference on Current Trends in Theory and Practice of Informatics, Milovy, Czech Republic*, number 1725 in Lecture Notes in Computer Science, pages 48–65. Springer, 1999.

—— Rytter uses the idea of Plandowsky [Pla94]. That

is, one can check in PTIME the equality of two words compressed by SLPs (close to gzip compressions). Rytter extends this result to pattern matching of two such words, plus membership for a compressed word in an uncompressed automaton (both in PTIME).

[SC02] Bikram Sengupta and Rance Cleaveland. Triggered Message Sequence Charts. In *SIGSOFT 2002/FSE-10*. ACM Press, 2002.

—— Definition of Triggered MSCs, example of WSR system. There is a nice idea of describing system by (parallel) composition of several local patterns, then restricting the global behavior. Anyway, the formalism is ad hoc for implementation, hence weird (postcondition before precondition). It is so because the semantic is per process instead than the MSC composition semantic.

[Sch78] Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 8th Annual ACM Symposium on Theory of Computing (STOC '78)*, pages 216–226, 1978.

—— This paper concerns dichotomy of SAT between PTIME and NP-complete problem. Schaefer claims the same result for Quantified SAT between PTIME and PSPACE-complete.

[Sch02] Philippe Schnoebelen. Verifying lossy channel systems has non primitive recursive complexity. *Information Processing Letter*, 83 (5) :251–261, 2002.

[Tho90] Wolfgang Thomas. On logical definability of trace languages. In *workshop of the ESPRIT BRA No 3166 : Algebraic and Syntactic Methods in Computer Science (ASMICS) 1989, Report TUM-I9002, Technical University of Munich*, pages 172–182, 1990.

[TW02] P.S. Thiagarajan and Igor Walukiewicz. An expressively complete linear time temporal logic. *Information and Computation*, 179(2) :230–249, 2002.

[usb96] Usb 1.1 specification, 1996. —— available at `http://www.usb.org/developers/docs/usbspec.zip`.

[Wal98] Igor Walukiewicz. Difficult Configurations - On the Complexity of LTrL. In *Proceedings of the 25th International colloquium on Automata, Languages and Programming (ICALP'98), 1998*, number 1443 in Lecture Notes in Computer Science, pages 140–151. Springer, 1998.

—— First paper where the idea of counters for Turing machine appears. It is used to encode exponential space TM with polynomial memory.

[Zie87] Wieslaw Zielonka. Note on finite asynchronous automata R.A.I.R.O. *Informatique Théorique et Applications*, 21 :99–135, 1987.

Dans cette thèse, nous nous intéressons à l'expressivité et la vérification de diverses structures communicantes par passage de messages. Ces structures génèrent des exécutions sous forme de diagrammes de séquence (Message Sequence Charts MSC). Ce sont les automates communicants, les (C)MSC-graphes et les formules MSO. Les questions de vérification qui nous intéressent sont le model-checking, ainsi que l'implémentation des structures à choix globaux en automates communicants (choix locaux). Les contributions de cette thèse sont les suivantes :

1- Nous exhibons une restriction assez expressive qui empêche l'indécidabilité de nombreux problèmes. Plus précisément, dés que les communications d'un langage peuvent être réordonner pour qu'il y ait toujours au plus k messages en transit, nous montrons comment représenter ce langage par un ensemble régulier.

2- Sous cette restriction, nous montrons que les automates communicants, les CMSC-graphes globalement coopératifs et les formules MSO sont expressivement équivalents. Cela permet également une algorithmique unifiée pour le model-checking de ces problèmes.

3- Nous proposons d'autres méthodes plus rapides dans des sous cas.

Mots clés : Message Sequence Charts, Concurrence, Scénario, Model-checking, Implementation, Automates, Hiérarchie, Complexité.