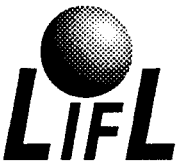


Laboratoire d'Informatique Fondamentale de Lille

Numéro d'ordre : 2408

THÈSE

Nouveau Régime

Présentée à

L'UNIVERSITÉ DES SCIENCES ET TECHNOLOGIES DE LILLE

Pour obtenir le titre de

DOCTEUR EN INFORMATIQUE

par

Isabelle RYL

LANGAGES DE SYNCHRONISATION

Thèse soutenue le 4 décembre 1998, devant la Commission d'Examen :

Président	:	Sophie TISON	Université de Lille 1
Rapporteurs	:	André ARNOLD	Université de Bordeaux I
		Antoine PETIT	E.N.S. Cachan
		Sheng YU	University of Western Ontario
Examinateurs	:	Mireille CLERBOUT	Université de Lille 1
		Yves ROOS	Université de Lille 1

Je tiens à remercier Sophie Tison qui m'a fait l'honneur de présider ce jury.

Qu'André Arnold, Professeur de l'université de Bordeaux 1, Antoine Petit, Professeur de l'École Normale Supérieure de Cachan et Sheng Yu, Professeur de l'university of Western Ontario, soient assurés de toute ma reconnaissance pour avoir accepté d'être rapporteurs de cette thèse. Je suis très touchée du soin avec lequel ils ont lu ce mémoire et je les remercie pour leurs remarques et leurs commentaires constructifs.

Je remercie tout particulièrement Mireille Clerbout et Yves Roos qui m'ont guidée durant la préparation de cette thèse. Ils se sont montrés particulièrement attentifs et disponibles, prodiguant généreusement conseils et suggestions. Ils ont essayé de me communiquer leur rigueur et leur enthousiasme, je leur dois énormément.

Je remercie également Michel Latteux pour l'intérêt qu'il a porté à ce travail, pour ses encouragements et son aide précieuse.

Enfin, je remercie tous ceux qui font de cet endroit un lieu où l'on a plaisir à travailler, tous ceux dont l'aide et l'amitié ne sont jamais comptées, tous ceux sans qui la vie du L.I.F.L. ne serait pas.

Table des matières

Introduction

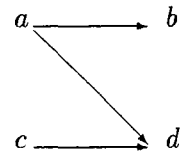
Au cours des quelques décennies d'expansion de l'informatique, il est vite apparu que les traditionnels ordinateurs « *mainframe* » allaient s'avérer inadéquats. Les recherches se sont alors orientées vers le développement de la programmation parallèle. Deux grandes familles de machines parallèles ont émergé, les machines SIMD dont les processeurs sont synchronisés, les machines MIMD dont les processeurs sont asynchrones. Dans cette dernière catégorie, qui nous intéresse ici, plusieurs activités différentes sont exécutées de manière concurrente, la difficulté pour le programmeur consiste évidemment à concevoir les différentes activités et à mettre en œuvre leur coopération.

De nombreux langages de programmation adaptés à la programmation distribuée ont été développés. L'un des principaux problèmes relatifs au développement d'une application distribuée est la synchronisation des différents flots d'exécution. Que ce soit dans le cas d'applications à mémoire partagée ou distribuée, il est souvent nécessaire de procéder à des synchronisations entre les différents acteurs d'un calcul pour mettre en commun des résultats intermédiaires par exemple, ou de protéger l'accès à une ressource partagée, etc ....

La plupart des premiers systèmes parallèles supportaient uniquement les langages de programmation traditionnels auxquels quelques outils de bas niveau permettant la création, la synchronisation et la terminaison de processus parallèles avaient été adjoints. Cette approche résultait de la facilité relative de son implantation et du manque d'outils de haut niveau largement reconnus. Dans la plupart des cas les adjonctions consistent en de simples librairies, cette solution étant peu coûteuse et pratique puisqu'elle n'entraîne pas de modifications du compilateur original du langage. Il est clair que l'utilisation d'outils tels que les sémaphores [24] est assez lourde et n'est pas immédiate pour tous les programmeurs (les sémaphores sont cependant un outil très puissant). De nombreux outils ont été développés parmi lesquels on peut citer les régions critiques conditionnelles [35, 6], les moniteurs [7, 36], les expressions de chemins [8], les CSP de Hoare [37] ou encore plus récemment le principe des transactions et les divers protocoles qui permettent de les manipuler [3, 31] introduits dans le cadre des bases de données. L'accent est souvent mis, dans le développement de nouveaux outils, sur la nécessité de les rendre accessibles à tout programmeur, sur la nécessité d'obtenir des programmes lisibles, faciles à modifier, à comprendre et à corriger.

Les expressions de synchronisation ont été introduites dans ce but : fournir un outil de haut niveau, simple et efficace, d'utilisation aisée. Les expressions de synchronisation sont basées sur l'utilisation des étiquettes. Une étiquette associée à un bloc d'intructions identifie l'exécution du bloc entier, une expression de synchronisation (expressions sur les étiquettes) exprime les contraintes de synchronisation que l'application doit respecter. Il s'agit plus ici de spécifier ce qui doit être synchronisé que d'exprimer comment cela doit être fait. Les opérateurs utilisés pour construire une expression sont la séquence ($\rightarrow$) pour forcer des blocs

d'actions à se dérouler en séquence, la concurrence (||) pour autoriser des blocs d'actions à se dérouler en parallèle, l'exclusion (|) pour exprimer l'exclusion mutuelle, la répétition (*) pour autoriser un bloc d'actions à se dérouler un nombre arbitraire de fois et enfin l'intersection (&) qui permet de combiner les contraintes élémentaires pour en bâtir de plus complexes. Par exemple l'expression $[(a \rightarrow b) \parallel (c \rightarrow d)]\&[(a \rightarrow d) \parallel c \parallel d]$ que nous reprendrons plus loin, désigne des contraintes entre les blocs d'actions étiquetés par a, b, c et d (nous utiliserons par la suite le terme action) qui peuvent être représentées par l'ordre partiel



Les expressions de synchronisation sont faciles à utiliser, il suffit d'étiqueter les blocs d'instructions sur lesquels porte la synchronisation et d'écrire, dans l'en-tête du programme l'expression correspondant au comportement souhaité. L'utilisation des expressions n'entraîne donc pour le programmeur aucune modification du programme source, les contraintes de synchronisation sont de ce fait très faciles à intégrer, à modifier, à corriger. Le modèle a cependant ses limitations et, si Guo donne dans [32] des solutions utilisant les expressions de synchronisation à quelques problèmes classiques tels que les philosophes ou le tampon fini, il semble assez difficile d'implanter le problème des lecteurs/rédacteurs avec priorité à l'aide des expressions de synchronisation (lorsque deux comportements sont autorisés, il n'y a aucune possibilité de donner la priorité à l'un d'entre-eux). Les expressions de synchronisation ne permettent donc pas de résoudre tous les problèmes de synchronisation mais jusquà présent, aucun outil n'est universel (voir [41] pour les limitations des sémaphores).

Un tel outil ne saurait exister sans une sémantique précise. Il est de plus nécessaire de l'implanter et de résoudre quelques problèmes élémentaires tels que décider si deux expressions expriment le même comportement. Pour toutes ces raisons, les langages de synchronisation ont été associés aux expressions de synchronisation.

Un langage de synchronisation associé à une expression e décrit toutes les exécutions possibles respectant les contraintes exprimées par e. Il correspond donc en quelque sorte à l'ensemble des traces d'exécution possibles et s'apparente aux traces de Hoare. Dans la sémantique de l'entrelacement (due en grande partie aux travaux de Hoare [38] et Milner [49]), le parallélisme est réduit au non déterminisme. Ici, les expressions de synchronisation exprimant des contraintes d'exécutions, il est nécessaire de pouvoir distinguer une exécution de deux actions a et b en séquence de l'exécution de ces deux mêmes actions en parallèle, surtout lorsque ces deux actions ne sont pas indépendantes. Un modèle dans lequel les comportements $a \parallel b$ et $ab + ba$ sont équivalents n'est donc pas adapté aux expressions de synchronisation. D'autre part, l'un des postulats fondamentaux de la sémantique de l'entrelacement est l'atomicité des actions. Dans la mesure où l'une des facilités offertes par les expressions de synchronisation est justement la possibilité d'étiqueter par un même label des blocs d'instructions de taille variable, il est clairement impossible de respecter ce postulat.

La solution choisie dans la définition des langages de synchronisation est l'utilisation d'actions atomiques marquant le début et la fin de chaque action utilisée dans une expression de synchronisation, cette solution respecte ainsi les hypothèses relativement fortes sur la nature des événements imposées par la sémantique de l'entrelacement :

« *The actual occurrence of each event in the life of an object should be regarded as an instantaneous or an atomic action without duration. Extended or time-consuming actions should be represented by pair of events, the first denoting its start and the second denoting its finish.* » *C.A.R. Hoare [38]*

À chaque action a apparaissant dans une expression e, correspondent dans le langage de synchronisation associé à e, les lettres a_s (s pour *start*) et a_t (t pour *termination*) qui marquent le début et la fin de l'action a. Il est donc possible de distinguer une réelle exécution en parallèle $a_s b_s a_t b_t$ d'une exécution en séquence $a_s a_t b_s b_t$ ce qui permettra, dans le cas des expressions de synchronisation d'autoriser ou non un début d'action b_s après un début d'action a_s en fonction des contraintes imposées par l'utilisateur.

Il peut sembler que l'atomicité des actions n'impose pas de réelle restriction puisque la durée des actions peut être traitée grâce au « *split* » d'actions en débuts et fins. Castellano, De Michelis et Pomello donnent dans [10] un petit exemple dans lequel une égalité sémantique définie dans la sémantique de l'entrelacement n'est pas préservée par raffinement alors qu'elle l'est dans un modèle basé sur les ordres partiels. D'autre part, van Glabbeeck et Vaandrager montrent dans [59] que cette représentation de la durée des actions dans des modèles entrelacés n'est pas possible dans le cadre général des systèmes concurrents non déterministes et en particulier lorsque l'auto-concurrence est permise. L'approche qui consiste à utiliser un modèle entrelacé avec un « *split* » d'actions en deux parties n'est donc pas universellement utilisable, elle est cependant adaptée dans de nombreux cas et convient tout-à-fait dans le cas des expressions de synchronisation.

Nous allons donc associer à une expression de synchronisation un langage de synchronisation construit sur l'alphabet des débuts et fins d'actions. Il est cependant nécessaire de garder une correcte association des débuts et fins d'actions ; en particulier, dans un mot représentant une exécution, il est nécessaire d'observer une séquence alternée de débuts et fins de chaque action. Un langage possédant cette propriété (équivalente pour les débuts et fins à la propriété des langages dits « bien parenthésés ») est appelé st-langage. Un langage associé à une expression est un st-langage construit par induction sur l'expression à partir de langages finis représentant l'exécution d'une action en utilisant l'union, l'intersection, la concaténation, le shuffle et l'étoile, les langages de synchronisation sont donc réguliers ce qui permet d'utiliser les automates finis pour implanter les expressions de synchronisation.

Les langages de synchronisation ne sont pas seulement un outil associé aux expressions de synchronisation, ils peuvent également être vus comme un modèle du comportement des programmes concurrents. L'informatique théorique entend, entre autres, apporter des méthodes formelles permettant de comprendre, d'analyser et de prouver les programmes pour améliorer leur utilisation. À ce titre, de nombreuses méthodes spécifiques ont été développées pour l'informatique parallèle. On peut considérer les réseaux de Petri, introduits par Petri en 1962 dans sa thèse « Kommunikation mit Automaten », comme des précurseurs de ces modèles. Il existe de nombreux autres modèles comme les systèmes de transitions (voir une synthèse dans [1]), les ordres partiels et les structures d'événements (introduites dans [50]), les traces de Mazurkiewicz (introduites dans [43], voir aussi [23, 22] pour des synthèses) et, bien sûr, tous les modèles basés sur l'entrelacement (une comparaison de ces modèles est proposée dans [58]). Chaque modèle a bien entendu des avantages et des inconvénients, il est bien adapté à un type de problème ou à la vérification de certaines propriétés. Les langages de synchronisation ont l'avantage d'être réguliers, on dispose ainsi de toute la théorie des automates finis,

de la logique qui leur est associée, ... Les limites du pouvoir d'expression des langages de synchronisation sont bien sûr liées à celles des expressions de synchronisation.

Les langages de synchronisation donnent une sémantique aux expressions de synchronisation. Inversement, si l'on considère des langages de synchronisation comme des ensembles de traces d'exécution, une expression peut être vue comme un moyen de présenter un comportement de façon synthétique. Il semble plus évident de comprendre un comportement par la donnée d'une expression que par la donnée d'un ensemble de traces d'exécution. Si l'on disposait d'un algorithme permettant de construire une expression de synchronisation à partir d'un langage ou de quelques mots, on pourrait envisager d'utiliser les expressions de synchronisation pour la mise au point d'applications distribuées après exécution. En effet, le résultat de l'enregistrement d'une exécution est une trace d'exécution souvent assez longue et les outils existant pour l'analyser sont assez peu évolués. Une expression de synchronisation décrivant le « plus petit comportement » (au sens de comportement le plus contraint) compatible avec un ensemble d'exécutions tests serait une bonne approximation du comportement de l'application. Ceci est bien sûr dépendant de la donnée d'un algorithme permettant de construire une expression de synchronisation à partir d'un langage donné ainsi que d'un algorithme de simplification des expressions pour les rendre le plus synthétique possible (trouver une forme normale pour les expressions de synchronisation est un problème ouvert).

Il n'est pas immédiat de trouver une expression de synchronisation associée à un langage donné. En effet, même si ce langage est rationnel, il est nécessaire de décomposer le langage en union, concaténation, étoile mais aussi intersection et shuffle non pas de langages réguliers mais de st-langages réguliers. Une première question à résoudre, avant d'essayer de construire une expression, est de trouver un critère pour décider si un langage donné est un langage de synchronisation. Pour ce faire, Guo, Salomaa et Yu, ont proposé, dans [33, 34], d'utiliser un système de réécriture. L'idée est de trouver un système de réécriture tel qu'un st-langage régulier soit un langage de synchronisation si et seulement s'il est clos par ce système. D'autre part, il serait agréable de pouvoir retrouver le plus petit langage de synchronisation qui contient un mot en calculant sa clôture par un système de réécriture.

On peut remarquer qu'en considérant un langage de synchronisation L et une relation ϱ irréflexive et symétrique (une commutation partielle) qui contient tous les couples (a_s, b_s) et (a_t, b_t) pour a et b distincts, les classes d'équivalence éléments du quotient du langage par la congruence engendrée par ϱ (les traces) contiennent des mots dans lesquels les mêmes occurrences d'actions sont en parallèle et les mêmes occurrences d'actions sont en séquence. En outre, une expression comme $a \parallel b$ autorise d'exécuter les actions a et b en même temps mais n'interdit pas de les exécuter en séquence. Utiliser un système de réécriture ne contenant que des règles du type $a_s b_s \leftrightarrow b_s a_s$ et $a_t b_t \leftrightarrow b_t a_t$ ne suffit donc pas : un langage de synchronisation contenant $a_s b_s a_t b_t$ doit également contenir les mots $a_s a_t b_s b_t$ et $b_s b_t a_s a_t$. Pour obtenir l'ensemble de mots provenant d'une expression de synchronisation donnée, il faut donc ajouter des règles non symétriques, du type $a_s b_t \rightarrow b_t a_s$, qui permettent d'échanger le début et la fin de deux actions distinctes afin de les mettre en séquence (ce qui revient à considérer une relation de semi-commutation que nous appellerons θ contenant les couples (a_s, b_s), (a_t, b_t) et (a_s, b_t) pour tout couple d'actions distinctes). Guo et al. montrent que θ n'est pas suffisante pour obtenir une bonne caractérisation, il est facile de trouver un exemple de st-langage clos par θ qui n'est pas un langage de synchronisation (la clôture de $a_s b_s a_t a_s b_t b_s a_t b_t$ par exemple). Guo et al. conjecturent dans [34] que le système de réécriture R, union de θ et d'un système

– que nous appellerons Ω – contenant les règles du type

$$a_{1t}\dots a_{nt}a_{1s}\dots a_{ns}b_{1t}\dots b_{mt}b_{1s}\dots b_{ms} \leftrightarrow b_{1t}\dots b_{mt}b_{1s}\dots b_{ms}a_{1t}\dots a_{nt}a_{1s}\dots a_{ns}$$

pour toute série d'actions deux à deux distinctes, permet de caractériser les langages de synchronisation.

La partie Ω du système de réécriture est assez peu intuitive aussi est-il naturel de se demander pourquoi n'avoir pas envisagé d'utiliser des extensions des relations de commutations partielles plus connues et plus naturelles telles que les relations d'indépendance sous contexte. Dans une telle relation, il ne s'agit plus de considérer des relations d'indépendances de deux actions comme dans le cas des commutations partielles mais de considérer l'indépendance de deux actions sous un contexte gauche. Dans le cas classique du producteur/consommateur par exemple, les actions de produire « p » et de consommer « c » ne sont indépendantes que si elles sont précédées d'une autre production : les mots pc et cp ne sont pas équivalents par contre les mots ppc et pcp le sont. Biermann et Rozoy ont étudié les relations 1-contextuelles (c'est-à-dire où le contexte est réduit à une lettre) dans [5], Bauget et Gastin ont également étudié les congruences engendrées par des relations sous contexte et, plus particulièrement, leurs relations avec les ordres partiels dans [2], Arnold et Husson proposent un cadre commun d'étude des relations liées au contexte, les relations d'indépendance locale – r.i.l – (voir [40]). Une r.i.l. sur un alphabet Σ^* est une relation incluse dans $\Sigma^* \times \Sigma \times \Sigma$ irréflexive et symétrique par rapport aux deux derniers arguments.

Nous avons vu qu'en utilisant une commutation partielle, nous pouvons obtenir les mots qui possèdent les mêmes occurrences d'actions en parallèle. Les commutations partielles sont un cas limite des r.i.l., cette notion peut donc être exprimée par une r.i.l., par contre, dès que l'on considère également la remise en séquence des actions, il apparaît que la connaissance du contexte gauche ne suffit pas, les deux contextes sont nécessaires. Considérons par exemple le langage associé à l'expression $[(a \parallel b) \rightarrow a] \mid [a \rightarrow b \rightarrow b]$. Les mots $a_sb_sa_tb_ta_sa_t$ et $a_sa_tb_sb_ta_sa_t$ représentent des comportements qui respectent la contrainte décrite par le premier membre de l'expression, on souhaiterait qu'ils soient équivalents ou qu'au moins il soit possible de retrouver l'un à partir de l'autre. Avec une r.i.l., il faudrait supposer que a_t et b_s sont indépendants sous le contexte a_s, cependant dans le mot $a_sa_tb_sb_tb_sb_t$, qui appartient au même langage, les mêmes lettres, sous le même contexte gauche, ne sont pas indépendantes. La relation que nous recherchons ne peut donc pas avoir la forme d'une r.i.l. Les commutations sous contexte ne suffisant pas, on pourrait envisager d'utiliser une généralisation « plus large » des commutations partielles telle que les $\mathcal{G}$-relations introduites par Biermann et Rozoy dans [4], cependant, celle-ci ne convient pas davantage. Considérant un alphabet Σ, une $\mathcal{G}$-relation est une relation d'équivalence sur Σ^* pour laquelle les mots équivalents sont commutativement équivalents et qui est une demi-congruence droite (il ne s'agit donc plus de commutations de deux lettres sous contexte ou non mais d'une notion beaucoup plus large). Le problème vient toujours du fait que, dans le cas des langages de synchronisation, les contextes droits ou gauches seuls ne suffisent pas à décider si deux lettres sont indépendantes à un instant donné. Considérons par exemple le langage associé à l'expression $(a \parallel b) \mid (a \rightarrow b \rightarrow b)$. Dans ce langage, nous souhaitons que les mots $a_sb_sa_tb_t$ et $a_sa_tb_sb_t$ soient équivalents cependant, nous ne pouvons pas en déduire que $a_sb_sa_tb_tb_sb_t$ et $a_sa_tb_sb_tb_sb_t$ sont équivalents, nous n'avons donc pas une demi-congruence droite.

Dans le cas des langages de synchronisation, nous ne pouvons pas considérer que deux exécutions sont équivalentes dans l'absolu, $a_sa_tb_sb_t$ et $a_sb_sa_tb_t$ sont deux exécutions correctes

pour l'expression $a \parallel b$ mais seule $a_s a_t b_s b_t$ est correcte pour l'expression $a \rightarrow b$. L'avantage du système R est donc de fournir d'une part des règles symétriques qui permettent de passer d'un mot à un autre mot dans lequel les mêmes occurrences d'actions sont en parallèle – dans ce cas nous avons bien une équivalence – et d'autre part des règles non symétriques permettant de passer d'un mot à un mot de moindre degré de parallélisme. Il semble donc mieux adapté pour permettre de retrouver le plus petit langage de synchronisation contenant un mot donné (avec une relation d'équivalence, on n'obtiendrait pas le plus petit).

Le travail presenté dans cette thèse concerne l'étude des langages de synchronisation et leur caractérisation par des systèmes de réécriture. Dans le premier chapitre, nous rappelons les définitions des expressions et langages de synchronisation et les premiers résultats qui ont été obtenus par Guo et al. La relation θ étant une semi-commutation, nous serons amenés de nombreuses fois à utiliser des résultats classiques de la théorie des traces et des semi-traces aussi rappelons nous, succinctement, les résultats qui seront utilisés et uniquement ceux-ci notre objet n'étant pas l'étude des traces.

Dans le deuxième chapitre, nous étudions la conjecture de Guo et al. et nous montrons qu'elle est vraie dans le cas particulier de langages exprimant la synchronisation entre des séries de deux actions distinctes. Nous obtenons un algorithme de construction d'une expression de synchronisation associée à un langage régulier quelconque sur un alphabet de deux actions. Nous nous intéressons également à une famille de langages définie par Guo et al., les langages bien formés. Ces langages sont des st-langages qui peuvent être définis par une expression rationnelle dans laquelle toute sous-expression portée à l'étoile définit un st-langage. Guo et al. conjecturaient que les langages bien formés pouvaient être utilisés comme générateurs des langages de synchronisation. Nous donnons une caractérisation des langages bien formés basée sur la forme de leurs facteurs itérants qui nous permet de réfuter cette possibilité. Enfin, nous montrons que la conjecture de Guo et al. n'est pas vraie dans le cas général.

Partant du constat que certaines propriétés relatives aux projections sur des sous-alphabets de deux actions semblent manquer au système R, nous proposons de l'étendre dans le chapitre trois. Le nouveau système R' est basé sur R et est équivalent à R dans le cas des langages définis sur des alphabets de deux actions : un mot u se réécrit en un mot v par R' si et seulement si chaque projection sur un sous-alphabet de deux actions de u se réécrit par R en la projection de v. Nous nous intéressons aux propriétés techniques du système R' puis nous montrons que ce système permet de caractériser les langages de synchronisation finis. Nous obtenons un algorithme qui permet de construire une expression de synchronisation associée à un langage fini donné. Notre système étendu s'avère être le « plus grand » système de réécriture par lequel les langages de synchronisation sont clos et pourtant, nous montrons qu'il ne permet pas de les caractériser, ce résultat met fin à la recherche d'une caractérisation des langages de synchronisation par les systèmes de réécriture.

Dans le chapitre quatre, nous nous intéressons aux langages de synchronisation généralisés introduits par Salomaa et Yu dans [57] au regard de ces résultats négatifs. Salomaa et Yu ont étendu la sémantique des expressions de synchronisation augmentant ainsi leur pouvoir d'expression tout en restant dans le cadre des langages réguliers. Salomaa et Yu conjecturent que la classe des langages de synchronisation généralisés est exactement la classe des st-langages réguliers clos par θ et ils montrent que cette conjecture est vraie dans le cas des langages finis. Une telle caractérisation serait fort intéressante puisqu'il est possible de décider automatiquement si un langage régulier est clos par une semi-commutation à partir d'un automate le reconnaissant. Après quelques remarques sur les propriétés des langages de synchronisa-

tion généralisés, nous introduisons les st-morphismes, une classe de fonctions rationnelles très proches des morphismes strictement alphabétiques, qui préservent les st-langages. Ces st-morphismes nous permettent d'établir d'une part le lien entre les langages clos par θ et les langages clos par R' et d'autre part entre les deux familles de langages de synchronisation. Nous montrons que les familles des st-langages réguliers clos par R' et des images par st-morphismes des st-langages réguliers clos par θ sont égales et que les familles des images par st-morphismes des langages de synchronisation et de synchronisation généralisés sont égales.

Dans le dernier chapitre, nous montrons que la semi-commutation θ permet de caractériser les images par st-morphismes des langages de synchronisation généralisés. Nous proposons une preuve très simple pour les langages qui sont la clôture par θ de langages bien formés. Dans le cas général, nous montrons qu'il est possible d'utiliser un coloriage (une fonction rationnelle), une commutation partielle et un morphisme strictement alphabétique pour calculer la clôture d'un st-langage régulier par θ. Puis, nous proposons une construction à partir de l'automate asynchrone du langage colorié clos par commutation partielle qui permet d'obtenir un langage décomposable en produits de mixage. Une fois la décomposition terminée, il est aisé de montrer que le langage obtenu est un langage de synchronisation généralisé. Enfin, nous généralisons la construction pour montrer qu'elle peut être appliquée pour une semi-commutation et un langage quelconques.

Notations – Définitions

Notations

Le symbole $\oplus$ désigne le « ou » exclusif.

Les alphabets considérés seront toujours finis et seront en général notés Σ, X ou Y. Par convention, ε désigne le mot vide. L'ensemble de tous les mots finis sur un alphabet Σ est noté Σ^* et Σ^+ s'il est privé du mot vide.

Dans ce paragraphe, nous utiliserons l'alphabet Σ, les mots et les langages considérés sont définis sur Σ.

Nous notons respectivement $|u|$ et $|u|_a$ la longueur du mot u et le nombre d'occurrences de la lettre a dans u.

L'alphabet de u est noté $\mathrm{alph}(u)$ et est défini par:

$$\mathrm{alph}(u) = \{x \in \Sigma \mid |u|_x > 0\}.$$

Le mot u est un facteur du mot v s'il existe des mots u_1 et u_2 tels que $v = u_1 u u_2$, c'est un facteur propre si $u \neq v$.

Le mot u est un facteur gauche (resp. facteur droit) du mot v s'il existe un mot w tel que $v = uw$ (resp. $v = wu$), c'est un facteur gauche propre (resp. facteur droit propre) si w n'est pas vide.

Ces notions sont étendues aux langages: un langage M est facteur d'un langage L s'il existe deux langages L_1 et L_2 tels que $L = L_1.M.L_2$. Les autres notions sont étendues de la même manière.

L'ensemble des facteurs itérants d'un langage L est défini par:

$$\mathrm{fi}(L) = \{u \in \Sigma^* \mid \exists v, w \in \Sigma^*, vu^*w \subseteq L\}.$$

Le mot u est un sous-mot d'un mot v s'il existe des mots $u_1, \ldots, u_n$ et $v_0, \ldots, v_n$ tels que $u = u_1 \ldots u_n$ et $v = v_0 u_1 v_1 u_2 v_2 \ldots u_n v_n$.

Le mot u est un conjugué d'un mot v s'il existe deux mots u_1 et u_2 tels que $u = u_1 u_2$ et $v = u_2 u_1$.

La projection du mot u sur le sous-alphabet X de Σ est l'image de u par l'homomorphisme Π_X défini par:

$$\forall x \in \Sigma, \Pi_X(x) = \begin{cases} x \text{ si } x \in X, \\ \varepsilon \text{ si } x \notin X. \end{cases}$$

Le shuffle (produit de mélange) des mots u et v est noté $u \sqcup v$ et est défini par :

$$u \sqcup v = \{u_1 v_1 \dots u_n v_n \mid u_i, v_i \in \Sigma^*, u = u_1 \dots u_n, v = v_1 \dots v_n\}.$$

Le shuffle de deux langages L_1 et L_2 est :

$$L_1 \sqcup L_2 = \bigcup_{u \in L_1, v \in L_2} u \sqcup v.$$

Réécriture

Dans ce paragraphe, Σ est l'alphabet utilisé.

Considérant un système de réécriture R et deux mots u et v, nous écrivons

$$u \xrightarrow[R]{} v$$

s'il existe une règle $w \to w'$ dans R et deux mots u' et u'' tels que $u = u'wu''$ et $v = u'w'u''$. On note

$$u \xrightarrow[R]{*} v$$

s'il existe n mots ($n \geq 1$) $w_1, \dots, w_n$ tels que $w_1 = u$, $w_n = v$ et pour tout i, $1 \leq i < n$,

$$w_i \xrightarrow[R]{} w_{i+1}.$$

L'entier $n - 1$ est appelé longueur de la dérivation. Lorsque l'on considère une dérivation de longueur n on note

$$u \xrightarrow[R]{n} v.$$

Pour un mot u, $\mathrm{f}_R(u)$ est la clôture de u par R et est défini par

$$\mathrm{f}_R(u) = \{v \in \Sigma^* \mid u \xrightarrow[R]{*} v\},$$

de même, $\mathrm{f}_R(L)$ est la clôture par R du langage L et est définie par :

$$\mathrm{f}_R(L) = \bigcup_{u \in L} \mathrm{f}_R(u).$$

Graphes

Un graphe orienté (fini) est un couple (S, A) où S est un ensemble fini de sommets et A un ensemble d'arcs inclus dans $S \times S$.

Un chemin dans un graphe (S, A) est une suite d'arcs $a_0 \dots a_n$ telle que pour tout indice $0 < i \leq n$, $a_{i-1} = (s_j, s_k)$ et $a_i = (s_k, s_l)$. Un chemin $(s_0, s_1)(s_1, s_2) \dots (s_{n-1}, s_n)$ est dit étiqueté par le mot $s_0 s_1 \dots s_n$.

Un graphe (S, A) est dit connexe si, pour tout couple de sommets x et y de S, il existe un chemin de x à y dans $(S, A \cup A^{-1})$. Un graphe (S, A) est dit fortement connexe si, pour tout couple de sommets x et y de S, il existe un chemin de x à y dans (S, A).

Dans un graphe (S, A), un ensemble de sommets $S' \subseteq S$ est une clique si

$$(s_1 \in S' \text{ et } s_2 \in S') \Rightarrow ((s_1, s_2) \in A \text{ et } (s_2, s_1) \in A).$$

On appelle recouvrement par cliques d'un graphe $\mathcal{G} = (S, A)$ Les cliques $S_0, S_1, \dots, S_n$ d'un graphe $\mathcal{G} = (S, A)$ forment un recouvrement par cliques de $\mathcal{G}$ si $S_0 \cup \dots \cup S_n = S$ et si pour tout arc (s_1, s_2) de A, il existe un indice i tel que s_1 et s_2 appartiennent à S_i.

Chapitre 1

Notions de base

Nous donnons dans la première section de ce chapitre les définitions des st-langages, des expressions et langages de synchronisation ainsi que la définition du système de réécriture proposé par Guo, Salomma et Yu pour caractériser les langages de synchronisation. Puis, nous introduisons les commutations partielles et semi-commutations en ne mentionnant que les quelques résultats qui nous seront utiles dans les chapitres suivants. Enfin, dans la dernière section nous énonçons quelques propriétés générales des st-langages.

1.1 Expressions et langages de synchronisation

1.1.1 Expressions de synchronisation

Les expressions de synchronisation ont été introduites dans [30] dans le cadre du projet de développement d'un langage pour multiprocesseurs à mémoire partagée nommé *ParC*. La motivation première dans l'élaboration de ce langage était d'éviter au programmeur l'utilisation des outils de bas niveau pour spécifier le parallélisme et la synchronisation. Cette motivation a conduit à l'utilisation de nombreux outils de haut niveau tels que les variables partagées avec portée, des instructions spécifiques (`pexec` exécution de plusieurs traitements en parallèle, `pfor`, `for` parallèle sans contraintes de synchronisation, `sfor`, `for` parallèle avec contraintes de synchronisation, ...), les étiquettes identifiant l'exécution d'un traitement et enfin, les expressions de synchronisation.

Les expressions de synchronisation délivrent le programmeur de l'implantation de la synchronisation en lui imposant uniquement de spécifier les contraintes nécessaires. Des blocs d'instructions de taille variable peuvent être étiquetés et les contraintes sont exprimées par des expressions sur des étiquettes de blocs. Cet outil permet d'exprimer simplement les contraintes sans apporter de changement dans la structure du langage de base. Une étiquette de bloc est vue ici non comme un point d'entrée mais comme l'identificateur de l'exécution d'un traitement complet. Les expressions permettent en outre au programmeur de définir la synchronisation avec le grain de son choix et de modifier très facilement les contraintes choisies. Durant l'exécution, un traitement ne peut être effectué à un instant donné que si son exécution ne viole pas les contraintes imposées par l'expression, dans le cas contraire, l'exécution de ce traitement est différée. L'exemple suivant est tiré de [30] :

Exemple 1.1 La somme d'une longue séquence d'entiers peut être calculée par parties, en utilisant un langage de programmation parallèle offrant des variables partagées et une ins-

truction pfor permettant d'exécuter les différentes itérations d'une boucle en parallèle. Ici, la somme de 10000 entiers va être calculée par 100 processus qui vont chacun réaliser l'addition de 100 éléments de la séquence de départ.

```
shared int Sequence[10000], tmp[100];
shared int sum;

Tag T;
restrict (T*);

main() {
int i,j;
    pfor (i = 0; i <100; i++) {
        for (j = 0; j <100; j++) {
            .......
            tmp[i] = tmp[i] + Sequence[i*100+j];
        }
        T :: sum = sum + tmp[i];
    }
}
```

Les 100 sommes partielles peuvent naturellement être calculées en parallèle cependant, l'accès à la variable partagée sum doit être réalisé en exclusion mutuelle par tous les processus. Le tag T permet donc d'identifier le traitement sur lequel la contrainte de synchronisation va porter. L'expression de synchronisation indiquée après le mot clé restrict indique simplement que, durant l'exécution, les traitements T effectués par les différents processus devront l'être en séquence (mais dans un ordre quelconque : il n'y a aucune distinction entre les traitements T exécutés par différents processus).

Une expression de synchronisation est :

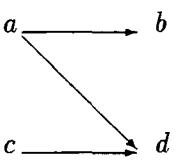
1. une étiquette désignant l'exécution d'un traitement ou ε désignant une action nulle ;
2. si e_1 et e_2 sont des expressions de synchronisation :
   - $(e_1 \rightarrow e_2)$ qui impose que l'exécution de e_2 ne débute que lorsque celle de e_1 est complètement terminée,
   - $(e_1 \mid e_2)$ qui signifie que l'exécution de e_1 ou e_2 peut se dérouler mais pas les deux,
   - $(e_1 \parallel e_2)$ qui signifie que les exécutions de e_1 et e_2 peuvent se chevaucher dans le temps. La nature de cet opérateur implique qu'une même étiquette ne peut apparaître dans les deux opérandes d'un opérateur $\parallel$,
   - $(e_1 \& e_2)$ qui impose que les contraintes exprimées par e_1 et e_2 soient satisfaites,
   - (e_1^*) qui exprime que l'exécution de e_1 peut être répétée un nombre arbitraire de fois.

Les expressions de synchronisation permettent donc d'exprimer la séquence ($\rightarrow$), l'exclusion ($|$), la répétition ($*$) et la mise en parallèle ($\|$). Considérons l'exemple suivant :

Exemple 1.2 Si deux traitements c et d ne peuvent s'effectuer que lorsque deux traitements a et b sont terminés mais qu'il n'y a pas de contrainte de synchronisation particulière entre a et b d'une part et entre c et d d'autre part, on obtient l'expression $(a \parallel b) \rightarrow (c \parallel d)$.

Une exécution durant laquelle a et b se déroulent au même instant puis, lorsque ces deux actions sont terminées, on observe l'exécution de c et d en séquence sera conforme aux contraintes imposées par l'expression. En effet, les actions autorisées à se dérouler en parallèle ne sont jamais contraintes de le faire, il ne s'agit pas d'indiquer un rendez-vous mais juste de spécifier l'indépendance. Les contraintes plus complexes devront être exprimées à l'aide de l'intersection (&). L'exemple suivant est donné par Guo, Salomaa et Yu dans [33] pour montrer la nécessité de l'utilisation de l'intersection, les contraintes exprimées par cette expression ne peuvent l'être par une expression n'utilisant que les quatre autres opérateurs.

Exemple 1.3 Si d'une part un traitement b ne peut s'effectuer qu'après un traitement a et que d'autre part un traitement d ne peut s'effectuer qu'après les deux traitements a et c, ce qui correspond au graphe de dépendance



on obtient l'expression $((a \rightarrow b) \parallel (c \rightarrow d))\&((a \rightarrow d) \parallel b \parallel c)$.

On peut également remarquer que la notion d'indépendance entre actions est totalement locale (même si nous avons vu dans l'introduction qu'elle ne dépend pas d'un contexte). Lors d'une même exécution, certaines occurrences d'actions peuvent être dépendantes tandis que d'autres occurrences des mêmes actions ne le sont pas.

Exemple 1.4 L'expression $(a \parallel b)^* \rightarrow (a \rightarrow b) \rightarrow (a^* \parallel b^*)$ décrit les contraintes suivantes : un nombre arbitraire d'exécutions d'une action a en parallèle avec une action b peut être effectué puis, lorsque l'on a observé l'exécution d'une nième occurrence de a en séquence avec la nième occurrence de b, les actions a et b sont totalement indépendantes.

On peut noter la différence entre les expressions $(a \parallel b)^*$ et $(a^* \parallel b^*)$. Dans le premier cas, les actions a et b peuvent être exécutées en parallèle mais, pour chaque occurrence de a, il doit y avoir une occurrence de b (et vice-versa) et seules ces occurrences « correspondantes » peuvent être exécutées en parallèle. Il y a une restriction sur l'indépendance entre a et b et une contrainte sur les nombres de a et b. Dans la deuxième expression, les actions a et b sont totalement indépendantes. Plus formellement, les expressions de synchronisation sont définies par :

Définition 1.5 *Soit Σ un alphabet d'actions (ou étiquettes). L'ensemble des expressions de synchronisation sur l'alphabet Σ, noté $\mathrm{ES}(\Sigma)$, est le plus petit sous-ensemble*

$$\mathrm{ES}(\Sigma) \subseteq (\Sigma \cup \{\rightarrow, \&, |, \|, *, (,)\})^*$$

tel que :

- *l'action nulle (notée ε) et toutes les actions de Σ appartiennent à $\mathrm{ES}(\Sigma)$;*

- *pour toutes expressions* e_1 *et* e_2 *de* $\mathrm{ES}(\Sigma)$, *les expressions* $(e_1 \rightarrow e_2)$, $(e_1 \mid e_2)$, $(e_1 \& e_2)$ *et* (e_1^*) *appartiennent à* $\mathrm{ES}(\Sigma)$;
- *pour toutes expressions* e_1 *et* e_2 *de* $\mathrm{ES}(\Sigma)$ *telles que* $alph(e_1) \cap alph(e_2) = \emptyset$ *(avec* $alph(e)$ *l'ensemble des étiquettes apparaissant dans l'écriture de* e*), l'expression* $(e_1 \parallel e_2)$ *appartient à* $\mathrm{ES}(\Sigma)$.

Les expressions de synchronisation permettent en premier lieu de décrire des contraintes de synchronisation dans une application distribuée, elles peuvent également être vues comme un outil donnant une représentation synthétique du comportement d'une application. Les langages qui sont associés aux expressions de synchronisation vont, pour chacun de ces points de vue, être essentiels pour leur utilisation.

1.1.2 Langages de synchronisation

Les langages de synchronisation sont associés aux expressions de synchronisation, ces langages permettent d'une part de donner une description sémantique précise des expressions et d'autre part d'implanter les expressions : pour *ParC*, la structure de contrôle d'exécution d'un programme est constituée de l'automate du langage associé à l'expression de synchronisation écrite par le programmeur. De plus, en pratique, un compilateur doit pouvoir tester si deux expressions sont équivalentes, contradictoires ou si l'une est plus contrainte que l'autre de manière à pouvoir contrôler la cohérence des programmes. Les langages associés aux expressions permettent de décider de l'inclusion et de l'équivalence de deux expressions.

Nous avons vu que les langages associés aux expressions de synchronisation utilisent la représentation d'une action de durée quelconque par deux actions atomiques marquant son début et sa fin. À partir d'un alphabet d'actions Σ, nous allons donc construire deux alphabets contenant respectivement les actions atomiques marquant les débuts et fins des actions de Σ :

Définition 1.6 *Soit* Σ *un alphabet fini, les alphabets* Σ_s *et* Σ_t *sont définis par la relation :*

$$(a \in \Sigma) \Leftrightarrow (a_s \in \Sigma_s) \Leftrightarrow (a_t \in \Sigma_t).$$

À chaque action étiquetée par une lettre a, correspondent dans un langage de synchronisation les lettres a_s (s pour *start*) et a_t (t pour *termination*) qui marquent le début et la fin de l'action a. Il est donc possible de distinguer une réelle exécution en parallèle $a_s b_s a_t b_t$ d'une exécution en séquence $a_s a_t b_s b_t$ ce qui permettra, dans le cas des expressions de synchronisation d'autoriser ou non un début d'action b_s après un début d'action a_s en fonction des contraintes imposées par l'utilisateur.

À une expression de synchronisation est associé un langage de synchronisation construit sur l'alphabet des débuts et fins d'actions. Il est cependant nécessaire de garder une correcte association des débuts et fins d'actions, en particulier, dans un mot représentant une exécution, il est nécessaire d'observer, pour toute action, une séquence correctement alternée de débuts et fins de cette action. Nous considérons donc la famille des langages pour lesquelles la correspondance des débuts et fins est respectée.

Définition 1.7 *Soit* Σ *un alphabet d'actions. Un mot* u *de* $(\Sigma_s \cup \Sigma_t)^*$ *est un st-mot si et seulement si pour toute action* x *de* Σ,

$$\Pi_{\{x_s, x_t\}}(u) \in (x_s x_t)^*.$$

Un langage dont tous les mots sont des st-mots est appelé st-langage.

Notation 1.8 *Nous notons* ST_Σ *l'ensemble des st-mots sur l'alphabet* $\Sigma_s \cup \Sigma_t$.

Nous nous servirons également des définitions suivantes :

Définition 1.9 *Un mot st-primitif est un mot non vide dont aucun facteur gauche propre n'est un st-mot. Les facteurs st-primitifs d'un st-mot* u *sont les mots st-primitifs* $u_1, \ldots, u_n$ *tels que* $u = u_1 \ldots u_n$.

Nous pouvons donc à présent donner la définition formelle des langages de synchronisation qui sont construits par induction à partir des expressions de synchronisation.

Définition 1.10 *Soit* Σ *un alphabet d'actions. On définit pour une expression de synchronisation* $e \in SE(\Sigma)$ *le langage de synchronisation* $L(e) \subseteq (\Sigma_s \cup \Sigma_t)^*$ *de la façon suivante :*

- $L(\varepsilon) = \varepsilon$;
- $\forall a \in \Sigma,\ L(a) = a_s a_t$;
- *si* $e = e_1 \rightarrow e_2$ *alors* $L(e) = L(e_1).L(e_2)$;
- *si* $e = e_1 \mid e_2$ *alors* $L(e) = L(e_1) \cup L(e_2)$;
- *si* $e = e_1 \& e_2$ *alors* $L(e) = L(e_1) \cap L(e_2)$;
- *si* $e = e_1 \parallel e_2$ *alors* $L(e) = L(e_1) \sqcup L(e_2)$;
- *si* $e = e_1^*$ *alors* $L(e) = (L(e_1))^*$.

Clairement, par construction, les langages de synchronisation sont réguliers. Ceci va permettre, entre autres, d'utiliser les automates finis pour implanter les expressions de synchronisation : l'implantation proposée par Guo dans [32] utilise les « reversed alternating finite automata » ([26, 11]).

Exemple 1.11 Pour $e = a \parallel b$ nous obtenons le langage :

$$L(e) = \{a_s a_t b_s b_t, a_s b_s a_t b_t, a_s b_s b_t a_t, b_s a_s a_t b_t, b_s a_s b_t a_t, b_s b_t a_s a_t\}.$$

Ce langage contient bien sûr les séquences $a_s a_t b_s b_t$ et $b_s b_t a_s a_t$, nous avons vu que l'opérateur de concurrence n'indique pas un rendez-vous mais l'indépendance des actions.

L'équivalence et l'inclusion de deux expressions de synchronisation sont décidables, pour comparer deux expressions, il suffit de tester l'égalité ou l'inclusion des langages qui leur sont associés.

Exemple 1.12 Les expressions $(a \rightarrow b) \parallel c$ et $[(a \parallel c) \rightarrow b] \mid [a \rightarrow (b \parallel c)]$ ne sont pas équivalentes. Intuitivement, une exécution durant laquelle l'action c est commencée pendant le a et terminée pendant le b respecte les contraintes de la première expression mais pas de la deuxième. Pour montrer ceci, il suffit de calculer les langages associés aux deux expressions :

$$L((a \rightarrow b) \parallel c) = (a_s a_t.b_s b_t) \sqcup c_s c_t,$$
$$L([(a \parallel c) \rightarrow b] \mid [a \rightarrow (b \parallel c)]) = [(a_s a_t \sqcup c_s c_t).b_s b_t] + [a_s a_t.(b_s b_t \sqcup c_s c_t)].$$

On constate que ces deux langages sont différents, le mot $c_s a_s a_t b_s b_t c_t$ par exemple appartient au premier mais pas au deuxième.

Suite à cette définition des langages de synchronisation, il est naturel de se demander s'il est possible de décider si un langage donné est un langage de synchronisation. Ceci d'une part dans le but d'obtenir une caractérisation décidable de la famille des langages de synchronisation et d'autre part dans le but d'agrandir le champ d'application des expressions de synchronisation. Comme les expressions de synchronisation permettent de décrire de manière synthétique le comportement d'une application, on peut envisager de les utiliser pour présenter à un programmeur le résultat d'une exécution (ou d'un jeu d'exécutions) plutôt que de lui fournir les traces d'exécution brutes. Un tel outil pourrait être utilisé pour le déverminage d'applications distribuées. Dans cette optique, il est nécessaire de pouvoir reconstruire, à partir d'une trace d'exécution ou d'un st-langage, une expression de synchronisation la plus proche possible du jeu d'essais présenté.

1.1.3 Un système de réécriture

Guo, Salomaa et Yu proposent, dans [33], de caractériser les langages de synchronisation en termes de langages clos par système de réécriture. Ils ont, pour cela, défini un système de réécriture (ou plus exactement une famille de systèmes de réécriture en fonction des alphabets d'actions utilisés) :

Définition 1.13 *Soit Σ un alphabet d'actions. Le système de réécriture R_Σ est l'union des systèmes θ_Σ et Ω_Σ avec:*

$$\theta_\Sigma = \{a_s b_s \leftrightarrow b_s a_s, a_t b_t \leftrightarrow b_t a_t, a_s b_t \rightarrow b_t a_s \mid a, b \in \Sigma, a \neq b\},$$

et

$$\Omega_\Sigma =$$
$$\{a_{1t} \ldots a_{nt} a_{1s} \ldots a_{ns} b_{1t} \ldots b_{mt} b_{1s} \ldots b_{ms} \leftrightarrow b_{1t} \ldots b_{mt} b_{1s} \ldots b_{ms} a_{1t} \ldots a_{nt} a_{1s} \ldots a_{ns}$$
$$\mid a_1, \ldots, a_n, b_1, \ldots, b_m \textit{ sont des lettres de } \Sigma \textit{ deux à deux distinctes}\}.$$

Lorsqu'aucune confusion n'est possible ou lorsque nous voulons parler des systèmes en général, nous utilisons R, θ et Ω à la place de R_Σ, θ_Σ et Ω_Σ.

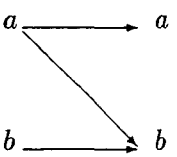
La première partie du système R est relativement intuitive. Si nous considérons un mot d'un langage de synchronisation et que, dans ce mot, nous trouvons un mot $ua_s b_s v$, il est clair que les occurrences de a et b mises en évidence sont autorisées à se dérouler en parallèle. Dans ce cas, l'ordre dans lequel les exécutions de a et b sont commencées n'a aucune importance, le mot $ub_s a_s v$ doit donc forcément appartenir au langage considéré. Les règles du type $a_t b_t \leftrightarrow b_t a_t$ s'expliquent avec des arguments semblables.

Si nous trouvons un mot $ua_s b_t v$, les occurrences de a et b mises en évidence sont autorisées à se dérouler en parallèle, cependant nous avons vu qu'il ne pouvait s'agir d'une obligation, il est tout-à-fait possible d'exécuter ces deux occurrences (qui sont indépendantes) en séquence, le mot $ub_t a_s v$ appartient donc naturellement au langage considéré. Par contre, si un langage de synchronisation contient un mot $ub_t a_s v$, l'observation de ce seul mot ne permet pas de déduire si les occurrences concernées sont autorisées à se dérouler en parallèle ou non. C'est pourquoi les règles du type $a_s b_t \rightarrow b_t a_s$ ne sont pas symétriques.

La deuxième partie du système R est peut-être moins intuitive. Elle a été motivée par la constatation du fait que θ n'est pas suffisante pour caractériser les langages de synchronisation,

il est assez aisé de constater ce fait sur l'exemple suivant :

Exemple 1.14 Le langage $L = \mathrm{f}_\theta(a_s b_s a_t a_s b_t b_s a_t b_t)$ n'est pas un langage de synchronisation. Le plus petit langage de synchronisation, au sens de l'inclusion, contenant le mot $a_s b_s a_t a_s b_t b_s a_t b_t$ est le langage $M = L(a^2 \parallel b^2)$. L'ordre partiel qui peut être associé à ce mot est



or les expressions de synchronisation ne permettent pas d'exprimer des contraintes aussi fines entre les différentes occurrences d'actions. Comme le mot $u = a_s b_s b_t b_s a_t a_s a_t b_t$ appartient à M mais pas à L, L n'est pas un langage de synchronisation. Pour obtenir le mot u à partir du mot de départ, il est nécessaire d'utiliser d'autres règles que celles de θ, ici, il suffit d'appliquer la règle $a_t a_s b_t b_s \to b_t b_s a_t a_s$, on peut constater que $\mathrm{f}_R(a_s b_s a_t a_s b_t b_s a_t b_t)$ et M sont égaux.

Clairement, les systèmes θ et Ω préservent les st-mots, Guo et al. montrent dans [34] que le système de réécriture R est adapté à l'étude des langages de synchronisation et ils conjecturent que ce système permet de caractériser les langages de synchronisation.

Théorème 1.15 (Guo et al.) *Tout langage de synchronisation défini sur Σ est clos par le système de réécriture $R(\Sigma)$.*

Conjecture I (Guo et al.) *Tout st-langage régulier défini sur $\Sigma_s \cup \Sigma_t$ et clos par $R(\Sigma)$ est un langage de synchronisation.*

Voici posés les définitions des expressions et langages de synchronisation ainsi que les problèmes qui vont nous intéresser. Dans la section suivante, nous rappelons quelques définitions et propriétés des commutations partielles et semi-commutations qui seront utilisées durant l'étude des langages de synchronisation.

1.2 Commutations partielles et semi-commutations

La notion très étendue et très largement étudiée de langage trace a été introduite par Mazurkiewicz [43] dans le but d'étudier le comportement des réseaux de Petri « one safe » (indépendamment, Cartier et Foata ont étudié le monoïde partiellement commutatif libre pour résoudre des problèmes de réarrangement en combinatoire [9]). Cette idée d'exprimer le parallélisme par une relation d'indépendance entre actions s'est avérée être une notion essentielle dans la modélisation du parallélisme et a donné naissance à de nombreux travaux et de nombreuses extensions telles que les semi-commutations que nous utilisons ou encore les traces infinies [21, 27].

1.2.1 Quelques définitions

Notre but n'est pas ici l'étude des traces mais nous utiliserons à de nombreuses reprises les commutations partielles pour profiter des outils qui leur sont associés. D'autre part, le

système de réécriture θ étant une semi-commutation, nous disposerons, pour son étude, des propriétés des semi-commutations. Nous rappelons donc ici uniquement les propriétés qui seront utilisées dans les chapitres suivants. La théorie des langages trace est largement plus développée et a donné lieu à un ouvrage de synthèse, le « Book of Traces » [23].

L'idée de base des commutations partielles est d'exprimer le parallélisme par l'indépendance d'actions. Chaque action est représentée par une lettre d'un alphabet fini Σ, un mot de Σ^* représente une trace d'exécution et, si deux actions a et b sont indépendantes, deux exécutions $uabv$ et $ubav$ sont équivalentes *i.e.* amènent le système dans un même état.

Définition 1.16 *Une relation de commutation partielle ϱ définie sur un alphabet Σ est une relation irréflexive et symétrique incluse dans $\Sigma \times \Sigma$.*

Clerbout et Latteux ont introduit les semi-commutations, une généralisation naturelle des commutations partielles. Une semi-commutation est une relation irréflexive sur un alphabet Σ, la différence par rapport à une commutation partielle est donc l'absence de symétrie. Les semi-commutations ont été introduites dans le but de modéliser des comportements de type producteur/consommateur qui ne pouvaient pas l'être à l'aide des commutations partielles. Un comportement de type producteur/consommateur se résume par l'expression « *on ne peut consommer que ce que l'on a produit* », si l'on nomme p l'action de produire (un jeton par exemple) et c l'action de consommer, un comportement $(pc)^*$ est correct. Il est également possible d'anticiper la production avec, comme limite, un comportement du type $p^n c^n$, par contre, il est impossible d'anticiper une consommation. Il existe donc une dépendance non symétrique entre le producteur et le consommateur, une exécution $ucpv$ peut être remplacée par $upcv$ mais pas l'inverse ce qui conduit à une semi-commutation $\varrho = \{(c, p)\}$.

Définition 1.17 (Clerbout, Latteux [12, 13]) *Une relation de semi-commutation ϱ définie sur un alphabet Σ est une relation irréflexive incluse dans $\Sigma \times \Sigma$.*

Clairement, toute relation de commutation-partielle est une relation de semi-commutation.

On peut noter ici que les réseaux de Petri sont bâtis sur le modèle production et consommation de jetons, les liens entre les semi-commutations et les réseaux de Petri ont donc naturellement été étudiés [39, 52, 53, 14].

Aux commutations partielles et aux semi-commutations sont associés des systèmes de réécriture, par abus de langage, nous utiliserons par la suite la même notation pour désigner la relation et le système de réécriture qui lui est associé.

Définition 1.18 *À toute relation de semi-commutation ou de commutation partielle ϱ sur un alphabet d'actions Σ est associé un système de réécriture S_ϱ défini par :*

$$S_\varrho = \{ab \to ba \mid (a, b) \in \varrho\}.$$

Remarquons que, dans le cas d'une commutation partielle, le système de réécriture obtenu est symétrique.

1.2.2 Quelques propriétés de base

Nous utiliserons principalement trois propriétés de base, le Lemme de Projection, le Lemme des Distances et le Lemme de Levi.

Le Lemme de Projection permet très souvent de se ramener au cas simple d'un alphabet de deux lettres, il s'énonce comme suit :

Lemme 1.19 (Lemme de Projection [12]) *Soient ϱ une semi-commutation sur un alphabet Σ et u et v deux mots de Σ^*. Alors, nous avons :*

$$(u \xrightarrow[\varrho]{*} v) \Leftrightarrow (\forall x, y \in \Sigma, \Pi_{\{x,y\}}(u) \xrightarrow[\varrho]{*} \Pi_{\{x,y\}}(v)).$$

La notion de distance que nous allons utiliser ici est une distance entre les mots commutativement équivalents qui se définit intuitivement par « *le nombre de permutations de deux lettres qu'il faut effectuer pour passer de l'un à l'autre* ».

Définition 1.20 *Deux mots u et v définis sur un alphabet Σ sont commutativement équivalents si et seulement si pour toute lettre x de Σ, les nombres d'occurrences de x dans u et dans v sont égaux. L'ensemble des mots commutativement équivalents au mot u est :*

$$\mathrm{com}(u) = \{w \mid \forall x \in \Sigma, |u|_x = |w|_x\}.$$

Pour définir la distance entre deux mots, nous allons utiliser la notion de numérotation introduite dans [54].

Définition 1.21 *Soit Σ un alphabet et ϱ une semi-commutation sur Σ. L'alphabet numéroté et la semi-commutation numérotée correspondant sont définis par :*

- $\Sigma_{num} = \Sigma \times \mathbb{N}$;
- $\varrho_{num} = \{((a,i)(b,j)) \mid i,j \in \mathbb{N}, (a,b) \in \varrho\}$.

L'application num *de Σ^* dans Σ^*_{num} est définie inductivement par :*

- $\mathrm{num}(\varepsilon) = \varepsilon$;
- $\forall u \in \Sigma^*, \forall a \in \Sigma, \mathrm{num}(ua) = \mathrm{num}(u)(a, |u|_a)$.

Le morphisme denum *de Σ^*_{num} dans Σ^* est défini par :*

- $\forall x = (a,i) \in \Sigma_{num}, \mathrm{denum}(x) = a$.

Cette numérotation peut effectivement être utilisée :

Lemme 1.22 (Lemme de numérotation) *Soit ϱ une semi-commutation sur un alphabet Σ. Nous avons :*

$$\forall u, v \in \Sigma^*, (u \xrightarrow[\varrho]{n} v) \Leftrightarrow (\mathrm{num}(u) \xrightarrow[\varrho_{num}]{n} \mathrm{num}(v)),$$
$$\forall u', v' \in \Sigma^*_{num}, (u' \xrightarrow[\varrho_{num}]{n} v') \Rightarrow (\mathrm{denum}(u') \xrightarrow[\varrho]{n} \mathrm{denum}(v')).$$

Définition 1.23 *Soient u et v deux mots commutativement équivalents de Σ^*. La distance de u à v est notée* $\mathrm{d}(u,v)$ *et est égale à* $Card(\{(a,b) \in \Sigma_{num} \times \Sigma_{num} \mid \mathrm{num}(u) = xaybz\} \setminus \{(a,b) \in \Sigma_{num} \times \Sigma_{num} \mid \mathrm{num}(v) = x'ay'bz'\})$.

Le Lemme des Distances assure que tout pas de dérivation qui fait décroître la distance entre le mot de départ et le mot cible est un « bon » pas.

Lemme 1.24 (Lemme des Distances [29]) *Soient ϱ une semi-commutation sur un alphabet Σ et u et v deux mots de Σ^* tels que v appartient à $\mathrm{f}_\varrho(u)$. Alors, nous avons:*

$$(u \underset{\varrho}{\longrightarrow} w \text{ avec } \mathrm{d}(w,v) < \mathrm{d}(u,v)) \Rightarrow (v \in \mathrm{f}_\varrho(w)).$$

Nous utiliserons en fait un corollaire de ce Lemme. Intuitivement, la distance entre deux mots est la longueur minimale d'une dérivation par une commutation partielle qui permet de passer de l'un à l'autre. Le Corollaire suivant nous assure qu'une telle dérivation existe toujours.

Corollaire 1.25 *Soient ϱ une semi-commutation sur un alphabet Σ, u un mot de Σ^* et v un mot de $\mathrm{f}_\varrho(u)$. Il existe une dérivation de u à v de longueur* $\mathrm{d}(u,v)$.

Enfin, rappelons le Lemme de Levi pour les semi-traces énoncé par Clerbout, Latteux et Roos (et précédemment par Cori et Perrin [18] dans le cas des traces) :

Lemme 1.26 (Lemme de Levi pour les semi-traces [14]) *Soient ϱ une semi-commutation définie sur un alphabet Σ, u, v, α, β des mots de Σ^* tels que $\alpha\beta$ appartient à $\mathrm{f}_\varrho(uv)$. Il existe des mots u', v', u'' et v'' tels que:*

$$\begin{array}{l} \alpha \in \mathrm{f}_\varrho(u'v') \text{ et } \beta \in \mathrm{f}_\varrho(u''v''), \\ u'u'' \in \mathrm{f}_\varrho(u) \text{ et } v'v'' \in \mathrm{f}_\varrho(v), \\ \mathrm{alph}(u'') \times \mathrm{alph}(v') \subseteq \varrho. \end{array}$$

1.2.3 Commutations et langages réguliers

Les propriétés concernant les langages réguliers sont un point important de l'étude des commutations et pour l'utilisation que nous faisons des commutations, il s'agit d'un point essentiel puisque les langages de synchronisation sont réguliers. La question principale dans ce domaine est de savoir dans quelle mesure la clôture d'un langage régulier par une commutation est régulière. Avant d'y répondre, il est nécessaire de parler des graphes associés aux fonctions de commutation.

Définition 1.27 *À toute semi-commutation (ou commutation partielle) ϱ définie sur un alphabet Σ est associé un graphe de non commutation $(\Sigma, \bar{\varrho})$ où $\bar{\varrho} = (\Sigma \times \Sigma) \setminus \varrho$.*

Définition 1.28 *Soient ϱ une commutation partielle (ou une semi-commutation) définie sur un alphabet Σ et u un mot de Σ. Le mot u est dit connexe (respectivement fortement connexe) pour ϱ si le graphe $(\mathrm{alph}(u), \bar{\varrho})$, restriction du graphe de non commutation de ϱ à l'alphabet de u, est connexe (respectivement fortement connexe).*

Nous ne disposons malheureusement que d'une condition suffisante pour décider si la clôture d'un langage régulier est régulière. Cette condition porte sur la connexité des facteurs itérants du langage et a été établie par Clerbout et Latteux dans [13] pour les semi-commutations et, de façon indépendante, par Ochmański dans [51] et Métivier dans [44, 45, 47] pour les commutations partielles.

Théorème 1.29 *Soient ϱ une commutation partielle définie sur un alphabet Σ et L un langage régulier inclus dans Σ^*. Si tout facteur itérant de L est connexe pour ϱ alors $\mathrm{f}_\varrho(L)$ est un langage régulier.*

Théorème 1.30 *Soient ϱ une semi-commutation définie sur un alphabet Σ et L un langage régulier inclus dans Σ^*. Si tout facteur itérant de L est fortement connexe pour ϱ alors $\mathrm{f}_\varrho(L)$ est un langage régulier.*

Dans le cas des langages réguliers, il est également possible de décider, de façon automatique, si un langage donné est clos par une commutation partielle ou par une semi-commutation en testant une propriété de l'automate fini minimal déterministe qui le reconnaît. Dans le cas des commutations partielles, nous avons la propriété « *diamant* » :

Proposition 1.31 (Diamant [46]) *Soient ϱ une commutation partielle définie sur l'alphabet Σ et L un langage régulier de Σ^*. Notons $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ l'automate déterministe minimal complet reconnaissant L. Le langage L est clos par ϱ si et seulement si pour tout couple (a, b) de ϱ et pour tout état q de Q, $\delta(q, ab) = \delta(q, ba)$. Cette propriété de l'automate est appelée propriété de diamant.*

Dans le cas des semi-commutations, le résultat est dû à Gonzalez :

Proposition 1.32 (Semi-diamant [29]) *Soient ϱ une semi-commutation définie sur l'alphabet Σ et L un langage régulier de Σ^*. Notons $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ l'automate minimal déterministe complet reconnaissant L. Le langage L est clos par ϱ si et seulement si pour tout couple (a, b) de ϱ et pour tout état q de Q tels que $\delta(q, ab) = q'$, il existe un état q'' de Q tel que $\delta(q, ba) = q''$ et tel que le langage reconnu par $(\Sigma, Q, \delta, q', F)$ est inclus dans le langage reconnu par $(\Sigma, Q, \delta, q'', F)$ (si (b, a) appartient également à ϱ, ces deux langages sont égaux). Cette propriété de l'automate est appelée propriété de semi-diamant.*

1.3 Quelques propriétés des st-langages

Nous énonçons, dans cette dernière section, quelques propriétés relatives à la famille des st-langages et aux systèmes de réécriture particuliers que nous allons utiliser. Remarquons tout d'abord que l'alphabet des facteurs itérants d'un st-langage n'est pas quelconque :

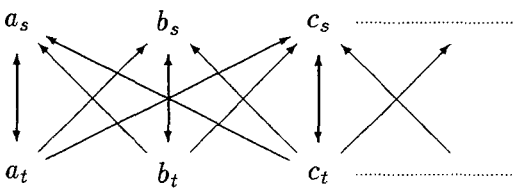
Fait 1.33 *Soit L un st-langage sur l'alphabet d'actions Σ. Nous avons :*

$$\forall u \in \mathrm{fi}(L),\ \forall x \in \Sigma,\ |u|_{x_s} = |u|_{x_t}.$$

De ce fait nous pouvons déduire, grâce à la condition suffisante du Théorème 1.30, que la clôture par θ d'un st-langage régulier est toujours régulière. Ceci nous permet de donner une preuve très simple du théorème suivant de Salomaa et Yu :

Théorème 1.34 (Salomaa et Yu [57]) *Soit L un st-langage régulier défini sur l'alphabet d'actions Σ. Le langage $\mathrm{f}_{\theta_\Sigma}(L)$ est régulier.*

Preuve. Tout facteur itérant d'un st-langage contient, pour toute action x, autant d'occurrences de x_s que d'occurrences de x_t (Fait 1.33). En conséquence, tout facteur itérant d'un st-langage est fortement connexe, son graphe de non commutation étant du type :



D'après le Théorème 1.30, la clôture par θ de tout st-langage régulier est donc régulière. □

Remarque 1.35 *Dans [46], Métivier introduit une procédure* S, *qui donne un algorithme pour calculer la clôture par une commutation partielle d'un langage régulier à partir de l'automate déterministe minimal du langage lorsque tous ses facteurs itérants sont connexes pour la commutation partielle (ceci a été montré par Métivier, Richomme et Wacrenier dans [48], une présentation plus récente de* S *peut être trouvée dans [22]). Cette procédure pouvant également être appliquée pour les semi-commutations, elle permet de calculer, dans tous les cas, la clôture d'un st-langage par θ puisque tous les facteurs itérants d'un st-langage régulier sont fortement connexes pour θ.*

Nous avons encore à remarquer une propriété intéressante de la clôture des st-langages par le système de réécriture R par rapport aux opérations de concaténation, étoile et union. Tout d'abord, dans le cas des st-mots, nous avons le lemme suivant :

Lemme 1.36 *Soient u et v deux st-mots définis sur l'alphabet d'actions Σ. Alors, nous avons :*

$$\mathrm{f}_{R_\Sigma}(uv) = \mathrm{f}_{R_\Sigma}(u).\mathrm{f}_{R_\Sigma}(v).$$

Preuve. Clairement, il suffit de montrer

$$(uv \underset{R_\Sigma}{\longrightarrow} w) \Rightarrow (w \in \mathrm{f}_{R_\Sigma}(u).\mathrm{f}_{R_\Sigma}(v)).$$

Comme u et v sont des st-mots, la dernière lettre de u appartient à Σ_t et la première lettre de v à Σ_s or, aucune règle de θ ne permet de commuter une lettre de Σ_t et une lettre de Σ_s dans cet ordre. Si la règle utilisée dans cette dérivation est une règle de θ, nous obtenons le résultat.

Si la règle utilisée dans cette dérivation est une règle de Ω. Nous avons

$$uv = w_1 a_{1t} \dots a_{nt} a_{1s} \dots a_{ns} b_{1t} \dots b_{mt} b_{1s} \dots b_{ms} w_2$$

$$w = w_1 b_{1t} \dots b_{mt} b_{1s} \dots b_{ms} a_{1t} \dots a_{nt} a_{1s} \dots a_{ns} w_2.$$

Comme u et v sont des st-mots, soit u est un facteur gauche de w_1, soit v est un facteur droit de w_2. Dans les deux cas, w appartient à $\mathrm{f}_{R_\Sigma}(u).\mathrm{f}_{R_\Sigma}(v)$. □

Lemme 1.37 *Soient L_1 et L_2 des st-langages réguliers définis sur l'alphabet d'actions Σ. Nous avons:*

$$\mathrm{f}_{R_\Sigma}(L_1 + L_2) = \mathrm{f}_{R_\Sigma}(L_1) + \mathrm{f}_{R_\Sigma}(L_2),$$
$$\mathrm{f}_{R_\Sigma}(L_1.L_2) = \mathrm{f}_{R_\Sigma}(L_1).\mathrm{f}_{R_\Sigma}(L_2),$$
$$\mathrm{f}_{R_\Sigma}((L_1)^*) = (\mathrm{f}_{R_\Sigma}(L_1))^*.$$

Preuve. L'égalité $\mathrm{f}_{R_\Sigma}(L_1 + L_2) = \mathrm{f}_{R_\Sigma}(L_1) + \mathrm{f}_{R_\Sigma}(L_2)$ est triviale.

Soit $L = L_1.L_2$. Pour tout mot u de $\mathrm{f}_{R_\Sigma}(L)$, il existe v appartenant à L_1 et w appartenant à L_2 tels que u appartient à $\mathrm{f}_{R_\Sigma}(vw)$. D'après le Lemme précédent, $\mathrm{f}_{R_\Sigma}(vw) = \mathrm{f}_{R_\Sigma}(v).\mathrm{f}_{R_\Sigma}(w)$ donc u appartient à $\mathrm{f}_{R_\Sigma}(L_1).\mathrm{f}_{R_\Sigma}(L_2)$. Réciproquement il est clair que $\mathrm{f}_{R_\Sigma}(L_1).\mathrm{f}_{R_\Sigma}(L_2)$ est inclus dans $\mathrm{f}_{R_\Sigma}(L)$.

Soit $L = (L_1)^*$. Tout mot u de L se décompose en st-mots de L_1. En utilisant la même méthode que ci-dessus, il est facile de montrer que les lettres de deux mots de L_1 consécutifs ne peuvent jamais commuter entre elles donc, $\mathrm{f}_{R_\Sigma}(L) = (\mathrm{f}_{R_\Sigma}(L_1))^*$. □

Cette propriété termine les remarques générales sur les st-langages et ce chapitre de présentation des différentes notions. Nous avons présenté les expressions et langages de synchronisation, les résultats existants ainsi que les principales notions ayant trait aux commutations partielles que nous utiliserons par la suite. Nous allons à présent entamer l'étude proprement dite des langages de synchronisation et de leur caractérisation.

Chapitre 2

Étude du système R

Dans ce chapitre, nous nous intéressons au système de réécriture R défini par Guo, Salomaa et Yu. Nous verrons dans un premier temps que, dans le cas particulier d'un alphabet d'actions ne contenant que deux actions, le système R permet effectivement de caractériser les langages de synchronisation comme proposé dans la conjecture I. Puis, nous nous intéresserons à une famille particulière de langages, les langages bien formés, proposés par Guo, Salomaa et Yu comme générateurs des langages de synchronisation et enfin, nous répondrons de façon négative à la conjecture I dans le cas général.

2.1 Langages définis sur des alphabets de 2 actions

Dans cette première partie, nous nous intéressons aux langages définis sur des alphabets de deux actions (soit quatre lettres) et nous montrons que la conjecture I est vraie dans ce cas particulier. Dans toute cette partie, nous allons supposer que $\Sigma = \{a, b\}$ et, comme nous aurons à considérer de nombreuses dérivations, nous allons distinguer les différentes parties de R pour en faciliter la manipulation. Nous utiliserons les notations suivantes : $\theta = \theta_1 \cup \theta_2 \cup \theta_3$, avec :

$$\begin{aligned} \theta_1 = &\ \{a_s b_s \longrightarrow b_s a_s, b_s a_s \longrightarrow a_s b_s\}, \\ \theta_2 = &\ \{a_t b_t \longrightarrow b_t a_t, b_t a_t \longrightarrow a_t b_t\}, \\ \theta_3 = &\ \{a_s b_t \longrightarrow b_t a_s, b_s a_t \longrightarrow a_t b_s\}. \end{aligned}$$

Rappelons que pour l'alphabet d'actions considéré, nous avons :

$$\Omega = \{a_t a_s b_t b_s \longrightarrow b_t b_s a_t a_s, b_t b_s a_t a_s \longrightarrow a_t a_s b_t b_s\}.$$

2.1.1 Générateur d'un st-langage régulier et R-clos

L'idée est de reconstruire les langages de synchronisation à l'aide de mots élémentaires. Nous appelons générateur d'un langage L clos par un système de réécriture tout langage ayant L pour clôture et nous allons exhiber des langages générateurs des langages de synchronisation à partir du langage

$$\mathbb{M} = (a_s a_t + b_s b_t + a_s b_s (a_t a_s)^* (b_t b_s)^* a_t b_t)^*.$$

Les mots $a_s a_t$ et $b_s b_t$ représentent l'exécution d'une action a et d'une action b et un facteur $a_s b_s (a_t a_s)^p (b_t b_s)^q a_t b_t$ représente une exécution de $p+1$ actions a et $q+1$ actions b contenant le plus de parallélisme possible.

Nous allons montrer qu'un mot sur notre alphabet de deux actions est un st-mot s'il peut être obtenu par R à partir d'un mot de $\mathbb{M}$. Remarquons auparavant le fait suivant :

Fait 2.1 *L'ensemble des mots st-primitifs sur l'alphabet $\Sigma = \{a, b\}$ est le langage :*

$$a_s a_t + b_s b_t + (a_s b_s + b_s a_s)(a_t a_s + b_t b_s)^*(a_t b_t + b_t a_t).$$

Lemme 2.2 *Un mot $u \in (a_s + a_t + b_s + b_t)^*$ est un st-mot si et seulement s'il existe un mot $v \in \mathbb{M}$ tel que $v \in \mathrm{f_R}(u)$ et $u \in \mathrm{f_R}(v)$.*

Preuve. La condition est clairement suffisante, comme $\mathbb{M}$ est un st-langage et que le système R préserve la st-propriété, si un mot v de $\mathbb{M}$ se réécrit en u, alors u est un st-mot.

Pour montrer que la condition est nécessaire, il suffit de considérer les mots st-primitifs. C'est clairement le cas si $u = a_s a_t$ ou $u = b_s b_t$ car alors $u \in \mathbb{M}$. Si $u \in (a_s b_s + b_s a_s)(a_t a_s + b_t b_s)^*(a_t b_t + b_t a_t)$, alors nous avons

$$u \in (a_s b_s + b_s a_s)(a_t a_s + b_t b_s)^m (a_t b_t + b_t a_t)$$

$$* \downarrow \ \theta_1$$

$$u_1 \in a_s b_s (a_t a_s + b_t b_s)^m (a_t b_t + b_t a_t)$$

$$* \downarrow \ \theta_2$$

$$u_2 \in a_s b_s (a_t a_s + b_t b_s)^m a_t b_t$$

$$* \downarrow \ \Omega$$

$$v = a_s b_s (a_t a_s)^p (b_t b_s)^q a_t b_t, p + q = m.$$

Comme v appartient à $\mathrm{f_R}(u)$ et à $\mathbb{M}$ et comme toutes les règles utilisées pour la dérivation sont symétriques, u appartient à $\mathrm{f_R}(v)$. La condition est donc nécessaire pour les mots primitifs.

Considérons u un st-mot et sa décomposition en mots st-primitifs $u = u_1 u_2 \dots u_n$. Nous avons montré qu'alors il existe $v_1, v_2, \dots, v_n$ appartenant à $\mathbb{M}$ tels que pour tout indice i, $u_i \in \mathrm{f_R}(v_i)$ et $v_i \in \mathrm{f_R}(u_i)$, le mot $v = v_1 v_2 \dots v_n$ appartient donc à $\mathbb{M}$ avec $u \in \mathrm{f_R}(v)$.

□

En conséquence du lemme précédent, nous allons pouvoir extraire du langage $\mathbb{M}$ des générateurs des st-langages réguliers et R-clos.

Corollaire 2.3 *Soit $L \subseteq (a_s + a_t + b_s + b_t)^*$ un st-langage régulier et R-clos. Il existe un langage régulier $K \subseteq \mathbb{M}$ tel que $L = f_R(K)$.*

Preuve. Il suffit de prendre $K = L \cap \mathbb{M}$. En effet $L \cap \mathbb{M}$ est régulier puisque L et $\mathbb{M}$ sont tous deux réguliers et d'après le lemme précédent, pour tout mot u de L, il existe un mot v de $\mathbb{M}$ tel que $u \in \mathrm{f_R}(v)$ et $v \in \mathrm{f_R}(u)$. Comme $L = f_R(L)$, v appartient à L, tout mot de L appartient donc à la clôture par R d'un mot de $L \cap \mathbb{M}$. □

Nous allons à présent montrer la réciproque de ce corollaire, c'est-à-dire, montrer que la clôture par R d'un langage régulier inclus dans $\mathbb{M}$ est régulière. Montrons tout d'abord que l'on peut décomposer la clôture par R d'un langage L en deux étapes en appliquant d'abord le système Ω puis le système θ.

Lemme 2.4 *Soit L un st-langage inclus dans $(a_s+a_t+b_s+b_t)^*$. Nous avons $\mathrm{f_R}(L) = \mathrm{f}_\theta \circ \mathrm{f}_\Omega(L)$.*

Preuve. Soient u un mot de L et w un mot de $\mathrm{f_R}(u)$. Montrons par récurrence sur la longueur de la dérivation de u à w que w appartient à $\mathrm{f}_\theta \circ \mathrm{f}_\Omega(u)$ ce qui est clairement vrai pour une dérivation de longueur nulle. Considérons une dérivation

$$u \xrightarrow[R]{n} v \xrightarrow[R]{} w,$$

par hypothèse de récurrence, nous avons

$$u \xrightarrow[\Omega]{*} v' \xrightarrow[\theta]{*} v \xrightarrow[R]{} w.$$

Si la règle appliquée pour passer de v à w est une règle de θ, nous avons terminé. Sinon, nous avons

$$u \xrightarrow[\Omega]{*} v' \xrightarrow[\theta]{*} v = w_1 a_t a_s b_t b_s w_2 \xrightarrow[R]{} w_1 b_t b_s a_t a_s w_2 = w.$$

Considérons le dernier pas de dérivation par θ impliquant $a_t a_s b_t b_s$, qui peut être

$$w'_1 a_t b_t a_s b_t b_s w'_2 \xrightarrow[\theta]{} w'_1 b_t a_t a_s b_t b_s w'_2 \text{ avec } w'_1 b_t \xrightarrow[\theta]{*} w_1 \text{ et } w'_2 \xrightarrow[\theta]{*} w_2$$

ou

$$w'_1 a_t a_s b_t a_s b_s w'_2 \xrightarrow[\theta]{} w'_1 a_t a_s b_t b_s a_s w'_2 \text{ avec } w'_1 \xrightarrow[\theta]{*} w_1 \text{ et } a_s w'_2 \xrightarrow[\theta]{*} a_s w_2.$$

Comme ni le facteur $b_t a_s b_t$ ni le facteur $a_s b_t a_s$ ne peuvent appartenir à un st-mot, si la règle utilisée pour passer de v à w appartient à Ω alors v et v' sont égaux et w appartient à $\mathrm{f}_\Omega(u)$. Dans chacun des cas, w appartient à $\mathrm{f}_\theta \circ \mathrm{f}_\Omega(u)$. □

Le calcul de la clôture d'un langage par R peut être décomposé en deux étapes dont la première est le calcul de la clôture du langage par Ω. Nous allons montrer que la clôture par Ω préserve la régularité grâce à une propriété vérifiée par les facteurs itérants du langage $\mathbb{M}$.

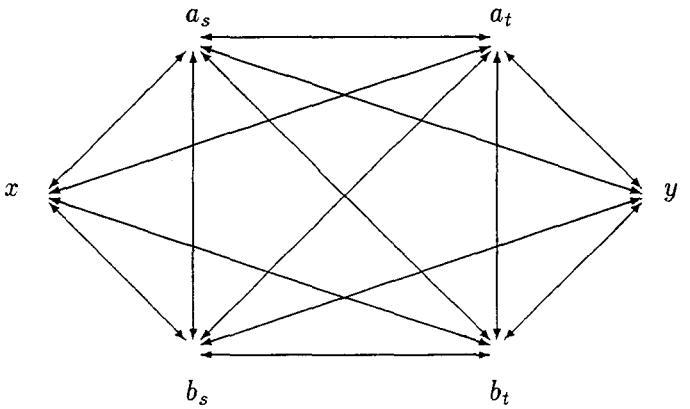
Lemme 2.5 *Soit L un langage régulier inclus dans $\mathbb{M}$. Le langage $\mathrm{f}_\Omega(L)$ est un langage régulier.*

Preuve. Considérons le nouvel alphabet $\mathrm{X} = \{x, y, a_s, a_t, b_s, b_t\}$, le morphisme

$$\begin{array}{rccc} h : & \mathrm{X}^* & \longrightarrow & (\Sigma_s \cup \Sigma_t)^* \\ & x & \longmapsto & a_t a_s \\ & y & \longmapsto & b_t b_s \\ & z \in \Sigma_s \cup \Sigma_t & \longmapsto & z \end{array}$$

et le système de réécriture $S = \{xy \longrightarrow yx, yx \longrightarrow xy\}$.

Clairement, nous avons $h \circ \mathrm{f}_S \circ h^{-1}(L) = \mathrm{f}_\Omega(L)$. Le langage L étant régulier, son image par le morphisme inverse h^{-1} est régulière. Le système S est une commutation partielle dont le graphe de non commutation est le suivant :



D'après la forme de la décomposition en facteurs st-primitifs d'un st-mot sur $\{a, b\}$ que nous avons vue précédemment, dans un mot de $h^{-1}(L)$, x et y peuvent uniquement être voisins dans un facteur du type $a_s b_s (x + a_t a_s)^* x^+ y^+ (y + b_t b_s)^* a_t b_t$. En conséquence, aucun facteur itérant du langage n'a pour alphabet $\{x, y\}$. Tous les facteurs itérants de $h^{-1}(L)$ sont connexes donc d'après le Théorème 1.29, $\mathrm{f}_S \circ h^{-1}(L)$ est régulier et son image par un morphisme, $h \circ \mathrm{f}_S \circ h^{-1}(L)$, est régulière. En conclusion, la clôture par Ω d'un langage inclus dans $\mathbb{M}$ est régulière. □

Les deux lemmes précédents nous permettent d'obtenir le lemme :

Lemme 2.6 *La clôture par R d'un langage régulier inclus dans $\mathbb{M}$ est régulière.*

Preuve. Soit L un langage régulier inclus dans $\mathbb{M}$. D'après le Lemme 2.4, $\mathrm{f}_{\mathrm{R}}(L) = \mathrm{f}_\theta \circ \mathrm{f}_\Omega(L)$. Comme $L \subseteq \mathbb{M}$, d'après le Lemme 2.5, $\mathrm{f}_\Omega(L)$ est un langage régulier. D'après le Théorème 1.34, la clôture par θ du st-langage régulier $\mathrm{f}_\Omega(L)$ est régulière. □

Du Corollaire 2.3 et du Lemme 2.6, nous déduisons à présent la proposition :

Proposition 2.7 *Soit $L \subseteq (a_s + a_t + b_s + b_t)^*$. Le langage L est un st-langage régulier et R-clos si et seulement s'il existe un langage régulier $K \subseteq \mathbb{M}$ tel que $L = \mathrm{f}_{\mathrm{R}}(K)$.*

2.1.2 Décomposition d'un st-langage et R-clos

Nous allons à présent décomposer un st-langage L régulier et R-clos en utilisant la décomposition d'un langage $K \subseteq \mathbb{M}$ tel que $L = \mathrm{f}_{\mathrm{R}}(K)$ et montrer que l'on peut construire L par produit, union et étoile de langages très simples. À partir d'une telle décomposition, nous pourrons reconstruire une expression de synchronisation e telle que $L = L(e)$.

La première étape consiste à décomposer un langage inclus dans $\mathbb{M}$ en union, produit et étoile de langages élémentaires.

Lemme 2.8 *Soit $K \subseteq \mathbb{M} = (a_s a_t + b_s b_t + a_s b_s (a_t a_s)^* (b_t b_s)^* a_t b_t)^*$ un langage régulier. Le langage K peut être décrit par une expression rationnelle sur $a_s a_t$, $b_s b_t$ et des sous-ensembles réguliers de $a_s b_s (a_t a_s)^* (b_t b_s)^* a_t b_t$.*

Preuve. Considérons $\mathcal{A} = (\{a_s, a_t, b_s, b_t\}, Q, q_0, \delta, F)$ l'automate déterministe minimal reconnaissant K. Pour toute paire d'états q et q' de $\mathcal{A}$, nous notons $R_{qq'}$ le langage :

$$R_{qq'} = \{w \in \{a_s, a_t, b_s, b_t\}^* \mid \delta(q, w) = q'\}.$$

Soit X le nouvel alphabet défini par :

$$\mathrm{X} = \{x_{qq'}, y_{qq'}, z_{qq'} \mid q, q' \in Q\}$$

et soit σ la substitution :

$$\begin{array}{rccl} \sigma & : & \mathrm{X}^* & \longrightarrow & (\Sigma_s \cup \Sigma_t)^* \\ & & x_{qq'} & \longmapsto & a_s a_t \cap R_{qq'} \\ & & y_{qq'} & \longmapsto & b_s b_t \cap R_{qq'} \\ & & z_{qq'} & \longmapsto & a_s b_s (a_t a_s)^* (b_t b_s)^* a_t b_t \cap R_{qq'}. \end{array}$$

Nous allons montrer que $K = \sigma(\sigma^{-1}(K) \cap T)$ avec

$$\begin{array}{rl} T = \{\alpha_1 \alpha_2 \ldots \alpha_n \mid & \alpha_i \in \mathrm{X}, \\ & \alpha_1 \in \{x_{q_0 q_i}, y_{q_0 q_i}, z_{q_0 q_i} \mid q_i \in Q\}, \\ & \alpha_n \in \{x_{q_i q_f}, y_{q_i q_f}, z_{q_i q_f} \mid q_i \in Q, q_f \in F\}, \\ & (\exists j, \alpha_i \in \{x_{q_i q_j}, y_{q_i q_j}, z_{q_i q_j} \mid q_i \in Q\}) \Rightarrow \\ & (\alpha_{i+1} \in \{x_{q_j q_k}, y_{q_j q_k}, z_{q_j q_k} \mid q_k \in Q\})\}. \end{array}$$

Le langage T est clairement rationnel puisque c'est un langage local.

Montrons en premier lieu l'inclusion $\sigma(\sigma^{-1}(K) \cap T) \subseteq K$. Soit w appartenant à $\sigma^{-1}(K) \cap T$. Le mot w s'écrit $w = \alpha_{q_0 q_1} \alpha_{q_1 q_2} \ldots \alpha_{q_{n-1} q_n}$ où, pour tout indice j, $\alpha_{q_j q_{j+1}}$ appartient à $\{x_{q_j q_{j+1}}, y_{q_j q_{j+1}}, z_{q_j q_{j+1}}\}$.

Pour tout u appartenant à $\sigma(w)$, il existe un découpage $u = u_1 \ldots u_n$ tel que pour tout indice i, le mot u_i appartient à $\sigma(\alpha_{q_{i-1} q_i})$ par construction, il existe, dans l'automate $\mathcal{A}$, un chemin $(q_0, u_1, q_1) \ldots (q_{n-1}, u_n, q_n)$ avec q_n état final donc u appartient à K.

Montrons à présent l'inclusion $K \subseteq \sigma(\sigma^{-1}(K) \cap T)$. Pour tout mot w de K, il existe un découpage $w = w_1 w_2 \ldots w_n$ avec pour tout indice i :

$$w_i \in \{a_s a_t, b_s b_t\} \cup \{a_s b_s (a_t a_s)^p (b_t b_s)^q a_t b_t | p, q \in \mathbb{N}\}.$$

Il existe donc un unique chemin $(q_0, w_1, q_1)(q_1, w_2, q_2) \ldots (q_{n-1}, w_n, q_n)$ avec q_n état final de l'automate $\mathcal{A}$.

Considérons le mot $\alpha = \alpha_{q_0 q_1} \alpha_{q_1 q_2} \ldots \alpha_{q_{n-1} q_n}$ où pour tout indice j, $\alpha_{q_{j-1} q_j}$ est égal respectivement à $x_{q_{j-1} q_j}$, $y_{q_{j-1} q_j}$ ou $z_{q_{j-1} q_j}$ quand w_j est respectivement égal à $a_s a_t$, $b_s b_t$ ou $a_s b_s (a_t a_s)^p (b_t b_s)^q a_t b_t$. Par construction, w appartient à $\sigma(\alpha)$, par conséquent, α appartient à $\sigma^{-1}(K)$ de plus, clairement, α appartient à T, donc w appartient à $\sigma(\sigma^{-1}(K) \cap T)$. □

En conséquence, un langage régulier inclus dans $\mathbb{M}$ peut se décomposer en produit, union et étoile de langages élémentaires qui sont $a_s a_t$, $b_s b_t$ ou des langages réguliers inclus dans $a_s b_s (a_t a_s)^* (b_t b_s)^* a_t b_t$. Clairement, les langages élémentaires $a_s a_t$ et $b_s b_t$ sont des langages de synchronisation correspondant aux expressions a et b. Pour associer une expression de synchronisation à la clôture d'un langage régulier inclus dans $a_s b_s (a_t a_s)^* (b_t b_s)^* a_t b_t$ il est nécessaire de procéder à une nouvelle décomposition. Montrons en premier lieu que la clôture par R d'un langage du type $a_s b_s (a_t a_s)^{k_1} ((a_t a_s)^{k_2})^* (b_t b_s)^{k_3} ((b_t b_s)^{k_4})^* a_t b_t$ est un langage de synchronisation.

Lemme 2.9 *Soit L un st-langage régulier de la forme*

$$L = a_s b_s (a_t a_s)^{k_1} ((a_t a_s)^{k_2})^* (b_t b_s)^{k_3} ((b_t b_s)^{k_4})^* a_t b_t.$$

Alors $\mathrm{f_R}(L)$ est un langage de synchronisation avec

$$\mathrm{f_R}(L) = \Pi_{\{a_s, a_t\}}(L) \sqcup \Pi_{\{b_s, b_t\}}(L).$$

Preuve. Clairement, nous avons $f_R(L) \subseteq \Pi_{\{a_s,a_t\}}(L) \sqcup \Pi_{\{b_s,b_t\}}(L)$.

Nous allons tout d'abord montrer l'inclusion inverse dans le cas d'un mot. Soit $u = a_s b_s (a_t a_s)^p (b_t b_s)^q a_t b_t$ avec p et q positifs ou nuls. Montrons que $\Pi_{\{a_s,a_t\}}(u) \sqcup \Pi_{\{b_s,b_t\}}(u) \subseteq f_R(u)$ par induction sur la longueur de u. Pour une longueur égale à 4, nous avons

$$\mathrm{f_R}(a_s b_s a_t b_t) = \{a_s a_t b_s b_t, a_s b_s a_t b_t, a_s b_s b_t a_t, b_s a_s a_t b_t, b_s a_s b_t a_t, b_s b_t a_s a_t\} = a_s a_t \sqcup b_s b_t.$$

Considérons un mot u de longueur $n+2$ et un mot $w \in \Pi_{\{a_s,a_t\}}(u) \sqcup \Pi_{\{b_s,b_t\}}(u)$. D'après le Fait 2.1, la décomposition de w en facteurs st-primitifs $w = w_1 \dots w_k$ ne contient que des mots de $a_s a_t + b_s b_t + (a_s b_s + b_s a_s)(a_t a_s + b_t b_s)^*(a_t b_t + b_t a_t)$. Si $k = 1$, soit le mot w appartient à $\{a_s a_t, b_s b_t\}$ et dans ce cas nous avons forcément $u = w$, soit w appartient à $(a_s b_s + b_s a_s)(a_t a_s + b_t b_s)^*(a_t b_t + b_t a_t)$ et dans ce cas nous avons déjà vu (dans la preuve du Lemme 2.2) que nous avons une dérivation :

$$w \in (a_s b_s + b_s a_s)(a_t a_s + b_t b_s)^m (a_t b_t + b_t a_t) \xrightarrow[\theta]{*} u_2 \in a_s b_s (a_t a_s + b_t b_s)^m a_t b_t$$
$$u_2 \xrightarrow[\Omega]{*} a_s b_s (a_t a_s)^p (b_t b_s)^q a_t b_t = u.$$

Nous allons donc considérer le cas $k > 1$.

- Si $w_1 = a_s a_t$ et $p = 0$, il est facile de voir que l'on peut ramener a_t sur la gauche. Si $p \neq 0$, une dérivation par θ de longueur 2 permet de passer de u à $a_s a_t a_s b_s (a_t a_s)^{p-1} (b_t b_s)^q a_t b_t$. Par hypothèse de récurrence, $w_2 \dots w_k$ appartient à $\mathrm{f_R}(a_s b_s (a_t a_s)^{p-1} (b_t b_s)^q a_t b_t)$, donc w appartient à $\mathrm{f_R}(u)$.

- Si $w_1 = b_s b_t$ et $q = 0$, il est facile de voir que l'on peut commuter a_s et b_s et ramener b_t sur la gauche. Si $q \neq 0$, alors une dérivation par Ω de longueur p permet de passer de u à $a_s b_s b_t b_s (a_t a_s)^p (b_t b_s)^{q-1} a_t b_t$ puis, par deux pas de dérivation par θ, on obtient $b_s b_t a_s b_s (a_t a_s)^p (b_t b_s)^{q-1} a_t b_t$. Par hypothèse de récurrence, $w_2 \dots w_k$ appartient à $\mathrm{f_R}(a_s b_s (a_t a_s)^p (b_t b_s)^{q-1} a_t b_t)$, donc w appartient à $\mathrm{f_R}(u)$.

- Dans le dernier cas, w_1 appartient à $(a_s b_s + b_s a_s)(a_t a_s + b_t b_s)^*(a_t b_t + b_t a_t)$. Notons respectivement $i+1$ et $j+1$ les nombres de a_s et b_s présents dans w_1. Nous avons vu (Lemme 2.2), qu'alors le mot $a_s b_s (a_t a_s)^i (b_t b_s)^j a_t b_t$ se réécrit par R, en utilisant uniquement des règles symétriques, en w_1. Nous allons isoler ce facteur durant la dérivation. Nous avons :

  $$u \xrightarrow[\Omega]{*} a_s b_s (a_t a_s)^i (b_t b_s)^j (a_t a_s)^{p-i} (b_t b_s)^{q-j} a_t b_t.$$

  Clairement, si $q - j$ est nul, nous pouvons ramener par θ_3 le dernier b_t sur la gauche pour terminer. De même, si $p - i$ est nul, nous pouvons ramener par une alternance de θ_3 et θ_2 le dernier a_t sur la gauche pour terminer. Sinon, nous avons :

$$\begin{array}{c} a_s b_s (a_t a_s)^i (b_t b_s)^j (a_t a_s)^{p-i} (b_t b_s)^{q-j} a_t b_t \\ \Omega \downarrow (p-i) \\ a_s b_s (a_t a_s)^i (b_t b_s)^{j+1} (a_t a_s)^{p-i} (b_t b_s)^{q-j-1} a_t b_t \\ \theta_3 \downarrow \\ a_s b_s (a_t a_s)^i (b_t b_s)^j b_t a_t b_s a_s (a_t a_s)^{p-i-1} (b_t b_s)^{q-j-1} a_t b_t \\ \theta_2 \downarrow \\ a_s b_s (a_t a_s)^i (b_t b_s)^j a_t b_t b_s a_s (a_t a_s)^{p-i-1} (b_t b_s)^{q-j-1} a_t b_t \\ \theta_1 \downarrow \\ a_s b_s (a_t a_s)^i (b_t b_s)^j a_t b_t a_s b_s (a_t a_s)^{p-i-1} (b_t b_s)^{q-j-1} a_t b_t \end{array}$$

Par hypothèse de récurrence, $w_2 \dots w_k$ appartient à $\mathrm{f_R}(a_s b_s (a_t a_s)^{p-i-1} (b_t b_s)^{q-j-1} a_t b_t)$, donc w appartient à $\mathrm{f_R}(u)$.

En conséquence, pour tout mot u de la forme $a_s b_s (a_t a_s)^p (b_t b_s)^q a_t b_t$, $\Pi_{\{a_s,a_t\}}(u) \sqcup\!\sqcup \Pi_{\{b_s,b_t\}}(u)$ est inclus dans $\mathrm{f_R}(u)$.

Revenons à présent au cas des langages. Pour tout $(a_s a_t)^p$ appartenant à $\Pi_{\{a_s,a_t\}}(L)$ et pour tout $(b_s b_t)^q$ appartenant à $\Pi_{\{b_s,b_t\}}(L)$, le mot $a_s b_s (a_s a_t)^{p-1} (b_s b_t)^{q-1} a_t b_t$ appartient à L donc, pour tout u de $\Pi_{\{a_s,a_t\}}(L)$ et pour tout v de $\Pi_{\{b_s,b_t\}}(L)$, il existe un mot w de L tel que $\Pi_{\{a_s,a_t\}}(w) = u$ et $\Pi_{\{b_s,b_t\}}(w) = v$. Nous avons :

$$\mathrm{f_R}(L) = \Pi_{\{a_s,a_t\}}(L) \sqcup\!\sqcup \Pi_{\{b_s,b_t\}}(L) = [(a_s a_t)^{k_1+1} ((a_s a_t)^{k_2})^*] \sqcup\!\sqcup [(b_s b_t)^{k_3+1} ((b_s b_t)^{k_4})^*].$$

Les langages $[(a_s a_t)^{k_1+1} ((a_s a_t)^{k_2})^*]$ et $[(b_s b_t)^{k_3+1} ((b_s b_t)^{k_4})^*]$ sont des langages de synchronisation c'est pourquoi leur shuffle, $\mathrm{f_R}(L)$, est un langage de synchronisation. □

Dans le but de montrer qu'un langage $L = \mathrm{f_R}(K)$ avec $K \subseteq \mathbb{M}$ est un langage de synchronisation, nous avons exprimé K comme produit, union et étoile de langages élémentaires. Or, nous savons que la clôture par R d'un produit, d'une union ou de l'étoile de st-langages correspond au produit, à l'union ou à l'étoile de leur clôture 1.37, nous obtenons donc la proposition suivante :

Proposition 2.10 *Soit Σ un alphabet de deux actions. Tout st-langage régulier clos par R défini sur $\Sigma_s \cup \Sigma_t$ est un langage de synchronisation dont on peut effectivement construire une expression.*

Preuve. Considérons $\Sigma = \{a, b\}$ et L un st-langage régulier clos par R défini sur $\Sigma_s \cup \Sigma_t$. D'après le Corollaire 2.3, il existe un langage régulier K inclus dans $(a_s a_t + b_s b_t + a_s b_s (a_t a_s)^* (b_t b_s)^* a_t b_t)^*$ qui génère L. D'après le Lemme 2.8, nous savons que K se décompose en produit, union et étoile de langages réguliers K_i égaux à $\{a_s a_t\}$, $\{b_s b_t\}$ ou inclus dans $a_s b_s (a_t a_s)^* (b_t b_s)^* a_t b_t$. Donc, d'après le Lemme 1.37, cette décomposition de K peut être utilisée pour construire une décomposition (ayant la même structure) de L en produit, union et étoile des $\mathrm{f_R}(K_i)$. Il suffit donc de montrer que les $\mathrm{f_R}(K_i)$ sont des langages de synchronisation.

Clairement, $\{a_s a_t\}$ et $\{b_s b_t\}$ sont des langages de synchronisation correspondant aux expressions a et b.

Un langage régulier inclus dans $a_s b_s (a_t a_s)^* (b_t b_s)^* a_t b_t$, est de la forme $a_s b_s M a_t b_t$ avec $M \subseteq (a_t a_s)^* (b_t b_s)^*$ donc, d'après les résultats de Ginsburg et Spanier [28], ce langage est une union finie de langages du type $a_s b_s (a_t a_s)^i ((a_t a_s)^j)^* (b_t b_s)^k ((b_t b_s)^l)^* a_t b_t$. La clôture de chacun de ces langages est, d'après le Lemme 2.9, un langage de synchronisation associé à l'expression $(a^i \rightarrow (a^j)^*) \parallel (b^k \rightarrow (b^l)^*)$.

À partir de la décomposition de L en produit, union et étoile d'un nombre fini de langages de synchronisation, il est facile de contruire une expression de synchronisation en remplaçant chaque langage de synchronisation par son expression et les opérateurs union par exclusion et produit par mise en séquence. □

Remarque 2.11 *La construction que nous obtenons nous montre que l'opérateur & n'est pas nécessaire à l'écriture des expressions de synchronisation ne comportant que deux actions distinctes.*

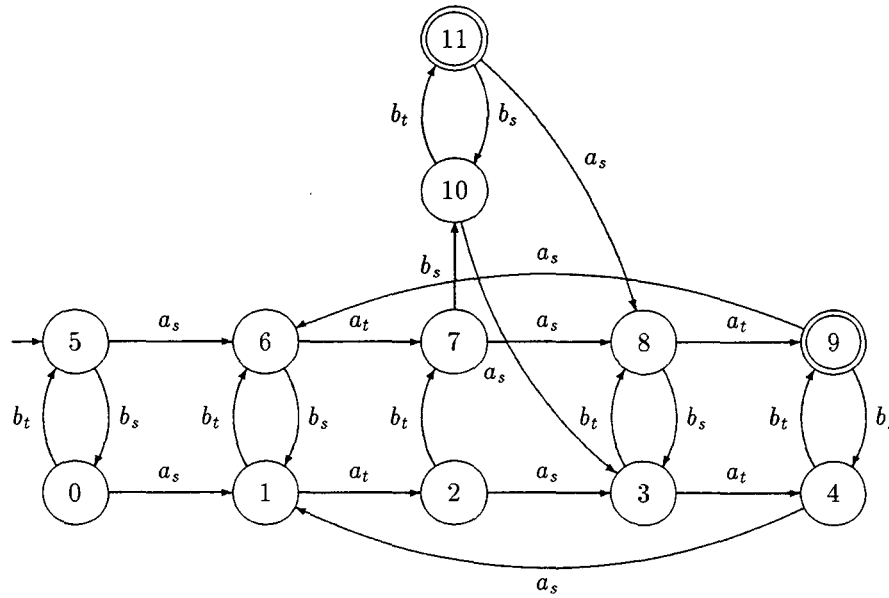


FIG. 2.1 - *Le langage L*

2.1.3 Exemple de construction

Nous allons construire une expression de synchronisation associée au st-langage régulier L défini par l'automate de la Figure 2.1 en utilisant la méthode décrite depuis le début du chapitre.

Le langage L est clos par R puisque l'automate ci-dessous possède les propriétés suivantes :

- la propriété diamant pour les règles symétriques de θ (voir Proposition 1.31) ;
- la propriété de semi-diamant pour les règles non symétriques de θ (voir Proposition 1.32) ;
- la généralisation de la propriété diamant pour la règle $a_t a_s b_t b_s \longrightarrow b_t b_s a_t a_s$ de Ω. En effet, il est facile de vérifier sur l'automate du langage L que pour toute paire d'états q et q' tels qu'il existe un chemin de q à q' étiqueté par $a_t a_s b_t b_s$ (resp. $b_t b_s a_t a_s$) – ici il s'agit des paires d'états $(1,3)$ et $(3,1)$ – il existe un chemin de q à q' étiqueté par $b_t b_s a_t a_s$ (resp. $a_t a_s b_t b_s$).

D'après le Lemme 2.3, il existe un langage K inclus dans $\mathbb{M}$ tel que $L = \mathrm{f_R}(K)$ et il suffit de prendre $K = L \cap \mathbb{M}$. Le langage K est défini par l'automate donné Figure 2.2.

À présent, il nous faut calculer $\sigma^{-1}(K) \cap T$ d'après la construction de la preuve du Lemme 2.8 pour pouvoir décomposer K en langages élémentaires. Nous obtenons pour $\sigma^{-1}(K) \cap T$ l'automate donné Figure 2.3.

Comme pour toute paire d'états q_i et q_j de l'automate ci-dessus, $\sigma(x_{q_i q_j}) = a_s a_t$ et $\sigma(y_{q_i q_j}) = b_s b_t$, il suffit d'utiliser les lettres x et y. D'autre part, nous avons les égalités $\sigma(z_{q_0 q_3}) = \sigma(z_{q_{11} q_9}) = \sigma(z_{q_3 q_9}) = \sigma(z_{q_9 q_3}) = a_s b_s (a_t a_s a_t a_s)^* (b_t b_s)^* a_t b_t$ et $\sigma(z_{q_9 q_9}) = \sigma(z_{q_0 q_9}) = a_s b_s a_t a_s (a_t a_s a_t a_s)^* (b_t b_s)^* a_t b_t$, nous utiliserons donc respectivement z_1 et z_2, nous obtenons donc l'automate donnée Figure 2.4 :

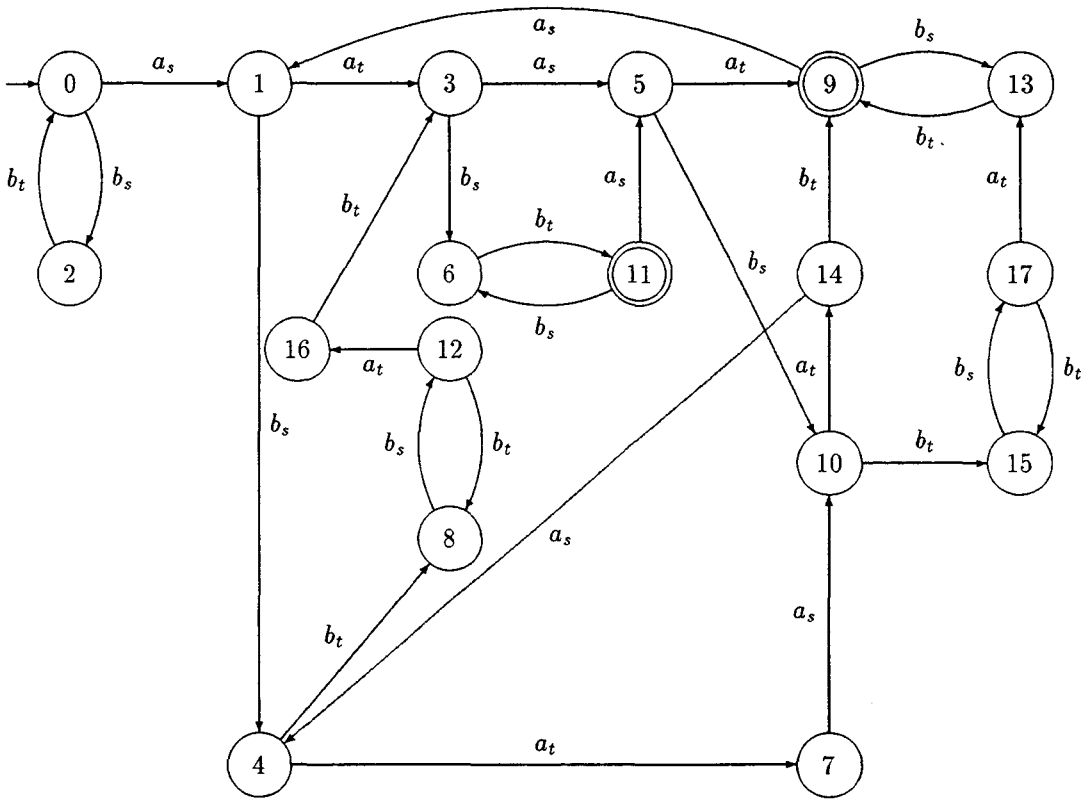


FIG. 2.2 - *Le langage K*

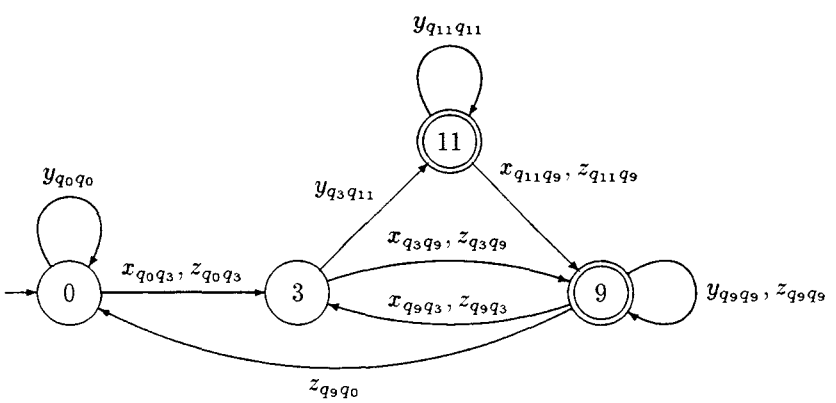


FIG. 2.3 - *Le langage* $\sigma^{-1}(K) \cap T$

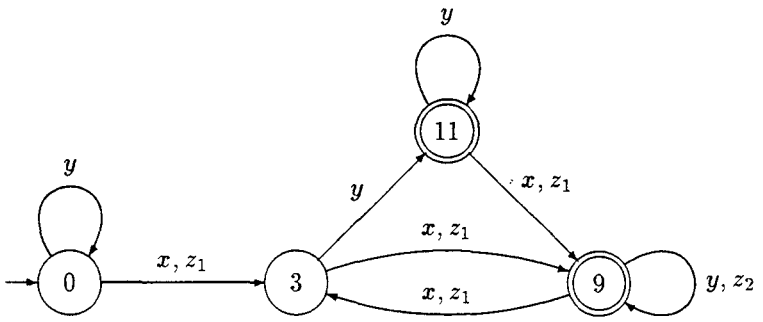


FIG. 2.4 - *Le langage $\sigma^{-1}(K) \cap T$ utilisé*

Nous obtenons pour $\sigma^{-1}(K) \cap T$ l'expression rationnelle :

$$y^*(x + z_1)y^*[y + (x + z_1)((y + z_2) + (x + z_1)y^*(x + z_1))^*](\varepsilon + (x + z_1)y^+)$$

Il suffit donc d'appliquer σ à chacun des membres de l'expression pour obtenir une décomposition de K.

$$\begin{array}{c}(b_sb_t)^*(a_sa_t + a_sb_s(a_ta_sa_ta_s)^*(b_tb_s)^*a_tb_t)(b_sb_t)^* \\ (b_sb_t + (a_sa_t + a_sb_s(a_ta_sa_ta_s)^*(b_tb_s)^*a_tb_t) \\ ((b_sb_t + a_sb_sa_ta_s(a_ta_sa_ta_s)^*(b_tb_s)^*a_tb_t) + (a_sa_t + a_sb_s(a_ta_sa_ta_s)^*(b_tb_s)^*a_tb_t) \\ (b_sb_t)^*(a_sa_t + a_sb_s(a_ta_sa_ta_s)^*(b_tb_s)^*a_tb_t))^*) \\ (\varepsilon + (a_sa_t + a_sb_s(a_ta_sa_ta_s)^*(b_tb_s)^*a_tb_t)(b_sb_t)^+)\end{array}$$

Les expressions de synchronisation associées aux langages élémentaires sont les suivantes :

$$\begin{array}{rcl} a_sa_t & \longmapsto & a \\ b_sb_t & \longmapsto & b \\ a_sb_s(a_ta_sa_ta_s)^*(b_tb_s)^*a_tb_t & \longmapsto & ((a \rightarrow (a \rightarrow a)^*) \parallel b^+) \\ a_sb_sa_ta_s(a_ta_sa_ta_s)^*(b_tb_s)^*a_tb_t & \longmapsto & ((a \rightarrow a)^+) \parallel b^+) \end{array}$$

Nous obtenons donc, pour L, l'expression de synchronisation :

$$\begin{array}{c} b^*(a + ((a \rightarrow (a \rightarrow a)^*) \parallel b^+))b^* \\ (b + (a + ((a \rightarrow (a \rightarrow a)^*) \parallel b^+)) \\ ((b + ((a \rightarrow a)^+) \parallel b^+)) + (a + ((a \rightarrow (a \rightarrow a)^*) \parallel b^+)) \\ b^*(a + ((a \rightarrow (a \rightarrow a)^*) \parallel b^+)))^*) \\ (\varepsilon + (a + ((a \rightarrow (a \rightarrow a)^*) \parallel b^+))b^+) \end{array}$$

L'expression de synchronisation obtenue par la méthode systématique n'est pas, en général, la « plus courte » possible, elle dépend totalement de l'expression rationnelle obtenue pour $\sigma^{-1}(K) \cap T$.

Nous avons apporté un réponse positive à la conjecture I dans le cas de langages définis sur des alphabets de deux actions. De plus, il est possible de construire, de façon systématique, une expression de synchronisation associée à un tel langage. Nous allons à présent nous intéresser au cas d'alphabets quelconques.

2.2 Langages bien formés

De même que nous avons utilisé, dans la section précédente, le langage $\mathbb{M}$ comme générateur des langages définis sur des alphabets de deux actions, Guo a proposé, dans sa thèse [32] (et également dans [34]), une famille de langages, les langages bien formés, comme générateurs des langages de synchronisation.

2.2.1 Définitions

Guo, Salomaa et Yu définissent dans [34] les expressions rationnelles et les langages bien formés :

Définition 2.12 *Une expression rationnelle α est bien formée si pour toute sous-expression $(\beta)^*$ de α, β est l'expression rationnelle d'un st-langage.*

Définition 2.13 *Un langage est bien formé s'il peut être défini par une expression rationnelle bien formée.*

Il est bien entendu qu'un langage possède de nombreuses expressions rationnelles, elles ne sont cependant pas toutes bien formées si le langage l'est. Par exemple $a_s b_s a_t (a_s a_t)^* a_s a_t b_t$ est une expression rationnelle bien formée mais pas $a_s b_s a_t a_s (a_t a_s)^* a_t b_t$, pourtant ces deux expressions définissent le même langage bien formé. Guo et al. ont montré, dans le même article, le théorème suivant et proposé une conjecture liant les langages bien formés aux langages de synchronisation.

Théorème 2.14 (Guo et al. [34]) *La clôture par θ d'un st-langage bien formé est un langage bien formé.*

Conjecture II (Guo et al. [34])

a - La clôture par R de tout st-langage bien formé est un langage de synchronisation.

b - Tout langage de synchronisation est la clôture par R d'un st-langage bien formé.

Un langage est bien formé s'il possède une expression rationnelle bien formée. Décider si un langage est bien formé n'est donc pas immédiat, par exemple le langage L_1 défini par l'expression rationnelle $a_s b_s c_s (b_t c_t b_s c_s + b_t b_s)^* a_t b_t c_t$ est bien formé alors que le langage L_2 défini par l'expression rationnelle $a_s b_s c_s (a_t b_t a_s b_s + c_t b_t c_s b_s)^* a_t b_t c_t$ ne l'est pas. En effet le langage L_1 peut être défini par l'expression rationnelle bien formée

$$a_s b_s c_s a_t b_t c_t + a_s b_s c_s b_t (b_s b_t)^* b_s a_t b_t c_t +$$
$$a_s b_s c_s (b_t (b_s b_t)^* b_s + \varepsilon) b_t c_t (b_s c_s b_t c_t + b_s c_s b_t (b_s b_t)^* b_s b_t c_t)^* b_s c_s (b_t (b_s b_t)^* b_s + \varepsilon) a_t b_t c_t.$$

Pourtant, les expressions rationnelles données pour les langages L_1 et L_2 semblent relativement similaires. Nous avons donc, dans un premier temps, mis en évidence une autre caractérisation des langages bien formés qui ne dépend pas de l'expression rationnelle choisie.

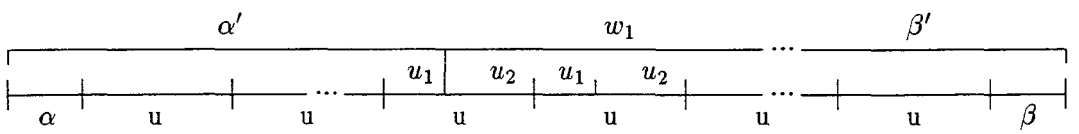
2.2.2 Caractérisation des st-langages bien formés

Nous allons, pour caractériser les langages bien formés, mettre en évidence une propriété de leurs facteurs itérants et proposer une condition nécessaire et suffisante pour décider si un st-langage régulier est un langage bien formé.

Lemme 2.15 *Tout facteur itérant d'un st-langage bien formé est le conjugué d'un st-mot.*

Preuve. Soit L un st-langage bien formé sur l'alphabet $\Sigma_s \cup \Sigma_t$. Il existe une expression rationnelle E pour L telle que pour toute sous-expression $(\beta)^*$ de E, β est l'expression rationnelle d'un st-langage. Notons $L_S = \{w \in \beta \mid (\beta)^* \text{ est une sous-expression de } E\}$.

Par définition, pour tout facteur itérant u de L, il existe α et β appartenant à $(\Sigma_s \cup \Sigma_t)^*$ tels que $\alpha u^* \beta$ est inclus dans L. Pour tout k positif ou nul, posons $v_k = \alpha u^k \beta$. Nous avons également $v_k = \alpha' w_1 .. w_n \beta'$ avec α' et β' appartenant à $(\Sigma_s \cup \Sigma_t)^*$, n positif ou nul et pour tout indice i de l'intervalle $1..n$, w_i appartient à L_S. Pour k assez grand, on peut prendre $|\alpha'| > |\alpha|$, $|\beta'| > |\beta|$ et n strictement positif. Nous avons :



Les w_i sont des st-mots donc $u_2 u_1$ est facteur gauche de st-mot et, pour toute action x de Σ, nous avons :

$$\Pi_{\{x_s, x_t\}}(u_2 u_1) \in (x_s x_t)^*(x_s + \varepsilon).$$

De plus, comme u est un facteur itérant de L, pour toute action x, nous avons $|u|_{x_s} = |u|_{x_t}$, donc pour toute action x :

$$\Pi_{\{x_s, x_t\}}(u_2 u_1) \in (x_s x_t)^*.$$

Le mot $u_2 u_1$, conjugué de u, est un st-mot. □

Cette condition nécessaire nous permet d'énoncer la remarque suivante :

Remarque 2.16 *Soit* $L = b_s(a_s a_t)^* a_s b_t (b_s b_t)^* a_t$. *Comme pour tout entier* n, *nous avons :*

$$b_s(a_s a_t)^n a_s b_t (b_s b_t)^n a_t \xrightarrow[\Omega]{*} b_s a_s (b_t b_s a_t a_s)^n b_t a_t,$$

le langage $b_s a_s (b_t b_s a_t a_s)^* b_t a_t$ *est inclus dans* $\mathrm{f_R}(L)$. *Le mot* $b_t b_s a_t a_s$ *est un facteur itérant de* $\mathrm{f_R}(L)$ *mais il n'est pas le conjugué d'un st-mot donc* $\mathrm{f_R}(L)$ *n'est pas un langage bien formé. La clôture par R d'un langage bien formé n'est donc pas toujours bien formée.*

Nous allons à présent montrer la réciproque du Lemme 2.15 en utilisant une série de lemmes techniques. Définissons tout d'abord l'alphabet « nonst » d'un langage, c'est-à-dire l'ensemble des actions pour lesquelles le langage ne correspond pas à une succession correcte de débuts et de fins.

Définition 2.17 *Soit* $L \subseteq (\Sigma_s \cup \Sigma_t)^*$. *L'alphabet* $\mathrm{nonst}(L)$ *est défini par :*

$$\mathrm{nonst}(L) = \{x \in \Sigma \mid \exists u \in L \text{ tel que } \Pi_{\{x_s, x_t\}}(u) \notin (x_s x_t)^*\}.$$

Lemme 2.18 *Soient L et L' des langages de $(\Sigma_s \cup \Sigma_t)^*$. Si L est un st-langage tel que chacun de ses mots contient au moins une occurrence de x_s pour tout x appartenant à* nonst(L') *alors, pour tous langages L_1 et L_2 tels que $L = L_1.L_2$, nous avons :*

$$(L_1.(L')^*.L_2 \text{ est un facteur de st-langage}) \Leftrightarrow (L_1.(L')^*.L_2 \text{ est un st-langage}).$$

Preuve. L'implication de droite à gauche est triviale, montrons l'implication de gauche à droite. Pour montrer que le langage $L_1.(L')^*.L_2$ est un st-langage, il suffit de montrer que pour toute action x de Σ, sa projection sur $\{x_s, x_t\}$ est un st-langage. Soit x une action de Σ. Par hypothèse, le langage $L_1.(L')^*.L_2$ est un facteur de st-langage donc tout mot u de L' est un facteur itérant de st-langage, nous avons :

$$\forall u \in L', \Pi_{\{x_s,x_t\}}(u) \in (x_s x_t)^* \text{ ou } \Pi_{\{x_s,x_t\}}(u) \in x_t(x_s x_t)^* x_s.$$

Considérons deux mots distincts de L', u et u'. Le mot uu' est un facteur itérant de st-langage donc, $\Pi_{\{x_s,x_t\}}(uu')$ appartient à $(x_s x_t)^*$ ou $\Pi_{\{x_s,x_t\}}(uu')$ appartient à $x_t(x_s x_t)^* x_s$, c'est-à-dire que si $\Pi_{\{x_s,x_t\}}(u)$ appartient à $x_t(x_s x_t)^* x_s$ alors forcément $\Pi_{\{x_s,x_t\}}(u')$ appartient à $x_t(x_s x_t)^* x_s + \varepsilon$ et réciproquement. Donc, nous avons :

$$\Pi_{\{x_s,x_t\}}(L') \subseteq (x_s x_t)^* \text{ ou } \Pi_{\{x_s,x_t\}}(L') \subseteq x_t(x_s x_t)^* x_s + \varepsilon.$$

Le langage $L_1.L_2$ est un st-langage, en conséquence $\Pi_{\{x_s,x_t\}}(L_1)$ est inclus dans $(x_s x_t)^*$ et $\Pi_{\{x_s,x_t\}}(L_2)$ dans $(x_s x_t)^*$ ou $\Pi_{\{x_s,x_t\}}(L_1)$ dans $(x_s x_t)^* x_s$ et $\Pi_{\{x_s,x_t\}}(L_2)$ dans $x_t(x_s x_t)^*$. Nous allons considérer deux cas en fonction de la projection de L' sur l'alphabet $\{x_s, x_t\}$.

1. Si $\Pi_{\{x_s,x_t\}}(L')$ est inclus dans $(x_s x_t)^*$, comme $L_1.(L')^*.L_2$ est un facteur de st-langage, $\Pi_{\{x_s,x_t\}}(L_1)$ et $\Pi_{\{x_s,x_t\}}(L_2)$ sont inclus dans $(x_s x_t)^*$ et donc $\Pi_{\{x_s,x_t\}}(L_1.(L')^*.L_2)$ l'est également.

2. Si $\Pi_{\{x_s,x_t\}}(L')$ est inclus dans $x_t(x_s x_t)^* x_s + \varepsilon$ et si $\Pi_{\{x_s,x_t\}}(L')$ contient au moins un mot non vide, comme $L_1.(L')^*.L_2$ est un facteur de st-langage, $\Pi_{\{x_s,x_t\}}(L_1)$ est inclus dans $(x_s x_t)^* x_s$ et $\Pi_{\{x_s,x_t\}}(L_2)$ dans $x_t(x_s x_t)^*$ ou ils ne contiennent tous deux que le mot vide. Ce dernier cas ne peut survenir car, par hypothèse, pour tout x appartenant à nonst(L'), tout mot de L contient au moins une occurrence de x_s, en conséquence, $\Pi_{\{x_s,x_t\}}(L_1.(L')^*.L_2)$ est inclus dans $(x_s x_t)^*$.

Comme pour toute action x de Σ, la projection de $L_1.(L')^*.L_2$ sur l'alphabet $\{x_s, x_t\}$ est incluse dans $(x_s x_t)^*$, $L_1.(L')^*.L_2$ est un st-langage. □

Nous allons à présent nous intéresser à l'étoile d'un langage bien formé. Dans la décomposition d'un langage d'après son expression rationnelle en vue de montrer qu'il est bien formé, le cas de l'étoile est le plus problématique aussi allons nous, avant de commencer cette décomposition, en étudier quelques propriétés.

Lemme 2.19 *Soit M un langage bien formé tel que M^* est un facteur de st-langage et tel que tout mot de M^* est le conjugué d'un st-mot. Alors, nous avons $M = \bigcup_{i=1}^{n} d_i f_i$ où pour tout i :*

- *$f_i d_i$ est un st-langage ;*

- d_i *et* f_i *sont des langages bien formés ;*
- *pour tout* x *appartenant à* $\mathrm{nonst}(d_i f_i)$ *et pour tout mot* w *de* f_i, $|w|_{x_s} \neq 0$;
- $\mathrm{nonst}(d_i f_i)$ *est inclus dans* $\mathrm{nonst}(d_{i+1} f_{i+1})$.

Preuve. Comme M est un langage bien formé, il possède une expression rationnelle bien formée dont nous pouvons déduire une factorisation de M :

$$M = \bigcup_{i \in I} T_{i,0}^* u_{i,1} T_{i,1}^* \dots u_{i,p_i} T_{i,p_i}^*$$

telle que pour tous indices i et k, $T_{i,k}^*$ est un st-langage bien formé et $u_{i,k}$ est un mot.

Notons $u_i = u_{i,1} \dots u_{i,p_i}$ pour tout indice i de I. Comme u_i est un mot de M, il est le conjugué d'un st-mot, il existe un indice k_i tel que $u_i = u_{i,1} \dots u'_{i,k_i} u''_{i,k_i} \dots u_{i,p_i}$ et tel que $u''_{i,k_i} \dots u_{i,p_i} u_{i,1} \dots u'_{i,k_i}$ est un st-mot. Posons

$$d_i = T_{i,0}^* u_{i,1} \dots T_{i,k_i-1}^* u'_{i,k_i} \text{ et } f_i = u''_{i,k_i} T_{i,k_i}^* \dots u_{i,p_i} T_{i,p_i}^*.$$

Comme le langage $d_i f_i d_i f_i$ est inclus dans M^*, c'est un facteur de st-langage, donc le langage $f_i d_i$ est également un facteur de st-langage et même un st-langage car $u''_{i,k_i} \dots u_{i,p_i} u_{i,1} \dots u'_{i,k_i}$ est un st-mot et car chaque $T_{i,j}$ est un st-langage. Par construction, pour tout indice i, d_i et f_i possèdent une expression rationnelle bien formée donc d_i et f_i sont des langages bien formés. En conséquence, le langage M s'écrit :

$$M = \bigcup_{i \in I} d_i f_i,$$

sachant que, pour tout i, $f_i d_i$ est un st-langage et d_i et f_i des langages bien formés.

Il nous reste à montrer les deux derniers points du lemme. Remarquons en premier lieu que comme M^* est un facteur de st-langage, si x appartient à $\mathrm{nonst}(d_i f_i)$ alors $\Pi_{\{x_s,x_t\}}(d_i f_i)$ est inclus dans $x_t(x_s x_t)^* x_s$. Soient w appartenant à f_i et w' appartenant à d_i, si $|w|_{x_s} = 0$ alors $\Pi_{\{x_s,x_t\}}(ww')$ n'appartient pas à $(x_s x_t)^*$ ce qui est en contradiction avec le fait que $f_i d_i$ est un st-langage.

Pour le dernier point, supposons qu'il existe w appartenant à $d_i f_i$ et w' appartenant à $d_j f_j$, avec i différent de j, tels qu'il existe x appartenant à $\mathrm{nonst}(w) \setminus \mathrm{nonst}(w')$ et y appartenant à $\mathrm{nonst}(w') \setminus \mathrm{nonst}(w)$. Dans ce cas, le mot $\Pi_{\{x_s,x_t,y_s,y_t\}}(ww')$ appartient à $((x_t x_s)^+ \sqcup (y_s y_t)^*).((y_t y_s)^+ \sqcup (x_s x_t)^*)$ et donc ww' n'est pas le conjugué d'un st-mot d'où contradiction. □

Lemme 2.20 *Soit* L *un langage reconnaissable facteur de st-langage. Si tout facteur itérant de* L *est le conjugué d'un st-mot alors* L *est un langage bien formé.*

Preuve. Montrons ce lemme par induction sur la construction rationnelle du langage. Clairement, tout langage fini est bien formé.

Si $L = M_1 + M_2$ ou $L = M_1.M_2$ alors tout facteur itérant de M_1 (ou M_2) est le conjugué d'un st-mot puisqu'il est également facteur itérant de L. De plus, M_1 (ou M_2) est clairement un facteur de st-langage donc, par hypothèse de récurrence, c'est un langage bien formé. Le langage L est donc bien formé.

Si $L = (M)^*$, tout facteur itérant de M est également facteur itérant de L, il est donc conjugué d'un st-mot. Par hypothèse de récurrence, M est bien formé. Si M est un st-langage, comme M possède une expression rationnelle bien formée E, L possède une expression rationnelle bien formée $(E)^*$ donc L est bien formé. Si M n'est pas un st-langage, comme tout mot de M^* est un facteur itérant de L, tout mot de M^* est le conjugué d'un st-mot, nous pouvons donc appliquer le Lemme 2.19 pour obtenir la décomposition de M, $M = \bigcup_{i \in I} d_i f_i$ avec pour $i \in I = \{0, \ldots, n\}$:

- $f_i d_i$ st-langage ;
- d_i et f_i langages bien formés ;
- pour tout x appartenant à $\text{nonst}(d_i f_i)$, pour tout w appartenant à f_i, $|w|_{x_s} \neq 0$;
- $\text{nonst}(d_i f_i)$ est inclus dans $\text{nonst}(d_{i+1} f_{i+1})$ pour tout $0 \leq i < n$.

À partir de cette décomposition de M, nous allons pouvoir construire une expression bien formée pour $(M)^*$ par récurrence sur la taille de I. Nous savons que l'ensemble $\{d_i f_i \mid i \in I\}$ est ordonné de la façon suivante : $\text{nonst}(d_0 f_0) \subseteq \text{nonst}(d_1 f_1) \subseteq \ldots \subseteq \text{nonst}(d_n f_n)$. Si $I = \{0\}$, $M^* = d_0(f_0 d_0)^* f_0 + \varepsilon$. Comme $f_0 d_0$ est un st-langage et comme f_0 et d_0 possèdent une expression rationnelle bien formée, M^* possède une expression rationnelle bien formée.

Si $I' = I \cup \{n\}$, le langage M^* peut être écrit sous la forme :

$$M^* = (\bigcup_{i \in I} d_i f_i)^*.(d_n(f_n(\bigcup_{i \in I} d_i f_i)^* d_n)^* f_n + \varepsilon).(\bigcup_{i \in I} d_i f_i)^*.$$

Par hypothèse de récurrence $(\bigcup_{i \in I} d_i f_i)^*$ possède une expression bien formée, il suffit donc que $f_n(\bigcup_{i \in I} d_i f_i)^* d_n$ soit un st-langage pour obtenir une expression bien formée pour M^*.

Le langage $f_n d_n$ est un st-langage. Pour tout i appartenant à I, nous avons l'inclusion $\text{nonst}(d_i f_i) \subseteq \text{nonst}(d_n f_n)$ donc, pour tout x appartenant à $\text{nonst}(\bigcup_{i \in I} d_i f_i)$, tout mot de f_n contient au moins une occurence de x_s. D'après le Lemme 2.18, $f_n(\bigcup_{i \in I} d_i f_i)^* d_n$ est un st-langage. □

Tout st-langage étant un facteur de st-langage, la réciproque du Lemme 2.15 se déduit immédiatement du lemme précédent, ce qui nous permet d'énoncer la proposition :

Proposition 2.21 *Un st-langage régulier est bien formé si et seulement si chacun de ses facteurs itérants est le conjugué d'un st-mot.*

La preuve du Lemme 2.20 étant constructive, nous allons voir, sur un exemple, comment transformer une expression rationnelle en expression rationnelle bien formée. Considérons le langage L défini par l'expression rationnelle

$$a_s b_s c_s \underbrace{(a_t b_t c_t a_s b_s c_s + a_t a_s + a_t b_t a_s b_s)}_{M}{}^* a_t b_t c_t.$$

Nous pouvons ordonner les mots de M en fonction de leur alphabet « nonst », nous avons :

$$a_t b_t c_t a_s b_s c_s > a_t b_t a_s b_s > a_t a_s.$$

Pour construire une expression rationnelle bien formée pour M^*, nous allons appliquer la construction proposée dans la récurrence de la preuve. Nous allons donc considérer successivement les facteurs du plus petit au plus grand. Pour le langage $(a_t a_s)^*$ nous obtenons l'expression $E_1 = \varepsilon + a_t(a_s a_t)^* a_s$, pour le langage $(a_t b_t a_s b_s + a_t a_s)^*$ nous obtenons l'expression $E_2 = E_1(a_t b_t(a_s b_s.E_1.a_t b_t)^* a_s b_s + \varepsilon)E_1$ et enfin pour le langage M^* nous obtenons l'expression $E_2(a_t b_t c_t(a_s b_s c_s.E_2.a_t b_t c_t)^* a_s b_s c_s + \varepsilon)E_2$. C'est-à-dire que le langage M^* peut être défini par l'expression rationnelle bien formée:

$$(\varepsilon + a_t(a_s a_t)^* a_s)(a_t b_t(a_s b_s(\varepsilon + a_t(a_s a_t)^* a_s)a_t b_t)^* a_s b_s + \varepsilon)(\varepsilon + a_t(a_s a_t)^* a_s)$$
$$(a_t b_t c_t(a_s b_s c_s$$
$$(\varepsilon + a_t(a_s a_t)^* a_s)(a_t b_t(a_s b_s(\varepsilon + a_t(a_s a_t)^* a_s)a_t b_t)^* a_s b_s + \varepsilon)(\varepsilon + a_t(a_s a_t)^* a_s)$$
$$a_t b_t c_t)^* a_s b_s c_s + \varepsilon)$$
$$(\varepsilon + a_t(a_s a_t)^* a_s)(a_t b_t(a_s b_s(\varepsilon + a_t(a_s a_t)^* a_s)a_t b_t)^* a_s b_s + \varepsilon)(\varepsilon + a_t(a_s a_t)^* a_s).$$

La caractérisation des langages bien formés que nous avons donnée va nous permettre de montrer que la clôture par le système R d'un langage bien formé n'est pas toujours un langage de synchronisation pour apporter une réponse négative à la première partie de la conjecture II.

2.2.3 Clôture des langages bien formés

Dans cette sous-section, nous allons considérer le langage bien formé

$$\mathbb{L} = b_s(a_s a_t)^* c_s b_t(a_s a_t)^* c_t$$

et montrer que sa clôture par R n'est pas un langage de synchronisation. Pour cela, nous allons utiliser une propriété relativement technique des langages de synchronisation.

Lemme 2.22 *Soient Σ un alphabet d'actions contenant a et c et L un langage de synchronisation défini sur l'alphabet $\Sigma_s \cup \Sigma_t$. Il existe un entier N tel que :*

$$((a_s a_t)^p c_s(a_s a_t)^q c_t \in L \text{ avec } p, q > N) \Rightarrow ((a_s a_t)^{p-1} c_s(a_s a_t)^{q+1} c_t \in L).$$

Preuve. Si l'un des deux ensembles $\{p \mid (a_s a_t)^p c_s(a_s a_t)^q c_t \in L\}$ et $\{q \mid (a_s a_t)^p c_s(a_s a_t)^q c_t \in L\}$ est borné par la valeur *sup*, il suffit de prendre $N = sup$. Dans le cas contraire, aucun de ces deux ensembles n'est borné, considérons alors le langage $K = L \cap (a_s a_t)^* c_s(a_s a_t)^* c_t$. Le langage K, intersection de deux langages réguliers, est régulier, il existe donc un intervalle fini F tel que :

$$K = \bigcup_{i \in F} (a_s a_t)^{\alpha_i}((a_s a_t)^{\beta_i})^* c_s(a_s a_t)^{\gamma_i}((a_s a_t)^{\delta_i})^* c_t.$$

Par définition, tout mot $u = (a_s a_t)^p c_s(a_s a_t)^q c_t$ appartenant à L appartient à K donc,

$$(a_s a_t)^p c_s(a_s a_t)^q c_t \in \bigcup_{i \in F} (a_s a_t)^{\alpha_i}((a_s a_t)^{\beta_i})^* c_s(a_s a_t)^{\gamma_i}((a_s a_t)^{\delta_i})^* c_t,$$

il existe donc un indice i de F et deux entiers n et m tels que $p = \alpha_i + n\beta_i$ et $q = \gamma_i + m\delta_i$.

Considérons $N = \max_{i \in F}\{\alpha_i + \delta_i\beta_i\}$. Si $(a_s a_t)^p c_s(a_s a_t)^q c_t$ appartient à L avec p et q strictement supérieurs à N alors $p = \alpha_i + n\beta_i$ et $q = \gamma_i + m\delta_i$ avec $n > \delta_i$ donc il existe

$p' = \alpha_i + (n - \delta_i)\beta_i$ et $q' = \gamma_i + (m + \beta_i)\delta_i$ tels que $w = (a_s a_t)^{p'} c_s (a_s a_t)^{q'} c_t$ appartient à K et donc à L. Or, nous avons :

$$w \xrightarrow[R]{*} w' = (a_s a_t)^{\alpha_i + (n-\delta_i)\beta_i + (\beta_i\delta_i - 1)} c_s (a_s a_t)^{\gamma_i + (m+\beta_i)\delta_i - (\beta_i\delta_i - 1)} c_t$$

avec

$$\begin{cases} \alpha_i + (n - \delta_i)\beta_i + (\beta_i\delta_i - 1) = p - 1, \\ \gamma_i + (m + \beta_i)\delta_i - (\beta_i\delta_i - 1) = q + 1. \end{cases}$$

Comme L est un langage de synchronisation, L est fermé par le système de réécriture R donc w' appartient à L. □

Lemme 2.23 *Soit L un langage de synchronisation. Il existe un entier N_L tel que pour tous p et q strictement supérieurs à N_L,*

$$(b_s(a_s a_t)^p c_s b_t (a_s a_t)^q c_t \in L) \Rightarrow (b_s(a_s a_t)^{p-1} c_s a_s a_t b_t (a_s a_t)^q c_t \in L).$$

Preuve. Montrons le Lemme par induction sur la construction du langage de synchronisation. Si L est fini alors il suffit de prendre $N_L = \max\{p, q \mid b_s(a_s a_t)^p c_s b_t (a_s a_t)^q c_t \in L\}$. Si $L = L_1 + L_2$ ou $L = L_1 \cap L_2$, il suffit de prendre $N_L = \max\{N_{L_1}, N_{L_2}\}$. Si $L = (L_1)^*$ ou $L = L_1.L_2$, comme le mot $b_s(a_s a_t)^p c_s b_t (a_s a_t)^q c_t$ est st-primitif, il suffit de prendre respectivement $N_L = N_{L_1}$ ou $N_L = \max\{N_{L_1}, N_{L_2}\}$. Reste à étudier le cas $L = L_1 \sqcup L_2$. Nous considérons la construction de L comme langage de synchronisation, les alphabets de L_1 et L_2 sont donc disjoints, trois cas peuvent survenir :

- si a_s, b_s et c_s appartiennent à l'alphabet de L_1, il suffit de prendre $N_L = N_{L_1}$;
- si a_s et b_s appartiennent à l'alphabet de L_1 et c_s à celui de L_2 ou b_s et c_s à l'alphabet de L_1 et a_s à celui de L_2, il suffit de prendre $N_L = 0$;
- si a_s et c_s appartiennent à l'alphabet de L_1 et b_s à l'alphabet de L_2 alors, si le mot $b_s(a_s a_t)^p c_s b_t (a_s a_t)^q c_t$ appartient à $L_1 \sqcup L_2$, $(a_s a_t)^p c_s (a_s a_t)^q c_t$ appartient à L_1 et $b_s b_t$ appartient à L_2. D'après le Lemme 2.22, il existe un entier N tel que si le mot $(a_s a_t)^p c_s (a_s a_t)^q c_t$ appartient à L_1 avec p et q strictement supérieur à N alors $(a_s a_t)^{p-1} c_s (a_s a_t)^{q+1} c_t$ appartient à L_1, donc $(a_s a_t)^{p-1} c_s (a_s a_t)^{q+1} c_t \sqcup b_s b_t$ est inclus dans $L_1 \sqcup L_2$ c'est-à-dire $b_s(a_s a_t)^{p-1} c_s a_s a_t b_t (a_s a_t)^q c_t$ appartient à L. Il suffit donc de prendre $N_L = N$.

□

Cette propriété va nous permettre de conclure cette sous-partie en répondant à la première partie de la Conjecture II.

Contre-exemple 2.24 *Le langage $\mathbb{L} = b_s(a_s a_t)^* c_s b_t (a_s a_t)^* c_t$ est un st-langage régulier bien formé dont la clôture par le système de réécriture R n'est pas un langage de synchronisation.*

Preuve. Pour tous entiers p et q, $b_s(a_s a_t)^p c_s b_t (a_s a_t)^q c_t$ appartient à $\mathbb{L}$. Aucune règle du système de réécriture R ne permet d'obtenir le facteur $c_s a_s a_t b_t$ dans un mot de la clôture de $\mathbb{L}$ donc, aucun mot de type $b_s(a_s a_t)^{p-1} c_s a_s a_t b_t (a_s a_t)^q c_t$ n'appartient à $\mathrm{f}_R(\mathbb{L})$. D'après le Lemme 2.23, $\mathrm{f}_R(\mathbb{L})$ n'est pas un langage de synchronisation. □

Ce contre-exemple clôt l'étude de la première partie de la Conjecture II, nous allons à présent nous intéresser à la deuxième partie de cette conjecture.

2.2.4 Langages de synchronisation comme clôtures de langages bien formés

Dans la deuxième partie de la Conjecture II, il est supposé que tout langage de synchronisation est la clôture par le système R d'un langage bien formé. Cette conjecture peut sembler naturelle car, dans la construction des langages de synchronisation, on calcule uniquement l'étoile de st-langages. Cependant, nous allons montrer que cette conjecture n'est pas vraie en utilisant la Caractérisation 2.21 des langages bien formés.

Contre-exemple 2.25 *Le langage de synchronisation décrit par l'expression* $[(a \parallel b) \rightarrow (a \parallel b)^*] \parallel [d \rightarrow (c \rightarrow d)^*]$ *n'est la clôture par le système de réécriture* R *d'aucun langage bien formé.*

Preuve. Notons L le langage considéré, M_1 le langage $a_s b_s a_t b_t (a_s b_s a_t b_t)^*$, M_2 le langage $d_s d_t (c_s c_t d_s d_t)^*$ et L_1 et L_2 les clôtures par R de M_1 et M_2. Clairement, L_1 et L_2 sont des langages de synchronisation correspondant aux expressions $(a \parallel b) \rightarrow (a \parallel b)^*$ et $d \rightarrow (c \rightarrow d)^*$ et donc $L = L_1 \sqcup L_2$.

Soit $N = a_s b_s d_s (d_t c_s a_t b_t a_s b_s c_t d_s)^* a_t b_t d_t$. Clairement, ce langage est inclus dans le shuffle de M_1 et M_2. Cependant, nous allons montrer qu'il n'existe pas de langage bien formé inclus dans L qui engendre N.

La projection sur $\{c_s, c_t, d_s, d_t\}$ de tout mot de L appartient à L_2 donc, dans tout mot de L, pour tout facteur $d_t u d_s$, u contient au moins le sous-mot $c_s c_t$ (et réciproquement pour tout facteur $c_t u c_s$, u contient au moins le sous-mot $d_s d_t$). De ce fait, les lettres correspondant à des actions c et d ne pourront jamais intervenir dans une règle de Ω. De plus, la projection sur $\{a_s, a_t, b_s, b_t\}$ de tout mot de L appartient à L_1 donc aucun mot de L ne contient de facteur $a_t a_s b_t b_s$ ou $b_t b_s a_t a_s$ puisque chaque action a peut être exécutée en parallèle avec une seule action b. Donc, pour tout u appartenant à L, $\mathrm{f_R}(u) = \mathrm{f}_\theta(u)$. Notons

$$Max_\theta(L) = \{u \in L \mid \forall v \in L, v \xrightarrow[\theta]{*} u \Rightarrow u \xrightarrow[\theta]{*} v\}.$$

Montrons que N est une partie de $Max_\theta(L)$. Pour tout u appartenant à N, on cherche un mot v de L tel que $v \longrightarrow_\theta^* u$ en utilisant une règle non symétrique. Nous allons montrer qu'un tel mot n'existe pas. Considérons :

$$\mathrm{f}_{\theta_1,\theta_2}(N) = (a_s \sqcup b_s \sqcup d_s).(d_t c_s.(a_t b_t + b_t a_t).(a_s b_s + b_s a_s).c_t d_s)^*.(a_t \sqcup b_t \sqcup d_t)$$

et montrons que si l'on applique une règle $x_t y_s \longrightarrow y_s x_t$ à un des mots de ce langage, le mot obtenu n'appartient pas à L. Si on applique une telle règle impliquant les actions c et d, le mot obtenu n'appartient pas à L puisque la projection de tout mot de L sur l'alphabet $\{c_s, c_t, d_s, d_t\}$ appartient à M_2 qui est un ensemble de séquences. Si l'on applique une règle $b_t a_s \longrightarrow a_s b_t$ à un mot de $\mathrm{f}_{\theta_1,\theta_2}(N)$, le mot obtenu n'appartient pas à L puisque, dans L, chaque action a ne peut être exécutée en parallèle qu'avec un seul b, le cas de la règle $a_t b_s \longrightarrow b_s a_t$ est similaire. Donc, si un mot u de L se réécrit en un mot v de N, la dérivation n'utilise que des règles symétriques.

Comme N est inclus dans $Max_\theta(L)$, d'après la forme de $\mathrm{f}_{\theta_1,\theta_2}(N)$, tout langage engendrant L possède un facteur itérant appartenant au langage $(d_t c_s.(a_t b_t + b_t a_t).(a_s b_s + b_s a_s).c_t d_s)^+$,

ce facteur itérant n'est pas le conjugué d'un st-mot. En conclusion, il n'existe pas de langage bien formé dont L soit la clôture. □

En conclusion de cette partie sur les langages bien formés, nous avons une réponse négative à la Conjecture II. Cette réponse est surprenante en raison de la structure des langages bien formés qui semblait bien adaptée à la structure des langages de synchronisation. Nous allons à présent revenir à la Conjecture I pour conclure l'étude du système R.

2.3 Langages de synchronisation et système R

Le langage $\mathrm{f_R}(b_s(a_sa_t)^*c_sb_t(a_sa_t)^*c_t)$ que nous avons utilisé comme contre-exemple à la Conjecture IIa fournit également un contre-exemple à la Conjecture I.

Contre-exemple 2.26 *Le langage* $\mathrm{f_R}(b_s(a_sa_t)^*c_sb_t(a_sa_t)^*c_t)$ *est un st-langage régulier et R-clos qui n'est pas un langage de synchronisation.*

Preuve. Nous avons montré précédemment que $\mathrm{f_R}(\mathbb{L})$ n'est pas un langage de synchronisation, cependant c'est un st-langage clos par R, il nous reste à montrer que c'est un langage régulier.

Comme tout mot de $\mathbb{L}$ contient exactement une occurrence de chacune des lettres b_s, b_t, c_s et c_t, il est impossible d'appliquer une règle de Ω à l'un de ses mots. Nous avons donc $\mathrm{f_R}(\mathbb{L}) = \mathrm{f}_\theta(\mathbb{L})$, c'est-à-dire que l'on calcule la clôture de $\mathbb{L}$ par la semi-commutation θ. Or, tout facteur itérant de $\mathbb{L}$ a pour alphabet $\{a_s, a_t\}$, donc tout facteur itérant de $\mathbb{L}$ est fortement connexe. D'après le Théorème 1.30, $\mathrm{f}_\theta(\mathbb{L})$ est régulier. Le langage $\mathrm{f}_\theta(\mathbb{L})$ est donc un st-langage régulier clos par R mais pas un langage de synchronisation. □

Le système R ne permet donc pas de caractériser les langages de synchronisation. Au regard de l'étude que nous avons menée, deux problèmes essentiels semblent se dégager apportant chacun une piste dans la recherche d'une caractérisation des langages de synchronisation. En premier lieu, dans le cas du contre-exemple ci-dessus, il semble impossible de « distinguer » les deux groupes d'actions a. Au départ, nous disposons du langage $\mathbb{L} = b_s(a_sa_t)^*c_sb_t(a_sa_t)^*c_t$ et, lorsque nous calculons la clôture de ce langage, nous n'obtenons jamais le facteur $c_sa_sa_tb_t$ c'est-à-dire que nous sommes en présence d'une série d'exécutions durant lesquelles il est possible de mettre en parallèle l'action c et l'action b, un nombre arbitrairement grand de a en parallèle avec l'action b, un nombre arbitrairement grand de a en parallèle avec l'action c mais en aucun cas une même action a en parallèle avec le b et le c. Une telle restriction ne peut être exprimée à l'aide des expressions de synchronisation, en effet nous ne disposons pas de la négation — on peut d'ailleurs remarquer que les langages de synchronisation ne sont pas fermés par complémentaire. Par contre, si on distingue les deux groupes d'actions a par un renommage, nous obtenons un langage de synchronisation :

$$M = \mathrm{f_R}(b_s(a_{1s}a_{1t})^*c_sb_t(a_{2s}a_{2t})^*c_t) = L(((b \to a_2^*) \parallel (a_1^* \to c))\&((a_1^* \to a_2^*) \parallel b \parallel c)).$$

Le langage $\mathrm{f}_\theta(\mathbb{L})$ est l'image par le morphisme strictement alphabétique

$$\begin{array}{rccccccc} \varphi : & a_{1s} & \longmapsto & a_s & a_{1t} & \longmapsto & a_t \\ & a_{2s} & \longmapsto & a_s & a_{2t} & \longmapsto & a_t \\ & b_s & \longmapsto & b_s & b_t & \longmapsto & b_t \\ & c_s & \longmapsto & c_s & c_t & \longmapsto & c_t \end{array}$$

du langage de synchronisation M, il semble donc utile, dans certains cas, de considérer les images par morphismes des langages de synchronisation.

Le second type de problèmes que nous avons rencontré apparaît dans l'exemple suivant.

Exemple 2.27 Considérons deux mots

$$u = a_s b_s c_s a_t b_t b_s b_t b_s a_s c_t c_s a_t b_t c_t \text{ et } v = a_s b_s c_s c_t c_s a_t b_t b_s b_t b_s a_s a_t b_t c_t.$$

Toutes les projections de u sur des sous-alphabets de deux actions permettent d'obtenir, par R, les projections de v :

$$\begin{aligned} a_s b_s a_t b_t b_s b_t b_s a_s a_t b_t &\longrightarrow_R^0 a_s b_s a_t b_t b_s b_t b_s a_s a_t b_t, \\ a_s c_s a_t a_s c_t c_s a_t c_t &\longrightarrow_R^1 a_s c_s c_t c_s a_t a_s a_t c_t, \\ b_s c_s b_t b_s b_t b_s c_t c_s b_t c_t &\longrightarrow_R^2 b_s c_s c_t c_s b_t b_s b_t b_s b_t c_t, \end{aligned}$$

pourtant v n'appartient pas à $\mathrm{f_R}(u)$.

Dans cet exemple, le plus petit langage de synchronisation au sens de l'inclusion auquel le mot u appartient est défini par l'expression

$$\begin{aligned} & [((a \parallel b) \rightarrow b \rightarrow (a \parallel b)) \parallel (c \longrightarrow c)] \\ \& \quad & [((a \longrightarrow c) \parallel (a \parallel c)) \parallel (b \longrightarrow b \longrightarrow b)] \\ \& \quad & [(b \longrightarrow b \longrightarrow c) \parallel (b \parallel c)) \parallel (a \longrightarrow a)]. \end{aligned}$$

Or, le mot v appartient également à ce langage, c'est-à-dire que le système R ne permet pas non plus de caractériser les langages de synchronisation finis. Il semble qu'il manque au système R des propriétés de projection qui permettraient, dans notre exemple, de retrouver l'ensemble du langage de synchronisation à partir d'un seul mot. Nous allons, dans le chapitre suivant, proposer une extension du système R pour résoudre ce problème.

Chapitre 3

Langages de synchronisation et systèmes de réécriture

Nous avons vu dans le chapitre précédent que le système de réécriture R défini par Guo, Salomma et Yu ne permet pas de caractériser les langages de synchronisation dans le cas général. Il est alors naturel de se demander si les langages de synchronisation peuvent effectivement être caractérisés par un système de réécriture. Nous allons, dans ce chapitre, proposer une extension de R pour tenter de répondre à cette question.

3.1 Extension du système R

Le nouveau système de réécriture que nous allons définir doit avoir un comportement assez proche de celui de R, en particulier pour les langages définis sur des alphabets de deux actions qui sont complètement caractérisés par R. De plus, il doit résoudre des problèmes tels que celui de l'Exemple 2.27. Dans cet exemple, le problème semble venir du fait que R ne contient pas de règles du type :

$$a_t(b_tb_s)^n a_s c_t c_s \longrightarrow c_t c_s a_t (b_t b_s)^n a_s,$$

et, plus généralement du fait que nous n'avons pas pour R d'équivalent au Lemme de Projection pour les semi-commutations (Lemme 1.19). En suivant cette idée, si l'on veut obtenir un Lemme de Projection pour notre nouveau système, il faut également considérer des règles du type :

$$a_t c_s a_s b_t b_s \longrightarrow b_t b_s a_t c_s a_s$$

et bien d'autres encore. En fait, il est naturel de penser qu'il suffit d'ajouter à R des règles du type $uv \longrightarrow vu$ pour tous mots u et v tels que les projections de u sur des alphabets correspondant à une action commutent par R avec les projections de v sur des alphabets correspondant à une action. En appelant ts-mots les mots dont les projections sur les alphabets d'une action sont de la forme $(x_t x_s)^*$, on obtiendrait un système dont les règles permettent de commuter un mot facteur droit de ts-mot et un mot facteur gauche de ts-mot si ceux-ci n'ont aucune action en commun. Formellement, on obtiendrait le système suivant :

$$\begin{aligned} R'' = \theta \cup \{uv \longrightarrow vu \mid\ & \forall x \in \Sigma, \Pi_{\{x_s,x_t\}}(u) \in (x_t x_s)^* + x_s + \varepsilon, \\ & \forall x \in \Sigma, \Pi_{\{x_s,x_t\}}(v) \in (x_t x_s)^* x_s + x_t + \varepsilon, \\ & u \text{ et } v \text{ n'ont aucune action en commun}, \\ & \mathrm{alph}(u) \cap \Sigma_t \neq \emptyset, \mathrm{alph}(v) \cap \Sigma_s \neq \emptyset\}. \end{aligned}$$

Cette idée relativement intuitive ne suffit cependant pas à obtenir un Lemme de Projection, dans certains cas, il n'est pas possible d'utiliser des commutations de mots, les lettres en cause dans les différentes projections formant des facteurs complètement imbriqués dans le mot de départ. Considérons par exemple les mots $w = a_t z_t z_s y_t a_s b_t y_s b_s$ et $w' = b_t y_t b_s a_t y_s z_t z_s a_s$. Il est facile de vérifier que les projections sur les sous-alphabets de deux actions de w permettent d'obtenir celles de w' par R cependant, il n'est pas possible de passer de w à w' en appliquant des règles de R'' ou tout autre ensemble de règles du type $uv \longrightarrow vu$ qui reste compatible avec R.

Nous allons donc simplement considérer qu'un mot u donne un mot v par notre nouveau système de réécriture si et seulement si les projections de u sur les sous-alphabets de deux actions donnent les projections de v par R. Le système ainsi défini est naturellement infini.

Définition 3.1 *Soit Σ un alphabet d'actions. Le système de réécriture R'_Σ sur $\Sigma_s \cup \Sigma_t$ est défini par :*

$$\forall u, v \in (\Sigma_s \cup \Sigma_t)^*, (u \underset{R'_\Sigma}{\longrightarrow} v) \Leftrightarrow (\forall a, b \in \Sigma, \Pi_{\{a_s,a_t,b_s,b_t\}}(u) \xrightarrow[R_\Sigma]{*} \Pi_{\{a_s,a_t,b_s,b_t\}}(v)).$$

Par définition, ce système de réécriture possède la propriété suivante :

Propriété 3.2 *Les systèmes R'_Σ et R_Σ sont équivalents pour tout alphabet Σ de deux actions.*

Lorsqu'aucune confusion n'est possible, nous utiliserons simplement la notation R' à la place de R'_Σ. Clairement, le système R' préserve la st-propriété :

Propriété 3.3 *La clôture par R' d'un st-langage est un st-langage.*

La définition du système R' que nous venons de donner est relativement simple. Si R' est infini, il permet cependant toujours d'obtenir un mot commutativement équivalent au mot de départ. Pourtant, dans de nombreux cas, il est plus aisé de manipuler un système fini. Nous allons donc montrer que nous pouvons nous ramener à un système de réécriture fini mais, dans ce cas, les mots obtenus par dérivation ne seront plus commutativement équivalents au mot de départ.

Nous allons utiliser à de nombreuses reprises la projection sur un alphabet de deux actions aussi allons nous utiliser la notation suivante :

Notation 3.4 *Soit $\Sigma = \{a, b\}$ un alphabet d'actions. La projection $\Pi_{\Sigma_s \cup \Sigma_t}$ est notée Π_{ab}.*

3.2 Utilisation d'un système fini

3.2.1 Définition du système fini

Nous allons utiliser un système fini intermédiaire en nous basant sur la remarque suivante. Il est possible de simuler l'application d'une règle de Ω sur un facteur de deux actions en utilisant uniquement des règles agissant sur des facteurs de longueur deux et en marquant les lettres :

$$a_t a_s b_t b_s \longrightarrow a_t \bar{b}_t \bar{a}_s b_s \longrightarrow b_t \bar{a}_t \bar{a}_s b_s \longrightarrow b_t \bar{a}_t \bar{b}_s a_s \longrightarrow b_t b_s a_t a_s.$$

Ici, nous autorisons une commutation $a_t b_s \longrightarrow b_s a_t$ si ces lettres sont marquées, c'est-à-dire dans le cas où nous voulons « terminer » l'équivalent d'une règle de Ω. Cette idée peut être

étendue aux langages définis sur des alphabets d'actions quelconques, il suffit simplement de toujours se ramener aux projections sur les sous-alphabets de deux actions. Pour cela nous allons associer, à chaque lettre, autant de marques que de lettres dans l'alphabet d'actions afin de mémoriser les règles de type Ω en cours sur les divers sous-alphabets de deux actions. Les marques servent de mémoire au cours de la dérivation mais, comme la mémoire nécessaire est bornée, nous pouvons travailler sur un alphabet fini. La mémoire associée à chaque lettre de $\Sigma_s \cup \Sigma_t$ est représentée par un sous-ensemble de l'alphabet des actions. Nous allons désormais utiliser les alphabets suivants :

Σ l'alphabet d'actions ;
$\Sigma_e = \{(x, E) \mid E \subset \Sigma \text{ et } x \in (\Sigma_s \cup \Sigma_t) \setminus (E_s \cup E_t)\}$;
$\Sigma_0 = \{(x, \emptyset) \in \Sigma_e\}$.

L'alphabet Σ_e est l'alphabet étendu : par exemple, $(a_s, \{b, c\})$ est une lettre de Σ_e, a_s correspond au début d'action a et l'ensemble permet de garder trace des dernières commutations effectuées entre cette lettre et les lettres des sous-alphabets correspondant à une action — le b permet de garder trace de la dernière commutation effectuée avec une lettre de l'alphabet $\{b_s, b_t\}$.

Pour passer d'un alphabet à l'autre, nous utilisons des morphismes. Le morphisme h permet d'effacer la mémoire en projetant sur la première composante :

$$\begin{array}{rccc} h : & {\Sigma_e}^* & \to & (\Sigma_s \cup \Sigma_t)^* \\ & (x, E) & \mapsto & x. \end{array}$$

Le morphisme g ajoute un ensemble vide à une lettre « normale » permettant ainsi d'initialiser la mémoire :

$$\begin{array}{rccc} g : & (\Sigma_s \cup \Sigma_t)^* & \to & \Sigma_0^* \\ & x & \mapsto & (x, \emptyset). \end{array}$$

Nous considérons toujours les mots comme des enchaînements d'actions, c'est pourquoi nous étendons la notion de st-mot aux mots de ${\Sigma_e}^*$:

Définition 3.5 *Un mot u de ${\Sigma_e}^*$ est un st-mot si et seulement si $h(u)$ est un st-mot.*

Deux mots de ${\Sigma_e}^*$ peuvent représenter un même enchaînement d'actions mais différer par la valeur de la deuxième composante des lettres qui les composent, nous appellerons donc « squelette » d'un mot u de ${\Sigma_e}^*$ le mot $h(u)$.

De même, les lettres a, a_s, (a_t, E), appartiennent à différents alphabets mais représentent toutes la même action. Nous définissons donc l'action d'une lettre et les actions d'un mot :

Définition 3.6 *Pour tout $a \in \Sigma$, $\alpha(a_s) = \alpha(a_t) = a$ et pour tout $x \in \Sigma_e$, $\alpha(x) = \alpha(h(x))$.*
Pour un mot u, $\alpha(u) = \{\alpha(x) \mid x \in \text{alph}(u)\}$.

Il nous sera également nécessaire de considérer, comme dans le cas de R', les projections sur des sous-alphabets de deux actions :

Définition 3.7 *Pour tout couple d'actions $a, b \in \Sigma$, le morphisme Π_{ab} de ${\Sigma_e}^*$ dans ${\Sigma_e}^*$ est défini par :*

$$\forall (x, E) \in \Sigma_e, \Pi_{ab}((x, E)) = \begin{cases} (x, E \cap \{a, b\}) \text{ si } \alpha(x) \in \{a, b\}, \\ \varepsilon \text{ dans les autres cas.} \end{cases}$$

Enfin, nous pouvons définir le système de réécriture fini que nous allons associer à R', ce système est nommé $R_{e\Sigma}$, pour R_Σ sur l'alphabet étendu. Lorsqu'aucune confusion n'est possible, le système de réécriture est simplement nommé R_e.

Définition 3.8 *Le système de réécriture symétrique $R_{e\Sigma}$ défini sur Σ_e est l'union des ensembles de règles suivants:*

$$\begin{array}{ll}
s_1 = \{(x_s,E)(y_s,F) \longrightarrow (y_s,F)(x_s,E) & \mid x,y \in \Sigma_e, x \notin F, y \notin E\}\ ; \\
s_2 = \{(x_s,E)(y_s,F) \longrightarrow (y_s,F\cup\{x\})(x_s,E\setminus\{y\}) & \mid x,y \in \Sigma_e, x \notin F, y \in E\}\ ; \\
s_3 = \{(x_t,E)(y_t,F) \longrightarrow (y_t,F)(x_t,E) & \mid x,y \in \Sigma_e, x \notin F, y \notin E\}\ ; \\
s_4 = \{(x_t,E)(y_t,F) \longrightarrow (y_t,F\setminus\{x\})(x_t,E\cup\{y\}) & \mid x,y \in \Sigma_e, x \in F, y \notin E\}\ ; \\
s_5 = \{(x_s,E)(y_t,F) \longrightarrow (y_t,F\cup\{x\})(x_s,E\cup\{y\}) & \mid x,y \in \Sigma_e, x \notin F, y \notin E\}\ ; \\
s_6 = \{(x_t,E)(y_s,F) \longrightarrow (y_s,F\setminus\{x\})(x_t,E\setminus\{y\}) & \mid x,y \in \Sigma_e, x \in F, y \in E\}.
\end{array}$$

Les règles de s_1 et s_2 correspondent aux règles de R du type $x_s y_s \longrightarrow y_s x_s$, les règles de s_3 et s_4 aux règles du type $x_t y_t \longrightarrow y_t x_t$, les règles de s_5 aux règles du type $x_s y_t \longrightarrow y_t x_s$ et les règles s_6 permettent de réaliser la commutation d'une fin et d'un début d'action lorsque la situation l'autorise. Ainsi, pour reprendre l'exemple du début de cette section, on peut effectuer, pas à pas, l'équivalent d'une règle de Ω:

Exemple 3.9 Une règle $a_t a_s b_t b_s \longrightarrow b_t b_s a_t a_s$ peut être simulée par:

$$\begin{array}{rl}
(a_t,\emptyset)(a_s,\emptyset)(b_t,\emptyset)(b_s,\emptyset) & \longrightarrow (a_t,\emptyset)(b_t,\{a\})(a_s,\{b\})(b_s,\emptyset) \\
& \longrightarrow (b_t,\emptyset)(a_t,\{b\})(a_s,\{b\})(b_s,\emptyset) \\
& \longrightarrow (b_t,\emptyset)(a_t,\{b\})(b_s,\{a\})(a_s,\emptyset) \\
& \longrightarrow (b_t,\emptyset)(b_s,\emptyset)(a_t,\emptyset)(a_s,\emptyset).
\end{array}$$

Remarquons que les ensembles permettent uniquement de simuler les règles de Ω, ils témoignent des échanges en cours puisque l'on dérive pas à pas. Lorsque l'échange correspondant à la règle de Ω est terminé, les ensembles sont à nouveau vides. L'utilisation des ensembles n'empêche jamais d'effectuer une commutation correspondant à une ancienne règle de θ.

Exemple 3.10 Considérons un exemple de dérivation:

$$\begin{array}{rl}
 & (a_s,\emptyset)(b_s,\emptyset)(a_t,\emptyset)(c_s,\emptyset)\underbrace{(a_s,\emptyset)(b_t,\emptyset)}(b_s,\emptyset)(a_t,\emptyset)(b_t,\emptyset)(c_t,\emptyset) \\
\longrightarrow_{R_e} & (a_s,\emptyset)(b_s,\emptyset)(a_t,\emptyset)\underbrace{(c_s,\emptyset)(b_t,\{a\})}(a_s,\{b\})(b_s,\emptyset)(a_t,\emptyset)(b_t,\emptyset)(c_t,\emptyset) \\
\longrightarrow_{R_e} & (a_s,\emptyset)(b_s,\emptyset)\underbrace{(a_t,\emptyset)(b_t,\{a,c\})}(c_s,\{b\})(a_s,\{b\})(b_s,\emptyset)(a_t,\emptyset)(b_t,\emptyset)(c_t,\emptyset) \\
\longrightarrow_{R_e} & (a_s,\emptyset)(b_s,\emptyset)(b_t,\{c\})(a_t,\{b\})(c_s,\{b\})\underbrace{(a_s,\{b\})(b_s,\emptyset)}(a_t,\emptyset)(b_t,\emptyset)(c_t,\emptyset) \\
\longrightarrow_{R_e} & (a_s,\emptyset)(b_s,\emptyset)(b_t,\{c\})(a_t,\{b\})\underbrace{(c_s,\{b\})(b_s,\{a\})}(a_s,\emptyset)(a_t,\emptyset)(b_t,\emptyset)(c_t,\emptyset) \\
\longrightarrow_{R_e} & (a_s,\emptyset)(b_s,\emptyset)(b_t,\{c\})\underbrace{(a_t,\{b\})(b_s,\{a,c\})}(c_s,\emptyset)(a_s,\emptyset)(a_t,\emptyset)(b_t,\emptyset)(c_t,\emptyset) \\
\longrightarrow_{R_e} & (a_s,\emptyset)(b_s,\emptyset)(b_t,\{c\})(b_s,\{c\})(a_t,\emptyset)(c_s,\emptyset)(a_s,\emptyset)(a_t,\emptyset)(b_t,\emptyset)(c_t,\emptyset)
\end{array}$$

Il ne reste aucune marque concernant a et b puisque l'on a effectué l'équivalent d'une règle Ω complète; par contre, il reste des marques concernant b et c puisque l'on a effectué ce qui pourrait être le début d'une règle de Ω.

Clairement, le système R_e préserve les st-langages:

Propriété 3.11 *La clôture par R_e d'un st-langage est un st-langage.*

Nous ne sommes pas en présence de commutations partielles puisque les règles du système de réécriture R_e, si elles permettent de permuter deux lettres représentant des débuts ou fins d'actions, modifient les deuxièmes composantes de ces lettres et donc, l'alphabet des mots. Cependant, par abus de langage, nous parlerons de commutations de lettres ne considérant que l'image des lettres par h. De plus, afin d'augmenter la lisibilité, nous introduisons la notation suivante :

Notation 3.12 *Lors de la commutation de deux lettres x et y, les seuls éléments des ensembles pouvant être modifiés sont $\alpha(x)$ et $\alpha(y)$, il n'est donc pas nécessaire de considérer les autres éléments. Aussi, lorsque l'on exhibe deux lettres $x = (a, E)$ et $y = (b, F)$ d'un mot auquel on applique une règle, elles sont notées :*

$$\begin{aligned} x &= a \text{ si } b \notin E, \\ x &= \bar{a} \text{ si } b \in E, \\ y &= b \text{ si } a \notin F, \\ y &= \bar{b} \text{ si } a \in F. \end{aligned}$$

Par exemple si, dans la dérivation

$$\begin{gathered} (a_s, \emptyset)(b_s, \emptyset)(c_s, \emptyset)(a_t, \emptyset)(b_t, \emptyset)(c_t, \emptyset) \\ R_e \downarrow \\ (a_s, \emptyset)(b_s, \emptyset)(a_t, \{c\})(c_s, \{a\})(b_t, \emptyset)(c_t, \emptyset), \end{gathered}$$

on désire mettre en évidence la règle utilisée, il n'est pas nécessaire de conserver les informations concernant l'action b,

$$(c_s, \emptyset)(a_t, \emptyset) \longrightarrow (a_t, \{c\})(c_s, \{a\})$$

peut être simplifiée en :

$$c_s a_t \longrightarrow \bar{a}_t \bar{c}_s.$$

Dans la suite de cette partie, nous étudions les propriétés de R_e, ces propriétés nous fournissent une boîte à outils efficace pour manipuler le système R_e mais surtout, elles nous permettront, par la suite, d'établir le lien entre R' et R_e. Nous verrons que pour calculer la clôture d'un mot (ou d'un langage) par R', il suffit d'utiliser le morphisme g pour « ajouter » un ensemble vide à chacune de ses lettres, de calculer la clôture par R_e du mot ainsi obtenu puis d'utiliser le morphisme h pour revenir à l'alphabet initial.

3.2.2 Propriétés du système fini

Nous allons à présent étudier les propriétés du système de réécriture R_e. Celles-ci sont, en grande partie semblables aux propriétés classiques des systèmes de commutation cependant, elles ne sont pas immédiates en raison de la modification de l'alphabet des mots en cours de dérivation. Dans de nombreux cas, nous énoncerons des propriétés des mots de Σ_0^*, celles-ci nous seront en effet fort utiles par la suite pour établir le lien avec R'.

En premier lieu, le nouveau système que nous avons défini est bien adapté à l'étude des st-langages.

Lemme 3.13 *Soient $u \in \Sigma_0^*$ un st-mot et $v \in \mathrm{f}_{\mathrm{R_e}}(u)$. Pour tout découpage $v = v_1(x, E)v_2$ avec $E \neq \emptyset$, nous avons :*

$$(x = a_s) \Rightarrow (\forall y \in E, |h(v_1)|_{y_t} + |h(v_1)|_{a_t} \geq 1), \tag{3.1}$$

$$(x = a_t) \Rightarrow (\forall y \in E, |h(v_2)|_{y_s} + |h(v_2)|_{a_s} \geq 1). \tag{3.2}$$

Preuve. Montrons en premier lieu l'implication (3.1). Clairement, u vérifie (3.1), il suffit donc de montrer que cette propriété est conservée lors d'un pas de dérivation. Considérons un mot v tel que pour tout découpage $v = v_1(a_s, E)v_2$ avec $E \neq \emptyset$, nous avons :

$$\forall y \in E, |h(v_1)|_{y_t} + |h(v_1)|_{a_t} \geq 1$$

et une dérivation $v = w_1xyw_2 \longrightarrow_{R_e} w_1y'x'w_2 = w$. Il nous faut montrer que quelle que soit la règle utilisée pour passer de v à w, le mot w vérifie (3.1). Nous allons détailler l'application d'une règle s_2, c'est-à-dire, le cas $x = (a_s, E)$ et $y = (b_s, F)$ avec $b \in E$. Nous avons donc $w = w_1(b_s, F \cup \{a\})(a_s, E \setminus \{b\})w_2$. Par hypothèse, nous avons :

- pour tout découpage $v = w_{11}(c_s, G)w_{12}xyw_2$ avec $G \neq \emptyset$, le mot w_{11} vérifie :

$$\forall y \in G, |h(w_{11})|_{y_t} + |h(w_{11})|_{c_t} \geq 1 \ ;$$

- pour tout découpage $v = w_1xyw_{21}(c_s, G)w_{22}$ avec $G \neq \emptyset$, le mot w_1xyw_{21} vérifie :

$$\forall y \in G, |h(w_1xyw_{21})|_{y_t} + |h(w_1xyw_{21})|_{c_t} \geq 1.$$

Ces deux points ne sont aucunement modifiés par l'application de la règle $xy \longrightarrow y'x'$. Il nous faut donc vérifier que

$$\forall y \in F \cup \{a\}, |h(w_1)|_{y_t} + |h(w_1)|_{b_t} \geq 1.$$

Par hypothèse, pour tout $y \in F$ nous avons $|h(w_1)|_{y_t} + |h(w_1)|_{b_t} \geq 1$ et pour tout $y \in E$, nous avons $|h(w_1)|_{y_t} + |h(w_1)|_{a_t} \geq 1$, donc, comme $b \in E$, nous avons $|h(w_1)|_{a_t} + |h(w_1)|_{b_t} \geq 1$. De même, il nous faut vérifier que

$$\forall y \in E \setminus \{b\}, |h(w_1)|_{y_t} + |h(w_1)|_{a_t} \geq 1,$$

ce qui se déduit immédiatement de l'hypothèse pour tout $y \in E$, $|h(w_1)|_{y_t} + |h(w_1)|_{a_t} \geq 1$. De même, quelle que soit la règle de R_e utilisée, le mot obtenu vérifie (3.1).

D'une manière totalement similaire, on montre l'implication (3.2). □

Nous déduisons immédiatement de ce lemme le corollaire :

Corollaire 3.14 *Soit $u \in \Sigma_0^*$ un st-mot. Nous avons :*

$$\mathrm{f}_{\mathrm{R_e}}(u) \in g(\Sigma_s){\Sigma_e}^* g(\Sigma_t).$$

Les deuxièmes composantes des lettres d'un mot de ${\Sigma_e}^*$ ne sont pas dues au hasard, celles-ci témoignent des dérivations en cours et certaines séquences de lettres de l'alphabet Σ_e ne peuvent intervenir au cours d'une dérivation d'un mot de Σ_0^*. Nous allons donc définir le langage rationnel L_Σ dont les mots sont des facteurs de st-mots de Σ_e dont les projections sur les sous-alphabets de deux actions ne contiennent pas certains facteurs « interdits ». Ces facteurs « interdits » sont des facteurs qui ne peuvent apparaître lors d'une dérivation par R_e d'un mot de Σ_0^*. Le langage considéré n'est en aucun cas le langage obtenu par clôture de l'ensemble des st-mots de Σ_0^*, c'est un langage plus grand (au sens de l'inclusion) mais suffisamment précis pour restreindre le champ d'étude dans de nombreux cas.

Définition 3.15 *On appelle L_Σ le langage des facteurs de st-mots défini sur l'alphabet Σ_e tel que pour tous $a, b \in \Sigma$, $a \neq b$,*

$$\Pi_{ab}(L_\Sigma) \cap \{a_s, a_t, b_s, b_t, \bar{a}_s, \bar{a}_t, \bar{b}_s, \bar{b}_t\}^* M_{ab} \{a_s, a_t, b_s, b_t, \bar{a}_s, \bar{a}_t, \bar{b}_s, \bar{b}_t\}^* = \emptyset.$$

Le langage M_{ab} contient les facteurs que l'on « interdit » :

$$\begin{aligned} M_{ab} = \; & a_s\bar{b}_s + a_s\bar{b}_t + \bar{a}_s\bar{b}_s + \bar{a}_s b_t + \bar{a}_s\bar{b}_t + a_t\bar{a}_s + a_t\bar{b}_s + \bar{a}_t b_t + \bar{a}_t\bar{b}_t + \bar{a}_t b_s + \bar{a}_t a_s + \\ & b_s\bar{a}_s + b_s\bar{a}_t + \bar{b}_s\bar{a}_s + \bar{b}_s a_t + \bar{b}_s\bar{a}_t + b_t\bar{b}_s + b_t\bar{a}_s + \bar{b}_t a_t + \bar{b}_t\bar{a}_t + \bar{b}_t a_s + \bar{b}_t b_s + \\ & (a_s + \bar{a}_s)(b_s + \bar{b}_s + b_t)^*\bar{b}_t + \bar{a}_s(a_s + a_t + \bar{a}_t)^*(b_t + \bar{b}_t) + \\ & (b_s + \bar{b}_s)(a_s + \bar{a}_s + a_t)^*\bar{a}_t + \bar{b}_s(b_s + b_t + \bar{b}_t)^*(a_t + \bar{a}_t). \end{aligned}$$

Montrons que ce langage est clos par R_e :

Lemme 3.16 *Pour tout mot u de L_Σ, $\mathrm{f}_{\mathrm{R}_e}(u)$ est inclus dans L_Σ.*

Preuve. Clairement, il suffit de vérifier la propriété pour une dérivation de longueur 1. Soient u appartenant à L_Σ et v obtenu en appliquant une règle de R_e à u. Il existe u_1, u_2 de ${\Sigma_e}^*$ et une règle $zt \longrightarrow t'z'$ de R_e tels que $u = u_1 z t u_2$ et $v = u_1 t' z' u_2$. Il suffit alors de vérifier que $\Pi_{\alpha(z)\alpha(t)}(v)$ ne contient aucun facteur interdit sachant que $\Pi_{\alpha(z)\alpha(t)}(u)$ ne contient aucun facteur interdit. Notons a et b les lettres de Σ telles que $\alpha(z) = a$ et $\alpha(t) = b$ et étudions les différents cas :

- si $\Pi_{ab}(u) = u'_1 a_s b_s u'_2$ et $\Pi_{ab}(v) = u'_1 b_s a_s u'_2$, comme u appartient à L_Σ, u'_1 est le mot vide ou se termine par a_t ou b_t et u'_2 commence par a_t ou b_t donc, v est un mot de L_Σ ;
- si $\Pi_{ab}(u) = u'_1 \bar{a}_s b_s u'_2$ et $\Pi_{ab}(v) = u'_1 \bar{b}_s a_s u'_2$, comme u appartient à L_Σ, u'_1 est le mot vide ou se termine par $\bar{a}_t$ ou $\bar{b}_t$ et u'_2 commence par a_t ou b_t donc, v est un mot de L_Σ ;
- si $\Pi_{ab}(u) = u'_1 a_t b_t u'_2$ et $\Pi_{ab}(v) = u'_1 b_t a_t u'_2$, comme u appartient à L_Σ, u'_1 se termine par a_s ou b_s et u'_2 est le mot vide ou commence par a_s ou b_s donc, v est un mot de L_Σ ;
- si $\Pi_{ab}(u) = u'_1 a_t \bar{b}_t u'_2$ et $\Pi_{ab}(v) = u'_1 b_t \bar{a}_t u'_2$, comme u appartient à L_Σ, u'_1 se termine par a_s ou b_s et u'_2 est le mot vide ou commence par $\bar{a}_s$ ou $\bar{b}_s$ donc, v est un mot de L_Σ ;
- si $\Pi_{ab}(u) = u'_1 a_s b_t u'_2$ et $\Pi_{ab}(v) = u'_1 \bar{b}_t \bar{a}_s u'_2$, comme u appartient à L_Σ, u'_1 se termine par a_t, b_s ou $\bar{b}_s$ et u'_2 commence par a_t ou b_s donc, v est un mot de L_Σ ;
- si $\Pi_{ab}(u) = u'_1 \bar{a}_t \bar{b}_s u'_2$ et $\Pi_{ab}(v) = u'_1 b_s a_t u'_2$, comme u appartient à L_Σ, u'_1 se termine par a_s, $\bar{a}_s$ ou b_t et u'_2 commence par a_s, b_t ou $\bar{b}_t$ donc, v est un mot de L_Σ.

Dans chacun des cas, le mot v appartient à L_Σ. □

Le lemme suivant montre qu'il est possible de simuler une dérivation par R d'un facteur de st-mot u de $(\Sigma_s \cup \Sigma_t)^*$ à partir d'un mot quelconque de L_Σ ayant u pour squelette.

Lemme 3.17 *Soit u appartenant à $(\Sigma_s \cup \Sigma_t)^*$ un facteur de st-mot. Pour tout v de $\mathrm{f}_{\mathrm{R}}(u)$ et tout u' de $h^{-1}(u) \cap L_\Sigma$, il existe un mot v' de $h^{-1}(v) \cap L_\Sigma$ tel que $v' \in \mathrm{f}_{\mathrm{R}_e}(u')$.*

Preuve. Nous allons montrer le lemme par récurrence sur la longueur de la dérivation par R. La propriété est trivialement vérifiée pour une dérivation de longueur nulle. Si

$$u \xrightarrow[R]{n} w \xrightarrow[R]{} v,$$

par hypothèse de récurrence, pour tout u' appartenant à $h^{-1}(u) \cap L_\Sigma$, il existe un w' appartenant à $h^{-1}(w) cap L_\Sigma$ tel que $w' \in \mathrm{f}_{\mathrm{R}_e}(u')$. Il suffit donc de montrer qu'il existe un mot v'

de $f_{R_e}(w')$ tel que $h(v') = v$. Pour passer de w à v, on peut appliquer une règle de θ ou de Ω, nous allons considérer séparement chacun de ces cas.

- Si $w \longrightarrow_\theta v$, il existe un découpage $w = w_1xyw_2$ avec $v = w_1yxw_2$ et $xy \longrightarrow yx$ une règle de θ. Comme $h(w') = w$, il existe un découpage de $w' = w'_1ztw'_2$ avec $h(w'_1) = w_1$, $h(w'_2) = w_2$, $h(z) = x$ et $h(t) = y$. Le mot w' appartient à L_Σ et xy est le membre gauche d'une règle de θ donc, il existe une règle $zt \longrightarrow t'z'$ de R_e qui nous donne :
$$w'_1ztw'_2 \xrightarrow[R_e]{} w'_1t'z'w'_2 = v' \text{ avec } h(v') = v.$$

- Si $w \longrightarrow_\Omega v$, il existe des découpages de w et v :
$$w = w_1a_{1t}\dots a_{kt}a_{1s}\dots a_{ks}b_{1t}\dots b_{lt}b_{1s}\dots b_{ls}w_2,$$
$$v = w_1b_{1t}\dots b_{lt}b_{1s}\dots b_{ls}a_{1t}\dots a_{kt}a_{1s}\dots a_{ks}w_2,$$
avec $a_1, \dots, a_k, b_1, \dots, b_l$ des lettres de Σ deux à deux distinctes. Comme $h(w') = w$, $w' = w'_1x_1\dots x_ky_1\dots y_kz_1\dots z_lt_1\dots t_lw'_2$ avec, pour tout indice i compris entre 1 et k, $h(x_i) = a_{it}$, $h(y_i) = a_{is}$ et pour indice j compris entre 1, et l, $h(z_j) = b_{jt}$, $h(t_j) = b_{js}$, $h(w'_1) = w_1$ et $h(w'_2) = w_2$. Comme w' appartient à L_Σ, pour tout i et tout j, nous avons :
$$\Pi_{a_ib_j}(w') = \Pi_{a_ib_j}(w'_1)a_{it}a_{is}b_{jt}b_{js}\Pi_{a_ib_j}(w'_2).$$
En conséquence, nous avons :
$$\forall (x, E) \in \text{alph}(x_1\dots x_ky_1\dots y_k),\ E \cap \alpha(z_1\dots z_lt_1\dots t_l) = \emptyset,$$
$$\forall (x, E) \in \text{alph}(z_1\dots z_lt_1\dots t_l), E \cap \alpha(x_1\dots x_ky_1\dots y_k) = \emptyset.$$
Donc, en utilisant l fois une règle de s_5 entre chaque y_i et tous les $z_1, \dots, z_l$, nous obtenons :
$$w' = w'_1x_1\dots x_ky_1\dots y_kz_1\dots z_lt_1\dots t_lw'_2$$
$$s_5 \downarrow (k \times l)$$
$$w'_1x_1\dots x_kz'_1\dots z'_ly'_1\dots y'_kt_1\dots t_lw'_2$$
avec $y'_i = (y, E \cup \alpha(z_1\dots z_l))$ si $y_i = (y, E)$ et $z'_i = (z, F \cup \alpha(y_1\dots y_k))$ si $z_i = (z, F)$. Il est donc possible de poursuivre la dérivation :

$$w'_1x_1\dots x_kz'_1\dots z'_ly'_1\dots y'_kt_1\dots t_lw'_2$$

$$s_4 \downarrow (k \times l)$$

$$w'_1z_1\dots z_lx'_1\dots x'_ky'_1\dots y'_kt_1\dots t_lw'_2 \quad \begin{cases} x'_i = (x, E \cup (\alpha(z_1\dots z_l)) \\ \text{si } x_i = (x, E) \end{cases}$$

$$s_2 \downarrow (k \times l)$$

$$w'_1z_1\dots z_lx'_1\dots x'_kt'_1\dots t'_ly_1\dots y_kw'_2 \quad \begin{cases} t'_i = (t, E \cup \alpha(y_1\dots y_k)) \\ \text{si } t_i = (t, E) \end{cases}$$

$$s_6 \downarrow (k \times l)$$

$$w'_1z_1\dots z_lt_1\dots t_lx_1\dots x_ky_1\dots y_kw'_2 = v'$$

Nous obtenons $h(v') = w_1b_{1t}\dots b_{lt}b_{1s}\dots b_{ls}a_{1t}\dots a_{kt}a_{1s}\dots a_{ks}w_2 = v$. De plus, d'après le Lemme 3.16, v' appartient à L_Σ.

□

Nous allons considérer, dans la suite, de nombreuses dérivations qui permettent d'échanger deux mots sans bouleverser l'ordre des lettres qui les composent. Dans une telle dérivation, les squelettes des deux mots ne sont pas modifiés. Pour plus de clarté nous allons adopter la convention suivante :

Notation 3.18 *Soit u appartenant à $\Sigma_e{}^*$. Nous utiliserons les notations $u', u'', u''', \ldots$ pour désigner des mots ayant le même squelette que u (c'est-à-dire tels que $h(u) = h(u') = h(u'') = h(u''') = \ldots$).*

Contrairement au système de réécriture R, le système R_e comporte uniquement des règles permettant de commuter deux lettres. Nous allons voir que cette particularité permet d'organiser les dérivations.

Lemme 3.19 *Soit xu un mot de Σ^* tel que x est une lettre de Σ_e et tel que $\alpha(x)$ n'est pas une action de $\alpha(u)$. Nous avons :*

$$(u \xrightarrow[R_e]{*} v \text{ et } xv \xrightarrow[R_e]{|v|} v'x') \Leftrightarrow (xu \xrightarrow[R_e]{|u|} u'x' \text{ et } u' \xrightarrow[R_e]{*} v').$$

Preuve. Clairement, il suffit de vérifier la propriété pour une dérivation de u à v de longueur 1. Posons $u = u_1yzu_2$, $v = u_1z'y'u_2$ et $v' = u_1'z''y''u_2'$. De la longueur de la dérivation de xv à $v'x'$, nous pouvons déduire les quatre dérivations suivantes :

$$\begin{array}{lll} xu_1 & \longrightarrow_{R_e}^{|u_1|} & u_1'x'', \\ x''z' & \longrightarrow_{R_e} & z''x''', \\ x'''y' & \longrightarrow_{R_e} & y''x^{(4)}, \\ x^{(4)}u_2 & \longrightarrow_{R_e}^{|u_2|} & u_2'x'. \end{array}$$

Nous avons :

$$x''yz \xrightarrow[R_e]{} x''z'y' \xrightarrow[R_e]{} z''x'''y' \xrightarrow[R_e]{} z''y''x^{(4)}.$$

Rappelons que pour deux lettres a et b de Σ_e, l'existence d'une règle $ab \to b'a'$ dans R_e ainsi que $\Pi_{\alpha(a)\alpha(b)}(b'a')$ dépendent uniquement de $\Pi_{\alpha(a)\alpha(b)}(ab)$, de plus pour toute action c de Σ différente de $\alpha(a)$ et $\alpha(b)$, nous avons $\Pi_{\alpha(a)c}(a') = \Pi_{\alpha(a)c}(a)$ et $\Pi_{\alpha(b)c}(b') = \Pi_{\alpha(b)c}(b)$. Des égalités $\Pi_{\alpha(x)\alpha(y)}(x'') = \Pi_{\alpha(x)\alpha(y)}(x''')$ et $\Pi_{\alpha(x)\alpha(y)}(y) = \Pi_{\alpha(x)\alpha(y)}(y')$, nous pouvons donc déduire l'implication :

$$(z''x'''y' \xrightarrow[R_e]{} z''y''x^{(4)}) \Rightarrow (x''yz \xrightarrow[R_e]{} y'''x^{(5)}z)$$

avec $\Pi_{\alpha(x)\alpha(y)}(x^{(5)}) = \Pi_{\alpha(x)\alpha(y)}(x^{(4)})$ et $\Pi_{\alpha(x)\alpha(y)}(y''') = \Pi_{\alpha(x)\alpha(y)}(y'')$. Nous pouvons poursuivre de la même manière. Des égalités $\Pi_{\alpha(x)\alpha(z)}(x'') = \Pi_{\alpha(x)\alpha(z)}(x^{(5)})$ et $\Pi_{\alpha(x)\alpha(z)}(z') = \Pi_{\alpha(x)\alpha(z)}(z)$, nous pouvons donc déduire l'implication :

$$(x''z'y' \xrightarrow[R_e]{} z''x'''y') \Rightarrow (y'''x^{(5)}z \xrightarrow[R_e]{} y'''z'''x^{(4)})$$

avec $\Pi_{\alpha(x)\alpha(z)}(x^{(4)}) = \Pi_{\alpha(x)\alpha(z)}(x''')$ et $\Pi_{\alpha(x)\alpha(z)}(z''') = \Pi_{\alpha(x)\alpha(z)}(z'')$. Enfin, des égalités $\Pi_{\alpha(y)\alpha(z)}(y''') = \Pi_{\alpha(y)\alpha(z)}(y)$ et $\Pi_{\alpha(y)\alpha(z)}(z''') = \Pi_{\alpha(x)\alpha(z)}(z)$, nous pouvons donc déduire l'implication :

$$(x''yz \xrightarrow[R_e]{} x''z'y') \Rightarrow (y'''z'''x^{(4)} \xrightarrow[R_e]{} z''y''x^{(4)}).$$

Nous avons donc :

$$x''yz \xrightarrow[R_e]{2} y'''z'''x^{(4)} \xrightarrow[R_e]{} z''y''x^{(4)}.$$

Nous obtenons donc la dérivation suivante :

$$xu = xu_1yzu_2 \xrightarrow[R_e]{|u|} u_1'y'''z'''u_2'x' \xrightarrow[R_e]{} u_1'z''y''u_2'x' = v'x',$$

avec :

$$u' = u_1'y'''z'''u_2' \xrightarrow[R_e]{} u_1'z''y''u_2' = v'.$$

La preuve de l'implication de droite à gauche est totalement identique. □

Lemme 3.20 (Lemme de base) *Soient u appartenant à Σ_e^* et x une lettre de Σ_e tels que*

$$xu \xrightarrow[R_e]{*} u_1x'u_2 \text{ avec } h(x) = h(x') \text{ et } \alpha(x) \notin \alpha(u_1).$$

Alors, nous avons

$$u \xrightarrow[R_e]{*} u_1'u_2 \text{ et } xu_1' \xrightarrow[R_e]{|u_1|} u_1x' \text{ avec } h(u_1) = h(u_1').$$

Preuve. Nous allons montrer le lemme par récurrence sur la longueur de la dérivation. La propriété est trivialement vérifiée pour une dérivation de longueur nulle. Considérons la dérivation

$$xu \xrightarrow[R_e]{n} w \xrightarrow[R_e]{} u_1x'u_2.$$

Le mot w se décompose en $w = w_1x''w_2$, $\alpha(x)$ n'étant pas une action de $\alpha(w_1)$. Par hypothèse de récurrence, nous avons :

$$u \xrightarrow[R_e]{*} w_1'w_2 \text{ et } xw_1' \xrightarrow[R_e]{|w_1|} w_1x'' \text{ avec } h(w_1) = h(w_1').$$

Il nous faut mettre en évidence la dernière règle appliquée lors de la dérivation et étudier les cas possibles.

- Si $xu \longrightarrow_{R_e}^n w_1x''yu_2 \longrightarrow_{R_e} w_1y'x'u_2$ (ici $w_2 = yu_2$ et $u_1 = w_1y'$), nous avons :

$$xu \xrightarrow[R_e]{*} xw_1'w_2 = xw_1'yu_2 \xrightarrow[R_e]{|w_1|} w_1x''yu_2 \xrightarrow[R_e]{} w_1y'x'u_2.$$

Donc $u_1x'u_2$ appartient à $\mathrm{f}_{R_e}(xu)$ avec

$$\begin{cases} u_1 = w_1y', \\ u \longrightarrow_{R_e}^* w_1'yu_2, \\ xw_1'y \longrightarrow_{R_e}^k w_1x''y \longrightarrow_{R_e} w_1y'x', \\ |w_1y'| = k+1, \\ h(w_1'y) = h(w_1')h(y) = h(w_1)h(y') = h(w_1y'). \end{cases}$$

- Si $xu \longrightarrow_{R_e}^{n} v_1yzv_2x'u_2 \longrightarrow_{R_e} v_1z'y'v_2x'u_2$ (ici $w_1 = v_1yzv_2$, $u_1 = v_1z'y'v_2$ et $u_2 = w_2$), nous avons :

$$u \xrightarrow[R_e]{*} v_1'y''z''v_2'u_2 \text{ et } xv_1'y''z''v_2' \xrightarrow[R_e]{|v_1yzv_2|} v_1yzv_2x' \text{ avec } \alpha(x) \notin \alpha(v_1'y''z''v_2').$$

Nous avons donc :

$$xv_1'y''z''v_2' \xrightarrow[R_e]{|v_1yzv_2|} v_1yzv_2x' \text{ et } v_1yzv_2 \xrightarrow[R_e]{*} v_1z'y'v_2.$$

Du Lemme 3.19, nous déduisons :

$$v_1'y''z''v_2' \xrightarrow[R_e]{*} v_1'z'''y'''v_2' \text{ et } xv_1'z'''y'''v_2' \xrightarrow[R_e]{|v_1'z'''y'''v_2'|} v_1z'y'v_2x'.$$

Donc $u_1x'u_2$ appartient à $\mathrm{f}_{R_e}(xu)$ avec

$$\begin{cases} u_1 = v_1z'y'v_2, \\ u \longrightarrow_{R_e}^{*} v_1'z'''y'''v_2'u_2, \\ xv_1'z'''y'''v_2'u_2 \longrightarrow_{R_e}^{k} v_1z'y'v_2x'u_2, \\ k = |v_1z'y'v_2|, \\ h(v_1z'y'v_2) = h(v_1'z'''y'''v_2'). \end{cases}$$

- Si $xu \longrightarrow_{R_e}^{n} u_1yx''w_2 \longrightarrow_{R_e} u_1x'y'w_2$ (ici $w_1 = u_1y$ et $u_2 = y'w_2$), nous avons :

$$u \xrightarrow[R_e]{*} u_1'y''w_2 \text{ et } xu_1'y'' \xrightarrow[R_e]{|u_1y|} u_1yx''.$$

Donc $xu \longrightarrow_{R_e}^{*} xu_1'y''w_2 \longrightarrow_{R_e}^{|u_1y-1|} u_1x'''y''w_2$, comme toutes les règles de R_e sont symétriques et que $x'''y'' \longrightarrow yx'' \longrightarrow x'y'$, nous avons $x''' = x'$ et $y'' = y'$. Donc $u_1x'u_2$ appartient à $\mathrm{f}_{R_e}(xu)$ avec

$$\begin{cases} u_2 = y'w_2, \\ u \longrightarrow_{R_e}^{*} u_1'y''w_2, \\ xu_1'y''w_2 \longrightarrow_{R_e}^{k} u_1x'y'w_2, \\ k = |u_1|, \\ h(u_1) = h(u_1'). \end{cases}$$

- Si $xu \longrightarrow_{R_e}^{n} u_1x'v_1yzv_2 \longrightarrow_{R_e} u_1x'v_1z'y'v_2$ (ici $w_1 = u_1$, $w_2 = v_1yzv_2$ et $u_2 = v_2z'y'v_2$), nous avons :

$$u \xrightarrow[R_e]{*} xu_1'v_1yzv_2 \xrightarrow[R_e]{} xu_1'v_1z'y'v_2 \xrightarrow[R_e]{|u_1|} u_1x'v_1z'y'v_2.$$

Donc, nous savons immédiatement que $u_1x'u_2$ appartient à $\mathrm{f}_{R_e}(xu)$ avec

$$\begin{cases} u_2 = v_1z'y'v_2, \\ u \longrightarrow_{R_e}^{*} u_1'v_1z'y'v_2, \\ xu_1' \longrightarrow_{R_e}^{k} u_1x', \\ k = |u_1|, \\ h(u_1') = h(u_1). \end{cases}$$

□

Dans la mesure où il est possible d'effectuer des dérivations pas à pas, nous allons définir une notion de distance. Pour cela, nous étendons la Définition 1.23 donnée au Chapitre 1 :

Définition 3.21 *Soient $u, v \in \Sigma_e^*$. On appelle distance de u à v,*

$$\mathrm{d}(u, v) = \mathrm{d}(h(u), h(v)).$$

Par définition, si v appartient à $\mathrm{f}_{\mathrm{R}_e}(u)$, la distance de u à v est la borne inférieure des longueurs des dérivations permettant de passer de u à v. Cette borne peut effectivement être atteinte :

Lemme 3.22 (Lemme des Distances) *Soient u appartenant à Σ_e^* et v un mot de $\mathrm{f}_{\mathrm{R}_e}(u)$. Nous avons :*

$$(\mathrm{d}(u, v) = k) \Rightarrow (u \xrightarrow[R_e]{k} v).$$

Preuve. Montrons le lemme par récurrence sur la longueur de u. La propriété est trivialement vérifiée si u est de longueur nulle. Soient u et v appartenant à Σ_e^* tels que v appartient à $\mathrm{f}_{\mathrm{R}_e}(u)$. Posons $u = xu_1$ avec $x \in \Sigma_e$ et $v = v_1 x' v_2$ avec $\alpha(x) \notin \alpha(v_1)$. D'après le Lemme 3.20, nous avons :

$$u_1 \xrightarrow[R_e]{*} v_1' v_2 \text{ et } x v_1' \xrightarrow[R_e]{i} v_1 x' \text{ avec } i = |v_1|.$$

Par hypothèse de récurrence, il existe une dérivation par R_e de u_1 à $v_1' v_2$ de longueur $j = \mathrm{d}(u_1, v_1' v_2)$. Donc nous avons :

$$x u_1 \xrightarrow[R_e]{j} x v_1' v_2 \xrightarrow[R_e]{i} v_1 x' v_2.$$

La longueur de la dérivation est $i + j = |v_1| + \mathrm{d}(u_1, v_1' v_2)$, comme $h(v_1) = h(v_1')$, $i + j = \mathrm{d}(xu_1, v_1 x' v_2)$. □

Le lemme précédent nous permet d'énoncer le corollaire :

Corollaire 3.23 *Soient u appartenant à Σ_e^* et v appartenant à $\mathrm{f}_{\mathrm{R}_e}(u)$. Si u et v ont même image par h alors ces deux mots sont égaux.*

Preuve. Si $h(u) = h(v)$ alors la distance de u à v est nulle. D'après le Lemme 3.22, u se réécrit en v par une dérivation de longueur nulle donc u et v sont égaux. □

Montrons à présent qu'un mot u ne peut se dériver en deux mots différents ayant le même squelette :

Corollaire 3.24 *Soient u, v_1, v_2 des mots de Σ_e^* tels que v_1 et v_2 appartiennent à $\mathrm{f}_{\mathrm{R}_e}(u)$. Si $h(v_1) = h(v_2)$ alors v_1 et v_2 sont égaux.*

Preuve. Toutes les règles de R_e sont symétriques, donc si v_1 appartient à $\mathrm{f}_{\mathrm{R}_e}(u)$, u appartient à $\mathrm{f}_{\mathrm{R}_e}(v_1)$ donc, v_2 est un mot de $\mathrm{f}_{\mathrm{R}_e}(v_1)$. Comme $h(v_1) = h(v_2)$, d'après le Corollaire 3.23, les mots v_1 et v_2 sont égaux. □

Ce corollaire termine l'énoncé des propriétés élémentaires du système R_e. Grâce aux lemmes techniques dont nous disposons à présent, nous allons pouvoir dégager les propriétés essentielles du système. En premier lieu, nous pouvons énoncer un Lemme de Levi.

Lemme 3.25 (Lemme de Levi) *Soient u_1, u_2, v_1 et v_2 des mots de $\Sigma_e{}^*$ tels que v_1v_2 appartient à $\mathrm{f}_{\mathrm{R}_e}(u_1u_2)$. Il existe $\alpha_1, \alpha_2, \beta_1, \beta_2, \alpha'_1, \alpha'_2, \beta'_1, \beta'_2$ appartenant à $\Sigma_e{}^*$ tels que :*

$$u_1 \xrightarrow[R_e]{*} \alpha_1\alpha_2 \text{ et } u_2 \xrightarrow[R_e]{*} \beta_1\beta_2 \text{ avec } \alpha(\alpha_2) \cap \alpha(\beta_1) = \emptyset$$

et

$$\alpha_2\beta_1 \xrightarrow[R_e]{*} \beta'_1\alpha'_2, \alpha_1\beta'_1 \xrightarrow[R_e]{*} v_1 \text{ et } \alpha'_2\beta_2 \xrightarrow[R_e]{*} v_2.$$

Preuve. Soient u_1, u_2, v_1 et v_2 des mots de $\Sigma_e{}^*$ tels que v_1v_2 appartient à $\mathrm{f}_{\mathrm{R}_e}(u_1u_2)$. Montrons le lemme par récurrence sur la longueur de u_1. Si u_1 est de longueur nulle, le lemme est trivialement vérifié sinon, posons $u_1 = xw$. Il y a alors deux découpages possibles pour v_1v_2.

- Dans le premier cas, nous avons $v_1v_2 = \nu_1x'\nu_2v_2$, $\alpha(x)$ n'étant pas une action de $\alpha(\nu_1)$. D'après le Lemme 3.20, nous avons :

  $$wu_2 \xrightarrow[R_e]{*} \nu'_1\nu_2v_2 \text{ et } x\nu'_1 \xrightarrow[R_e]{*} \nu_1x'.$$

  Par hypothèse de récurrence, comme $\nu'_1\nu_2v_2$ appartient à $\mathrm{f}_{\mathrm{R}_e}(wu_2)$ et comme w est de taille strictement inférieure à celle de u, nous avons :

  $$w \xrightarrow[R_e]{*} \alpha_1\alpha_2 \text{ et } u_2 \xrightarrow[R_e]{*} \beta_1\beta_2 \text{ avec } \alpha(\alpha_2) \cap \alpha(\beta_1) = \emptyset$$

  et

  $$\alpha_2\beta_1 \xrightarrow[R_e]{*} \beta'_1\alpha'_2, \alpha_1\beta'_1 \xrightarrow[R_e]{*} \nu'_1\nu_2 \text{ et } \alpha'_2\beta_2 \xrightarrow[R_e]{*} v_2.$$

  Donc, nous avons immédiatement :

  $$xw \xrightarrow[R_e]{*} x\alpha_1\alpha_2 \text{ et } u_2 \xrightarrow[R_e]{*} \beta_1\beta_2 \text{ avec } \alpha(\alpha_2) \cap \alpha(\beta_1) = \emptyset$$

  et

  $$\alpha_2\beta_1 \xrightarrow[R_e]{*} \beta'_1\alpha'_2, x\alpha_1\beta'_1 \xrightarrow[R_e]{*} x\nu'_1\nu_2 \xrightarrow[R_e]{*} \nu_1x'\nu_2 = v_1 \text{ et } \alpha'_2\beta_2 \xrightarrow[R_e]{*} v_2.$$

- Dans le deuxième cas, nous avons $v_1v_2 = v_1\nu_1x'\nu_2$, $\alpha(x)$ n'étant pas une action de $\alpha(v_1\nu_1)$. D'après le Lemme 3.20, nous avons :

  $$wu_2 \xrightarrow[R_e]{*} v'_1\nu'_1\nu_2 \text{ et } xv'_1\nu'_1 \xrightarrow[R_e]{*} v_1\nu_1x'.$$

  Par hypothèse de récurrence, comme $v'_1\nu'_1\nu_2$ appartient à $\mathrm{f}_{\mathrm{R}_e}(wu_2)$ et comme w est de taille strictement inférieure à celle de u, nous avons :

  $$w \xrightarrow[R_e]{*} \alpha_1\alpha_2 \text{ et } u_2 \xrightarrow[R_e]{*} \beta_1\beta_2 \text{ avec } \alpha(\alpha_2) \cap \alpha(\beta_1) = \emptyset$$

  et

  $$\alpha_2\beta_1 \xrightarrow[R_e]{*} \beta'_1\alpha'_2, \alpha_1\beta'_1 \xrightarrow[R_e]{*} v'_1 \text{ et } \alpha'_2\beta_2 \xrightarrow[R_e]{*} \nu'_1\nu_2.$$

Comme $v_1\nu_1x'$ appartient à $\mathrm{f}_{\mathrm{R}_e}(xv_1'\nu_1')$, d'après le Lemme des Distances 3.22, nous pouvons considérer une dérivation minimale donc $v_1x''\nu_1'$ appartient à $\mathrm{f}_{\mathrm{R}_e}(xv_1'\nu_1')$ et $v_1\nu_1x'$ appartient à $\mathrm{f}_{\mathrm{R}_e}(v_1x''\nu_1')$, nous avons :

$$xv_1'\nu_1' \xrightarrow[R_e]{*} v_1x''\nu_1' \xrightarrow[R_e]{*} v_1\nu_1x'.$$

Comme v_1' appartient à $\mathrm{f}_{\mathrm{R}_e}(\alpha_1\beta_1')$ et v_1x'' à $\mathrm{f}_{\mathrm{R}_e}(xv_1')$, nous déduisons des Lemmes 3.19 et 3.22 :

$$x\alpha_1\beta_1' \xrightarrow[R_e]{*} \alpha_1'x'''\beta_1' \xrightarrow[R_e]{*} \alpha_1'\beta_1''x'' \xrightarrow[R_e]{*} v_1x''.$$

Donc, nous avons :

$$xw \xrightarrow[R_e]{*} x\alpha_1\alpha_2 \xrightarrow[R_e]{*} \alpha_1'x'''\alpha_2 \text{ et } u_2 \xrightarrow[R_e]{*} \beta_1\beta_2$$

avec $\alpha(x'''\alpha_2) \cap \alpha(\beta_1) \neq \emptyset$ ainsi que :

$$x'''\alpha_2\beta_1 \xrightarrow[R_e]{*} x'''\beta_1'\alpha_2' \xrightarrow[R_e]{*} \beta_1''x''\alpha_2', \alpha_1'\beta_1' \xrightarrow[R_e]{*} v_1 \text{ et } x''\alpha_2'\beta_2 \xrightarrow[R_e]{*} \nu_1'\nu_2 \xrightarrow[R_e]{*} \nu_1x'\nu_2.$$

□

Nous pouvons déduire du lemme de Levi un Lemme de Simplification :

Lemme 3.26 (Lemme de Simplification pour R_e) *Soient u, v, u_1 et v_1 des mots de $\Sigma_e{}^*$ tels que les squelettes de u_1 et v_1 sont respectivement commutativement équivalents à ceux de u et v et tels que u_1v_1 appartient à $\mathrm{f}_{\mathrm{R}_e}(uv)$. Alors, nous avons*

$$u \xrightarrow[R_e]{*} u_1 \text{ et } v \xrightarrow[R_e]{*} v_1.$$

Preuve. D'après le Lemme de Levi 3.25, nous avons :

$$u \xrightarrow[R_e]{*} \alpha_1\alpha_2 \text{ et } v \xrightarrow[R_e]{*} \beta_1\beta_2 \text{ avec } \alpha(\alpha_2) \cap \alpha(\beta_1) = \emptyset$$

et

$$\alpha_2\beta_1 \xrightarrow[R_e]{*} \beta_1'\alpha_2', \alpha_1\beta_1' \xrightarrow[R_e]{*} u_1 \text{ et } \alpha_2'\beta_2 \xrightarrow[R_e]{*} v_1.$$

Or nous savons que $h(u)$ et $h(u_1)$ sont commutativement équivalents donc, $\beta_1 = \alpha_2 = \varepsilon$. Nous avons donc :

$$u \xrightarrow[R_e]{*} \alpha_1 \xrightarrow[R_e]{*} u_1 \text{ et } v \xrightarrow[R_e]{*} \beta_2 \xrightarrow[R_e]{*} v_1.$$

□

Dans la mesure où notre but est d'établir un lien entre R' et R_e, il semble nécessaire que ce dernier vérifie un Lemme de Projection semblable à celui énoncé pour les semi-commutations puisque, par définition, R' possède cette propriété.

Lemme 3.27 (Lemme de Projection) *Soient u et v deux mots de $\Sigma_e{}^*$. Nous avons :*

$$(u \xrightarrow[R_e]{*} v) \Leftrightarrow (\forall a, b \in \Sigma, \Pi_{ab}(u) \xrightarrow[R_e]{*} \Pi_{ab}(v)).$$

Preuve. Montrons l'implication de gauche à droite par récurrence sur la longueur de la dérivation par R_e de u à v. Clairement, cette implication est vraie pour une dérivation de longueur nulle. Considérons une dérivation

$$u \xrightarrow[R_e]{n} w \xrightarrow[R_e]{} v.$$

Par hypothèse de récurrence, pour toute paire d'actions a et b de Σ, $\Pi_{ab}(w)$ appartient à $\mathrm{f}_{\mathrm{R}_e}(\Pi_{ab}(u))$. Considérons la dérivation de w à v :

$$w = w_1(x,E)(y,F)w_2 \xrightarrow[R_e]{} v = w_1(y,F')(x,E')w_2.$$

La règle suivante appartient donc à R_e :

$$(x, E \cap \{y\})(y, F \cap \{x\}) \longrightarrow (y, F' \cap \{x\})(x, E' \cap \{y\}).$$

Pour toute paire d'actions a et b de Σ distinctes de $\alpha(x)$ et $\alpha(y)$, $\Pi_{ab}(w) = \Pi_{ab}(v)$, il suffit donc de considérer la projection sur les actions $\alpha(x)$ et $\alpha(y)$ pour laquelle on obtient la dérivation :

$$\begin{array}{c} \Pi_{\alpha(x)\alpha(y)}(w) = \Pi_{\alpha(x)\alpha(y)}(w_1)(x, E \cap \{y\})(y, F \cap \{x\})\Pi_{\alpha(x)\alpha(y)}(w_2) \\ R_e \downarrow \\ \Pi_{\alpha(x)\alpha(y)}(w_1)(y, F' \cap \{x\})(x, E' \cap \{y\})\Pi_{\alpha(x)\alpha(y)}(w_2) = \Pi_{\alpha(x)\alpha(y)}(v). \end{array}$$

Donc, pour toute paire d'actions a et b de Σ, $\Pi_{ab}(v)$ appartient à $\mathrm{f}_{\mathrm{R}_e}(\Pi_{ab}(u))$.

Montrons à présent l'implication de droite à gauche par récurrence sur la longueur de u. Si u est de longueur nulle, l'implication est trivialement vérifiée, sinon, considérons un mot v de ${\Sigma_e}^*$ tel que pour toute paire d'actions a et b de Σ, $\Pi_{ab}(v)$ appartient à $\mathrm{f}_{\mathrm{R}_e}(\Pi_{ab}(u))$. Posons $u = xu_1$ et $v = v_1x'v_2$ avec $\alpha(x) \notin \alpha(v_1)$ ainsi, $\Pi_{ab}(v_1x'v_2)$ appartient à $\mathrm{f}_{\mathrm{R}}(\Pi_{ab}(xu_1))$, c'est-à-dire que $\Pi_{ab}(v_1)\Pi_{ab}(x')\Pi_{ab}(v_2)$ appartient à $\mathrm{f}_{\mathrm{R}_e}(\Pi_{ab}(x)\Pi_{ab}(u_1))$. D'après le Lemme 3.20, nous avons :

$$\Pi_{ab}(u_1) \xrightarrow[R_e]{*} (\Pi_{ab}(v_1))'\Pi_{ab}(v_2) \text{ et } \Pi_{ab}(x)(\Pi_{ab}(v_1))' \xrightarrow[R_e]{k} \Pi_{ab}(v_1)\Pi_{ab}(x') \text{ avec } k = |\Pi_{ab}(v_1)|.$$

Donc, pour toute paire d'actions a et b de Σ, $(\Pi_{ab}(v_1))'\Pi_{ab}(v_2)$ appartient à $\mathrm{f}_{\mathrm{R}_e}(\Pi_{ab}(u_1))$.

Construisons un mot v_1' tel que $h(v_1') = h(v_1)$ de la façon suivante : pour tout découpage $v_1' = w_1(x,E)w_2$, $E = \bigcup_{b\in\Sigma} E_b$ avec, pour toute action b de Σ, $(\Pi_{\alpha(x)b}(v_1))' = \alpha(x,E_b)\beta$ avec $|h(w_1)|_x = |h(\alpha)|_x$. Par construction, pour toute paire d'actions a et b de Σ, $\Pi_{ab}(v_1') = (\Pi_{ab}(v_1))'$. Comme la taille de u_1 est strictement inférieure à celle de u, par hypothèse de récurrence, $v_1'v_2$ appartient à $\mathrm{f}_{\mathrm{R}_e}(u_1)$ donc, $xv_1'v_2$ appartient à $\mathrm{f}_{\mathrm{R}_e}(xu_1)$.

Montrons par récurrence sur la longueur de v_1' que v_1x' appartient à $\mathrm{f}_{\mathrm{R}_e}(xv_1')$. Nous savons que pour toute paire d'actions a et b de Σ, $\Pi_{ab}(x)\Pi_{ab}(v_1')$ se réécrit, par R_e, en $\Pi_{ab}(v_1)\Pi_{ab}(x')$ par une dérivation de longueur $|\Pi_{ab}(v_1)|$. Posons $v_1' = y'w_1'$ et $k = |\Pi_{\alpha(x)\alpha(y)}(v_1)|$ (on a donc $v_1 = yw_1$) et considérons la dérivation suivante :

$$\Pi_{\alpha(x)\alpha(y)}(x)\Pi_{\alpha(x)\alpha(y)}(v_1') \xrightarrow[R_e]{k} \Pi_{\alpha(x)\alpha(y)}(v_1)\Pi_{\alpha(x)\alpha(y)}(x').$$

Comme la longueur de la dérivation est $|\Pi_{\alpha(x)\alpha(y)}(v_1)|$, nous avons :

$$\Pi_{\alpha(x)\alpha(y)}(x)\Pi_{\alpha(x)\alpha(y)}(y'w_1') \xrightarrow[R_e]{} \Pi_{\alpha(x)\alpha(y)}(y')'\Pi_{\alpha(x)\alpha(y)}(x)''\Pi_{\alpha(x)\alpha(y)}(w_1'),$$

avec $\Pi_{\alpha(x)\alpha(y)}(y')' = \Pi_{\alpha(x)\alpha(y)}(y)$, c'est-à-dire que yx'' appartient à $\mathrm{f}_{\mathrm{R}_e}(xy')$ et nous avons:

$$\Pi_{\alpha(x)\alpha(y)}(x)''\Pi_{\alpha(x)\alpha(y)}(w_1') \xrightarrow[R_e]{k-1} \Pi_{\alpha(x)\alpha(y)}(w_1)\Pi_{\alpha(x)\alpha(y)}(x').$$

Par hypothèse de récurrence, w_1x' appartient à $\mathrm{f}_{\mathrm{R}_e}(x''w_1')$, donc nous avons:

$$xv_1' = xy'w_1' \xrightarrow[R_e]{*} yx''w_1' \xrightarrow[R_e]{*} yw_1x' = v_1x'.$$

En conclusion, $u = xu_1 \longrightarrow_{R_e}^{*} v_1x'v_2 = v$. □

Dans la dernière proposition de cette partie, nous allons voir comment, dans le cas de langages définis sur des alphabets de deux actions, R_e peut être utilisé pour calculer une clôture par R.

Proposition 3.28 *Soient Σ un alphabet de deux actions et u et v deux facteurs de st-mots de $(\Sigma_s \cup \Sigma_t)^*$. Alors, nous avons:*

$$(\exists w' \in h^{-1}(w), g(u) \xrightarrow[R_e]{*} w') \Leftrightarrow (u \xrightarrow[R]{*} w).$$

Preuve. Comme $g(u)$ appartient à L_Σ, l'implication de droite à gauche est une conséquence immédiate du Lemme 3.17. Il suffit donc de montrer la réciproque.

Posons $\Sigma = \{a, b\}$. Pour augmenter la lisibilité de cette preuve, nous utiliserons la Notation 3.12, $\Sigma_e = \{a_s, a_t, b_s, b_t, \bar{a_s}, \bar{a_t}, \bar{b_s}, \bar{b_t}\}$. Le système R_e est donc composé des règles:

$$\begin{array}{rcl} a_sb_s & \longleftrightarrow & b_sa_s, \\ \bar{a_s}b_s & \longleftrightarrow & \bar{b_s}a_s, \\ a_tb_t & \longleftrightarrow & b_ta_t, \\ a_t\bar{b_t} & \longleftrightarrow & b_t\bar{a_t}, \\ a_sb_t & \longleftrightarrow & \bar{b_t}\bar{a_s}, \\ b_sa_t & \longleftrightarrow & \bar{a_t}\bar{b_s}. \end{array}$$

Pour tout mot u de $(\Sigma_s \cup \Sigma_t)^*$, nous notons donc abusivement $g(u) = u$.

Considérons un mot u de $(\Sigma_s \cup \Sigma_t)^*$ et un mot w de $\mathrm{f}_{\mathrm{R}_e}(u)$. Montrons l'implication par récurrence sur la longueur de la dérivation minimale de u à w. Clairement, l'implication est vraie pour une dérivation de longueur nulle sinon, considérons la dérivation

$$u \xrightarrow[R_e]{n} w_1xyw_2 \xrightarrow[R_e]{} w_1y'x'w_2 = w \text{ avec } n+1 = \mathrm{d}(u, w).$$

Nous pouvons distinguer deux cas en fonction de la dernière règle appliquée.

- Premier cas: $xy \longrightarrow yx \notin \{\bar{a_t}\bar{b_s} \longrightarrow b_sa_t, \bar{b_t}\bar{a_s} \longrightarrow a_sb_t\}$. Par hypothèse de récurrence, $h(w_1xyw_2)$ appartient à $\mathrm{f}_{\mathrm{R}}(u)$, comme il existe une règle $h(x)h(y) \longrightarrow h(y)h(x)$ de θ correspondant à la règle $xy \longrightarrow y'x'$ de R_e, nous avons immédiatement

$$u \xrightarrow[R]{*} h(w_1xyw_2) \xrightarrow[\theta]{} h(w_1)h(y')h(x')h(w_2) = h(w).$$

- Deuxième cas : $xy \longrightarrow yx \in \{\bar{a}_t\bar{b}_s \longrightarrow b_s a_t, \bar{b}_t\bar{a}_s \longrightarrow a_s b_t\}$. Il suffit de considérer l'une des deux possibilités, posons $xy = \bar{a}_t\bar{b}_s$. Comme u est un mot de Σ_0^* et que a_t et b_s sont barrées, nous avons appliqué, durant la dérivation une règle $b_s a_t \longrightarrow \bar{a}_t\bar{b}_s$ ou $a_s b_t \longrightarrow \bar{b}_t\bar{a}_s$ puis dans un ordre quelconque les règles $a_t\bar{b}_t \longrightarrow b_t\bar{a}_t$ et $\bar{a}_s b_s \longrightarrow \bar{b}_s a_s$ à ces deux lettres.

  Le premier cas est exclu car la dérivation est minimale. On peut écrire $w_1\bar{a}_t\bar{b}_s w_2 = u_1 b' u_2 \bar{a}_t\bar{b}_s u_3 a' u_4$ avec $h(b') = b_t$, $h(a') = a_s$, $h(u_2) \in (a_s + a_t)^*$ et $h(u_3) \in (b_s + b_t)^*$. Comme la dérivation est minimale, b' n'a commuté avec aucune lettre de u_1 après avoir commuté avec $\bar{a}_t$, on peut donc regrouper toutes les dérivations qui impliquent des lettres de u_2 et b' (on procède de la même manière pour a'). Les dernières règles impliquant $\bar{a}_t$, $\bar{b}_s$, a' et b' sont $a_s b_t \longrightarrow \bar{b}_t\bar{a}_s$ puis $a_t\bar{b}_t \longrightarrow b_t\bar{a}_t$ et $\bar{a}_s b_s \longrightarrow \bar{b}_s a_s$, on peut donc également regrouper ces règles en fin de dérivation :

$$\begin{array}{lllll} u & \longrightarrow_{R_e}^{i} & u_1 u_2' a_t a_s b_t b_s u_3' w_4 & \longrightarrow_{R_e} & u_1 u_2' a_t \bar{b}_t\bar{a}_s b_s u_3' u_4 \\ & \longrightarrow_{R_e} & u_1 u_2' b_t\bar{a}_t\bar{a}_s b_s u_3' w_4 & \longrightarrow_{R_e} & u_1 u_2' b_t\bar{a}_t\bar{b}_s a_s u_3' w_4 \\ & \longrightarrow_{R_e} & u_1 u_2' b_t b_s a_t a_s u_3' w_4 & \longrightarrow_{R_e}^{j} & u_1 b' u_2 b_s a_t a_s u_3' u_4 \\ & \longrightarrow_{R_e}^{k} & u_1 b' u_2 b_s a_t u_3 a' u_4 & = & w \end{array}$$

  avec $i + 4 + j + k = n + 1$. Comme i est strictement inférieur à $n + 1$, par hypothèse de récurrence, v_0 appartient à $\mathrm{f}_{\mathrm{R}}(u)$ avec

$$v_0 = h(u_1 u_2' a_t a_s b_t b_s u_3' u_4) = h(u_1)h(u_2')a_t a_s b_t b_s h(u_3')h(u_4)$$
$$= h(u_1)h(u_2)a_t a_s b_t b_s h(u_3)h(u_4) \underset{R}{\longrightarrow} h(u_1)h(u_2)b_t b_s a_t a_s h(u_3)h(u_4).$$

  De plus, nous savons que $h(u_2)$ est inclus dans $(a_s + a_t)^*$ et $h(u_3)$ dans $(b_s + b_t)^*$, nous pouvons donc terminer :

$$\begin{array}{lll} u \longrightarrow_R^* v_0 & \longrightarrow_R & h(u_1)h(u_2)b_t b_s a_t a_s h(u_3)h(u_4) \\ & \longrightarrow_\theta^* & h(u_1)b_t h(u_2)b_s a_t h(u_3)a_s h(u_4) = h(w). \end{array}$$

En conclusion, comme w appartient à $\mathrm{f}_{\mathrm{R}_e}(g(u))$, $h(w)$ appartient à $\mathrm{f}_{\mathrm{R}}(u)$. □

Nous avons vu de nombreuses propriétés de R_e qui sont, pour la plupart, le pendant des propriétés des commutations partielles. Ces résultats ne sont pas immédiats en raison de la particularité de l'alphabet et de l'évolution de celui-ci au cours de la dérivation. Ce travail étant terminé, nous disposons à présent d'une « boîte à outils » complète permettant de manipuler notre système.

3.2.3 Propriétés de R'

Nous allons à présent nous intéresser aux propriétés du système R' et, en premier lieu, nous intéresser aux liens entre R_e et R'. Deux choses interviennent dans la définition du système R', les projections sur les sous-alphabets de deux actions et le système R. Or, nous avons vu que le système R_e vérifie un Lemme de Projection semblable à celui énoncé pour les semi-commutations et qu'il peut être utilisé pour calculer la clôture par R dans le cas de langages définis sur des alphabets de deux actions. En utilisant le Lemme de Projection 3.27 et la Proposition 3.28 nous allons donc pouvoir établir le lien entre R_e et R'.

Proposition 3.29 *Soient u et v deux facteurs de st-mots de $(\Sigma_s \cup \Sigma_t)^*$. Nous avons :*

$$(u \underset{R'}{\longrightarrow} v) \Leftrightarrow (\exists w \in h^{-1}(v), g(u) \underset{R_e}{\overset{*}{\longrightarrow}} w).$$

Preuve. Soient u et v deux facteurs de st-mots de $(\Sigma_s \cup \Sigma_t)^*$. Nous avons:

$$(\exists w \in h^{-1}(v), g(u) \xrightarrow[R_e]{*} w) \Rightarrow (\forall a, b \in \Sigma, \Pi_{ab}(g(u)) \xrightarrow[R_e]{*} \Pi_{ab}(w)), \tag{3.3}$$

$$\Rightarrow (\forall a, b \in \Sigma, g(\Pi_{ab}(u)) \xrightarrow[R_e]{*} \Pi_{ab}(w)), \tag{3.4}$$

$$\Rightarrow (\forall a, b \in \Sigma, \Pi_{ab}(u) \xrightarrow[R]{*} h(\Pi_{ab}(w))), \tag{3.5}$$

$$\Rightarrow (\forall a, b \in \Sigma, \Pi_{ab}(u) \xrightarrow[R]{*} \Pi_{ab}(v)), \tag{3.6}$$

$$\Rightarrow (u \xrightarrow[R']{} v). \tag{3.7}$$

L'implication 3.3 se déduit de Lemme de Projection 3.27. Pour les implications 3.4 et 3.6, il suffit de remarquer que $\Pi_{ab} \circ g = g \circ \Pi_{ab}$ et $h \circ \Pi_{ab} = \Pi_{ab} \circ h$. La Proposition 3.28 permet de déduire l'implication 3.5 et enfin, l'implication 3.7 est l'application de la définition de R'.

Pour montrer l'implication réciproque, on peut, en utilisant la définition de R', la Proposition 3.28 et l'égalité $\Pi_{ab} \circ g = g \circ \Pi_{ab}$, facilement obtenir les implications suivantes:

$$\begin{aligned}(u \xrightarrow[R']{} v) &\Rightarrow (\forall a, b \in \Sigma, \Pi_{ab}(u) \xrightarrow[R]{*} \Pi_{ab}(v),\\ &\Rightarrow (\forall a, b \in \Sigma, \exists w_{ab} \in h^{-1}(\Pi_{ab}(v)), g(\Pi_{ab}(u)) \xrightarrow[R_e]{*} w_{ab},\\ &\Rightarrow (\forall a, b \in \Sigma, \exists w_{ab} \in h^{-1}(\Pi_{ab}(v)), \Pi_{ab}(g(u)) \xrightarrow[R_e]{*} w_{ab}).\end{aligned}$$

Soit w un mot de $h^{-1}(v)$ tel que pour tout découpage $w = w_1(x, E)w_2$, on a $E = \bigcup_{b\in\Sigma}(E_b)$ avec pour toute action b de Σ, $w_{\alpha(x)b} = v_1(x, E_b)v_2$ avec $|h(v_1)|_x = |h(w_{\alpha(x)b})|_x$. Par construction, pour toute paire d'actions a et b de Σ, $\Pi_{ab}(w) = w_{ab}$ et $h(w) = v$. Nous pouvons donc appliquer le Lemme de Projection 3.27 pour obtenir:

$$(\forall a, b \in \Sigma, \Pi_{ab}(g(u)) \xrightarrow[R_e]{*} \Pi_{ab}(w)) \Rightarrow (g(u) \xrightarrow[R_e]{*} w).$$

□

Nous allons voir, en reprenant un des exemples du début de ce chapitre, comment utiliser R_e à la place de R'. Nous avions considéré l'exemple des mots $w = a_t z_t z_s y_t a_s b_t y_s b_s$ et $w' = b_t y_t b_s a_t y_s z_t z_s a_s \in \mathrm{f}_{R'}(w)$ que l'on ne pouvait obtenir en utilisant des règles du type $uv \longrightarrow vu$, nous allons voir qu'en utilisant R_e, nous pouvons placer une à une les lettres de w pour constituer w' sans se soucier de l'ordre dans lequel on applique les règles de dérivation:

$$\begin{aligned}g(w) = &\ (a_t,\emptyset)(z_t,\emptyset)(z_s,\emptyset)(y_t,\emptyset)(a_s,\emptyset)(b_t,\emptyset)(y_s,\emptyset)(b_s,\emptyset)\\ \xrightarrow[R_e]{5} &\ (\mathbf{b_t},\emptyset)(a_t,\{b\})(z_t,\{b\})(z_s,\{b\})(y_t,\emptyset)(a_s,\{b\})(y_s,\emptyset)(b_s,\emptyset)\\ \xrightarrow[R_e]{3} &\ (\mathbf{b_t},\emptyset)(\mathbf{y_t},\emptyset)(a_t,\{b\})(z_t,\{b,y\})(z_s,\{b,y\})(a_s,\{b\})(y_s,\emptyset)(b_s,\emptyset)\\ \xrightarrow[R_e]{5} &\ (\mathbf{b_t},\emptyset)(\mathbf{y_t},\emptyset)(\mathbf{b_s},\emptyset)(a_t,\emptyset)(z_t,\{y\})(z_s,\{y\})(a_s,\emptyset)(y_s,\emptyset)\\ \xrightarrow[R_e]{3} &\ (\mathbf{b_t},\emptyset)(\mathbf{y_t},\emptyset)(\mathbf{b_s},\emptyset)(\mathbf{a_t},\emptyset)(\mathbf{y_s},\emptyset)(\mathbf{z_t},\emptyset)(\mathbf{z_s},\emptyset)(\mathbf{a_s},\emptyset)(\mathbf{y_s},\emptyset) \in h^{-1}(w').\end{aligned}$$

Nous pouvons à présent utiliser le système R_e pour calculer la clôture par R'. Nous allons donc pouvoir déduire des propriétés de R_e deux propriétés de R' qui s'avèreront utiles par la suite. Du corollaire 3.14, nous déduisons une propriété importante de la clôture par $\mathrm{f}_{R'}$:

Lemme 3.30 *Soient $L_1, L_2 \subseteq (\Sigma_s \cup \Sigma_t)^*$ deux st-langages. Alors nous avons:*

$$\mathrm{f}_{R'}(L_1.L_2) = \mathrm{f}_{R'}(L_1).\mathrm{f}_{R'}(L_2)$$

et

$$\mathrm{f}_{R'}(L_1^*) = (\mathrm{f}_{R'}(L_1))^*.$$

Preuve. Clairement $f_{R'}(L_1).f_{R'}(L_2)$ est inclus dans $f_{R'}(L_1.L_2)$, il suffit donc de montrer que pour deux st-mots u_1 et u_2 de Σ_0^*, $f_{R_e}(u_1u_2)$ est inclus dans $f_{R_e}(u_1).f_{R_e}(u_2)$. Nous avons immédiatement l'inclusion $f_{R_e}(u_1u_2) \subseteq f_{R_e}(f_{R_e}(u_1).f_{R_e}(u_2))$ or, d'après le Corollaire 3.14, comme u_1 et u_2 appartiennent à Σ_0^*, $f_{R_e}(u_1)$ et $f_{R_e}(u_2)$ appartiennent à $g(\Sigma_s){\Sigma_e}^*g(\Sigma_t)$. Comme aucune partie gauche de règle de R_e n'appartient à $g(\Sigma_t)g(\Sigma_s)$, nous avons $f_{R_e}(u_1u_2) = f_{R_e}(u_1).f_{R_e}(u_2)$. De ce résultat, nous déduisons immédiatement $f_{R'}(L_1^*) = (f_{R'}(L_1))^*$. □

Enfin, du Lemme de Simplification pour R_e, nous déduisons immédiatement un Lemme de Simplification pour R'.

Corollaire 3.31 (Lemme de Simplification pour R') *Soient u, v, u_1 et v_1 des facteurs de st-mots de $(\Sigma_s \cup \Sigma_t)^*$ tels que u_1 et v_1 sont respectivement commutativement équivalents à u et v et tels que u_1v_1 appartient à $f_{R_e}(uv)$. Alors, nous avons:*

$$u \xrightarrow[R']{} u_1 \text{ et } v \xrightarrow[R']{} v_1.$$

Preuve. D'après la Proposition 3.29, il existe un mot $u_1'v_1'$ appartenant à $f_{R_e}(g(u)g(v))$ avec $h(u_1') = u_1$ et $h(v_1') = v_1$. D'après le Lemme 3.26, u_1' appartient à $f_{R_e}(g(u))$ et v_1' appartient à $f_{R_e}(g(v))$. D'après la Proposition 3.29, nous obtenons:

$$u \xrightarrow[R']{} u_1 \text{ et } v \xrightarrow[R']{} v_1.$$

□

Nous avons donc montré que le système de réécriture R' est un système qui possède de nombreuses propriétés intéressantes d'un point de vue technique, nous allons montrer, dans la section suivante, qu'il est parfaitement adapté à l'étude des langages de synchronisation.

3.3 Langages de synchronisation et systèmes de réécriture

Nous allons, dans cette partie, étudier la relation entre le système de réécriture R' et les langages de synchronisation. Nous allons ensuite nous intéresser aux autres systèmes de réécriture que nous comparerons à R'.

3.3.1 Langages de synchronisation et R'

En premier lieu, le système de réécriture R' conserve la propriété intéressante du système R: les langages de synchronisation sont clos par R'.

Proposition 3.32 *Tout langage de synchronisation est clos par R'.*

Preuve. Montrons le lemme par induction sur la construction des langages de synchronisation. Soient L, L_1, L_2 des langages de synchronisation définis sur l'alphabet $\Sigma_s \cup \Sigma_t$.

- Si $L = \varepsilon$ ou $L = x_sx_t$ avec x une action de Σ, alors $L = f_{R'}(L)$.
- Si $L = L_1 + L_2$, pour tout mot u de L, il existe un indice i de $\{1,2\}$ tel que $u \in L_i$. Comme L_i est clos par R', pour tout v appartenant à $f_{R'}(u)$, v appartient à L_i et donc v appartient à L.

- Si $L = L_1 \cap L_2$, tout mot u de L appartient à L_1 et L_2. Comme L_1 et L_2 sont clos par R', tout mot v appartenant à $\mathrm{f}_{R'}(u)$ appartient à L_1 et L_2 et donc à $L_1 \cap L_2 = L$.

- Si $L = L_1 \sqcup L_2$, montrons que $L = \mathrm{f}_{R'}(L)$ c'est-à-dire que $L_1 \sqcup L_2 = \mathrm{f}_{R'}(L_1 \sqcup L_2)$. Clairement, $L_1 \sqcup L_2$ est inclus dans $\mathrm{f}_{R'}(L_1 \sqcup L_2)$ il suffit donc de montrer l'inclusion inverse.

  Si v est un mot de $\mathrm{f}_{R'}(L_1 \sqcup L_2)$, il existe un mot u de $L_1 \sqcup L_2$ tel que $v \in \mathrm{f}_{R'}(u)$. Par définition de R', pour toute paire d'actions a et b de Σ, $\Pi_{ab}(v)$ appartient à $\mathrm{f}_{R'}(\Pi_{ab}(u))$. Donc si $u_1 = \Pi_{\mathrm{alph}(L_1)}(u)$, $u_2 = \Pi_{\mathrm{alph}(L_2)}(u)$, $v_1 = \Pi_{\mathrm{alph}(L_1)}(v)$ et $v_2 = \Pi_{\mathrm{alph}(L_2)}(v)$, pour toute paire d'actions a et b de Σ, $\Pi_{ab}(v_1)$ appartient à $\mathrm{f}_{R'}(\Pi_{ab}(u_1))$ et $\Pi_{ab}(v_2)$ à $\mathrm{f}_{R'}(\Pi_{ab}(u_2))$. Par définition, v_1 appartient à $\mathrm{f}_{R'}(u_1)$ et v_2 à $\mathrm{f}_{R'}(u_2)$ donc v_1 est un mot de L_1, v_2 un mot de L_2 et v appartient à $L_1 \sqcup L_2$.

- Si $L = L_1.L_2$, d'après le Lemme 3.30, $\mathrm{f}_{R'}(L) = \mathrm{f}_{R'}(L_1).\mathrm{f}_{R'}(L_2)$. Comme L_1 et L_2 sont clos par R', nous avons immédiatement $\mathrm{f}_{R'}(L) = L_1.L_2 = L$.

- Si $L = L_1^*$, d'après le Lemme 3.30, $\mathrm{f}_{R'}(L) = (\mathrm{f}_{R'}(L_1))^*$. Comme L_1 est clos par R', on a immédiatement $\mathrm{f}_{R'}(L) = L_1^* = L$.

□

Le système R' nous permet également d'obtenir une caractérisation complète dans le cas des langages finis: les langages de synchronisation finis sont exactement les langages finis clos par R'. Nous proposons en outre, une construction pour associer une expression de synchronisation à la clôture d'un mot.

Lemme 3.33 *Soit u un st-mot de $(\Sigma_s \cup \Sigma_t)^*$, nous avons:*

$$\mathrm{f}_{R'}(u) = L(\underset{a \neq b \in \Sigma}{\&} e_{ab}),$$

avec pour toute paire d'actions distinctes a et b de Σ:

$$e_{ab} = e(\mathrm{f}_{R'}(\Pi_{ab}(u))) \underset{x \in \Sigma \setminus \{a,b\}}{\|} x^*,$$

si $e(\mathrm{f}_{R'}(\Pi_{ab}(u)))$ est une expression de synchronisation du langage $\mathrm{f}_{R'}(\Pi_{ab}(u))$.

Preuve. Par définition, nous avons l'égalité

$$\mathrm{f}_{R'}(u) = \bigcap_{a \neq b} [\mathrm{f}_{R'}(\Pi_{ab}(u)) \underset{x \in \Sigma \setminus \{a,b\}}{\sqcup\!\sqcup} (x_s x_t)^*].$$

D'autre part, d'après la Proposition 3.28, pour toute paire d'actions a et b, nous avons $\mathrm{f}_{R'}(\Pi_{ab}(u)) = \mathrm{f}_R(\Pi_{ab}(u))$ or d'après la Proposition 2.10, nous pouvons construire une expression de synchronisation pour $\mathrm{f}_R(\Pi_{ab}(u))$, nous pouvons donc construire une expression pour $\mathrm{f}_{R'}(\Pi_{ab}(u))$. □

Nous déduisons immédiatement du lemme précédent la proposition :

Proposition 3.34 *Tout st-langage fini clos par R' est un langage de synchronisation.*

La construction proposée, si elle n'aboutit pas en général à l'expression de synchronisation la plus « courte » associée à un langage donné, est cependant très simple à mettre en œuvre. Construisons par exemple l'expression associée au mot $u = a_s c_s a_t b_s c_t d_s b_t d_t$. Il nous faut d'abord construire les expressions associées aux clôtures par R' (ou R) des projections de u sur les sous-alphabets de deux actions :

$$\begin{aligned}
&e(\mathrm{f_R}(a_s a_t b_s b_t)) = a \rightarrow b,\\
&e(\mathrm{f_R}(a_s c_s a_t c_t)) = a \parallel c,\\
&e(\mathrm{f_R}(a_s a_t d_s d_t)) = a \rightarrow d,\\
&e(\mathrm{f_R}(c_s b_s c_t b_t)) = b \parallel c,\\
&e(\mathrm{f_R}(b_s d_s b_t d_t)) = b \parallel d,\\
&e(\mathrm{f_R}(c_s c_t d_s d_t)) = c \rightarrow d.
\end{aligned}$$

L'expression que nous obtenons en utilisant la construction systématique est donc :

$$\begin{aligned}
&((a \rightarrow b) \parallel c^* \parallel d^*)\&((a \parallel c) \parallel b^* \parallel d^*)\&((a \rightarrow d) \parallel b^* \parallel c^*)\&\\
&((b \parallel c) \parallel a^* \parallel d^*)\&((b \parallel d) \parallel a^* \parallel c^*)\&((c \rightarrow d) \parallel a^* \parallel b^*).
\end{aligned}$$

En éliminant les sous-expressions redondantes, nous obtenons :

$$((a \rightarrow b) \parallel c^* \parallel d^*)\&((a \rightarrow d) \parallel b^* \parallel c^*)\&((c \rightarrow d) \parallel a^* \parallel b^*),$$

on pourrait encore simplifier l'expression pour obtenir :

$$((a \rightarrow b) \parallel (c \rightarrow d))\&((a \rightarrow d) \parallel b^* \parallel c^*).$$

Le système de réécriture R' semble donc bien adapté à l'étude des langages de synchronisation, nous allons voir qu'aucun autre système de réécriture n'est plus satisfaisant.

3.3.2 Systèmes de réécriture

Nous utilisons la fonction R' pour étudier les langages de synchronisation malheureusement, nous n'avons pas la réciproque de la Proposition 3.32. En effet, nous avons montré dans le chapitre précédent que le langage

$$\mathrm{f_R}(b_s(a_s a_t)^* c_s b_t (a_s a_t)^* c_t)$$

n'est pas un langage de synchronisation. Or, nous avons

$$\mathrm{f_R}(b_s(a_s a_t)^* c_s b_t (a_s a_t)^* c_t) = \mathrm{f_{R'}}(b_s(a_s a_t)^* c_s b_t (a_s a_t)^* c_t).$$

Nous allons cependant montrer que le système R' est bien adapté à l'étude des langages de synchronisation car il n'existe pas de système plus grand qui vérifie les propriétés que nous recherchons.

Lemme 3.35 *Pour tout système de réécriture S_Σ tel que tout langage de synchronisation est clos par S_Σ, nous avons :*

$$\forall L \subseteq \mathrm{ST}_\Sigma,\ (L = \mathrm{f}_{R'_\Sigma}(L)) \Rightarrow (\mathrm{f}_{S_\Sigma}(L) = L).$$

Preuve. Soient L un st-langage sur l'alphabet d'actions Σ clos par R' et u un mot de L. D'après le Lemme 3.33, $\mathrm{f}_{R'_\Sigma}(u)$ est un langage de synchronisation. Par hypothèse, les langages de synchronisation sont clos par S_Σ, donc tout mot de $\mathrm{f}_{S_\Sigma}(u)$ appartient à $\mathrm{f}_{R'_\Sigma}(u)$ c'est-à-dire à L. □

Nous pouvons préciser le résultat du lemme précédent pour les systèmes de réécriture qui travaillent sur les facteurs de st-mots. L'utilisation de tels systèmes est en effet tout à fait naturelle dans l'étude des st-langages.

Lemme 3.36 *Pour tout système de réécriture*

$$S_\Sigma \subseteq \{(u,v) \mid \exists u_1, u_2 \ t.q. \ u_1 u u_2 \in \mathrm{ST}_\Sigma\}$$

tel que tout langage de synchronisation est clos par S_Σ, nous avons :

$$((u,v) \in S_\Sigma) \Rightarrow (u \underset{R'_\Sigma}{\longrightarrow} v).$$

Preuve. Soit $(u,v) \in S_\Sigma$. Il existe deux mots u_1 et u_2 tels que $u_1 u u_2$ appartient à ST_Σ. Nous avons :

$$u_1 u u_2 \underset{S}{\longrightarrow} u_1 v u_2.$$

D'après le Lemme 3.33, le langage $\mathrm{f}_{R'_\Sigma}(u_1 u u_2)$ est un langage de synchronisation et comme, par hypothèse, les langages de synchronisation sont clos par S_Σ, le mot $u_1 v u_2$ appartient à $\mathrm{f}_{R'_\Sigma}(u_1 u u_2)$. Du Lemme de Simplification 3.31, nous déduisons que v appartient à $\mathrm{f}_{R'_\Sigma}(u)$.

□

Du Lemme 3.35, nous déduisons la proposition :

Proposition 3.37 *Pour tout alphabet d'actions Σ tel que $|\Sigma| > 2$, il n'existe pas de système de réécriture S_Σ, fini ou infini, vérifiant :*

1. *tout langage de synchronisation est clos par S_Σ,*
2. *tout st-langage rationnel et clos par S_Σ est un langage de synchronisation.*

Preuve. Supposons qu'il existe un système de réécriture S_Σ, pour un alphabet Σ de $n > 2$ actions, qui vérifie les deux points. Considérons trois actions distinctes de Σ, a, b et c et le langage

$$L = \mathrm{f}_{R'_\Sigma}(b_s(a_s a_t)^* c_s b_t (a_s a_t)^* c_t).$$

D'après le Lemme 3.35, L est clos par S_Σ or, nous avons montré dans le chapitre précédent (contre-exemple 2.26) que L n'est pas un langage de synchronisation, d'où contradiction. □

Ce résultat met fin à la recherche d'une caractérisation des langages de synchronisation en termes de langages clos par un système de réécriture. Le système R', même s'il n'apporte qu'une réponse partielle, est le mieux adapté à l'étude de cette famille de langages. En outre, le résultat obtenu dans le cas des langages finis (qui n'était pas résolu avec le système R initial), nous donne un algorithme qui permet de reconstruire, à partir d'un langage fini, la plus petite expression de synchronisation (au sens de l'inclusion) qu'il respecte. Ce résultat est assez loin du cas général cependant, dans l'optique d'une utilisation effective des expressions de synchronisation pour la mise au point d'applications distribuées, il est tout à fait suffisant.

Chapitre 4

Langages de synchronisation généralisés

Nous avons montré précédemment que le système R ne permet pas de caractériser les langages de synchronisation dans le cas général. Cette constatation nous a conduits à chercher un autre système de réécriture susceptible de convenir. Nous venons de voir, dans le chapitre précédent que les langages de synchronisation ne peuvent être caractérisés en termes de langages clos par un système de réécriture. Parallèlement à cela, Salomaa et Yu ont poursuivi l'étude des langages de synchronisation et proposé, dans [57], d'étendre la définition des expressions de synchronisation augmentant ainsi leur pouvoir d'expression tout en conservant des langages associés réguliers. Pour éviter toute confusion nous nommerons ces expressions étendues « expressions de synchronisation généralisées » et les langages qui leur seront associés « langages de synchronisation généralisés ». Nous allons dans un premier temps présenter ces nouvelles expressions, les résultats de Salomaa et Yu et quelques remarques puis, nous nous attarderons sur la comparaison entre les expressions de synchronisation et les langages qui en découlent et la version généralisée.

4.1 Extension des langages et expressions de synchronisation

4.1.1 Expressions et langages de synchronisation généralisés

Salomaa et Yu ont étendu la définition des expressions de synchronisation en autorisant les deux opérandes d'un opérateur $\|$ à partager une même action, dans ce cas, les deux symboles correspondent à deux occurrences différentes d'un même traitement. Cette nouvelle définition autorise les expressions telles que $(a \parallel b) \parallel a$, cependant quelques restrictions sur la construction du langage vont être nécessaires afin de garder une certaine cohérence sémantique. L'auto-concurrence n'étant toujours pas admise, une telle expression est équivalente à $(a \longrightarrow a) \parallel b$. Cette modification de la définition peut paraître limitée cependant, nous verrons qu'elle modifie de façon significative l'ensemble des comportements qui peuvent être décrits.

Les expressions de synchronisation généralisées ont donc la même définition que les expressions de synchronisation excepté pour l'opérateur de concurrence $\|$, pour lequel il n'y a plus de restriction sur les alphabets des opérandes.

Définition 4.1 *Soit Σ un alphabet d'actions (ou étiquettes). L'ensemble des expressions de*

synchronisation généralisées sur l'alphabet Σ, *noté* $\mathrm{ES_G}(\Sigma)$, *est le plus petit sous-ensemble de* $(\Sigma \cup \{\rightarrow, \&, |, \|, *, (,)\})^*$ *tel que :*

- *l'action nulle (notée* ε*) et toutes les actions de* Σ *appartiennent à* $\mathrm{ES_G}(\Sigma)$ *;*
- *pour toutes expressions* e_1 *et* e_2 *de* $\mathrm{ES_G}(\Sigma)$, *les expressions* $(e_1 \rightarrow e_2)$, $(e_1 \mid e_2)$, $(e_1 \& e_2)$, $(e_1 \parallel e_2)$ *et* (e_1^*) *appartiennent à* $\mathrm{ES_G}(\Sigma)$.

Comme dans le cas des expressions de synchronisation, des langages sont associés aux expressions de synchronisation généralisées: les langages de synchronisation généralisés. En raison de la nouvelle définition de l'opérateur $\|$, le produit de shuffle ne peut plus lui être associé. En effet, si l'on considère simplement l'expression $a \parallel a$, le langage associé à cette expression est construit à partir des langages $a_s a_t$ et $a_s a_t$, l'utilisation du produit de shuffle conduirait au langage $a_s a_t \sqcup\!\sqcup a_s a_t$ qui n'est pas un st-langage. C'est pourquoi Salomaa et Yu ont introduit une nouvelle opération, le st-shuffle qui est une restriction du shuffle classique. Le st-shuffle de deux mots est simplement l'ensemble des st-mots de leur shuffle (une méthode générale pour définir différents types de shuffle a récemment été introduite dans [42]).

Définition 4.2 *Soient* L_1 *et* L_2 *deux st-langages sur l'alphabet d'actions* Σ. *Le st-shuffle de* L_1 *et* L_2 *est noté* $L_1 \sqcup\!\sqcup_{st} L_2$ *et est défini par :*

$$L_1 \sqcup\!\sqcup_{st} L_2 = (L_1 \sqcup\!\sqcup L_2) \cap \mathrm{ST}_\Sigma.$$

Les langages de synchronisation généralisés ont une définition tout à fait similaire à celle des langages de synchronisation, excepté pour le produit de shuffle qui est remplacé par le st-shuffle.

Définition 4.3 *Soit* Σ *un alphabet d'actions. Le langage de synchronisation généralisé* $L(e) \subseteq (\Sigma_s \cup \Sigma_t)^*$ *associé à l'expression de synchronisation généralisée* $e \in \mathrm{ES_G}(\Sigma)$ *est défini inductivement par :*

- $L(\varepsilon) = \varepsilon$ *;*
- $\forall a \in \Sigma,\ L(a) = a_s a_t$ *;*
- *si* $e = e_1 \rightarrow e_2$ *alors* $L(e) = L(e_1).L(e_2)$ *;*
- *si* $e = e_1 \mid e_2$ *alors* $L(e) = L(e_1) \cup L(e_2)$ *;*
- *si* $e = e_1 \& e_2$ *alors* $L(e) = L(e_1) \cap L(e_2)$ *;*
- *si* $e = e_1 \parallel e_2$ *alors* $L(e) = L(e_1) \sqcup\!\sqcup_{st} L(e_2)$ *;*
- *si* $e = e_1^*$ *alors* $L(e) = (L(e_1))^*$.

Clairement, les langages de synchronisation généralisés sont réguliers. Nous pouvons également remarquer que si une expression de synchronisation généralisée est construite de telle sorte que les opérandes de tous les opérateurs $\|$ sont définis sur des alphabets disjoints alors c'est une expression de synchronisation. Dans ce cas, le langage de synchronisation qui lui est associé est égal au langage de synchronisation généralisé qui lui est associé. De fait, tout langage de synchronisation est un langage de synchronisation généralisé.

4.1.2 Langages de synchronisation généralisés et systèmes de réécriture

La question de la décidabilité de l'appartenance d'un langage à la famille des langages de synchronisation se pose également dans le cas des langages de synchronisation généralisés.

L'utilisation du nouvel opérateur de concurrence augmente le pouvoir d'expression des expressions de synchronisation et précise considérablement les comportements décrits. Considérons par exemple le mot

$$u = a_s b_s a_t a_s b_t b_s a_t b_t.$$

Il est aisé de se convaincre que le plus petit langage de synchronisation contenant le mot u est associé à l'expression $(a \rightarrow a) \parallel (b \rightarrow b)$. En effet, nous avons montré que l'intersection n'est pas nécessaire lorsque l'on considère un alphabet de deux actions, le mot u est st-primitif éliminant ainsi l'emploi de la séquence, il est fini donc ne nécessite pas l'emploi de l'étoile et les deux opérandes de l'opérateur de concurrence doivent êtres définies sur des alphabets disjoints. Le plus petit langage de synchronisation contenant u est donc le langage qui décrit la mise en parallèle de deux actions a et de deux actions b c'est-à-dire le plus grand langage de synchronisation dont les mots contiennent exactement deux a et deux b. La description obtenue par une expression de synchronisation généralisée est beaucoup plus précise : il est possible d'exprimer le fait que la deuxième action a peut être exécutée en parallèle avec les deux actions b mais que la première occurrence de l'action a ne peut être exécutée en parallèle qu'avec un seul b. Nous obtenons alors l'expression $(a \rightarrow b) \parallel (a \parallel b)$.

Dans la perspective d'une caractérisation des langages de synchronisation généralisés en termes de langages clos par un système de réécriture, cette amélioration de la qualité de la description des comportements est à prendre en considération. En effet, s'il est toujours naturel de considérer des règles de réécriture telles que la commutation de deux débuts d'actions et, en général, les règles de θ, les règles du système Ω n'ont plus aucune raison d'être puisqu'elles présupposent un certain degré minimal de parallélisme. Par exemple l'emploi d'une règle

$$a_t a_s b_t b_s \longrightarrow b_t b_s a_t a_s$$

permet d'obtenir à partir de notre mot u un mot contenant le même degré de parallélisme,

$$v = a_s b_s b_t b_s a_t a_s a_t b_t$$

mais dans lequel nous avons à présent la première occurrence de a en parallèle avec deux actions b et la deuxième occurrence de l'actions a en parallèle avec un seul b. Dans le cas des langages de synchronisation, il est naturel et même heureux d'obtenir le mot v puisque ce mot appartient au plus petit langage de synchronisation contenant u, on peut remarquer que le plus petit langage de synchronisation contenant v est également le langage associé à l'expression $(a \rightarrow a) \parallel (b \rightarrow b)$. Par contre, dans le cas des langages de synchronisation généralisés, le mot v n'appartient pas au langage associé à $(a \rightarrow b) \parallel (a \parallel b)$ et le plus petit langage de synchronisation généralisé contenant le mot v est associé à l'expression $(b \rightarrow a) \parallel (a \parallel b)$. Il n'est donc pas souhaitable de considérer un système de réécriture permettant d'obtenir le mot v à partir du mot u.

Salomaa et Yu ont étudié la possibilité d'une caractérisation des langages de synchronisation généralisés par θ. Pour rappel, nous ne considérons par la semi-commutation θ que des commutations de deux débuts d'actions, de deux fins d'actions ou d'un début et d'une fin d'actions dans cet ordre. Salomaa et Yu ont montré le lemme suivant :

Lemme 4.4 (Salomaa et Yu [57]) *Soit Σ un alphabet d'actions. Tout langage de synchronisation généralisé défini sur $\Sigma_s \cup \Sigma_t$ est clos par la semi-commutation θ_Σ.*

Ils conjecturent que la semi-commutation θ permet de caractériser les langages de synchronisation généralisés:

Conjecture III (Salomaa et Yu [57]) *Soit Σ un alphabet d'actions. Tout st-langage régulier défini sur $\Sigma_s \cup \Sigma_t$ et clos par θ_Σ est un langage de synchronisation généralisé.*

Une telle caractérisation serait fort utile puisqu'il est possible de vérifier de façon automatique qu'un langage régulier donné est clos par une semi-commutation en utilisant la propriété de semi-diamant de l'automate minimal du langage (Proposition 1.32). Salomaa et Yu montrent cette conjecture dans le cas particulier des langages finis.

Théorème 4.5 (Salomaa et Yu [57]) *Soit Σ un alphabet d'actions. Un st-langage fini défini sur Σ est un langage de synchronisation généralisé si et seulement s'il est clos par θ_Σ.*

Nous allons voir, dans la sous-section suivante, quelques remarques.

4.1.3 Remarques

Cette extension des expressions et langages de synchronisation permet de résoudre certains problèmes que nous avions rencontrés avec les premières expressions. Reprenons par exemple le langage $\mathbb{L} = b_s(a_s a_t)^* c_s b_t (a_s a_t)^* c_t$ donné dans le Contre-exemple 2.26. La clôture par R de ce langage n'est pas un langage de synchronisation, nous avions remarqué qu'il est impossible, dans une expression de synchronisation, de distinguer les actions a qui se déroulent respectivement durant l'action b ou l'action c. Il se trouve que la clôture par R de ce langage et sa clôture par θ sont égales (nous avions vu qu'il n'était jamais possible d'appliquer une règle de Ω à un mot de $\mathrm{f_R}(\mathbb{L})$). En fait, $\mathrm{f_R}(\mathbb{L}) = \mathrm{f}_\theta(\mathbb{L})$ est un langage de synchronisation généralisé correspondant à l'expression

$$(b \longrightarrow a^*) \parallel (a^* \longrightarrow c).$$

La nouvelle définition de l'opérateur $\parallel$ nous permet de différencier les deux groupes de a en fonction de leur rapport à b et c, ce qui n'était pas possible avec l'ancienne définition. Il semble donc que les langages de synchronisation généralisés, parce qu'ils permettent d'exprimer des comportements plus précis, évitent certaines faiblesses des langages de synchronisation que nous avions constatées.

Avant de nous lancer dans la comparaison des langages de synchronisation et des langages de synchronisation généralisés qui fera l'objet de la fin de ce chapitre, nous allons faire quelques remarques. Salomaa et Yu, dans [57], énoncent quelques propriétés de la semi-commutation θ qui sont essentielles dans l'étude des langages clos par θ. En utilisant les résultats des chapitres précédents ainsi que des propriétés générales des semi-commutations, nous allons voir qu'il est possible de donner d'autres preuves, souvent plus courtes, de ces résultats.

Nous avons donné, au Chapitre 1, une preuve du Théorème: *la clôture par θ d'un st-langage régulier est régulière* utilisant une propriété de l'alphabet des facteurs itérants d'un st-langage et le Théorème 1.30. La preuve originale donnée dans [57] est une preuve par induction sur la construction d'une expression rationnelle définissant le langage considéré. Cette propriété de θ est très importante puisque nous considérons des familles de langages réguliers, nous évitons ainsi des problèmes qui survenaient dans l'étude des langages clos par R. Par exemple le st-langage régulier $a_s(b_s a_t a_s b_t)^* a_t$ est un langage dont la clôture par R n'est

pas régulière, par contre, sa clôture par θ est régulière, il est donc potentiellement générateur d'un langage de synchronisation généralisé (en fait nous verrons que sa clôture correspond à l'expression $((a \rightarrow b)^* \rightarrow a) \parallel (a \parallel b)^*)$. De fait, si la conjecture III s'avère exacte, tout st-langage régulier est générateur d'un langage de synchronisation généralisé.

D'un point de vue plus technique, la propriété suivante est également importante, elle est utilisée dans de nombreuses preuves par récurrence sur la construction rationnelle d'un langage. Il s'agit en fait d'une extension d'un lemme de Salomaa et Yu montrant que la clôture de la concaténation de deux st-mots est égale à la concaténation de leur clôtures.

Lemme 4.6 *Soient L_1 et L_2 des st-langages. Nous avons:*

$$\begin{aligned} \mathrm{f}_\theta(L_1 + L_2) &= \mathrm{f}_\theta(L_1) + \mathrm{f}_\theta(L_2), \\ \mathrm{f}_\theta(L_1.L_2) &= \mathrm{f}_\theta(L_1).\mathrm{f}_\theta(L_2), \\ \mathrm{f}_\theta(L_1^*) &= (\mathrm{f}_\theta(L_1))^*. \end{aligned}$$

Preuve. La preuve de ce Lemme se déduit immédiatement de la preuve du Lemme 1.37. □

La dernière propriété technique est un Lemme de Projection. L'extension du système R avait été suscitée par la remarque qu'un tel lemme lui faisait défaut. Ici, le lemme de projection sur les sous-alphabets de deux actions est une conséquence directe du Lemme de Projection 1.19 pour les semi-commutations.

Lemme 4.7 (Salomaa et Yu [57]) *Soient u et v deux mots de $(\Sigma_s \cup \Sigma_t)^*$. Nous avons:*

$$(u \xrightarrow[\theta]{*} v) \Leftrightarrow (\forall a, b \in \Sigma, \Pi_{ab}(u) \xrightarrow[\theta]{*} \Pi_{ab}(v)).$$

Preuve. Pour l'implication de gauche à droite, d'après le Lemme de Projection pour les semi-commutations, pour toute paire de lettres x et y de $(\Sigma_s \cup \Sigma_t)$, $\Pi_{\{x,y\}}(v)$ appartient à $\mathrm{f}_\theta(\Pi_{\{x,y\}}(u))$. En appliquant à nouveau le Lemme de Projection, nous obtenons, pour toute paire d'actions a et b de Σ, $\Pi_{ab}(v)$ appartient à $\mathrm{f}_\theta(\Pi_{ab}(u))$.

Pour l'implication de droite à gauche, il suffit de projeter à nouveau les $\Pi_{ab}(u)$ et $\Pi_{ab}(v)$ pour toute paire d'actions a et b, le Lemme de Projection pour les semi-commutations implique que pour toute paire de lettres x et y de $(\Sigma_s \cup \Sigma_t)$, $\Pi_{\{x,y\}}(v)$ appartient à $\mathrm{f}_\theta(\Pi_{\{x,y\}}(u))$. On applique à nouveau le Lemme de Projection pour obtenir $v \in \mathrm{f}_\theta(u)$. □

Revenons à présent à la conjecture III. Salomma et Yu montrent cette conjecture dans le cas des langages finis en utilisant une propriété technique de la décomposition d'un mot en produit de shuffle qui conduit à une preuve assez longue et technique. Nous proposons ici une preuve de ce résultat plus courte de laquelle nous pouvons extraire deux lemmes préliminaires. Nous allons construire une expression de synchronisation généralisée associée à la clôture par θ d'un mot en utilisant la méthode proposée au chapitre précédent pour calculer une expression de synchronisation associée à la clôture d'un mot par R'. Pour cela, nous utiliserons une numérotation des actions pour nous ramener à des mots ne contenant qu'une occurrence de chaque action.

Le premier lemme permet, sous certaines conditions, de calculer le langage associé à l'image par un morphisme strictement alphabétique d'une expression de synchronisation en appliquant un morphisme similaire au langage associé à l'expression. Ce que nous entendons par image par un morphisme strictement alphabétique h d'une expression de synchronisation e et que

nous noterons abusivement $h(e)$ est l'expression obtenue en appliquant le morphisme h à toutes les actions de e et en conservant la structure de e.

Lemme 4.8 *Soient h un morphisme strictement alphabétique de Σ^* dans X^* étendu de façon naturelle de $(\Sigma_s \cup \Sigma_t)^*$ dans $(\mathrm{X}_s \cup \mathrm{X}_t)^*$ et e une expression de synchronisation généralisée sur Σ. Si pour toute sous-expression $e_1 \& e_2$ de e, $L(h(e_1 \& e_2)) = L(h(e_1)) \& L(h(e_2))$ alors nous avons :*

$$L(h(e)) = h(L(e)) \cap \mathrm{ST_X}.$$

Preuve. Ce lemme se montre par induction sur la construction de l'expression e. Si $e = a \in \Sigma$, le lemme est clairement vérifié. Sinon, détaillons le cas $e = e_1 \parallel e_2$, les autres cas se traitant de la même manière. Nous avons :

$$L(h(e_1 \parallel e_2)) = L(h(e_1) \parallel h(e_2)) = [L(h(e_1)) \sqcup L(h(e_2))] \cap \mathrm{ST_X}.$$

Par hypothèse de récurrence, nous avons :

$$[L(h(e_1)) \sqcup L(h(e_2))] \cap \mathrm{ST_X} = [(h(L(e_1)) \cap \mathrm{ST_X}) \sqcup (h(L(e_2)) \cap \mathrm{ST_X})] \cap \mathrm{ST_X}.$$

Dans tout mot de $h(L(e_1))$ (ou $h(L(e_2))$) tout facteur gauche contient, pour toute action x au moins autant de x_s que de x_t. De cette remarque nous pouvons déduire que si l'on considère le shuffle de deux mots appartenant respectivement à $h(L(e_1))$ et $h(L(e_2))$ et si l'un au moins de ces deux mots n'est pas un st-mot, alors leur shuffle ne contient aucun st-mot. C'est pourquoi nous obtenons :

$$\begin{aligned}
&[(h(L(e_1)) \cap \mathrm{ST_X}) \sqcup (h(L(e_2)) \cap \mathrm{ST_X})] \cap \mathrm{ST_X} \\
&\quad = [(h(L(e_1)) \sqcup h(L(e_2))) \cap \mathrm{ST_X}] \cap \mathrm{ST_X} \\
&\quad = h((L(e_1) \sqcup L(e_2)) \cap \mathrm{ST_X}) \cap \mathrm{ST_X} \\
&\quad = h(L(e_1 \parallel e_2)) \cap \mathrm{ST_X}.
\end{aligned}$$

□

Nous pouvons également remarquer que, dans le cas des mots ne contenant qu'une occurrence de chaque action, il y a égalité entre les clôtures par θ ou par R'.

Lemme 4.9 *Soit u un st-mot sur l'alphabet d'action Σ ne contenant qu'une occurrence de chaque action. Nous avons :*

$$\mathrm{f}_\theta(u) = \mathrm{f}_{\mathrm{R}'}(u).$$

Preuve. Nous avons les équivalences suivantes :

$$(u \xrightarrow[\theta]{*} v) \Leftrightarrow (\forall a, b \in \Sigma, \Pi_{ab}(u) \xrightarrow[\theta]{*} \Pi_{ab}(v)) \tag{4.1}$$

$$\Leftrightarrow (\forall a, b \in \Sigma, \Pi_{ab}(u) \xrightarrow[R]{*} \Pi_{ab}(v)) \tag{4.2}$$

$$\Leftrightarrow (u \xrightarrow[R']{} v) \tag{4.3}$$

L'équivalence 4.1 est donnée par le Lemme de projection sur les sous-alphabets de deux actions 4.7. Comme u ne contient qu'une occurrence de chaque action, il n'est jamais possible d'appliquer une règle de Ω à une projection sur deux actions de u, ce qui nous conduit à l'équivalence 4.2. Enfin, l'équivalence 4.3 est donnée par la définition de R'. □

En utilisant ces deux lemmes, nous pouvons à présent donner une preuve très simple du théorème de Salomaa et Yu :

Preuve du Théorème 4.5. Clairement, il suffit de montrer le résultat pour la clôture d'un mot. Considérons un st-mot u sur l'alphabet d'actions Σ et sa clôture $\mathrm{f}_\theta(u)$. D'après le Lemme de numérotation 1.22, nous avons:

$$\mathrm{f}_\theta(u) = \mathrm{denum}_{\Sigma_s \cup \Sigma_t}(\mathrm{f}_{\theta_{\mathrm{num}_{\Sigma_s \cup \Sigma_t}}}(\mathrm{num}_{\Sigma_s \cup \Sigma_t}(u))).$$

Comme $\mathrm{num}_{\Sigma_s \cup \Sigma_t}(u)$ ne contient qu'une occurrence de chaque action, nous pouvons appliquer le Lemme 4.9 et, si nous notons $e(\mathrm{num}_\Sigma(u))$ l'expression de synchronisation associée à $\mathrm{f}_{\theta_{\mathrm{num}_{\Sigma_s \cup \Sigma_t}}}(\mathrm{num}_{\Sigma_s \cup \Sigma_t}(u))$ par la méthode de la Proposition 3.33, nous avons:

$$\mathrm{denum}_{\Sigma_s \cup \Sigma_t}(\mathrm{f}_{\theta_{\mathrm{num}_{\Sigma_s \cup \Sigma_t}}}(\mathrm{num}_{\Sigma_s \cup \Sigma_t}(u))) = \mathrm{denum}_{\Sigma_s \cup \Sigma_t}(L(e(\mathrm{num}_{\Sigma_s \cup \Sigma_t}(u)))).$$

Comme $\mathrm{f}_\theta(u)$ est un st-langage,

$$\mathrm{denum}_{\Sigma_s \cup \Sigma_t}(L(e(\mathrm{num}_{\Sigma_s \cup \Sigma_t}(u)))) = \mathrm{denum}_{\Sigma_s \cup \Sigma_t}(L(e(\mathrm{num}_{\Sigma_s \cup \Sigma_t}(u)))) \cap \mathrm{ST}_\Sigma$$

et, comme denum_Σ et $e(\mathrm{num}_{\Sigma_s \cup \Sigma_t}(u))$ vérifient les hypothèses (sur h et e) du Lemme 4.8, nous obtenons:

$$\mathrm{denum}_{\Sigma_s \cup \Sigma_t}(L(e(\mathrm{num}_{\Sigma_s \cup \Sigma_t}(u)))) \cap \mathrm{ST}_\Sigma = L(\mathrm{denum}_\Sigma(e(\mathrm{num}_\Sigma(u)))).$$

□

Suite au résultat sur les langages finis, Salomaa et Yu proposent une idée de construction pour apporter une réponse partielle à cette conjecture dans le cas des langages qui sont la clôture par θ de langages bien formés. En effet, dans le cas général, il semble assez difficile de dégager une méthode de construction d'une expression et même, la plupart du temps, de trouver une expression associée à un langage donné « à la main ». Dans [57] est proposé l'exemple du langage

$$\mathrm{f}_\theta(a_s(b_s a_t a_s b_t)^* a_t),$$

ce langage correspond à l'expression

$$((a \longrightarrow b)^* \longrightarrow a) \parallel (a \parallel b)^*,$$

qui, apparemment, ne présente pas de ressemblance structurelle avec le langage de départ. Salomaa et Yu se restreignent donc au cas particulier des langages qui sont la clôture d'un langage bien formé, et pensent que, dans ce cas, la conjecture peut être montrée en utilisant une méthode similaire à celle qu'ils ont utilisée pour les langages finis.

Par la méthode proposée, le but est de construire inductivement, pour une expression rationnelle bien formée γ, une expression de synchronisation généralisée e_γ correspondant à la clôture par θ du langage L défini par γ. Lorsque l'on s'intéresse à une expression γ, la concaténation de st-langages, l'étoile d'un st-langage ou l'union ne posent aucun problème particulier. Le seul cas non trivial est le cas où tout mot de L est un mot st-primitif et où $\gamma = \beta_1 a_s \beta_2 a_t$ avec β_1 et β_2 des concaténations de lettres et d'étoiles d'expressions. Une paire d'occurrences (c_s, c_t) est un diviseur de γ si c_s apparaît dans β_1 et l'occurrence de c_t correspondante dans β_2 — en dehors des étoiles puisque γ est bien formée. On note $c_1, \ldots, c_k$

les occurrences d'actions appartenant à des diviseurs et β'_1 et β'_2 les expressions obtenues respectivement à partir de β_1 et β_2 en effaçant toutes les occurrences de lettres appartenant à des diviseurs. Salomaa et Yu conjecturent que :

$$e_\gamma = (e_{\beta_1.\beta_2} \parallel a)\&[(e_{\beta'_1} \longrightarrow (a \parallel e_{\beta'_2})) \parallel c_1 \parallel \cdots \parallel c_k].$$

Cette idée de construction, si elle est intuitive, ne convient pas aux langages bien formés. Considérons $\gamma = b_s(c_sc_t)^*a_sb_t(c_sc_t)^*a_t$. En utilisant la notation décrite ci-dessus, nous avons $\beta_1 = b_s(c_sc_t)^*$ et $\beta_2 = b_t(c_sc_t)^*$. Comme (b_s, b_t) est le seul diviseur de γ, nous obtenons $\beta'_1 = (c_sc_t)^*$ et $\beta'_2 = (c_sc_t)^*$. Nous avons $(e_{\beta'_1} \longrightarrow (a \parallel e_{\beta'_2})) = a \parallel c^*$, de plus, l'expression $e_{\beta_1.\beta_2}$ est l'expression associée au langage $b_s(c_sc_t)^*b_t(c_sc_t)^*$ soit $(b \parallel c^*)$, nous aurions donc

$$e_\gamma = ((b \parallel c^*) \parallel a)\&[c^* \parallel a \parallel b].$$

Cette expression ne peut convenir puisqu'elle décrit le langage contenant tous les mots dans lesquels apparaissent une occurrence de a, une occurrence de b et un nombre arbitraire de c.

Nous pensons que la conjecture III est fausse et qu'il existe des st-langages réguliers clos par θ qui ne sont pas des langages de synchronisation généralisés. En particulier, nous conjecturons que le langage

$$\mathrm{f}_\theta(a_sb_s(b_tb_s(a_ta_s)^+b_tb_s)^*a_tb_t)$$

est un contre-exemple cependant, nous n'avons pour le moment ni d'expression de synchronisation généralisée associée à ce langage ni de preuve qu'il n'en existe pas. Le problème présent dans ce langage peut être rapproché de celui du langage du Contre-exemple 2.26. Ici, il nous faut différencier les actions b qui peuvent se dérouler en parallèle avec trois actions a des actions b qui ne peuvent se dérouler en parallèle qu'avec une seule action a en assurant une alternance d'actions b de chaque type. Dans la mesure où le nombre d'actions b est potentiellement illimité, il n'est pas possible de nommer chaque occurrence explicitement comme lorsque le nombre de b est fini. Par exemple, si on considère simplement

$$\mathrm{f}_\theta(a_sb_sb_tb_s(a_ta_s)^+b_tb_sa_tb_t),$$

nous obtenons l'expression

$$((a \parallel b) \rightarrow a^* \rightarrow (a \parallel b)) \parallel b,$$

comme le nombre de b est fini, nous pouvons les positionner correctement dans l'expression.

Au delà de la caractérisation des langages de synchronisation généralisés, il semble intéressant d'étudier les rapports entre les familles des langages de synchronisation et des langages de synchronisation généralisés d'une part et entre les familles des st-langages réguliers clos par θ et par R' d'autre part. On peut remarquer en effet que la preuve que nous donnons pour la conjecture III dans le cas des langages finis, est basée sur la construction d'une expression de synchronisation — non généralisée — à laquelle on se ramène par un morphisme. D'autre part, nous avions remarqué, dans le Contre-exemple 2.26, que le langage $\mathrm{f}_\theta(b_s(a_sa_t)^*c_sb_t(a_sa_t)^*)$, s'il n'est pas un langage de synchronisation, est l'image par un morphisme strictement alphabétique d'un langage de synchronisation. En utilisant la même idée, nous pouvons remarquer que les langages de synchronisation et les langages de synchronisation généralisés sont relativement proches. Intuitivement il suffirait de renommer les symboles d'actions apparaissant dans les deux opérandes d'un opérateur $\parallel$ dans une expression de synchronisation généralisée

pour obtenir une expression de synchronisation. De la même manière, la différence entre R' et θ vient essentiellement de la seconde partie de R, le système Ω. Si l'on « empêche » par un renommage approprié l'application de règles de Ω, on peut espérer obtenir une équivalence entre la clôture par θ et par R'. Dans la section suivante, nous allons établir le lien entre les familles de langages clos par θ ou par R'.

4.2 Lien entre les familles de langages clos par θ et par R'

Pour étudier et comparer les différentes familles de langages qui nous intéressent, nous allons utiliser une classe particulière de fonctions rationnelles qui, d'un certain point de vue, sont très proches des morphismes strictement alphabétiques. En effet, nous allons utiliser des renommages ou des numérotations d'actions c'est-à-dire que l'image d'une action sera toujours une unique action. Cependant, les morphismes devant être appliqués à des st-langages, il est nécessaire de prendre quelques précautions pour s'assurer que le début et la fin d'une même action auront bien pour images le début et la fin d'une même action. Pour cela, nous allons considérer les morphismes d'actions, famille de morphismes strictement alphabétiques.

Définition 4.10 *Soient* Σ *et* X *deux alphabets d'actions. Un morphisme strictement alphabétique de* Σ^* *dans* X^* *est appelé morphisme d'actions. Un morphisme d'actions* φ *de* Σ^* *dans* X^* *est étendu de façon naturelle pour obtenir* φ *de* $(\Sigma_s \cup \Sigma_t)^*$ *dans* $(X_s \cup X_t)^*$.

Cette famille de morphismes strictement alphabétiques nous permet d'obtenir des mots relativement cohérents, par exemple, l'image d'un mot $a_s a_t b_s b_t$ sera toujours une séquence de deux actions, quels que soient leurs noms, et jamais un mot du type $a_s c_t c_s b_t$. Cependant, cette restriction de la classe des morphismes strictement alphabétiques n'est pas encore suffisante pour obtenir des st-langages comme images des st-langages. Considérons le langage

$$L = \{a_s a_t, a_s b_s a_t b_t\}$$

et le morphisme d'actions

$$\begin{array}{rccc} \varphi : & \{a,b\}^* & \longrightarrow & \{a\}^* \\ & a & \longmapsto & a \\ & b & \longmapsto & a. \end{array}$$

Le langage $\varphi(L) = \{a_s a_t, a_s a_s a_t a_t\}$ n'est pas un st-langage, φ ne paraît pas « adapté » au langage L. Pour obtenir un st-langage, il faudrait vérifier, avant d'appliquer φ, qu'aucun mot de L ne contient, en parallèle, d'actions dont les images par φ sont égales. Nous avons choisi une autre solution, à savoir calculer l'image d'un langage donné par le morphisme choisi, qu'il soit bien adapté au langage en question ou non, et ne conserver ensuite que l'ensemble des st-mots obtenus ce qui nous conduit à la définition suivante :

Définition 4.11 *À tout morphisme d'actions* φ *de* Σ^* *dans* X^*, *nous associons une fonction rationnelle* $\hat{\varphi}$ *appelée st-morphisme :*

$$\hat{\varphi} = \{(u, \varphi(u)) \mid u \in \mathrm{ST}_\Sigma \text{ et } \varphi(u) \in \mathrm{ST}_X\}.$$

Notons que $\hat{\varphi}$ *est égal à* $(\cap \mathrm{ST}_X) \circ \varphi \circ (\cap \mathrm{ST}_\Sigma)$. *Nous notons* Φ_{st} *la famille des st-morphismes.*

Nous utiliserons aussi bien les st-morphismes que leurs inverses, les st-morphismes inverses. Afin d'éviter toute confusion sur la signification exacte de l'inversion d'un st-morphisme, nous allons en fixer la définition.

Définition 4.12 *Soit $\hat{\varphi}$ un st-morphisme de Σ^* dans X^*. Le st-morphisme inverse $\hat{\varphi}^{-1}$ est défini par :*

$$\hat{\varphi}^{-1} = \{(u, v) \mid u \in \mathrm{ST}_X \text{ et } v \in \varphi^{-1}(u)\}.$$

Notons que $\hat{\varphi}^{-1}$ est égal à $\varphi^{-1} \circ (\cap \mathrm{ST}_X)$. Nous notons Φ_{st}^{-1} la famille des st-morphismes inverses.

Nous pouvons dès à présent remarquer quelques propriétés des st-morphismes. Tout d'abord pour la composition :

Lemme 4.13 *La composition de deux st-morphismes est un st-morphisme, la composition de deux st-morphismes inverses est un st-morphisme inverse.*

Preuve. Soient $\hat{\varphi}$ et $\hat{\psi}$ des st-morphismes respectivement associés aux morphismes d'actions φ de X^* dans Υ^* et ψ de Υ^* dans Σ^*. Nous avons :

$$\hat{\varphi} = \{(u, \varphi(u)) \mid u \in \mathrm{ST}_X \text{ et } \varphi(u) \in \mathrm{ST}_\Upsilon\},$$
$$\hat{\psi} = \{(u, \psi(u)) \mid u \in \mathrm{ST}_\Upsilon \text{ et } \psi(u) \in \mathrm{ST}_\Sigma\}.$$

La composition de ces deux st-morphismes est :

$$\hat{\psi} \circ \hat{\varphi} = \{(u, \psi \circ \varphi(u)) \mid u \in \mathrm{ST}_X, \varphi(u) \in \mathrm{ST}_\Upsilon \text{ et } \psi \circ \varphi(u) \in \mathrm{ST}_\Sigma\}.$$

Comme l'appartenance du mot $\psi \circ \varphi(u)$ à ST_Σ implique que $\varphi(u)$ appartient à ST_Υ, $\hat{\psi} \circ \hat{\varphi}$ est un st-morphisme. La composition de leurs inverses respectifs est :

$$\begin{aligned}
\hat{\varphi}^{-1} \circ \hat{\psi}^{-1} &= \{(u, v) \mid u \in \mathrm{ST}_\Sigma, w \in \psi^{-1}(u), w \in \mathrm{ST}_\Upsilon \text{ et } v \in \varphi^{-1}(w)\} \\
&= \{(u, v) \mid u \in \mathrm{ST}_\Sigma, v \in \varphi^{-1} \circ \psi^{-1}(u)\} \\
&= \{(u, v) \mid u \in \mathrm{ST}_\Sigma, v \in (\varphi \circ \psi)^{-1}(u)\} \\
&= (\hat{\varphi} \circ \hat{\psi})^{-1}
\end{aligned}$$

Comme la composition de deux st-morphismes est un st-morphisme, $\hat{\varphi} \circ \hat{\psi}$ est un st-morphisme et donc $\hat{\varphi}^{-1} \circ \hat{\psi}^{-1}$ est un st-morphisme inverse. □

D'autre part, il est toujours possible de remplacer la composition d'un st-morphisme inverse et d'un st-morphisme par celle d'un st-morphisme et d'un st-morphisme inverse :

Lemme 4.14 *Nous avons :*

$$\Phi_{st}^{-1} \circ \Phi_{st} \subseteq \Phi_{st} \circ \Phi_{st}^{-1}.$$

Preuve. Soient $\hat{\varphi}_1$ et $\hat{\varphi}_2$ deux st-morphismes avec respectivement φ_1 de Σ^* dans Y^* et φ_2 de X^* dans Y^*. Considérons le nouvel alphabet d'actions :

$$\Upsilon = \{(a, \alpha) \mid a \in \Sigma \text{ et } \alpha \in \varphi_2^{-1}(\varphi_1(a)\}$$

et les deux morphismes d'actions :

$$\begin{array}{rccc}
\psi_1 : & \Upsilon^* & \longrightarrow & \Sigma^* \\
 & (a, \alpha) & \longmapsto & a
\end{array}
\qquad
\begin{array}{rccc}
\psi_2 : & \Upsilon^* & \longrightarrow & X^* \\
 & (a, \alpha) & \longmapsto & \alpha
\end{array}$$

Clairement, nous avons $\hat{\varphi_2}^{-1} \circ \hat{\varphi_1} = \hat{\psi_2} \circ \hat{\psi_1}^{-1}$. □

Nous allons à présent revenir aux différentes familles de langages clos et utiliser les st-morphismes pour les étudier.

Lemme 4.15 *La famille des st-langages clos par θ est close par st-morphisme :*

$$\Phi_{\mathrm{st}}(\mathrm{L}_\theta) = \mathrm{L}_\theta.$$

Preuve. Clairement nous avons l'inclusion $\mathrm{L}_\theta \subseteq \Phi_{\mathrm{st}}(\mathrm{L}_\theta)$. Soit $L \subseteq (\mathrm{X}_s \cup \mathrm{X}_t)^*$ un st-langage clos par θ_{X}. Soient φ un morphisme d'actions d'un alphabet contenant X dans Σ et $\hat{\varphi}$ le st-morphisme associé. Montrons que $\hat{\varphi}(L)$ est clos par θ_Σ. Soit u appartenant à $\hat{\varphi}(L)$, il suffit de montrer que :

$$(u \in \hat{\varphi}(L) \text{ et } u \underset{\theta_\Sigma}{\longrightarrow} v) \Rightarrow (v \in \hat{\varphi}(L)).$$

Posons $u = u_1xyu_2$ et $v = u_1yxu_2$ avec (x, y) appartenant à θ_Σ. Comme u est un mot de $\hat{\varphi}(L)$, il existe un mot $w = w_1ztw_2$ de L tel que $\varphi(w_1) = u_1$, $\varphi(z) = x$, $\varphi(t) = y$ et $\varphi(w_2) = u_2$. Le couple (z, t) appartient à θ_{X} car $(\varphi(z), \varphi(t))$ appartient à θ_Σ. Comme L est clos, le mot w_1tzw_2 appartient à L donc, $\varphi(w_1tzw_2) = v$ est un mot de $\varphi(L)$ et comme θ_{X} et θ_Σ préservent la st-propriété, v appartient à $\hat{\varphi}(L)$. □

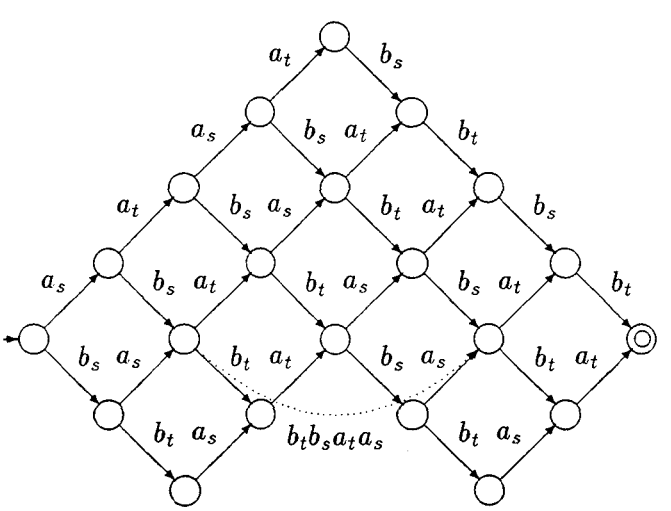
Nous déduisons immédiatement de ce lemme le corollaire :

Corollaire 4.16 *La famille des st-langages réguliers θ-clos est close par st-morphisme :*

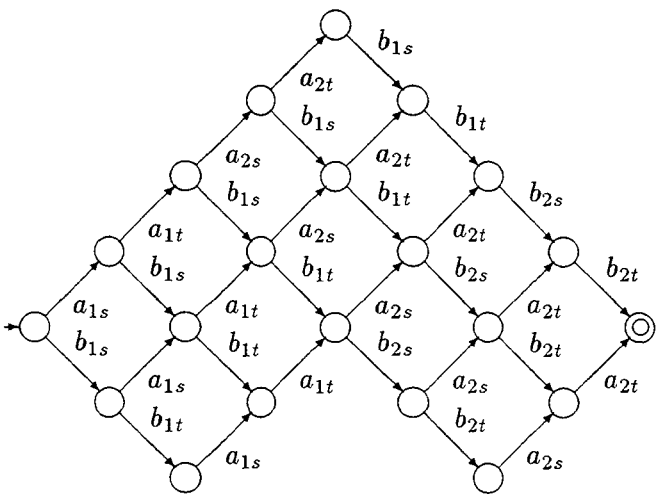
$$\Phi_{\mathrm{st}}(\mathrm{R}_\theta) = \mathrm{R}_\theta.$$

Preuve. Clairement nous avons les inclusions $\mathrm{R}_\theta \subseteq \Phi_{\mathrm{st}}(\mathrm{R}_\theta)$ et $\Phi_{\mathrm{st}}(\mathrm{R}_\theta) \subseteq \mathrm{L}_\theta$. Comme les st-morphismes préservent la régularité des langages, nous avons immédiatement $\Phi_{\mathrm{st}}(\mathrm{R}_\theta)) \subseteq \mathrm{R}_\theta$. □

Nous pouvons à présent établir la correspondance entre les familles de langages clos par θ et R', le principal résultat de cette section. La preuve de ce résultat utilise une numérotation alternative des actions et quelques propriétés techniques des systèmes de réécriture. Par exemple, il est ici nécessaire de se ramener au système fini R_e pour calculer des dérivations pas à pas. Cependant, l'idée de base reste très simple, nous allons considérer un exemple sur un alphabet de deux actions car dans ce cas R' et R sont équivalents. Considérons par exemple le langage $L = \mathrm{f}_\theta(a_sb_sa_ta_sb_tb_sa_tb_t)$ défini par l'automate suivant :



Ce langage n'est pas clos par R, il manque par exemple le chemin étiqueté par $b_t b_s a_t a_s$ indiqué par le pointillé. Cependant, si l'on numérote alternativement les occurrences d'une même action, il n'est plus possible d'appliquer une règle de Ω. Le langage $\mathrm{f}_\theta(a_{1s}b_{1s}a_{1t}a_{2s}b_{1t}b_{2s}a_{2t}b_{2t})$ donné par l'automate suivant est clos par R.



On peut remarquer que ces deux automates sont isomorphes, par la numérotation, les facteurs du type $a_t a_s$ ont été transformés en $a_{1t}a_{2s}$, il n'est donc plus possible de leur appliquer une règle de Ω. Nous avons donc construit un langage dont la clôture par R est égale à la clôture par θ et qui est en bijection avec L par le st-morphisme $\hat{\varphi}$ défini par:

$$\begin{array}{rccc} \varphi : & \{a_1, a_2, b_1, b_2\}^* & \longrightarrow & \{a, b\}^* \\ & a_1, a_2 & \longmapsto & a \\ & b_1, b_2 & \longmapsto & b. \end{array}$$

La preuve du lemme suivant utilise cette idée de numérotation dans le cas général.

Lemme 4.17 *La famille des images par st-morphismes des st-langages clos par R' et la famille des st-langages clos par θ coïncident:*

$$\Phi_{\mathrm{st}}(\mathrm{L}_{\mathrm{R}'}) = \mathrm{L}_\theta.$$

Preuve. De l'inclusion $\mathrm{L}_{\mathrm{R}'} \subseteq \mathrm{L}_\theta$ et du Lemme 4.15, nous déduisons immédiatement

$$\Phi_{\mathrm{st}}(\mathrm{L}_{\mathrm{R}'}) \subseteq \mathrm{L}_\theta.$$

Soit L un st-langage sur l'alphabet d'actions Σ clos par θ_Σ. Soient $\mathrm{X} = \bigcup_{a\in\Sigma}\{a_1, a_2\}$ un alphabet d'actions et φ le morphisme d'actions de X^* dans Σ^* tel que pour toute action a de Σ, $\varphi(a_1) = a$ et $\varphi(a_2) = a$. Nous notons $\hat{\varphi}$ le st-morphisme associé à φ. Posons

$$M = \varphi^{-1}(L) \cap \coprod_{a\in\Sigma} (a_{1s}a_{1t}a_{2s}a_{2t})^*(a_{1s}a_{1t} + \varepsilon).$$

Par construction, nous avons $L = \hat{\varphi}(M) = \varphi(M)$. Montrons que $M = \mathrm{f}_{R'_{\mathrm{X}}}(M)$. Soit u un mot de M et v un mot de $\mathrm{f}_{R'_{\mathrm{X}}}(u)$. Par définition de R'_{X}, nous avons,

$$\forall a, b \in \mathrm{X}, \Pi_{ab}(u) \xrightarrow[R_{\mathrm{X}}]{*} \Pi_{ab}(v).$$

Supposons qu'il soit possible de trouver une occurrence d'une lettre de X_t et une occurrence d'une lettre de X_s qui apparaissent dans cet ordre dans u et dans l'ordre inverse dans v. Nous avons $u = u_1 a_{it} u_2 b_{js} u_3$ avec i et j dans $\{1,2\}$, $a_{it}, b_{js} \notin \text{alph}(u_2)$ et $v = v_1 b_{js} v_2 a_{it} v_3$. Comme

$$\Pi_{a_i b_j}(u) \xrightarrow[R_X]{*} \Pi_{a_i b_j}(v),$$

nous appliquons, dans cette dérivation, une règle $a_{it}a_{is}b_{jt}b_{js} \longrightarrow b_{jt}b_{js}a_{it}a_{is}$, aux occurrences de lettres que nous considérons, donc nous pouvons déduire que $\Pi_{a_i b_j}(u_2) = a_{is}b_{jt}$. De plus, par construction,

$$\Pi_{a_1 a_2}(M) \subseteq (a_{1s}a_{1t}a_{2s}a_{2t})^*(a_{1s}a_{1t} + \varepsilon),$$

donc, nous avons $\Pi_{a_1 a_2}(u) = \Pi_{a_1 a_2}(v)$ et

$$\Pi_{a_i a_k b_j}(u) = \Pi_{a_i a_k b_j}(u_1) a_{it}a_{ks}a_{kt}a_{is}b_{jt}b_{js}\Pi_{a_i a_k b_j}(u_3),$$

avec $\{i,k\} = \{1,2\}$. À présent, nous pouvons déduire :

$$\Pi_{a_k b_j}(u) = \Pi_{a_k b_j}(u_1) a_{ks}a_{kt}b_{jt}b_{js}\Pi_{a_k b_j}(u_3).$$

Le problème est qu'un mot de cette forme ne peut donner, par R_X, un mot dans lequel les occurrences de a_{kt} et b_{js} sont dans l'ordre inverse, pourtant nous avons :

$$\Pi_{a_k b_j}(u) \xrightarrow[R_X]{*} \Pi_{a_k b_j}(v).$$

Ceci conduit à une contradiction. En conséquence, il n'est pas possible de trouver une occurrence d'une lettre de X_t et une occurrence d'une lettre de X_s qui soient dans cet ordre dans u et dans l'ordre inverse dans v.

Nous allons à présent considérer la dérivation de u par R'_X, comme v appartient à $f_{R'_\Sigma}(u)$, par définition nous avons :

$$\forall a, b \in X, \Pi_{ab}(u) \xrightarrow[R'_X]{} \Pi_{ab}(v).$$

Donc, par définition de R' d'après les projections sur des alphabets de deux actions, nous avons :

$$\forall a, b \in \Sigma, \Pi_{a_1 a_2 b_1 b_2}(u) \xrightarrow[R'_X]{} \Pi_{a_1 a_2 b_1 b_2}(v),$$

c'est-à-dire

$$g(\Pi_{a_1 a_2 b_1 b_2}(u)) \xrightarrow[R_{eX}]{*} w \in h^{-1}(\Pi_{a_1 a_2 b_1 b_2}(v)).$$

D'après le Lemme des Distances 3.22, il existe une dérivation par R_{eX} dans laquelle on n'applique jamais de règle $xy \longrightarrow yx$ avec $x \in h^{-1}(X_t)$ et $y \in h^{-1}(X_s)$ (puisque nous avons montré que de telles lettres sont dans le même ordre dans u et v). D'un autre côté, par construction, nous avons $\Pi_{a_1 a_2}(u) = \Pi_{a_1 a_2}(v)$ et $\Pi_{b_1 b_2}(u) = \Pi_{b_1 b_2}(v)$ donc nous ne commutons jamais deux lettres correspondant aux actions a_1 et a_2 ou b_1 et b_2, c'est-à-dire deux lettres dont les images par φ sont égales. Donc, pour toute règle $xy \longrightarrow y'x'$ de R_{eX} utilisée durant la dérivation, $\varphi(x)\varphi(y) \longrightarrow \varphi(y)\varphi(x)$ est une règle de θ_Σ. De plus, nous avons

$$\hat{\varphi}(\Pi_{a_1 a_2 b_1 b_2}(u)) = \Pi_{ab}(\hat{\varphi}(u)).$$

Grâce à ces remarques, nous pouvons facilement montrer par induction sur la longueur de la dérivation que nous pouvons copier pas à pas la dérivation

$$\Pi_{ab}(\hat{\varphi}(u)) \xrightarrow[\theta_\Sigma]{n} \Pi_{ab}(\hat{\varphi}(v)),$$

sur la dérivation

$$g(\Pi_{a_1a_2b_1b_2}(u)) \xrightarrow[R_{eX}]{n} w.$$

D'après le Lemme de Projection 4.7, nous avons donc :

$$\hat{\varphi}(u) \xrightarrow[\theta_\Sigma]{*} \hat{\varphi}(v).$$

Comme $\hat{\varphi}(M) = L$ et comme u appartient à M, $\hat{\varphi}(u)$ appartient à L. Le langage L étant clos par θ_Σ, $\hat{\varphi}(v)$ appartient à L. Nous savons également que pour toute action a de Σ, $\Pi_{a_1a_2}(v)$ appartient à $(a_{1s}a_{1t}a_{2s}a_{2t})^*(a_{1s}a_{1t} + \varepsilon)$ donc,

$$\varphi^{-1}(\hat{\varphi}(v)) \cap \bigsqcup_{a\in\Sigma} (a_{1s}a_{1t}a_{2s}a_{2t})^*(a_{1s}a_{1t} + \varepsilon) = v,$$

et nous pouvons conclure que v appartient à M donc M est clos par R'. □

Dans cette preuve, nous construisons un langage M qui satisfait certaines propriétés de clôture. La construction que nous utilisons préserve la régularité donc, si nous commençons avec un langage régulier, M est aussi régulier. Ceci conduit au résultat :

Proposition 4.18 *La famille des images par st-morphismes des st-langages réguliers et clos par R' et la famille des st-langages réguliers et clos par θ coïncident :*

$$\Phi_{\mathrm{st}}(\mathrm{R}_{\mathrm{R}'}) = \mathrm{R}_\theta.$$

Ceci termine l'étude des liens entre les familles des langages clos par θ et R', nous allons nous intéresser, dans la section suivante aux deux familles de langages de synchronisation, généralisés ou non.

4.3 Lien entre les familles LS et $\mathrm{LS_G}$

Nous allons à présent nous attacher à montrer que les familles $\Phi_{\mathrm{st}}(\mathrm{LS})$ et $\Phi_{\mathrm{st}}(\mathrm{LS_G})$ sont égales et, pour ce faire, nous allons montrer que $\mathrm{LS_G}$ est incluse dans $\Phi_{\mathrm{st}}(\mathrm{LS})$ en utilisant le fait que cette famille des images par st-morphismes des langages de synchronisation est close par produit, union, étoile (ce qui est évident) et par intersection et st-shuffle (ce que nous aurons à montrer).

Nous allons, par la suite, utiliser plusieurs fois le fait que les langages de synchronisation sont clos par st-morphisme inverse. En fait nous avons même une équivalence entre l'appartenance d'un langage et l'appartenance de son image par st-morphisme inverse à la famille LS (ou respectivement à la famille $\mathrm{LS_G}$).

Lemme 4.19 *Soient L un st-langage et $\hat{\varphi}$ un st-morphisme. Nous avons :*

$$(L \in \mathrm{LS}) \Leftrightarrow (\hat{\varphi}^{-1}(L) \in \mathrm{LS}),$$
$$(L \in \mathrm{LS_G}) \Leftrightarrow (\hat{\varphi}^{-1}(L) \in \mathrm{LS_G}).$$

Preuve. Nous allons tout d'abord montrer les implications de gauche à droite par induction sur la construction de L respectivement comme union, produit, étoile, intersection et shuffle de langages de synchronisation ou de langages de synchronisation généralisés. Les seuls cas non triviaux sont respectivement les cas où $L = L_1 \sqcup L_2$ ou $L = L_1 \sqcup_{st} L_2$. Considérons chacun de ces cas :

- si $L = L_1 \sqcup L_2$, comme L_1 et L_2 sont définis sur des alphabets disjoints, $\hat{\varphi}^{-1}(L_1)$ et $\hat{\varphi}^{-1}(L_2)$ sont également définis sur des alphabets disjoints. Comme ces deux langages sont des st-langages, leur shuffle est aussi un st-langage. Donc, nous avons

$$\hat{\varphi}^{-1}(L) = \hat{\varphi}^{-1}(L_1 \sqcup L_2) = \hat{\varphi}^{-1}(L_1) \sqcup \hat{\varphi}^{-1}(L_2).$$

  Par hypothèse de récurrence, $\hat{\varphi}^{-1}(L_1)$ et $\hat{\varphi}^{-1}(L_2)$ sont des langages de synchronisation, en conséquence, $\hat{\varphi}^{-1}(L)$ est un langage de synchronisation ;

- si $L = L_1 \sqcup_{st} L_2$, nous avons alors

$$\hat{\varphi}^{-1}(L) = \hat{\varphi}^{-1}(L_1 \sqcup L_2 \cap \mathrm{ST}_\Sigma) = \varphi^{-1}(L_1 \sqcup L_2 \cap \mathrm{ST}_\Sigma) \cap \mathrm{ST}_\mathrm{X}.$$

  Considérons le st-langage suivant :

$$M = \coprod_{x \in \Sigma} \Big(\bigcup_{y \in \varphi^{-1}(x)} y_s y_t \Big)^*,$$

  nous avons alors

$$\begin{aligned} \hat{\varphi}^{-1}(L) &= \varphi^{-1}(L_1 \sqcup L_2 \cap \mathrm{ST}_\Sigma) \cap \mathrm{ST}_\mathrm{X} \\ &= \varphi^{-1}(L_1) \sqcup \varphi^{-1}(L_2) \cap M \\ &= \hat{\varphi}^{-1}(L_1) \sqcup \hat{\varphi}^{-1}(L_2) \cap M. \end{aligned}$$

  Clairement M est un langage de synchronisation généralisé, par hypothèse de récurrence, $\hat{\varphi}^{-1}(L_1)$ et $\hat{\varphi}^{-1}(L_2)$ sont des langages de synchronisation généralisés, nous avons donc $\hat{\varphi}^{-1}(L) \in \mathrm{LS}_\mathrm{G}$.

Réciproquement, considérons un st-langage L défini sur un alphabet d'actions Υ et un morphisme d'actions φ défini de X^* dans Υ^* tel que $\hat{\varphi}^{-1}(L)$ soit un langage de synchronisation (respectivement un langage de synchronisation généralisé). Considérons la partition de l'alphabet X, $\mathrm{X} = \mathrm{X}_1 \cup \cdots \cup \mathrm{X}_n$ avec :

$$(\exists 1 \leq k \leq n \mid x, y \in \mathrm{X}_k) \Leftrightarrow (\varphi(x) = \varphi(y)).$$

Soient x_1 une lettre de X_1, ..., x_n une lettre de X_n. Posons :

$$M = \hat{\varphi}^{-1}(L) \cap [(x_{1s}x_{1t})^* \sqcup \ldots \sqcup (x_{ns}x_{nt})^*].$$

De cette égalité, nous déduisons que M appartient à LS (resp. à LS_G). Comme la restriction de φ de l'alphabet d'actions de M dans l'alphabet d'actions de L établit une bijection entre ces deux alphabets et comme $L = \varphi(M)$, L est un langage de synchronisation (resp. un langage de synchronisation généralisé). □

Le lemme précédent va nous permettre de montrer que la famille $\Phi_{st}(\mathrm{LS})$ est close par intersection.

Lemme 4.20 *La famille des images par st-morphismes des langages de synchronisation est close par intersection.*

Preuve. Soient L_1 et L_2 deux langages de synchronisation définis respectivement sur les alphabets d'actions X_1 et X_2. D'après le Lemme 4.13, la composition de deux st-morphismes est un st-morphisme donc nous pouvons toujours distinguer ces alphabets: sans perte de généralité, nous pouvons prendre $\mathrm{X}_1 \cap \mathrm{X}_2 = \emptyset$. Soient φ_1 et φ_2 deux morphismes d'actions respectivement définis de X_1^* dans Υ_1^* et de X_2^* dans Υ_2^*. Considérons les alphabets suivants:

$$\begin{aligned} \mathrm{X}_1' &= \{x \in \mathrm{X}_1 \mid \varphi_1(x) \notin \varphi_2(\mathrm{X}_2)\}, \\ \mathrm{X}_2' &= \{x \in \mathrm{X}_2 \mid \varphi_2(x) \notin \varphi_1(\mathrm{X}_1)\}, \\ \Sigma &= \{(x,y) \in \mathrm{X}_1 \times \mathrm{X}_2 \mid \varphi_1(x) = \varphi_2(y)\}, \end{aligned}$$

(notons que $\mathrm{X}_1' \cap \mathrm{X}_2' = \emptyset$) et les morphismes d'actions suivants:

$$\begin{array}{rccc} \psi_1 : & (\mathrm{X}_1' \cup \Sigma)^* & \longrightarrow & \mathrm{X}_1^* \\ & x \in \mathrm{X}_1' & \longmapsto & x \\ & (x,y) \in \Sigma & \longmapsto & x \end{array}$$

$$\begin{array}{rccc} \psi_2 : & (\mathrm{X}_2' \cup \Sigma)^* & \longrightarrow & \mathrm{X}_2^* \\ & x \in \mathrm{X}_2' & \longmapsto & x \\ & (x,y) \in \Sigma & \longmapsto & y \end{array}$$

$$\begin{array}{rccc} \psi : & \Sigma^* & \longrightarrow & (\Upsilon_1 \cap \Upsilon_2)^* \\ & (x,y) & \longmapsto & \varphi_1(x) = \varphi_2(y) \end{array}$$

Nous notons $\hat{\varphi}_1$, $\hat{\varphi}_2$, $\hat{\psi}$, $\hat{\psi}_1$ et $\hat{\psi}_2$ les st-morphismes induits par ces morphismes d'actions. Nous allons montrer que nous avons:

$$\hat{\varphi}_1(L_1) \cap \hat{\varphi}_2(L_2) = \hat{\psi}(\hat{\psi}_1^{-1}(L_1) \cap \hat{\psi}_2^{-1}(L_2)).$$

Remarquons tout d'abord les deux propriétés suivantes:

Assertion 1. Soient u_1 appartenant à L_1 et u_2 appartenant à L_2. Nous avons:

$$(\hat{\varphi}_1(u_1) = \hat{\varphi}_2(u_2)) \Leftrightarrow (\exists v \in \mathrm{ST}_\Sigma \mid \hat{\psi}_1(v) = u_1 \text{ et } \hat{\psi}_2(v) = u_2).$$

Pour l'implication de gauche à droite, il suffit de remarquer que si $u_1 = x_1 \ldots x_k$ et $u_2 = y_1 \ldots y_k$, le mot $v = (x_1, y_1) \ldots (x_k, y_k)$ appartient à ST_Σ avec $\hat{\psi}_1(v) = u_1$ et $\hat{\psi}_2(v) = u_2$. L'implication de droite à gauche se déduit des définitions des alphabets et des morphismes d'actions.

Assertion 2. Pour tout mot v de ST_Σ, nous avons:

$$\hat{\psi}(v) = \hat{\varphi}_1(\hat{\psi}_1(v)) = \hat{\varphi}_2(\hat{\psi}_2(v)).$$

De ces deux assertions, nous déduisons les équivalences suivantes :

$$\begin{array}{ll} & w \in \hat{\varphi}_1(L_1) \cap \hat{\varphi}_2(L_2) \\ \Leftrightarrow & \exists u_1 \in L_1, u_2 \in L_2 \mid \hat{\varphi}_1(u_1) = \hat{\varphi}_2(u_2) = w \\ \Leftrightarrow & \exists v \in \mathrm{ST}_\Sigma \mid \hat{\psi}_1(v) = u_1 \in L_1, \hat{\psi}_2(v) = u_2 \in L_2, \\ & w = \hat{\varphi}_1(\hat{\psi}_1(v)) = \hat{\varphi}_2(\hat{\psi}_2(v)) = \hat{\psi}(v) \\ \Leftrightarrow & \exists v \in \hat{\psi}_1^{-1}(L_1) \cap \hat{\psi}_2^{-1}(L_2) \mid w = \hat{\varphi}_1(\hat{\psi}_1(v)) = \hat{\varphi}_2(\hat{\psi}_2(v)) = \hat{\psi}(v) \\ \Leftrightarrow & w \in \hat{\psi}(\hat{\psi}_1^{-1}(L_1) \cap \hat{\psi}_2^{-1}(L_2)). \end{array}$$

Donc, nous avons

$$\hat{\varphi}_1(L_1) \cap \hat{\varphi}_2(L_2) = \hat{\psi}(\hat{\psi}_1^{-1}(L_1) \cap \hat{\psi}_2^{-1}(L_2)).$$

La famille des langages de synchronisation est close par st-morphisme inverse (d'après le Lemme 4.19) donc $\hat{\psi}_1^{-1}(L_1)$ et $\hat{\psi}_2^{-1}(L_2)$ sont des langages de synchronisation, nous avons :

$$\hat{\varphi}_1(L_1) \cap \hat{\varphi}_2(L_2) \in \Phi_{\mathrm{st}}(\mathrm{LS}).$$

□

Ce lemme nous permet également de montrer une équivalence semblable à celle du Lemme 4.19 pour les images par morphismes de langages de synchronisation. Cette équivalence ne nous est pas utile pour le moment cependant, montrer que deux familles ont des propriétés différentes est un moyen de les différencier, il était donc utile de vérifier cette équivalence pour la famille $\Phi_{\mathrm{st}}(\mathrm{LS})$ puisque nous l'avons pour la famille LS_{G}.

Lemme 4.21 *Soient L un st-langage et $\hat{\varphi}$ un st-morphisme. Nous avons :*

$$(L \in \Phi_{\mathrm{st}}(\mathrm{LS})) \Leftrightarrow (\hat{\varphi}^{-1}(L) \in \Phi_{\mathrm{st}}(\mathrm{LS})).$$

Preuve. Soit $L \in \Phi_{\mathrm{st}}(\mathrm{LS})$. Il existe un langage de synchronisation M et un st-morphisme $\hat{\varphi}_1$ tels que $L = \hat{\varphi}_1(M)$. Soit $\hat{\varphi}$ un st-morphisme, nous avons $\hat{\varphi}^{-1}(L) = \hat{\varphi}^{-1}(\hat{\varphi}_1(M))$. D'après le lemme 4.14, il existe deux st-morphismes $\hat{\psi}_1$ et $\hat{\psi}_2$ tels que $\hat{\varphi}^{-1}(L) = \hat{\psi}_2(\hat{\psi}_1^{-1}(M))$. D'après le Lemme 4.19, l'image par un st-morphisme inverse d'un langage de synchronisation est un langage de synchronisation donc $\hat{\varphi}^{-1}(L)$ appartient à $\Phi_{\mathrm{st}}(\mathrm{LS})$.

La preuve de la réciproque est totalement identique à celle de la réciproque du Lemme 4.19 puisque nous savons à présent que la famille des images par st-morphisme des langages de synchronisation est close par intersection. □

Il nous reste à présent à considérer le st-shuffle avant de pouvoir énoncer le principal résultat de ce chapitre.

Lemme 4.22 *La famille des images par st-morphisme des langages de synchronisation est close par st-shuffle.*

Preuve. Rappelons que le st-shuffle de deux langages est l'ensemble des st-mots de leur shuffle. Soient L_1 et L_2 deux langages de synchronisation respectivement définis sur les alphabets d'actions X_1 et X_2. Nous pouvons supposer que $\mathrm{X}_1 \cap \mathrm{X}_2 = \emptyset$. Soient φ_1 et φ_2 deux morphismes d'actions respectivement définis de X_1^* dans Υ_1^* et de X_2^* dans Υ_2^*. Nous allons considérer le morphisme d'actions :

$$\begin{array}{rccl} \varphi_3 : & (\mathrm{X}_1 \cup \mathrm{X}_2)^* & \longrightarrow & (\Upsilon_1 \cup \Upsilon_2)^* \\ & x \in \mathrm{X}_1 & \longmapsto & \varphi_1(x) \\ & x \in \mathrm{X}_2 & \longmapsto & \varphi_2(x) \end{array}$$

Comme L_1 et L_2 sont définis sur des alphabets disjoints, $L_1 \sqcup L_2$ est un st-langage et nous avons :

$$[\hat{\varphi}_1(L_1) \sqcup \hat{\varphi}_2(L_2)] \cap \mathrm{ST}_{\Upsilon_1 \cup \Upsilon_2} = \hat{\varphi}_3(L_1 \sqcup L_2).$$

□

Les propriétés de clôture de la famille des images par st-morphismes des langages de synchronisation que nous venons de montrer nous permettent d'établir le lien entre les langages de synchronisation et les langages de synchronisation généralisés.

Proposition 4.23 *La famille des images par st-morphismes des langages de synchronisation généralisés et la famille des images par st-morphismes des langages de synchronisation coïncident :*

$$\Phi_{\mathrm{st}}(\mathrm{LS_G}) = \Phi_{\mathrm{st}}(\mathrm{LS}).$$

Preuve. La famille $\Phi_{\mathrm{st}}(\mathrm{LS})$ contient tous les langages finis qui représentent l'exécution d'une seule action (comme $a_s a_t$) et elle est close par union, produit, étoile, intersection (Lemme 4.20) et st-shuffle (Lemme 4.22) donc, elle contient les langages de synchronisation généralisés :

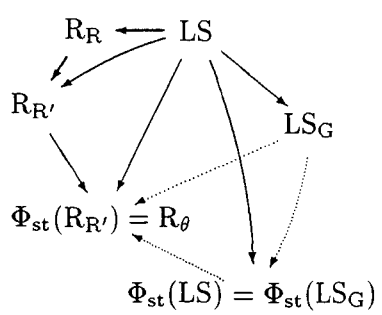
$$(\mathrm{LS_G} \subseteq \Phi_{\mathrm{st}}(\mathrm{LS})) \Rightarrow (\Phi_{\mathrm{st}}(\mathrm{LS_G}) \subseteq \Phi_{\mathrm{st}}(\mathrm{LS})).$$

De plus, la famille des langages de synchronisation est incluse dans la famille des langages de synchronisation généralisés, nous avons :

$$(\mathrm{LS} \subseteq \mathrm{LS_G}) \Rightarrow (\Phi_{\mathrm{st}}(\mathrm{LS}) \subseteq \Phi_{\mathrm{st}}(\mathrm{LS_G})).$$

□

Nous venons donc de montrer le lien entre les langages de synchronisation et les langages de synchronisation généralisés ainsi qu'entre les familles de langages clos par les systèmes que nous avons utilisés pour essayer de les caractériser. À ce stade, nous avons montré les inclusions suivantes (les flèches pleines désignent des inclusions strictes, les flèches en pointillés des inclusions larges) :



On peut néanmoins remarquer que nous n'avons toujours aucune inclusion d'une famille de langages clos par un système de réécriture dans une famille construite à partir des expressions de synchronisation et donc aucune caractérisation des familles de langages de synchronisation par un système de réécriture. Nous avons vu qu'une telle caractérisation n'existe pas dans le cas des langages de synchronisation mais la question reste ouverte dans les autres cas.

Chapitre 5

Langages de synchronisation généralisés et semi-commutations

Nous allons, dans ce chapitre, montrer que les st-langages réguliers clos par la semi-commutation θ sont les images par st-morphismes des langages de synchronisation généralisés (qui sont égales aux images des langages de synchronisation). Cette égalité, si elle n'est qu'une version affaiblie de la Conjecture III de Salomaa et Yu, représente cependant le premier lien entre une famille de langages clos par un système de réécriture et une famille de langages construits à partir des expressions de synchronisation. La preuve de ce résultat, bien que constructive, met en œuvre un grand nombre d'outils dont la manipulation est assez lourde. C'est pourquoi nous allons, dans un premier temps, montrer ce résultat dans le cas particulier des langages réguliers qui sont la clôture par θ d'un langage bien formé. Nous avons alors une preuve totalement différente et basée sur une construction beaucoup plus simple que celle du cas général.

5.1 Cas de clôtures par θ de langages bien formés

Dans cette section, nous proposons donc de montrer que les clôtures par θ des st-langages bien formés sont des images par st-morphismes de langages de synchronisation généralisés. Nous allons tout d'abord montrer que la clôture par θ d'un langage qui est la concaténation de mots et de langages de synchronisation généralisés dont les alphabets sont deux à deux disjoints est un langage de synchronisation généralisé.

Lemme 5.1 *Soient Σ un alphabet d'actions, $u_0, \ldots, u_n$ des mots de Σ^* et $L_1, \ldots L_n$ des langages de synchronisation généralisés sur Σ tels que les alphabets des L_i et de $u_0 \ldots u_n$ sont deux à deux disjoints. Alors le langage $\mathrm{f}_{\theta_\Sigma}(u_0 L_1 u_1 \ldots L_n u_n)$ est un langage de synchronisation généralisé pour lequel on peut construire une expression de synchronisation généralisée.*

Preuve. Nous allons durant cette preuve, noter Π_X la projection $\Pi_{X_s \cup X_t}$ pour tout alphabet d'actions X. Posons $u = u_0 \ldots u_n$ et montrons l'égalité

$$\mathrm{f}_{\theta_\Sigma}(u_0 L_1 \ldots L_n u_n) = \mathrm{f}_{\theta_\Sigma}(u) \sqcup\!\sqcup L_1.L_2 \ldots L_n \bigcap_{x \in \alpha(u)} M_x, \tag{5.1}$$

avec pour tout x appartenant à $\alpha(u)$:

$$M_x = \mathrm{f}_{\theta_\Sigma}(\Pi_{\{x\}}(u_0) L_1 \Pi_{\{x\}}(u_1) \ldots L_n \Pi_{\{x\}}(u_n)) \sqcup\!\sqcup \mathrm{f}_{\theta_\Sigma}(\Pi_{\alpha(u)\setminus\{x\}}(u)).$$

L'inclusion de gauche à droite se déduit du Lemme de Projection 4.7.

Soit v appartenant au membre droit de l'égalité. Comme les alphabets d'actions des L_i et de u sont deux à deux disjoints, il existe v_1 appartenant à L_1, ..., v_n appartenant à L_n tels que :

- v appartient à $\mathrm{f}_{\theta_\Sigma}(u) \sqcup\!\sqcup v_1.v_2 \ldots v_n$;
- pour tout x de $\alpha(u)$, v appartient à $\mathrm{f}_{\theta_\Sigma}(\Pi_x(u_0)v'_1 \ldots v'_n\Pi_x(u_n)) \sqcup\!\sqcup \mathrm{f}_{\theta_\Sigma}(\Pi_{\alpha(u)\setminus\{x\}}(u))$ où, pour tout indice i, v'_i est commutativement équivalent à v_i.

Considérons le mot $w = u_0v_1u_2 \ldots v_nu_n$ de $u_0L_1u_2 \ldots L_nu_n$. Nous allons montrer que pour tout couple d'actions a et b, $\Pi_{ab}(v)$ appartient à $\mathrm{f}_{\theta_\Sigma}(\Pi_{ab}(w))$:

- si a et b sont des actions de u, $\Pi_{ab}(w) = \Pi_{ab}(u)$ et, comme $\Pi_{ab}(v)$ appartient à $\mathrm{f}_{\theta_\Sigma}(\Pi_{ab}(u))$, nous pouvons immédiatement déduire que $\Pi_{ab}(v)$ appartient à $\mathrm{f}_{\theta_\Sigma}(\Pi_{ab}(w))$;
- si a est une action de L_i et b une action de L_j (que i et j soient distincts ou non), $\Pi_{ab}(v) = \Pi_{ab}(w)$;
- si a est une action de u et b une action de L_i, le mot $\Pi_{ab}(v)$ appartient au langage $\mathrm{f}_{\theta_\Sigma}(\Pi_{\{a\}}(u_0)\Pi_{\{b\}}(v'_1) \ldots \Pi_{\{b\}}(v'_n)\Pi_{\{a\}}(u_n))$. Comme pour tout indice i, $\Pi_{\{b\}}(v'_1)$ et $\Pi_{\{b\}}(v_1)$ sont égales, nous obtenons :

$$\Pi_{ab}(v) \in \mathrm{f}_{\theta_\Sigma}(\Pi_{ab}(w)).$$

Comme pour tout couple d'actions a et b, $\Pi_{ab}(v)$ appartient à $\mathrm{f}_{\theta_\Sigma}(\Pi_{ab}(w))$, d'après le Lemme de Projection 4.7, le mot v appartient à $\mathrm{f}_{\theta_\Sigma}(w)$. Ceci achève la preuve de l'égalité 5.1.

Les langages de la forme $\mathrm{f}_{\theta_\Sigma}(\Pi_{\alpha(u)\setminus\{x\}}(u))$ et $\mathrm{f}_{\theta_\Sigma}(u)$ sont des st-langages finis clos par θ_Σ, ce sont donc des langages de synchronisation généralisés. Le langage $L_1.L_2 \ldots L_n$, concaténation de langages de synchronisation généralisés, est un langage de synchronisation généralisé. Comme les langages de synchronisation généralisés sont clos par intersection et st-shuffle (ici nous calculons le shuffle de langages définis sur des alphabets disjoints, nous pouvons donc utiliser indifféremment shuffle ou st-shuffle), il suffit que les langages de la forme $\mathrm{f}_{\theta_\Sigma}(\Pi_x(u_0)L_1 \ldots L_n\Pi_x(u_n))$ soient des langages de synchronisation généralisés pour que $\mathrm{f}_{\theta_\Sigma}(u_0L_1 \ldots L_nu_n)$ en soit également un.

Chacun des L_i est un langage de synchronisation généralisé, pour chaque indice i, nous pouvons donc noter e_i une expression de synchronisation généralisée correspondant à L_i. Clairement, si $\Pi_x(u_0)L_1 \ldots L_n\Pi_x(u_n)$ est une concaténation de deux st-langages non vides, N_1 et N_2, $\mathrm{f}_{\theta_\Sigma}(u_0L_1 \ldots L_nu_n) = \mathrm{f}_{\theta_\Sigma}(N_1).\mathrm{f}_{\theta_\Sigma}(N_2)$ et une expression de synchronisation pour $\mathrm{f}_{\theta_\Sigma}(u_0L_1 \ldots L_nu_n)$ peut être obtenue par mise en séquence des expressions de $\mathrm{f}_{\theta_\Sigma}(N_1)$ et $\mathrm{f}_{\theta_\Sigma}(N_2)$. Si un langage de la forme $\Pi_x(u_0)L_1 \ldots L_n\Pi_x(u_n)$ ne possède pas de st-facteur gauche propre, alors il est égal à $x_s.L_1.L_2 \ldots L_n.x_t$. On vérifie aisément que la clôture d'un langage de cette forme correspond à l'expression :

$$x \parallel (e_1 \rightarrow \cdots \rightarrow e_n).$$

Le langage $\mathrm{f}_{\theta_\Sigma}(u_0L_1 \ldots L_nu_n)$ est donc un langage de synchronisation généralisé pour lequel on peut construire une expression de synchronisation généralisée, connaissant une expression pour chacun des L_i. □

Nous allons donc à présent utiliser le lemme précédent pour montrer que la clôture par θ d'un langage bien formé est un langage de synchronisation généralisé.

Lemme 5.2 *Soit N un st-langage bien formé. Il existe un morphisme d'actions φ et un langage de synchronisation généralisé L tels que :*

$$\mathrm{f}_\theta(N) = \varphi(L).$$

Preuve. Le langage N s'écrit sous la forme

$$N = \bigcup_{i \in F} u_{0,i} M_{1,i}^* u_{1,i} M_{2,i}^* \dots M_{k_i,i}^* u_{k_i,i}$$

où pour tout i et tout j, $M_{i,j}$ est un st-langage bien formé. Comme la clôture par θ d'une union de langages est égale à l'union des clôtures de ces langages, il nous suffit d'étudier la clôture de l'un des membres de l'union. Posons

$$M = u_0 M_1^* u_1 M_2^* \dots M_n^* u_n,$$

l'alphabet d'actions de M étant Σ et montrons que $\mathrm{f}_{\theta_\Sigma}(M)$ est l'image par un morphisme d'actions d'un langage de synchronisation généralisé par récurrence sur la hauteur d'étoile de M.

Si M est fini, $\mathrm{f}_{\theta_\Sigma}(M)$ est un langage de synchronisation généralisé. Sinon, nous avons :

$$\mathrm{f}_{\theta_\Sigma}(M) = \mathrm{f}_{\theta_\Sigma}(u_0 \mathrm{f}_{\theta_\Sigma}(M_1^*) u_1 \dots \mathrm{f}_{\theta_\Sigma}(M_n^*) u_n)$$

et, d'après le Lemme 4.6, nous obtenons :

$$\mathrm{f}_{\theta_\Sigma}(M) = \mathrm{f}_{\theta_\Sigma}(u_0 (\mathrm{f}_{\theta_\Sigma}(M_1))^* u_1 \dots (\mathrm{f}_{\theta_\Sigma}(M_n))^* u_n).$$

Par hypothèse de récurrence, pour tout indice i, il existe un morphisme d'actions φ_i et un langage de synchronisation généralisé L_i tels que $\mathrm{f}_{\theta_\Sigma}(M_i) = \varphi_i(L_i)$. Nous pouvons, sans perte de généralité, considérer que les alphabets des L_i et l'alphabet de $u = u_0 u_1 \dots u_n$ sont deux à deux disjoints. Nous obtenons :

$$\mathrm{f}_{\theta_\Sigma}(M) = \mathrm{f}_{\theta_\Sigma}(u_0 (\varphi_1(L_1))^* u_1 (\varphi_2(L_2))^* \dots (\varphi_n(L_n))^* u_n),$$

d'où :

$$\mathrm{f}_{\theta_\Sigma}(M) = \mathrm{f}_{\theta_\Sigma}(u_0 \varphi_1(L_1^*) u_1 \varphi_2(L_2^*) \dots \varphi_n(L_n^*) u_n).$$

Notons $Id_{\mathrm{alph}(u)}$ l'identité sur l'alphabet de u et posons $\varphi = \varphi_1 + \dots + \varphi_n + Id_{\mathrm{alph}(u)}$, nous avons :

$$\mathrm{f}_{\theta_\Sigma}(M) = \mathrm{f}_{\theta_\Sigma}(\varphi(u_0 L_1^* u_1 \dots L_n^* u_n)).$$

Notons que le langage $\varphi(u_0 L_1^* u_1 \dots L_n^* u_n)$ est un st-langage.

Il nous faut à présent montrer que, si X est l'alphabet d'actions de $u_0 L_1^* \dots L_n^* u_n$,

$$\mathrm{f}_{\theta_\Sigma}(\varphi(u_0 L_1^* u_1 \dots L_n^* u_n)) = \varphi(\mathrm{f}_{\theta_X}(u_0 L_1^* \dots L_n^* u_n)). \tag{5.2}$$

Soit u un mot de $\mathrm{f}_{\theta_\Sigma}(\varphi(u_0 L_1^* u_1 \dots L_n^* u_n))$. Il existe un mot v de $u_0 L_1^* u_1 \dots L_n^* u_n$ tel que u appartient à $\mathrm{f}_{\theta_\Sigma}(\varphi(v))$. Clairement, il suffit de vérifier pour une dérivation de longueur 1 que

$$(\varphi(v) \underset{\theta_\Sigma}{\longrightarrow} u) \Rightarrow (v \underset{\theta_X}{\longrightarrow} w \text{ avec } \varphi(w) = u).$$

Nous pouvons décomposer v en $v = v_1xyv_2$ pour mettre en évidence la règle appliquée :

$$\varphi(v) = \varphi(v_1)\varphi(x)\varphi(y)\varphi(v_2) \underset{\theta_\Sigma}{\longrightarrow} \varphi(v_1)\varphi(y)\varphi(x)\varphi(v_2) = u.$$

Pour toute règle $\varphi(x)\varphi(y) \longrightarrow \varphi(y)\varphi(x)$ de θ_Σ, la règle $xy \longrightarrow yx$ appartient à θ_X, nous avons donc :

$$v = v_1xyv_2 \underset{\theta_X}{\longrightarrow} v_1yxv_2 \text{ et } \varphi(v_1yxv_2) = u.$$

L'inclusion de gauche à droite de l'égalité 5.2 est donc vérifiée.

Soit u un mot de $\varphi(\mathrm{f}_{\theta_X}(u_0L_1^* \ldots L_n^*u_n))$. Il existe deux mots v de $\mathrm{f}_{\theta_X}(u_0L_1^* \ldots L_n^*u_n)$ et w de $u_0L_1^* \ldots L_n^*u_n$ tels que v appartient à $\mathrm{f}_{\theta_X}(w)$ et $\varphi(v) = u$. Clairement il suffit de montrer pour une dérivation de longueur 1 que

$$(w \underset{\theta_X}{\longrightarrow} v) \Rightarrow (\varphi(w) \underset{\theta_\Sigma}{\longrightarrow} \varphi(v)).$$

Comme $\varphi(u_0L_1^* \ldots L_n^*u_n)$ est un st-langage, pour toute paire d'actions x et y de X telles que $\varphi(x) = \varphi(y)$, nous avons $\Pi_{xy}(u_0L_1^* \ldots L_n^*u_n) \subseteq (x_sx_t + y_sy_t)^*$. Donc, pour toute règle $xy \longrightarrow yx$ de θ_X permettant de passer de w à v, il existe une règle $\varphi(x)\varphi(y) \longrightarrow \varphi(y)\varphi(x)$ de θ_Σ permettant de passer de $\varphi(w)$ à $\varphi(v)$ donc $\varphi(w) \longrightarrow_{\theta_\Sigma} \varphi(v)$. L'égalité 5.2 est vérifée.

Nous avons donc

$$\mathrm{f}_{\theta_\Sigma}(M) = \varphi(\mathrm{f}_{\theta_X}(u_0L_1^* \ldots L_n^*u_n)),$$

d'après le Lemme 5.1, le langage $\mathrm{f}_{\theta_X}(u_0L_1^* \ldots L_n^*u_n)$ est un langage de synchronisation généralisé, le langage $\mathrm{f}_{\theta_\Sigma}(M)$ est donc l'image par un morphisme d'actions d'un langage de synchronisation généralisé. Nous pouvons donc revenir au langage N de départ :

$$\mathrm{f}_{\theta_\Sigma}(N) = \mathrm{f}_{\theta_\Sigma}(\bigcup_{i \in F} u_{0,i}M_{1,i}^*u_{1,i}M_{2,i} \ldots M_{k_i,i}^*u_{k_i,i})$$

peut s'écrire :

$$\mathrm{f}_{\theta_\Sigma}(L) = \bigcup_{i \in F} \psi_i(L_{gi})$$

où pour tout indice i de F, ψ_i est un morphisme d'actions et L_{gi} un langage de synchronisation généralisé. Nous pouvons, sans perte de généralité, supposer que les L_{gi} sont définis sur des alphabets deux à deux disjoints donc, si l'on note $\psi = \psi_1 + \cdots + \psi_n$ et L le langage de synchronisation généralisé tel que $L = L_{g1} + \cdots + L_{gn}$, nous obtenons :

$$\mathrm{f}_{\theta_\Sigma}(N) = \psi(L).$$

□

Reprenons encore une fois l'exemple du st-langage bien formé $\mathbb{L} = b_s(a_sa_t)^*c_sb_t(a_sa_t)^*c_t$. Notons $\Sigma = \{a, b, c\}$ l'alphabet d'actions. Nous avons vu que $\mathrm{f}_{\theta_\Sigma}(\mathbb{L})$ n'est pas un langage de synchronisation, appliquons lui la construction que nous venons de voir. Nous avons :

$$\mathrm{f}_{\theta_\Sigma}(\mathbb{L}) = \mathrm{f}_{\theta_\Sigma}(b_s(\mathrm{f}_{\theta_\Sigma}(a_sa_t))^*c_sb_t(\mathrm{f}_{\theta_\Sigma}(a_sa_t))^*c_t.$$

Le langage $(\mathrm{f}_{\theta_\Sigma}(a_sa_t))^*$ correspond clairement à l'expression a^* cependant, comme nous voulons que les différents termes soient définis sur des alphabets disjoints, nous allons considérer

les langages de synchronisation généralisés $L_1 = (a_{1s}a_{1t})^*$ et $L_2 = (a_{2s}a_{2t})^*$ et les morphismes d'actions :

$$\begin{aligned} \varphi_1 &: a_1 \longmapsto a \\ \varphi_2 &: a_2 \longmapsto a. \end{aligned}$$

Nous avons :

$$\mathrm{f}_{\theta_\Sigma}(\mathbb{L}) = \mathrm{f}_{\theta_\Sigma}(b_s\varphi_1(L_1)c_sb_t\varphi_2(L_2)c_t) = \varphi(\mathrm{f}_{\theta_X}(b_s(a_{1s}a_{1t})^*c_sb_t(a_{2s}a_{2t})^*c_t)$$

où φ est le morphisme d'actions :

$$\begin{array}{rccc} \varphi : & \mathrm{X}^* & \longrightarrow & \Sigma^* \\ & a_1 & \longmapsto & a \\ & a_2 & \longmapsto & a \\ & b & \longmapsto & b \\ & c & \longmapsto & c. \end{array}$$

D'après le Lemme 5.1, le langage $\mathrm{f}_{\theta_X}(b_s(a_{1s}a_{1t})^*c_sb_t(a_{2s}a_{2t})^*c_t)$ est un langage de synchronisation généralisé. En appliquant la construction de la preuve du lemme, nous allons obtenir une expression de synchronisation généralisée :

$$\begin{aligned} \mathrm{f}_{\theta_X}(b_s(a_{1s}a_{1t})^*c_sb_t(a_{2s}a_{2t})^*c_t) =& [\mathrm{f}_{\theta_X}(b_sc_sb_tc_t) \sqcup\!\sqcup_{st} (a_{1s}a_{1t})^*(a_{2s}a_{2t})^*] \\ & \cap [\mathrm{f}_{\theta_X}(b_s(a_{1s}a_{1t})^*b_t(a_{2s}a_{2t})^*) \sqcup\!\sqcup_{st} c_sc_t] \\ & \cap [\mathrm{f}_{\theta_X}((a_{1s}a_{1t})^*c_s(a_{2s}a_{2t})^*c_t) \sqcup\!\sqcup_{st} b_sb_t]. \end{aligned}$$

Nous obtenons donc l'expression :

$$[(b \parallel c) \parallel (a_1^* \to a_2^*)] \& [(b \parallel a_1^*) \to a_2^*] \& [a_1^* \to (c \parallel a_2^*)].$$

La construction que nous avons utilisée dans la preuve du Lemme 5.2 ne peut être adaptée au cas général. Il est en effet difficile d'utiliser une preuve par récurrence sur la hauteur d'étoile pour des langages qui ne sont pas bien formés. Nous allons donc devoir mettre en œuvre une construction totalement différente.

5.2 Cas général des st-langages réguliers clos par θ

5.2.1 La transformation

Nous allons, dans le cas général, ramener la clôture par θ à une clôture par une commutation partielle afin de pouvoir disposer des nombreux outils, tels que les automates asynchrones, qui sont spécifiques à l'étude des commutations partielles et qui n'ont pas d'équivalent pour les semi-commutations. Nous montrons, dans cette partie, qu'il est possible d'utiliser, pour calculer la clôture d'un langage par θ, un coloriage, une commutation partielle et un morphisme « effaçant les couleurs ». L'idée de base, que nous allons voir sur un exemple, est très simple.

Considérons un alphabet de deux actions $\Sigma = \{a, b\}$ et un mot $u = a_sb_sa_tb_tb_sb_t$. Il nous faut nous rappeler que la clôture de la concaténation de deux st-mots est égale à la concaténation de leurs clôtures (Lemme 4.6). Nous avons :

$$\mathrm{f}_{\theta_\Sigma}(u) = \{a_sb_sa_tb_t, b_sa_sa_tb_t, a_sb_sb_ta_t, b_sa_sb_ta_t, a_sa_tb_sb_t, b_sb_ta_sa_t\}.\{b_sb_t\}.$$

Nous allons utiliser des « couleurs » (ici des indices) pour mettre en avant certaines dépendances entre actions. Tout d'abord, comme les lettres de deux facteurs st-primitifs ne peuvent jamais être mélangées, nous colorions alternativement les facteurs st-primitifs avec deux ensembles disjoints de couleurs et nous interdisons aux lettres qui ne sont pas colorées à l'aide du même ensemble de commuter. Cette première étape permet d'assurer l'intégrité des facteurs st-primitifs, elle n'est cependant pas suffisante pour mettre en avant toutes les dépendances nécessaires. En effet, dans un facteur st-primitif, certaines actions sont en séquence, nous allons donc distinguer certaines séquences grâce aux couleurs et les lettres portant une couleur commune ne seront pas autorisées à commuter (nous définirons par la suite un coloriage qui respecte ces deux points en utilisant un nombre fini de couleurs). Par exemple, le mot $v = a_{1s}b_{2s}a_{1t}b_{2t}b_{5s}b_{5t}$ est un coloriage de u qui satisfait les deux points. Nous pouvons à présent utiliser une commutation partielle, à la place d'une semi-commutation :

$$\varrho = \{(a_{1s}, b_{2s}), (a_{1s}, b_{2t}), (a_{1t}, b_{2s}), (a_{1t}, b_{2t})\},$$

et nous obtenons :

$$\begin{aligned} \mathrm{f}_\varrho(v) = \{ \quad & a_{1s}b_{2s}a_{1t}b_{2t}, b_{2s}a_{1s}a_{1t}b_{2t}, a_{1s}b_{2s}b_{2t}a_{1t}, \\ & b_{2s}a_{1s}b_{2t}a_{1t}, a_{1s}a_{1t}b_{2s}b_{2t}, b_{2s}b_{2t}a_{1s}a_{1t}\}.\{b_{5s}b_{5t}\}. \end{aligned}$$

En utilisant une commutation partielle, nous perdons la non symétrie. Cette perte est contrebalancée par l'utilisation du coloriage qui porte, lui-aussi, des informations sur les dépendances entre actions. Si le morphisme φ « efface » les couleurs, il est aisé de voir que $\mathrm{f}_{\theta_\Sigma}(u) = \varphi(\mathrm{f}_\varrho(v))$. En effet, ϱ permet uniquement de commuter les lettres correspondant à des actions différentes, n'appartenant pas à deux facteurs primitifs consécutifs (marquées avec des ensembles de couleurs différents) et n'appartenant pas à une séquence mise en évidence (n'ayant pas la même couleur).

Cette idée très simple peut être appliquée à un langage quelconque défini sur un alphabet d'actions quelconque. Il suffit pour cela de considérer les projections sur les sous-alphabets de deux actions et d'ajouter, à chaque lettre et pour chaque projection, un sous-ensemble de couleurs correspondant à sa position dans cette projection. Pour colorier un facteur primitif sur un alphabet de deux actions, nous utilisons trois couleurs et, comme nous marquons différemment deux facteurs primitifs consécutifs, nous allons utiliser deux ensembles de trois couleurs pour chaque sous-alphabet de deux actions. L'ensemble de couleurs d'une lettre sera donc l'union des ensembles qu'elle a reçus pour chaque projection.

Exemple 5.3 Prenons l'exemple du mot $u = a_s b_s b_t c_s a_t c_t$. Selon le principe précédent, nous allons utiliser deux ensembles de trois couleurs $\{1, 2, 3\}$ et $\{4, 5, 6\}$ pour colorier chaque projection, nous obtenons les coloriages suivants :

$$\begin{array}{l} a_{1s}b_{2s}b_{2t}a_{1t} \\ a_{1s}c_{2s}a_{1t}c_{2t} \\ b_{1s}b_{1t}c_{4s}c_{4t} \end{array}$$

La lettre a_s a reçu les couleurs 1 pour les actions a et b et 1 pour les actions a et c, nous la noterons donc $(a_s, \{ab_1, ac_1\})$ (il faut en effet que les ensembles de couleurs utilisés pour les différents alphabets soient disjoints) en répétant la même procédure pour toutes les lettres, nous obtenons le mot coloré v correspondant au mot u :

$$v = (a_s, \{ab_1, ac_1\})(b_s, \{ab_2, bc_1\})(b_t, \{ab_2, bc_1\})(c_s, \{ac_2, bc_4\})(a_t, \{ab_1, ac_1\})(c_t, \{ac_2, bc_4\}).$$

Les lettres $(b_t, \{ab_2, bc_1\})$ et $(c_s, \{ac_2, bc_4\})$ ne sont pas autorisées à commuter puisqu'elles ne sont pas colorées avec le même ensemble de couleurs pour $\{b, c\}$, par contre, les lettres $(a_s, \{ab_1, ac_1\})$ et $(b_s, \{ab_2, bc_1\})$ sont colorées avec le même ensemble de couleurs mais comme $\{ab_1, ac_1\} \cap \{ab_2, bc_1\} = \emptyset$, ces deux lettres peuvent commuter.

Nous allons à présent définir de façon formelle le coloriage et la commutation partielle que nous allons utiliser.

Définition 5.4 *Soit Σ un alphabet d'actions. À tout couple d'actions distinctes a et b de Σ, nous associons deux ensembles disjoints de trois couleurs notés $\mathrm{C}_{\{a,b\}}$ et $\bar{\mathrm{C}}_{\{a,b\}}$ tels que pour deux couples d'actions distincts les ensembles associés sont disjoints. Soit Σ_c l'alphabet coloré correspondant à Σ :*

$$\Sigma_c = \{(x, E) \mid x \in \Sigma, E \subseteq \bigcup_{a \neq x} \mathrm{C}_{\{a,x\}} + \bar{\mathrm{C}}_{\{a,x\}},$$

$$\forall a \neq x, E \cap \mathrm{C}_{\{a,x\}} = \emptyset \oplus E \cap \bar{\mathrm{C}}_{\{a,x\}} = \emptyset\}.$$

Notation 5.5 *Pour simplifier la manipulation des lettres de $\Sigma_{cs} \cup \Sigma_{ct}$, nous écrirons (a_s, E) (ou (a_t, E)) à la place de $(a, E)_s$ (ou $(a, E)_t$) et, pour chaque action a de Σ, nous notons*

$$\alpha((a_s, E)) = \alpha((a_t, E)) = a.$$

Il est nécessaire de définir quelques outils pour manipuler les mots colorés. En particulier, il est utile de pouvoir « effacer » les couleurs pour retrouver le mot de départ et également de « projeter » un mot coloré sur un alphabet d'actions puisque la notion de projection sur des sous-alphabets de deux actions est essentielle dans la définition du coloriage.

Définition 5.6 *Soit Σ un alphabet d'actions. Le morphisme d'actions φ_Σ « effaçant » les couleurs est défini par :*

$$\begin{array}{rccc} \varphi_\Sigma : & \Sigma_c^* & \longrightarrow & \Sigma^* \\ & (x, E) & \longmapsto & x. \end{array}$$

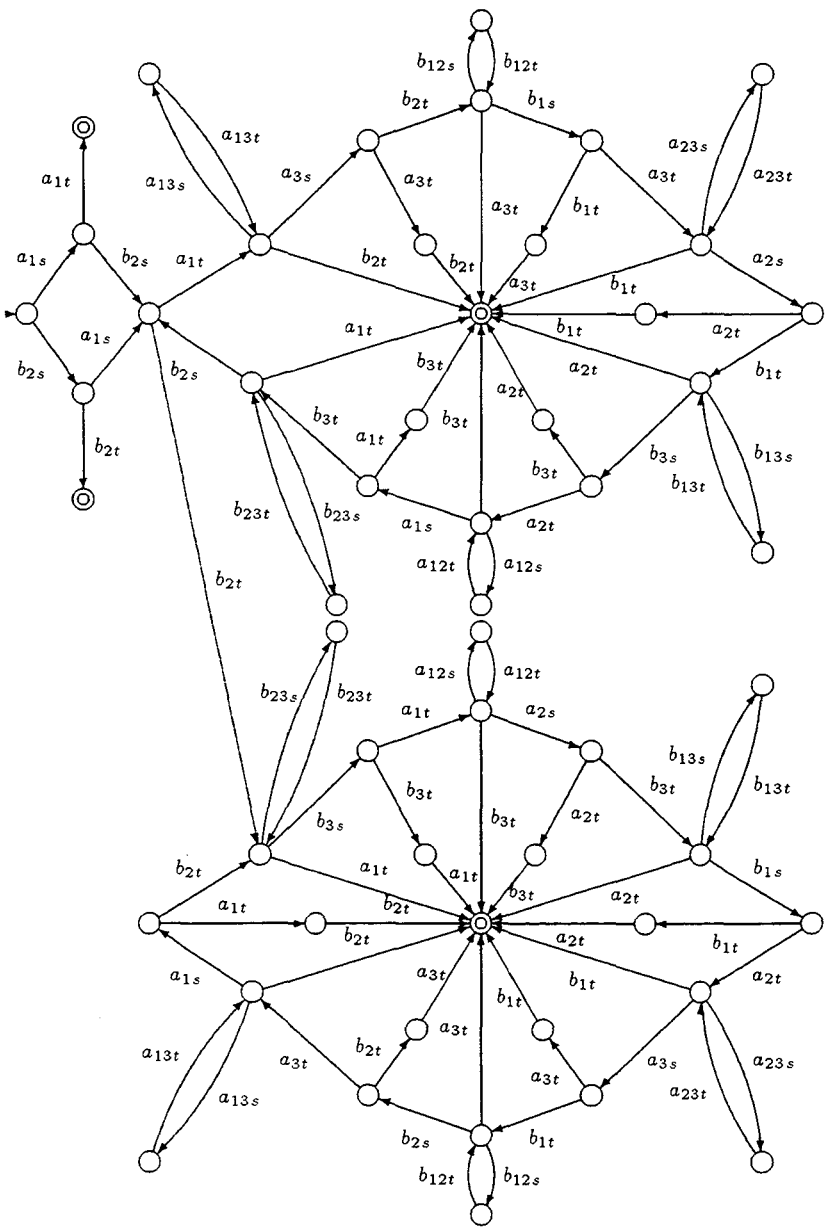
Pour toute paire d'actions $a, b \in \Sigma$, la projection sur les actions colorées a et b, notée Π_{ab}, est définie par :

$$\forall x \in \Sigma_{cs} \cup \Sigma_{ct}, \Pi_{ab}(x) = \begin{cases} x \text{ si } \alpha(x) \in \{a, b\}, \\ \varepsilon \text{ sinon.} \end{cases}$$

Comme nous l'avons déjà dit, le coloriage est basé sur les projections sur des sous-alphabets de deux actions. Nous allons donc en premier lieu définir le coloriage de langages définis sur des alphabets de deux actions.

Définition 5.7 *Soit $\Sigma = \{a, b\}$ un alphabet d'actions. Le langage P_{ab}, donné par la Figure 5.1 est le coloriage du langage des mots st- primitifs sur Σ à l'aide de l'ensemble de couleurs $\mathrm{C}_{\{a,b\}} = \{1, 2, 3\}$. Le langage $\bar{\mathrm{P}}_{\{a,b\}}$ est construit de la même manière en utilisant $\bar{\mathrm{C}}_{\{a,b\}}$.*

Comment le langage P_{ab} est-il construit ? On considère l'automate non déterministe reconnaissant les mots st-primitifs sur l'alphabet $\{a, b\}$ donné par la Figure 5.2. Les premières couleurs sont choisies arbitrairement, nous pouvons commencer par $a_{1s}b_{2s}$ ou $b_{2s}a_{1s}$. Ensuite, on alterne les facteurs $a_t a_s$ et $b_t b_s$, de sorte que deux actions a (ou b) consécutives ne portent pas la même couleur et de sorte que deux actions a et b se déroulant en parallèle ne portent



Les indices 1, 2 et 3 représentent les couleurs de l'ensemble $C_{\{a,b\}}$. Le langage reconnu par cet automate est P_{ab}. Pour obtenir $\bar{P}_{ab}$, il suffit de remplacer les couleurs 1, 2 et 3 respectivement par 4, 5 et 6 qui représenteront les couleurs de l'ensemble $\bar{C}_{\{a,b\}}$.

FIG. 5.1 - *Le langage* P_{ab}

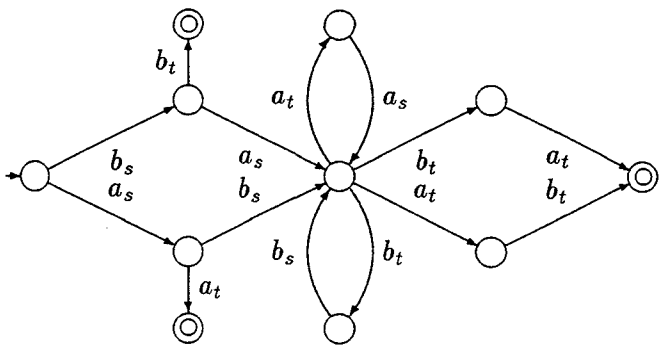


FIG. 5.2 - *Le langage des mots st-primitifs*

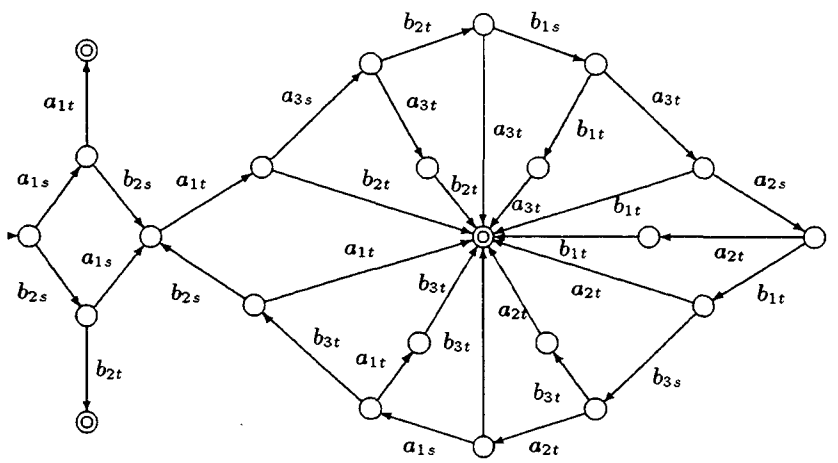


FIG. 5.3 - *Construction du coloriage*

pas la même couleur. On obtient ainsi l'automate donné Figure 5.3. Ensuite il suffit d'apporter quelques modifications pour autoriser les facteurs du type $(a_s a_t)^*$ ou $(b_s b_t)^*$. On remarque sur l'automate Figure 5.1 que ces facteurs itérants sont composés de lettres qui possèdent deux couleurs. En effet, si l'on considère par exemple un facteur

$$a_{1t}a_{3s}b_{2t}\mathbf{b}_{12s}\mathbf{b}_{12t}b_{1s}a_{3t}a_{2s},$$

l'action b_{12} ne peut ni commuter avec le premier a_1 ni avec le dernier a_2 pourtant, les occurrences d'actions correspondantes dans le mot de départ peuvent être voisines au cours d'une dérivation :

$$\mathbf{a}_t a_s b_t \mathbf{b}_s \mathbf{b}_t b_s a_t a_s \xrightarrow[\theta]{3} b_t \mathbf{a}_t \mathbf{b}_s a_s \mathbf{b}_t b_s a_t a_s,$$

$$a_t a_s b_t \mathbf{b}_s \mathbf{b}_t b_s a_t \mathbf{a}_s \xrightarrow[\theta]{3} a_t a_s b_t \mathbf{b}_s a_t \mathbf{b}_t \mathbf{a}_s b_s.$$

Il est donc nécessaire de marquer cette action b de la couleur 1 pour l'empêcher de commuter avec l'action a_1 qui la précède et de la couleur 2 pour l'empêcher de commuter avec l'action

a_2 qui la suit. Le même problème se pose pour tous les facteurs itérants du type $(a_s a_t)^*$ ou $(b_s b_t)^*$ c'est pourquoi toutes les actions correspondantes possèdent deux couleurs.

Remarquons quelques propriétés essentielles de ce coloriage qui seront utilisées par la suite et qui peuvent facilement être vérifiées sur l'automate. Chacune de ces propriétés est énoncée pour P_{ab} mais elle est également valable pour $\bar{\mathrm{P}}_{ab}$:

Propriété 5.8

a - *Le langage* P_{ab} *est un st-langage et* $\varphi_{\{a,b\}}(\mathrm{P}_{ab})$ *est égal à l'ensemble des mots st-primitifs sur* $\{a,b\}$.

b - *Pour une couleur donnée, la projection de* P_{ab} *sur le sous-alphabet des lettres qui possèdent cette couleur est une séquence.*

c - *Soit* (a_t,E) *et* (b_s,F) *deux lettres de* $\mathrm{alph}(\mathrm{P}_{ab})$. *Si* u *est un st-mot de* P_{ab} *de la forme* $u = u_1(a_t,E)u_2(b_s,F)u_3$ *avec* $E \cap F \cap \mathrm{C}_{ab} = \emptyset$, *alors* u_2 *contient un sous-mot* $(a_s,G)(a_t,G)$ *avec* $G \cap F \cap \mathrm{C}_{ab} \neq \emptyset$ *ou un sous-mot* $(b_s,G)(b_t,G)$ *avec* $G \cap E \cap \mathrm{C}_{ab} \neq \emptyset$.

En utilisant les deux langages que nous venons de présenter, nous pouvons définir la fonction rationnelle qui nous permet de colorier un langage à partir des coloriages sur les sous-alphabets de deux actions.

Définition 5.9 *Soit* Σ *un alphabet d'actions. Pour tout sous-alphabet de deux actions* $\mathrm{X} = \{a,b\} \subseteq \Sigma$, *nous notons* ψ_{ab} *le morphisme alphabétique :*

$$\begin{array}{rccl} \psi_{ab} & : & (\Sigma_{cs} \cup \Sigma_{ct})^* & \longrightarrow & (\mathrm{X}_{cs} \cup \mathrm{X}_{ct})^* \\ & & (x,E) & \longmapsto & (x, E \cap (\mathrm{C}_{\mathrm{X}} \cup \bar{\mathrm{C}}_{\mathrm{X}})) \text{ si } \alpha(x) \in \mathrm{X} \\ & & (x,E) & \longmapsto & \varepsilon \text{ sinon.} \end{array}$$

La fonction rationnelle τ_Σ *est définie par :*

$$\tau_\Sigma = \Big(\bigcap_{a,b \in \Sigma, a \neq b} \psi_{ab}^{-1}((\mathrm{P}_{ab}\bar{\mathrm{P}}_{ab})^*(\mathrm{P}_{ab} + \varepsilon)) \Big) \circ \varphi_\Sigma^{-1}.$$

Notons que pour un alphabet de deux actions $\Sigma = \{a,b\}$, nous obtenons simplement :

$$\tau_{\{a,b\}} = (\cap(\mathrm{P}_{ab}.\bar{\mathrm{P}}_{ab})^*(\mathrm{P}_{ab} + \varepsilon)) \circ \varphi_{\{a,b\}}^{-1},$$

c'est-à-dire que l'on colorie alternativement les facteurs st-primitifs en fonction de P_{ab} et de $\bar{\mathrm{P}}_{ab}$.

Enfin, nous pouvons définir la commutation partielle que nous allons utiliser. Cette commutation partielle est totalement liée au coloriage que nous avons choisi. Considérons par exemple deux lettres (a_t,E) et (b_s,F). Pour décider si ces deux lettres sont indépendantes ou non, il suffit de consulter les couleurs associées au sous-alphabet $\{a,b\}$. Si E est inclus dans $\mathrm{C}_{\{a,b\}}$ et F dans $\bar{\mathrm{C}}_{\{a,b\}}$, nous savons que ces deux lettres apparaissent dans deux facteurs st-primitifs différents, elles sont donc dépendantes. Sinon, comme nous utilisons des couleurs pour mettre en évidence certaines séquences, si E et F ne sont pas disjoints, les deux lettres appartiennent à une séquence et sont donc dépendantes. Dans les autres cas, nous autorisons la commutation. Nous obtenons donc la commutation partielle :

Définition 5.10 *Soit* Σ_c *un alphabet coloré. La commutation partielle* ϱ_Σ *est définie par :*

$$(((x,E),(y,F)) \in \varrho_\Sigma) \Leftrightarrow ([(x,y) \in \theta_\Sigma \text{ ou } (y,x) \in \theta_\Sigma]$$

$$et\ [(E \cup F) \cap \mathrm{C}_{\{\alpha(x),\alpha(y)\}} = \emptyset\ ou\ (E \cup F) \cap \bar{\mathrm{C}}_{\{\alpha(x),\alpha(y)\}} = \emptyset]$$
$$et\ [E \cap F \cap (\mathrm{C}_{\{\alpha(x),\alpha(y)\}} \cup \bar{\mathrm{C}}_{\{\alpha(x),\alpha(y)\}}) = \emptyset]).$$

Nous autorisons potentiellement trop de commutations, en particulier dans deux cas. Tout d'abord, dans le cas de facteurs st-primitifs différents. Par exemple le mot coloré correspondant au mot $a_s a_t a_s a_t b_s b_t$ est le mot

$$(a_s, \{1\})(a_t, \{1\})(a_s, \{4\})(a_t, \{4\})(b_s, \{2\})(b_t, \{2\})$$

(avec $\mathrm{C}_{\{a,b\}} = \{1, 2, 3\}$ et $\bar{\mathrm{C}}_{\{a,b\}} = \{4, 5, 6\}$). Le couple $((a_t, \{1\}), (b_s, \{2\}))$ appartient à $\varrho_{\{a,b\}}$, cependant ici ces deux lettres ne devraient pas être autorisées à commuter puisqu'elles n'appartiennent pas au même facteur st-primitif. Clairement, une telle commutation ne pourra jamais se produire puisque deux lettres appartenant à deux facteurs st-primitifs différents et colorées avec le même ensemble de couleurs sont toujours séparées par les lettres d'un facteur st-primitif coloré avec l'autre ensemble de couleurs.

Le deuxième cas concerne des occurences de lettres appartenant à un même facteur st-primitif. Considérons le mot $\mathbf{a}_s b_s \mathbf{a}_t a_s b_t b_s a_t a_s b_t \mathbf{b}_s \mathbf{b}_t a_t$, la première occurrence de l'action a et la dernière occurrence de l'action b sont en séquence et on ne pourra jamais commuter le a_t et le b_s correspondant dans une dérivation par θ. Cependant, le mot coloré correspondant est :

$$(a_s, \{1\})(b_s, \{2\})(a_t, \{1\})(a_s, \{3\})(b_t, \{2\})(b_s, \{1\})$$
$$(a_t, \{3\})(a_s, \{2\})(b_t, \{1\})(b_s, \{3\})(b_t, \{3\})(a_t, \{2\}).$$

Ici, les actions correspondant à la première occurrence de l'action a et à la dernière occurrence de l'action b sont a_1 et b_3 or, $((a_t, \{1\}), (b_s, \{3\}))$ appartient à $\varrho_{\{a,b\}}$. Le coloriage est construit de façon à ce que deux telles occurrences ne puissent jamais être voisines, la Propriété 5.8c nous assure que les occurrences considérées sont séparées, dans le mot de départ, par un sous-mot qui constitue un chemin permettant de passer de l'une à l'autre dans le graphe de non commutation de la commutation partielle et qui donc les séparera dans tout mot dérivé du mot de départ. Dans notre exemple, ce sous-mot est $(b_s, \{1\})(b_t, \{1\})$ ou $(a_s, \{3\})(a_s, \{3\})$.

Nous allons à présent montrer que l'on peut effectivement utiliser le coloriage et la commutation partielle pour calculer la clôture d'un langage par θ, en commençant par le cas particulier de langages définis sur des alphabets de deux actions.

Lemme 5.11 *Soient Σ un alphabet de deux actions et L un st-langage sur $\Sigma_s \cup \Sigma_t$. Nous avons :*

$$\mathrm{f}_{\theta_\Sigma}(L) = \varphi_\Sigma(\mathrm{f}_{\varrho_\Sigma}(\tau_\Sigma(L))).$$

Preuve. Posons $\Sigma = \{a, b\}$. Clairement, il suffit de montrer que, pour tout mot u de L, nous avons :

$$\mathrm{f}_{\theta_\Sigma}(u) = \varphi_\Sigma(\mathrm{f}_{\varrho_\Sigma}(\tau_\Sigma(u))).$$

Considérons la décomposition en facteurs st-primitifs de u, $u = u_1 \dots u_n$. Nous avons $\tau_\Sigma(u) = u'_1 \dots u'_n$ avec, pour tout indice impair, $u'_i = \varphi_\Sigma^{-1}(u_i) \cap \mathrm{P}_{ab}$ et, pour tout indice pair, $u'_i = \varphi_\Sigma^{-1}(u_i) \cap \bar{\mathrm{P}}_{ab}$. Comme nous avons les égalités $\mathrm{f}_{\varrho_\Sigma}(\tau_\Sigma(u)) = \mathrm{f}_{\varrho_\Sigma}(u'_1) \dots \mathrm{f}_{\varrho_\Sigma}(u'_n)$ et $\mathrm{f}_{\theta_\Sigma}(u) = \mathrm{f}_{\theta_\Sigma}(u_1) \dots \mathrm{f}_{\theta_\Sigma}(u_n)$, il suffit de montrer l'égalité $\mathrm{f}_{\theta_\Sigma}(u) = \varphi_\Sigma(\mathrm{f}_{\varrho_\Sigma}(\tau_\Sigma(u)))$ pour un mot u st-primitif et $\tau_\Sigma(u)$ appartenant à P_{ab}.

Montrons l'inclusion de gauche à droite par induction sur la longueur de la dérivation. Soit v appartenant à $\mathrm{f}_{\theta_\Sigma}(u)$. Clairement, l'inclusion est vérifiée pour une dérivation de longueur nulle. Dans les autres cas, d'après le Corollaire 1.25, nous pouvons considérer une dérivation de longueur minimale,

$$u \xrightarrow[\theta_\Sigma]{n} w = v_1xyv_2 \xrightarrow[\theta_\Sigma]{} v = v_1yxv_2.$$

Par hypothèse de récurrence, nous avons

$$\tau_\Sigma(u) \xrightarrow[\varrho_\Sigma]{n} w' = v_1'(x,E)(y,F)v_2',$$

avec $\varphi_\Sigma(v_1') = v_1$ et $\varphi_\Sigma(v_2') = v_2$. Comme (x,y) appartient à θ_Σ, $((x,E),(y,F))$ appartient à ϱ_Σ si $E \cap F \cap \mathrm{C}_{\{a,b\}} = \emptyset$, nous allons voir que c'est toujours le cas. Supposons que c appartient à $E \cap F \cap \mathrm{C}_{\{a,b\}}$ et posons $\mathrm{X} = \{(z,G) \in \Sigma_{cs} \cup \Sigma_{ct} \mid c \in G\}$. D'après la Propriété 5.8b, $\Pi_{\mathrm{X}}(\tau_\Sigma(u))$ est une séquence. Par définition de ϱ_Σ et d'après le Lemme de Projection pour les commutations partielles 1.19, les projections sur un tel alphabet X de w' et de $\tau_\Sigma(u)$ sont égales. C'est pourquoi (x,E) et (y,F), qui ne peuvent pas appartenir à une séquence (puisque $(x,y) \in \theta_\Sigma$), n'ont aucune couleur en commun pour l'alphabet $\{a,b\}$. Nous avons donc

$$\tau_\Sigma(u) \xrightarrow[\varrho_\Sigma]{n} w' = v_1'x'y'v_2' \xrightarrow[\varrho_\Sigma]{} v' = v_1'y'x'v_2' \in \varphi_\Sigma^{-1}(v).$$

Nous allons à présent montrer l'inclusion de droite à gauche par induction sur la longueur de la dérivation par ϱ_Σ. Soit v appartenant à $\mathrm{f}_{\varrho_\Sigma}(\tau_\Sigma(u))$. Clairement, l'inclusion est vraie pour $v = \tau_\Sigma(u)$ sinon, nous avons

$$\tau_\Sigma(u) \xrightarrow[\varrho_\Sigma]{n} w = v_1xyv_2 \xrightarrow[\varrho_\Sigma]{} v = v_1yxv_2.$$

Par hypothèse de récurrence, nous avons

$$u \xrightarrow[\theta_\Sigma]{n} w' = v_1'x'y'v_2',$$

avec $\varphi_\Sigma(v_1) = v_1'$, $\varphi_\Sigma(v_2) = v_2'$, $\varphi_\Sigma(x) = x'$ et $\varphi_\Sigma(y) = y'$. Nous avons immédiatement $(x',y') \in \theta_\Sigma$ excepté quand $\varphi_\Sigma(x)$ appartient à Σ_t et $\varphi_\Sigma(y)$ à Σ_s. Nous allons voir que ce cas ne peut survenir. Posons $x = (a_t,E)$ et $y = (b_s,F)$. Si (x,y) appartient à ϱ, nous avons $E \cap F \cap \mathrm{C}_{\{a,b\}} = \emptyset$. Comme ϱ_Σ est une commutation partielle, d'après le Corollaire 1.25, nous pouvons considérer une dérivation minimale donc x et y sont dans cet ordre dans $\tau_\Sigma(u)$, nous avons $\tau_\Sigma(u) = u_1xu_2yu_3$. D'après la Propriété 5.8c, ces deux occurrences x et y ne peuvent jamais être consécutives dans un mot de $\mathrm{f}_{\varrho_\Sigma}(\tau_\Sigma(u))$. □

Pour étendre ce résultat aux langages définis sur des alphabets d'actions quelconques, nous allons utiliser la notion d'image d'une semi-commutation par un morphisme strictement alphabétique.

Définition 5.12 (Clerbout, Latteux [13]) *Soit θ une semi-commutation définie sur un alphabet* X *et σ une substitution alphabétique de* X^* *dans les parties de* Σ^*. *L'image de θ par σ est le système défini par :*

$$\{(y_1,y_2) \mid y_1,y_2 \in \Sigma, y_1 \neq y_2, (x_1,x_2) \in \theta \text{ et } y_1 \in \sigma(x_1), y_2 \in \sigma(x_2)\}.$$

Lemme 5.13 (Roos 89 [54]) *Soient θ une semi-commutation définie sur l'alphabet* X *et g un morphisme strictement alphabétique de Υ^* dans* X^*. *En notant θ' l'image de θ par g^{-1}, nous avons :*

$$g^{-1} \circ \mathrm{f}_\theta = \mathrm{f}_{\theta'} \circ g^{-1} \text{ et } \mathrm{f}_{\theta'} \circ g = g \circ \mathrm{f}_\theta.$$

En utilisant ces notions, nous montrons la proposition suivante :

Proposition 5.14 *Soit L un st-langage sur l'alphabet $\Sigma_s \cup \Sigma_t$. Nous avons :*

$$\mathrm{f}_{\theta_\Sigma}(L) = \varphi_\Sigma(\mathrm{f}_{\varrho_\Sigma}(\tau_\Sigma(L))).$$

Preuve. Montrons tout d'abord l'inclusion $\varphi_\Sigma(\mathrm{f}_{\varrho_\Sigma}(\tau_\Sigma(L))) \subseteq \mathrm{f}_{\theta_\Sigma}(L)$. Soient u un mot de L et v un mot de $\varphi_\Sigma(\mathrm{f}_{\varrho_\Sigma}(\tau_\Sigma(u)))$. Nous avons :

$$u \xrightarrow{\tau_\Sigma} u' \xrightarrow[\varrho]{*} u'' \xrightarrow{\varphi_\Sigma} v.$$

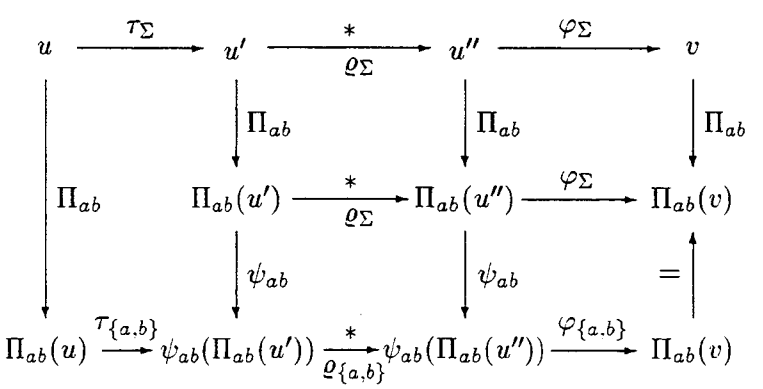
D'après le Lemme de Projection pour les commutations partielles 1.19, pour toute paire d'actions a et b, nous avons :

$$\Pi_{ab}(u') \xrightarrow[\varrho]{*} \Pi_{ab}(u'').$$

Considérons $\Pi_{ab}(u)$, clairement, nous avons :

$$\tau_{\{a,b\}}(\Pi_{ab}(u)) = \psi_{ab}(\Pi_{ab}(u')).$$

La restriction de la commutation partielle ϱ_Σ à l'alphabet de $\Pi_{ab}(u')$ est égale à $\psi_{ab}^{-1}(\varrho_{\{a,b\}})$. En conséquence, d'après le Lemme 5.13, nous avons $\mathrm{f}_{\varrho_{\{a,b\}}}(\psi_{ab}(\Pi_{ab}(u'))) = \psi_{ab}(\mathrm{f}_{\varrho_{\{a,b\}}}(\Pi_{ab}(u')))$. Nous avons :



Il est clair que $\varphi_{\{a,b\}}(\psi_{ab}(\Pi_{ab}(u')))$ et $\varphi_\Sigma(\Pi_{ab}(u''))$ sont égaux. D'après le Lemme 5.11, $\Pi_{ab}(v)$ appartient à $\mathrm{f}_{\theta_\Sigma}(\Pi_{ab}(u))$, donc nous pouvons appliquer le Lemme de Projection pour les commutations partielles 1.19 et nous obtenons $v \in \mathrm{f}_{\theta_\Sigma}(u)$.

Il nous reste à montrer l'inclusion $\mathrm{f}_{\theta_\Sigma}(L) \subseteq \varphi_\Sigma(\mathrm{f}_{\varrho_\Sigma}(\tau_\Sigma(L)))$. Soient u un mot de L et v un mot de $\mathrm{f}_{\theta_\Sigma}(u)$. Notons w le mot de $\varphi_\Sigma^{-1}(v)$ tel que pour toute action a,

$$(w = w_1 x w_2 \text{ avec } \alpha(x) = a)$$

$$\Updownarrow$$

$$(\tau_\Sigma(u) = u_1' x u_2' \text{ avec } |\varphi_\Sigma(u_1')|_{a_s} + |\varphi_\Sigma(u_1')|_{a_t} = |\varphi_\Sigma(w_1)|_{a_s} + |\varphi_\Sigma(w_1)|_{a_t}).$$

Nous allons montrer que w appartient à $\mathrm{f}_\varrho(\tau_\Sigma(u))$. D'après le Lemme de Projection 4.7, pour toute paire d'actions a et b, $\Pi_{ab}(v)$ appartient à $\mathrm{f}_{\theta_\Sigma}(\Pi_{ab}(u))$. Donc, d'après le Lemme 5.11, $\Pi_{ab}(v)$ appartient à $\varphi_{\{a,b\}}(\mathrm{f}_{\varrho_{\{a,b\}}}(\tau_{\{a,b\}}(\Pi_{ab}(u))))$, nous avons:

$$\Pi_{ab}(u) \xrightarrow{\tau_{\{a,b\}}} u_1 \underset{\varrho_{\{a,b\}}}{\xrightarrow{*}} u_2 \xrightarrow{\varphi_{\{a,b\}}} \Pi_{ab}(v).$$

Posons $u' = \tau_\Sigma(u)$. Clairement, nous avons $\psi_{ab}(\Pi_{ab}(u')) = u_1$. De plus, la restriction de ϱ_Σ à l'alphabet de $\Pi_{ab}(u')$ est l'image de $\varrho_{\{a,b\}}$ par ψ_{ab}^{-1} donc, nous pouvons appliquer le Lemme 5.13 et nous obtenons:

$$\begin{array}{ccccc}
u_1 & \underset{\varrho_{\{a,b\}}}{\xrightarrow{*}} & u_2 & \xrightarrow{\varphi_{ab}} & \Pi_{ab}(v) \\
\uparrow{\scriptstyle \psi_{ab}} & & \uparrow{\scriptstyle \psi_{ab}} & & \uparrow{\scriptstyle =} \\
\Pi_{ab}(u') & \underset{\varrho}{\xrightarrow{*}} & u_2' & \xrightarrow{\varphi} & \Pi_{ab}(v)
\end{array}$$

Par construction, nous avons

$$\Pi_{ab}(w) = u_2',$$

pour toute paire d'actions a et b donc, nous pouvons appliquer le Lemme de Projection pour les commutations partielles 1.19 et nous obtenons $w \in \mathrm{f}_{\varrho_\Sigma}(u')$. Nous avons:

$$\begin{array}{ccccccc}
\Pi_{ab}(u) & \xrightarrow{\tau_{\{a,b\}}} & u_1 & \underset{\varrho_{\{a,b\}}}{\xrightarrow{*}} & u_2 & \xrightarrow{\varphi_{ab}} & \Pi_{ab}(v) \\
\uparrow{\scriptstyle \Pi_{ab}} & & \uparrow{\scriptstyle \psi_{ab}} & & \uparrow{\scriptstyle \psi_{ab}} & & \uparrow{\scriptstyle =} \\
\Pi_{ab}(u') & & \Pi_{ab}(u') & \underset{\varrho}{\xrightarrow{*}} & u_2' & \xrightarrow{\varphi} & \Pi_{ab}(v) \\
\uparrow{\scriptstyle \Pi_{ab}} & & \uparrow{\scriptstyle \Pi_{ab}} & & \uparrow{\scriptstyle \Pi_{ab}} & & \uparrow{\scriptstyle \Pi_{ab}} \\
u & \xrightarrow{\tau} & u' & \underset{\varrho}{\xrightarrow{*}} & w & \xrightarrow{\varphi} & v
\end{array}$$

En conclusion, v appartient à $\varphi_\Sigma(\mathrm{f}_{\varrho_\Sigma}(\tau_\Sigma(u)))$. □

Nous avons montré que, pour calculer la clôture par θ d'un st-langage, il est possible d'utiliser une fonction rationnelle pour colorier le langage, de calculer la clôture par une commutation partielle et enfin d'appliquer un morphisme d'actions. Nous allons montrer que la clôture par ϱ du langage coloré est l'image par un st-morphisme d'un langage de synchronisation généralisé en utilisant les automates asynchrones et le produit de mixage.

5.2.2 Automates asynchrones et produit de mixage

Pour montrer que les st-langages réguliers clos par θ sont des langages de synchronisation généralisés, nous allons nous ramener à des langages clos par une commutation partielle et utiliser les automates asynchrones pour les décomposer en produits de mixage de langages de synchronisation. Nous allons donc, dans un premier temps, faire quelques rappels à propos des automates asynchrones et du produit de mixage.

Les automates asynchrones, introduits par Zielonka, sont utilisés pour reconnaître des langages réguliers clos par une commutation partielle. Chaque état d'un automate asynchrone est un vecteur de composantes locales. Chaque lettre de l'alphabet peut uniquement lire et modifier un sous-ensemble des composantes locales qui correspond à son domaine d'actions. Deux lettres indépendantes agissent sur des sous-ensembles de composantes disjoints et peuvent donc être exécutées dans un ordre quelconque.

Définition 5.15 (Zielonka [60]) *Un automate asynchrone déterministe (DAA) $\mathcal{A}$ sur les alphabets $(\Sigma_1, \ldots, \Sigma_n)$ est un quintuplet*

$$\mathcal{A} = ((\Sigma_1, \ldots, \Sigma_n), \bigotimes_{1 \leq i \leq n} Q_i, \delta, q_I, F)$$

où :

- $(\Sigma_1, \ldots, \Sigma_n)$ *est un n-uplet d'alphabets,* $\Sigma = \bigcup_{1 \leq i \leq n} \Sigma_i$ *est l'alphabet de* $\mathcal{A}$ *;*
- *l'ensemble d'états globaux* $\bigotimes_{1 \leq i \leq n} Q_i$ *est le produit direct des ensembles d'états locaux* Q_i *pour* $1 \leq i \leq n$ *;*
- q_I *est l'état (global) initial et* F *l'ensemble d'états (globaux) finaux ;*
- *le domaine d'une lettre* $a \in \Sigma$ *est* $I_a = \{1 \leq i \leq n \mid a \in \Sigma_i\}$ *;*
- *la fonction de transition* δ *est définie par les fonctions de transition partielles. Pour toute lettre* a*, la fonction partielle* δ_a *est un ensemble de transitions étiquetées par la lettre* a *:*

  $$\delta_a \subseteq (\bigotimes_{i \in I_a} Q_i) \times a \times (\bigotimes_{i \in I_a} Q_i),$$

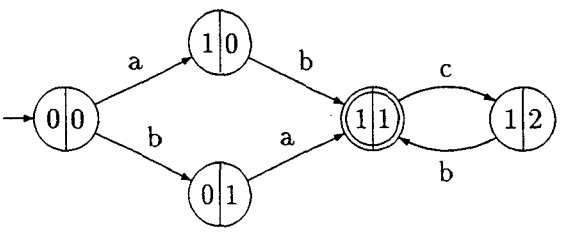
  et pour tout $q \in \bigotimes_{i \in I_a} Q_i$*, il existe au plus un* $q' \in \bigotimes_{i \in I_a} Q_i$ *tel que* $(q, a, q') \in \delta_a$*. La fonction* δ *est définie par :* $(q_1, \ldots, q_n), a, (q'_1, \ldots, q'_n) \in \delta$ *si et seulement si la transition partielle* $(((q_i)_{i \in I_a}), a, ((q'_i)_{i \in I_a}))$ *appartient à* δ_a *et si pour tout* $i \notin I_a$*,* $q_i = q'_i$*.*

Tout DAA induit une commutation partielle et reconnaît un langage clos par cette commutation partielle.

Considérons par exemple le langage défini par l'automate asynchrone

$$\mathcal{A} = ((\{a, c\}, \{b, c\}), Q_1 \times Q_2, \delta, (0, 0), \{(1, 1)\})$$

où $Q_1 = \{0,1\}$, $Q_2 = \{0,1,2\}$ et où δ est définie par les fonctions partielles $\delta_a = \{(0,a,1)\}$, $\delta_b = \{(0,b,1),(2,b,1)\}$ et $\delta_c = \{((1,1),c,(1,2))\}$:



Le langage reconnu par $\mathcal{A}$ est le langage $(ab + ba)(cb)^*$ qui est clos par la commutation partielle $\{(a,b)\}$, les lettres a et b agissent respectivement sur les première et deuxième composantes des états et on peut constater qu'à tout moment (ici uniquement dans l'état $(0,0)$) les lectures de ab et ba mènent dans le même état.

Le lien entre les automates asynchrones et les commutations partielles est établi par le théorème de Zielonka :

Théorème 5.16 (Zielonka [60]) *Tout langage régulier clos par une commutation partielle ϱ est reconnu par un automate asynchrone déterministe dont ϱ est la commutation partielle induite.*

Nous aurons besoin d'automates asynchrones définis sur des alphabets de notre choix, c'est pourquoi nous utiliserons des résultats plus précis permettant de « choisir » l'automate que nous utilisons.

Les automates 2-asynchrones sont des automates asynchrones tels que tout alphabet partiel Σ_i contient au plus deux lettres. Le théorème suivant établit que les automates 2-asynchrones ont la même puissance que les automates asynchrones.

Théorème 5.17 (Cori, Sopena, Latteux et Roos [19]) *Pour tout automate asynchrone $\mathcal{A}$, il existe un automate 2-asynchrone $\mathcal{A}'$ reconnaissant le même langage et tel que les domaines de deux lettres pour $\mathcal{A}$ sont disjoints si et seulement si les domaines de ces deux lettres pour $\mathcal{A}'$ sont disjoints.*

La construction donnée dans la preuve de ce théorème permet d'obtenir, à partir d'un automate asynchrone sur un alphabet $(\Sigma_1,\ldots,\Sigma_n)$ un automate 2-asynchrone sur $(\mathrm{X}_1,\ldots,\mathrm{X}_m)$ tel que pour tout indice i, quel que soit le sous-alphabet $\{a,b\}$ de Σ_i, il existe un indice j tel que $\{a,b\} = \mathrm{X}_j$. Ce théorème, associé au précédent, nous permet donc d'obtenir le corollaire :

Corollaire 5.18 *Soit L un langage régulier clos par une commutation partielle. Il existe un automate asynchrone reconnaissant L sur tout n-uplet d'alphabets formant un recouvrement par cliques du graphe de non commutation de la commutation partielle.*

Comme nous l'avons déjà dit, nous allons utiliser les automates asynchrones pour montrer que la clôture par ϱ d'un st-langage régulier coloré est l'image par un st-morphisme d'un langage de synchronisation généralisé. Plus précisément, nous allons utiliser une sous-classe des automates asynchrones définie par Duboc et liée au produit de mixage. La définition du

produit de mixage est celle de Duboc [25] qui est une version légèrement modifée de celle de De Simone [20] (ceci pour obtenir un produit associatif) :

Définition 5.19 *Soient* X *et* Σ *deux alphabets,* L *un langage de* X^* *et* M *un langage de* Σ^*. *On appelle produit de mixage de* L *et* M *dans* X *et* Σ, *l'ensemble* $L \sqcap M$ *défini par :*

$$L \sqcap M = \{u \in (X \cup \Sigma)^* \mid \Pi_X(u) \in L \text{ et } \Pi_\Sigma(u) \in M\}.$$

Duboc donne également dans [25] la définition des automates faiblement mixés (appelés « *loosly cooperating finite asynchronous automata* » par Zielonka [60]). Un automate faiblement mixé est un automate asynchrone avec une définition plus contraignante des fonctions de transition partielles, une lettre transforme toujours une composante de la même manière quelles que soient les autres composantes de son domaine.

Définition 5.20 *Un automate faiblement mixé déterministe (DLCA)* $\mathcal{A}$ *sur les alphabets* $(\Sigma_1, \ldots, \Sigma_n)$ *est un quintuplet*

$$\mathcal{A} = ((\Sigma_1, \ldots, \Sigma_n), \bigotimes_{1 \leq i \leq n} Q_i, \delta, q_I, F)$$

où :

- $(\Sigma_1, \ldots, \Sigma_n)$ *est un n-uplet d'alphabets,* $\Sigma = \bigcup_{1 \leq i \leq n} \Sigma_i$ *est l'alphabet de* $\mathcal{A}$;
- *l'ensemble d'états globaux* $\bigotimes_{1 \leq i \leq n} Q_i$ *est le produit direct des ensembles d'états locaux* Q_i *pour* $1 \leq i \leq n$;
- q_I *est l'état (global) initial et* F *l'ensemble d'états (globaux) finaux ;*
- *le domaine d'une lettre* $a \in \Sigma$ *est* $I_a = \{1 \leq i \leq n \mid a \in \Sigma_i\}$;
- *la fonction de transition* δ *est définie à partir des fonctions locales* δ_i. *Pour tout indice* $1 \leq i \leq n$, *la fonction locale* δ_i *est une partie de* $Q_i \times \Sigma_i \times Q_i$ *et, pour tout* q_i *de* Q_i *et tout* x *de* Σ_i, *il existe au plus un* q_i' *de* Q_i *tel que* (q_i, x, q_i') *appartient à* δ_i. *La fonction de transition* δ *est définie par :* $(q_1, \ldots, q_n), x, (q_1', \ldots, q_n') \in \delta$ *si et seulement si pour tout* $i \notin I_x$, $q_i = q_i'$ *et si pour tout* $i \in I_x$, $(q_i, x, q_i') \in \delta_i$.

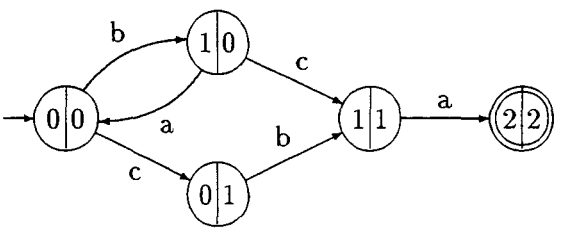
Le lien entre automates faiblement mixés et produit de mixage est établi par le théorème:

Théorème 5.21 (Duboc [25]) *Tout langage faiblement mixé (reconnu par un automate faiblement mixé) est une union finie de langages mixés.*

Cette union de langages mixés peut effectivement être construite. Dans la définition ci-dessus si $q_I = (q_1, \ldots, q_n)$, pour tout $f = (f_1, \ldots, f_n) \in F$ et tout $1 \leq i \leq n$, on pose $\mathcal{B}_f^i = (\Sigma_i, Q_i, q_i, f_i, \delta_i)$ et on a :

$$L(\mathcal{A}) = \bigcup_{f \in F} L(\mathcal{B}_f^1) \sqcap \ldots \sqcap L(\mathcal{B}_f^n).$$

La particularité d'un automate faiblement mixé est qu'une lettre donnée modifie toujours une composante indépendamment des autres composantes de son domaine. Par exemple, l'automate asynchrone suivant n'est pas un automate faiblement mixé :

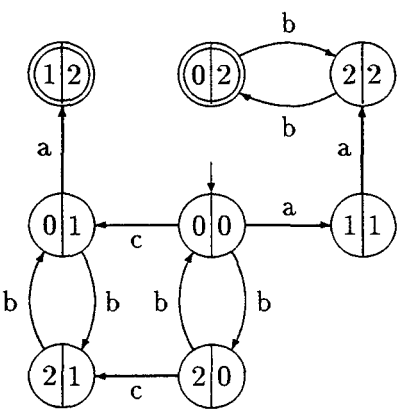


En effet, la fonction partielle δ_a contient les transitions $((1,0),a,(0,0))$ et $((1,1),a,(2,2))$, la lettre a permet donc de passer de l'état 1 en première composante aux états 0 ou 2 selon la deuxième composante. Par contre, l'automate asynchrone

$$\mathcal{A} = (\{\{a,b\},\{a,c\}), Q_1 \times Q_2, \delta, (0,0), \{(0,2),(1,2)\})$$

où $Q_1 = \{0,1,2\}$, $Q_2 = \{0,1,2\}$ et où δ est définie par les fonctions locales

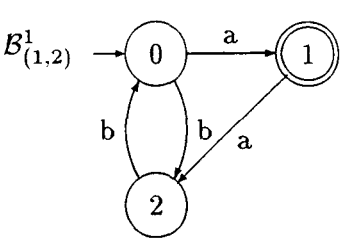
$\delta_1 = \{(0,a,1),(1,a,2),(0,b,2),(2,b,0)\}$ et $\delta_2 = \{(0,a,1),(1,a,2),(0,c,1)\}$:

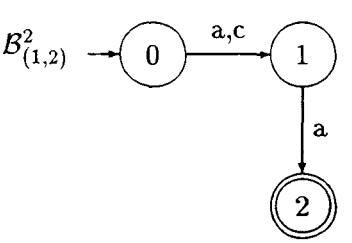


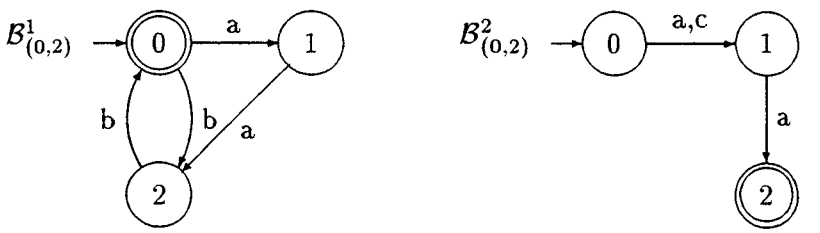
est un automate faiblement mixé. Le langage reconnu par cet automate peut donc être décomposé en produits de mixage d'après la construction donnée par Duboc. Nous avons :

$$L(\mathcal{A}) = [L(\mathcal{B}^1_{(1,2)}) \text{ m } L(\mathcal{B}^2_{(1,2)})] \cup [L(\mathcal{B}^1_{(0,2)}) \text{ m } L(\mathcal{B}^2_{(0,2)})]$$

avec







5.2.3 Construction

Arrivés à ce point, pour un st-langage régulier L sur l'alphabet d'actions Σ, nous avons $\mathrm{f}_{\theta_\Sigma}(L) = \varphi_\Sigma(\mathrm{f}_{\varrho_\Sigma}(\tau_\Sigma(L)))$. La seconde étape de la construction consiste à montrer que le langage $\mathrm{f}_{\varrho_\Sigma}(\tau_\Sigma(L))$ appartient à $\Phi_{\mathrm{st}}(\mathrm{LS_G})$. Comme les langages de synchronisation généralisés sont des langages réguliers, il est utile de s'intéresser en premier lieu à la régularité des langages que nous considérons. Nous montrons que, dans le cas présent, $\mathrm{f}_{\varrho_\Sigma}(\tau_\Sigma(L))$ est un langage régulier.

Lemme 5.22 *Soit L un st-langage régulier défini sur l'alphabet d'actions Σ. Le langage $\mathrm{f}_{\varrho_\Sigma}(\tau_\Sigma(L))$ est un langage régulier.*

Preuve. Soit L un st-langage régulier défini sur un alphabet d'actions Σ. Nous allons tout d'abord considérer le cas où Σ est un alphabet de deux actions, posons $\Sigma = \{a, b\}$. Tout facteur itérant de $\tau_{\{a,b\}}(L)$ est un facteur itérant de P_{ab}, $\bar{\mathrm{P}}_{ab}$, ou un mot de $(\mathrm{P}_{ab} + \bar{\mathrm{P}}_{ab})^*$. Si u est un facteur itérant de P_{ab} (ou $\bar{\mathrm{P}}_{ab}$), nous avons $\mathrm{alph}(u) = \{(a_s, E), (a_t, E)\}$, $\mathrm{alph}(u) = \{(b_s, F), (b_t, F)\}$ ou $\{(x, E) \mid |E| = 1\} \subseteq \mathrm{alph}(u) \subseteq \mathrm{alph}(\mathrm{P}_{ab})$: dans chacun des cas, le graphe de non commutation de u est connexe. Les mots de P_{ab} ou $\bar{\mathrm{P}}_{ab}$ sont connexes et, comme chaque lettre de l'alphabet de P_{ab} est connectée à toutes les lettres de l'alphabet de $\bar{\mathrm{P}}_{ab}$ et vice-versa, les mots de $(\mathrm{P}_{ab} + \bar{\mathrm{P}}_{ab})^*$ sont connexes.

Revenons au cas où Σ est un alphabet fini quelconque. Soit u un facteur itérant de L. Considérons la partition de l'alphabet de u, $\mathrm{alph}(u) = \mathrm{X}_1 \cup \cdots \cup \mathrm{X}_n$ telle que :

$$(\exists 1 \leq k \leq n, x, y \in \mathrm{X}_k) \Leftrightarrow (\alpha(x) = \alpha(y)).$$

Clairement, les X_k sont des cliques du graphe de non commutation de u. De plus, pour toute paire d'actions a et b de l'alphabet d'actions de u, $\Pi_{ab}(u)$ est un facteur itérant de $\Pi_{ab}(L)$, donc $\mathrm{alph}(\Pi_{ab}(u))$ est une composante connexe du graphe de non commutation de u c'est-à-dire que les cliques X_k sont deux-à-deux connectées. En conséquence, le mot u est connexe. Comme tout facteur itérant de $\tau_\Sigma(L)$ est connexe, d'après le Théorème 1.29, $\mathrm{f}_{\varrho_\Sigma}(\tau_\Sigma(L))$ est régulier.

□

Dans la mesure où nous considérons des langages réguliers, nous allons pouvoir utiliser les automates asynchrones. Duboc donne dans [25] une construction permettant de construire, à partir d'un automate asynchrone reconnaissant un langage L, un automate faiblement mixé reconnaissant un langage dont l'image par un morphisme strictement alphabétique est L. Nous allons utiliser la même idée pour construire un langage reconnu par un automate faiblement mixé que nous pourrons décomposer en produits de mixage. Cependant, nous ne pouvons pas utiliser directement la construction de Duboc, il nous faut trouver une construction qui

préserve les st-langages et, en premier lieu, montrer que les langages de synchronisation généralisés sont clos par produit de mixage.

Lemme 5.23 *Le produit de mixage dans* $\Sigma_{1s} \cup \Sigma_{1t}, \ldots, \Sigma_{ns} \cup \Sigma_{nt}$ *de* n *langages de synchronisation généralisés respectivement définis sur chacun des* n *alphabets est un langage de synchronisation généralisé.*

Preuve. Soient $L_1, \ldots, L_n$ des langages de synchronisation généralisés respectivement définis sur $\Sigma_{1s} \cup \Sigma_{1t}, \ldots, \Sigma_{ns} \cup \Sigma_{nt}$. Par définition, un mot u appartient au produit de mixage de ces langages si et seulement si pour tout $1 \leq i \leq n$, la projection de u sur $\Sigma_{is} \cup \Sigma_{it}$ appartient à L_i. Posons $\Sigma = \Sigma_1 \cup \cdots \cup \Sigma_n$, nous avons:

$$L_1 \sqcap \ldots \sqcap L_n = \bigcap_{1 \leq i \leq n} L_i \sqcup ((\Sigma_s \cup \Sigma_t) \setminus (\Sigma_{is} \cup \Sigma_{it}))^*.$$

Comme les langages de synchronisation généralisés sont des st-langages, nous pouvons utiliser des st-langages pour calculer le produit de mixage, nous pouvons remplacer le langage $((\Sigma_s \cup \Sigma_t) \setminus (\Sigma_{is} \cup \Sigma_{it}))^*$ par l'ensemble de ses st-mots:

$$L_1 \sqcap \ldots \sqcap L_n = \bigcap_{1 \leq i \leq n} L_i \sqcup \mathrm{ST}_{\Sigma \setminus \Sigma_i}.$$

Donc, le produit de mixage de $L_1, \ldots, L_n$ est un langage de synchronisation généralisé. □

Nous allons montrer qu'un st-langage régulier reconnu par un automate asynchrone sur des cliques telles que la projection du langage sur chacune des cliques est une séquence est l'image par un st-morphisme d'un langage de synchronisation généralisé. Pour cela, nous proposons une construction qui permet de transformer l'automate asynchrone en automate faiblement mixé qui reconnaît également un st-langage. Ensuite, en utilisant le Théorème 5.21 et le résultat précédent, nous pourrons montrer que le langage reconnu par l'automate faiblement mixé est un langage de synchronisation généralisé.

Proposition 5.24 *Soit* L *un st-langage sur l'alphabet d'actions* Σ. *Soient* $\Sigma_1, \ldots, \Sigma_n \subseteq \Sigma$ *des alphabets d'actions tels que pour tout* i, $\Pi_{\Sigma_{is} \cup \Sigma_{it}}(L)$ *décrit une séquence d'actions de* Σ_i. *Si* L *est reconnu par un automate asynchrone déterministe sur* $(\Sigma_{1s} \cup \Sigma_{1t}), \ldots, (\Sigma_{ns} \cup \Sigma_{nt})$ *alors* L *appartient à* $\Phi_{\mathrm{st}}(\mathrm{LS_G})$.

Preuve. Considérons $\mathcal{A} = ((\Sigma_{1s} \cup \Sigma_{1t}, \ldots, \Sigma_{ns} \cup \Sigma_{nt}), \bigotimes_{1 \leq i \leq n} Q_i, \delta, q_0, F)$, un automate asynchrone déterministe reconnaissant L dont les états sont tous accessibles et co-accessibles.

Cet automate possède quelques propriétés particulières que nous allons utiliser pour construire un automate faiblement mixé $\mathcal{A}'$ reconnaissant un langage de synchronisation généralisé dont l'image par un morphisme est le langage L. Comme la projection du langage L sur chaque sous-alphabet $\Sigma_{is} \cup \Sigma_{it}$ est une séquence, dans chaque mot du langage, un début d'action et la fin d'action correspondante peuvent uniquement être séparés par des lettres qui n'appartiennent à aucun des sous-alphabets associés au domaine de cette action. Cette propriété est essentielle puisqu'elle nous permet de prédire les transitions qui peuvent suivre une transition donnée. Supposons qu'une action a appartienne à $\Sigma_1, \ldots, \Sigma_p$, alors, pour toute transition

$$((q_1, \ldots, q_p), a_s, (q'_1, \ldots, q'_p)) \in \delta_{a_s},$$

il existe une transition

$$((q'_1, \ldots, q'_p), a_t, (q''_1, \ldots, q''_p)) \in \delta_{a_t}.$$

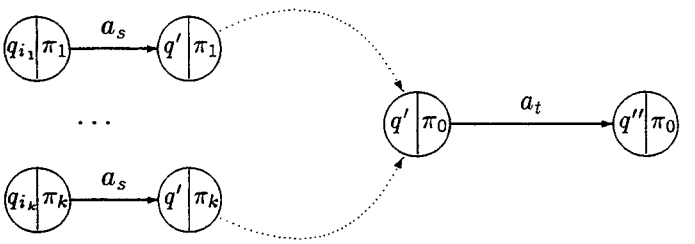
Cette propriété est même plus forte : le début d'une action ne peut être suivi que de la fin d'action correspondante, c'est-à-dire que s'il existe une transition

$$((q_1, \ldots, q_p, q_{p+1}, \ldots, q_n), a_s, (q'_1, \ldots, q'_p, q_{p+1}, \ldots, q_n)) \in \delta,$$

sur chaque chemin menant de l'état $q = (q'_1, \ldots, q'_p, q_{p+1}, \ldots, q_n)$ à un état final, il existe une transition

$$((q'_1, \ldots, q'_p, s_{p+1}, \ldots, s_n), a_t, (q''_1, \ldots, q''_p, s_{p+1}, \ldots, s_n))$$

et aucune transition sur le chemin entre q et $(q''_1, \ldots, q''_p, s_{p+1}, \ldots, s_n)$ ne peut modifier ou même lire les composantes $q'_1, \ldots, q'_p$, nous avons des schémas de la forme :



Nous allons utiliser cette propriété pour construire l'automate $\mathcal{A}'$.

Soient

$$\mathrm{X} = \{a_{(q,q',q'')} \mid q, q', q'' \in \bigotimes_{i \in I_{a_s}} Q_i, \exists (q, a_s, q') \in \delta_{a_s} \text{ et } \exists (q', a_t, q'') \in \delta_{a_t}\}$$

un nouvel alphabet d'actions et $\mathrm{X}_1, \ldots, \mathrm{X}_n$ les sous-alphabets tels que pour tout $1 \leq i \leq n$, $\mathrm{X}_i = \{a_{(q,q',q'')} \in \mathrm{X} \mid a \in \Sigma_i\}$. Posons

$$\mathcal{A}' = ((\mathrm{X}_{1s} \cup \mathrm{X}_{1t}, \ldots, \mathrm{X}_{ns} \cup \mathrm{X}_{nt}), \bigotimes_{1 \leq i \leq n} Q'_i, \delta', q_0, F).$$

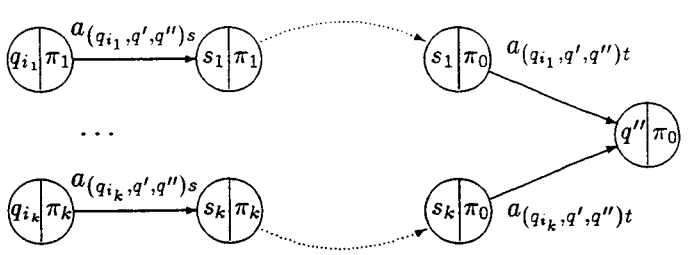
Les nouveaux ensembles d'états locaux Q'_i sont ceux de $\mathcal{A}$ augmentés des nouveaux états locaux que nous allons créer durant la définition de la fonction de transition. Les fonctions de transition partielles de $\mathcal{A}'$ sont définies comme suit. Pour toute action $a_{(q,q',q'')}$ de l'alphabet X, nous avons :

$$\delta'_{a_{(q,q',q'')_s}} = \{(q, a_{(q,q',q'')s}, s_{a(q,q',q'')})\}$$

et

$$\delta'_{a_{(q,q',q'')_t}} = \{(s_{a(q,q',q'')}, a_{(q,q',q'')t}, q'')\}$$

où $s_{a(q,q',q'')}$ est un p-uplet de nouveaux états locaux, nous obtenons donc des schémas du type (avec $s_1 = s_{a(q_{i_1},q',q'')}, \ldots, s_k = s_{a(q_{i_k},q',q'')}$) :



Si $q' = (r_1, \ldots, r_p)$ alors nous avons $s_{a(q,q',q'')} = (r_{a(q,q',q'')1}, \ldots, r_{a(q,q',q'')p})$. Les états r_i et $r_{a(q,q',q'')i}$ sont dits associés, par extension tout état local est associé à lui-même et deux n-uplets d'états sont associés si leurs composants sont deux-à-deux associés. L'automate $\mathcal{A}'$ est clairement un DLCA puisque chaque fonction de transition partielle contient au plus une transition.

Par construction, le langage obtenu est un st-langage, nous allons montrer que $L = \hat{\sigma}(L(\mathcal{A}'))$ si σ est le morphisme d'actions :

$$\begin{array}{rccc} \sigma : & \mathrm{X} & \longrightarrow & \Sigma \\ & a_{(q,q',q'')} & \longmapsto & a. \end{array}$$

Pour montrer l'égalité $L(\mathcal{A}) = \sigma(L(\mathcal{A}'))$, nous allons associer à chaque chemin C de $\mathcal{A}$ étiqueté par un mot u un chemin C' de $\mathcal{A}'$ (et vice versa), constitué d'états associés à ceux de C et étiqueté par un mot v tel que $\sigma(v) = u$. L'association des chemins est suffisante puisque les états initiaux et finaux de $\mathcal{A}$ et $\mathcal{A}'$ sont identiques et sont uniquement associés à eux-mêmes.

Pour l'inclusion $\sigma(L(\mathcal{A}')) \subseteq L(\mathcal{A}))$, remarquons qu'à tout état $(q'_1, \ldots, q'_n)$ de l'automate $\mathcal{A}'$ est associé un unique état $(q_1, \ldots, q_n)$ de $\mathcal{A}$ tel que pour tout i, q_i et q'_i sont associés. Par construction, pour toute transition $(q'_1, x, q'_2) \in \delta'$, il existe une transition $(q_1, \sigma(x), q_2) \in \delta$ telle que q_1 et q_2 sont respectivement associés à q'_1 et q'_2. Donc, tout chemin de l'automate $\mathcal{A}'$ qui reconnaît un mot u peut être associé à un chemin de l'automate $\mathcal{A}$ qui reconnaît le mot $\sigma(u)$.

Pour montrer l'inclusion inverse, nous ne pouvons pas associer un unique état de $\mathcal{A}'$ à chaque état de $\mathcal{A}$, cependant nous allons voir que pour toute transition (p, x, q) de δ, pour tout état p' associé à p, il existe un état q' associé à q et une lettre y tels que $(p', y, q') \in \delta'$ avec $\sigma(y) = x$. Posons $p = (p_1, \ldots, p_n)$, $p' = (p'_1, \ldots, p'_n)$, $q = (q_1, \ldots, q_n)$, nous pouvons supposer, sans perte de généralité, que $\alpha(x) = a$ et $I_{a_s} = \{1, \ldots, k\}$. Notons

$$p_a = (p_1, \ldots, p_k) \text{ et } q_a = (q_1, \ldots, q_k).$$

La transition (p_a, x, q_a) appartient à δ_x. Deux cas peuvent survenir :

- si $x = a_s$ alors pour tout $1 \leq i \leq k$, p_i et p'_i sont égaux et il existe une transition $((q_1, \ldots, q_k), a_t, (r_1, \ldots, r_k))$ appartenant à δ_{a_t}. Posons $r_a = (r_1, \ldots, r_k)$, par construction, nous avons :

  $$(p_a, a_{(p_a,q_a,r_a)s}, (s_{a(p_a,q_a,r_a)1}, \ldots, s_{a(p_a,q_a,r_a)k})) \in \delta'_{a_{(p_a,q_a,r_a)s}}.$$

  en conséquence, il existe un état

  $$q' = ((s_{a(p_a,q_a,r_a)1}, \ldots, s_{a(p_a,q_a,r_a)k}, q_{k+1}, \ldots, q_n)$$

  tel que $(p', a_{(p_a,q_a,r_a)s}, q')$ appartient à δ' avec $\sigma(a_{(p_a,q_a,r_a)s}) = a_s$;

- si $x = a_t$, pour tout état $s_{a(r_a,p_a,q_a)}$ associé à p_a, la transition $(s_{a(r_a,p_a,q_a)}, a_{(r_a,p_a,q_a)t}, q_a)$ appartient à $\delta'_{a_{(r_a,p_a,q_a)t}}$ (par construction). Donc, pour tout état p' associé à p, l'état

  $$q' = (q_1, \ldots, q_k, p'_{k+1}, \ldots, p'_n)$$

  est tel que $(p', a_{(r_a,p_a,q_a)t}, q')$ est une transition de δ' avec $\sigma(a_{(r_a,p_a,q_a)t}) = a_t$.

Nous pouvons construire un chemin de $\mathcal{A}$ étiqueté par un mot v tel que $\sigma(u) = v$. Ceci achève la preuve de l'égalité $\sigma(L(\mathcal{A}')) = L(\mathcal{A}) = L$.

De cette égalité nous pouvons immédiatement déduire $\hat{\sigma}(L(\mathcal{A}')) = L$ puisque L est un st-langage.

À présent, il nous reste à montrer que le langage $L(\mathcal{A}')$ reconnu par $\mathcal{A}'$ est un langage de synchronisation généralisé. L'automate $\mathcal{A}'$ est un DLCA, donc nous pouvons appliquer le Théorème 5.21 de Duboc :

$$L(\mathcal{A}') = \bigcup_{q_f \in F} L(\mathcal{B}^1_{q_0,q_f}) \sqcap \ldots \sqcap L(\mathcal{B}^n_{q_0,q_f}).$$

Pour tout $1 \leq i \leq n$ et tout $q_f \in F$, l'automate $\mathcal{B}^i_{q_0,q_f} = (\mathrm{X}_{is} \cup \mathrm{X}_{it}, Q_i, \delta'_i, q_i, q_{fi}))$ défini par :

- q_i et q_{fi} sont respectivement les ièmes composantes de q_0 et q_f ;
- une transition (s_i, x, s'_i) appartient à δ'_i si et seulement s'il existe, dans δ', une transition de $((s_1, \ldots, s_{i-1}, s_i, s_{i+1}, \ldots, s_n), x, (s'_1, \ldots, s'_{i-1}, s'_i, s'_{i+1}, \ldots, s'_n))$ avec $x \in \mathrm{X}_{is} \cup \mathrm{X}_{it}$.

Pour tout $1 \leq i \leq n$ et tout $q_f \in F$, le langage reconnu par l'automate $\mathcal{B}^i_{q_0,q_f}$ est une séquence, c'est donc un langage de synchronisation généralisé. Le produit de mixage de langages de synchronisation généralisés est un langage de synchronisation généralisé (d'après le Lemme 5.23), l'union également donc, $L(\mathcal{A}')$ est un langage de synchronisation généralisé. Comme le langage L est égal à $\hat{\sigma}(L(\mathcal{A}'))$, L est l'image par un st-morphisme d'un langage de synchronisation généralisé. □

Nous pouvons donc appliquer ce résultat à un langage coloré pour obtenir le résultat principal de ce chapitre.

Proposition 5.25 *La famille des st-langages réguliers clos par θ est égale à la famille des images par st-morphismes des langages de synchronisation généralisés :*

$$\mathrm{R}_\theta = \Phi_{\mathrm{st}}(\mathrm{LS}).$$

Preuve. L'inclusion de $\Phi_{\mathrm{st}}(\mathrm{LS})$ dans R_θ a été montrée au chapitre précédent. Soit L un st-langage régulier clos par θ. D'après le Lemme 5.14, $L = \varphi(\mathrm{f}_\varrho(\tau(L)))$. Le morphisme strictement alphabétique φ est un morphisme d'actions et, par construction, nous avons $\varphi(\mathrm{f}_\varrho(\tau(L))) = \hat{\varphi}(\mathrm{f}_\varrho(\tau(L)))$ donc L est l'image par un st-morphisme de $\mathrm{f}_\varrho(\tau(L))$. Supposons $\mathrm{f}_\varrho(\tau(L))$ défini sur un alphabet d'actions coloré Σ_c et considérons $(\Sigma_{cs} \cup \Sigma_{ct}, \bar{\varrho})$. Le début et la fin d'une même action colorée ne commutent jamais et deux lettres représentant des actions colorées différentes ne commutent pas si elles ont la même image par α, si elles ont une couleur en commun ou si elles ne sont pas coloriées avec le même sous-ensemble de trois couleurs. En considérant les cliques $(\Sigma_{1s} \cup \Sigma_{1t}), \ldots, (\Sigma_{ns} \cup \Sigma_{nt})$ constituées de la façon suivante :

$$\forall 1 \leq i \leq n, \Sigma_i = \{(x, E), (y, F) \mid \alpha(x) = \alpha(y) \text{ ou}$$
$$(E \cap \mathrm{C}_{\{x,y\}} = \emptyset \oplus F \cap \mathrm{C}_{\{x,y\}} = \emptyset) \text{ ou } E \cap F \cap (\mathrm{C}_{\{x,y\}} \cup \bar{\mathrm{C}}_{\{x,y\}}) \neq \emptyset\},$$

nous obtenons un recouvrement par cliques de $(\Sigma_{cs} \cup \Sigma_{ct}, \bar{\varrho})$. Notons que, par construction, la projection de $\mathrm{f}_\varrho(\tau(L))$ sur chacune de ces cliques est une séquence.

D'après le Corollaire 5.18, $\mathrm{f}_\varrho(\tau(L))$ est reconnu par un automate asynchrone déterministe sur $((\Sigma_{1s} \cup \Sigma_{1t}), \ldots, (\Sigma_{ns} \cup \Sigma_{nt}))$. D'après la Proposition 5.24, $\mathrm{f}_\varrho(\tau(L))$ est l'image par un

st-morphisme d'un langage de synchronisation. Comme la composition de st-morphismes est un st-morphisme (Lemme 4.13), L est l'image par un st-morphisme d'un langage de synchronisation. □

Comme les st-langages réguliers clos par R' sont clos par θ, nous déduisons de la proposition précédente le corollaire :

Corollaire 5.26 *La famille des st-langages réguliers clos par R' est incluse dans la famille des images par st-morphisme des langages de synchronisation :*

$$\mathrm{R}_{\mathrm{R}'} \subseteq \Phi_{\mathrm{st}}(\mathrm{LS}).$$

Nous obtenons donc une version affaiblie de la Conjecture III de Salomaa et Yu. Dans la construction que nous utilisons, le but est de ramener la clôture par une semi-commutation à la clôture par une commutation partielle. Une fois cette opération réalisée, les preuves des propriétés que nous voulons obtenir sont facilitées par les outils spécifiques aux commutations partielles. Il est naturel de se demander si une telle construction ne peut s'appliquer à d'autres problèmes cependant, le passage de la semi-commutation à la commutation partielle est une étape assez lourde aussi allons nous montrer, dans la section suivante, qu'il est possible de ramener toute semi-commutation à une commutation partielle en utilisant un coloriage générique.

5.3 Généralisation à toute semi-commutation

Le problème est le suivant : peut-on trouver, pour toute semi-commutation θ définie sur un alphabet Σ, un coloriage $\tau_{(\Sigma,\theta)}$, une commutation partielle ϱ_θ et un morphisme strictement alphabétique φ_Σ tels que pour tout langage L de Σ^*, $\mathrm{f}_\theta(L) = \varphi_\Sigma(\mathrm{f}_{\varrho_\theta}(\tau_{(\Sigma,\theta)}(L)))$?

5.3.1 Définition de la construction

Nous définissons un alphabet « coloré » dont les couleurs sont en fait représentées par des compteurs. Cet alphabet dépend uniquement de l'alphabet de départ et non de la semi-commutation considérée.

Définition 5.27 *Soient Σ un alphabet fini, u un mot de Σ^* et θ une semi-commutation sur Σ. L'alphabet coloré Σ_c est défini par :*

$$\begin{aligned}\Sigma_c = \; & \{(a,E) \mid a \in \Sigma, E \subseteq \{\{a,x,\gamma\} \mid x \in \Sigma, x \neq a, \gamma \in \mathbb{N}\} \\ & \textit{et } (\{a,x,\gamma_1\},\{a,y,\gamma_2\} \in E) \Rightarrow (x \neq y \textit{ ou } \gamma_1 = \gamma_2)\}.\end{aligned}$$

À chaque lettre a de l'alphabet de départ est ajouté un ensemble de compteurs, chaque compteur est associé à un sous-alphabet de deux lettres contenant a, par exemple la lettre $(a,\{\{a,b,3\},\{a,c,4\}\})$ est une lettre a qui a reçu les valeurs de compteurs 3 pour le sous-alphabet $\{a,b\}$ et 4 pour le sous-alphabet $\{a,c\}$. Le morphisme strictement alphabétique qui permet d'« effacer » les compteurs pour revenir à l'alphabet initial ne dépend, lui aussi, que de cet alphabet.

Définition 5.28 *Soit Σ un alphabet fini. Le morphisme strictement alphabétique φ_Σ est défini par :*

$$\begin{array}{rccc} \varphi_\Sigma : & \Sigma_c^* & \longrightarrow & \Sigma^* \\ & (a,E) & \longmapsto & a. \end{array}$$

Le coloriage, par contre, est entièrement lié à une semi-commutation particulière puisqu'il permet de prendre en compte la perte de non symétrie due au passage à une commutation partielle. Nous aurons donc une fonction de coloriage associée à chaque semi-commutation θ. Nous allons ajouter des compteurs aux lettres de l'alphabet de départ de sorte que pour tout couple (a,b) appartenant à θ et n'appartenant pas à θ^{-1}, deux lettres b et a consécutives dans la projection d'un mot sur l'alphabet $\{a,b\}$ reçoivent le même compteur. Prenons l'exemple du mot *ababab*. Le mot colorié correspondant sera

$$v = (a,\{a,b,0\})(b,\{a,b,1\})(a,\{a,b,1\})(b,\{a,b,2\})(a,\{a,b,2\})(b,\{a,b,3\}).$$

Si nous considérons la commutation partielle qui contient tous les couples de lettres correspondant à a et b et possédant des compteurs différents, nous obtenons dans la clôture du mot v les mots dans lesquels les b remontent en tête du mot. Ceci correspond exactement au phénomène qui se produit dans la clôture par θ du mot u. En marquant les lettres b et a consécutives avec le même compteur, les commutations non désirées sont « bloquées ». Cette manipulation n'est pas nécessaire dans le cas des couples de lettres n'appartenant pas à $\theta \cup \theta^{-1}$ ou appartenant à $\theta \cap \theta^{-1}$, par convention les compteurs associés à de tels couples de lettres seront toujours nuls.

Formellement, le coloriage et la commutation partielle associés à une semi-commutation donnée sont définis par :

Définition 5.29 *Soit θ une semi-commutation définie sur un alphabet Σ. La fonction $\tau_{(\Sigma,\theta)}$ de Σ^* dans Σ_c^* est définie par :*

$$\tau_{(\Sigma,\theta)}(\varepsilon) = \varepsilon \ ;$$
$$\tau_{(\Sigma,\theta)}(ua) = \tau_{(\Sigma,\theta)}(u)(a,E) \ ;$$

pour tout x appartenant à $\Sigma \setminus \{a\}$, $\{a,x,\gamma\} \in E$ avec :

- *si (a,x) n'appartient pas à $\theta \cup \theta^{-1}$ ou si (a,x) appartient à $\theta \cap \theta^{-1}$ alors $\gamma = 0$;*
- *si (a,x) appartient à θ et (x,a) n'appartient pas à θ alors :*
  - *$\gamma = 0$ si $|u|_x = 0$,*
  - *$\gamma = \gamma'$ si $\tau_{(\Sigma,\theta)}(u) = u_1(x,F)u_2$ avec $\{a,x,\gamma'\}$ appartenant à F et x n'appartenant pas à $\mathrm{alph}(\varphi_\Sigma(u_2))$;*
- *si (x,a) appartient à θ et (a,x) n'appartient pas à θ alors :*
  - *$\gamma = 0$ si $|u|_x = 0$,*
  - *$\gamma = \min(\mathbb{N} \setminus \{\gamma' \mid \exists (x,G) \in \mathrm{alph}(v)$ avec $\{a,x,\gamma'\} \in G\})$, avec $v = u_2$ s'il existe un découpage $\tau_{(\Sigma,\theta)}(u) = u_1(x,F)u_2$ tel que $x\varphi_\Sigma(u_2)a$ contient un sous-mot étiquetant un chemin de x à a dans le graphe de non commutation de θ et tel qu'aucun facteur droit propre de $x\varphi_\Sigma(u_2)a$ ne contient un tel chemin et $v = \tau_{(\Sigma,\theta)}(u)$ sinon.*

Définition 5.30 *Soit θ une semi-commutation sur un alphabet Σ. La commutation partielle ϱ_θ est définie par :*

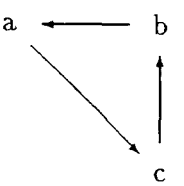
$$\varrho_\theta = \{((x,E),(y,F)) \in \Sigma_c^2 \mid x \neq y \text{et} ((x,y) \in \theta \cap \theta^{-1} \text{ ou } E \cap F = \emptyset)\}.$$

Avant de montrer que cette construction apporte effectivement le résultat escompté, considérons un exemple simple.

Exemple 5.31 Soient un alphabet $\Sigma = \{a, b, c\}$, $\theta = \{(a, b), (b, c), (c, a)\}$ une semi-commutation et un mot $u = abcab$. Le mot coloré correspondant est :

$$\begin{aligned}\tau_{(\Sigma,\theta)}(u) = \quad & (a, \{\{a, b, 0\}, \{a, c, 0\}\})(b, \{\{a, b, 1\}, \{b, c, 0\}\})(c, \{\{a, c, 0\}, \{b, c, 1\}\}) \\ & (a, \{\{a, b, 1\}, \{a, c, 1\}\})(b, \{\{a, b, 0\}, \{b, c, 1\}\}).\end{aligned}$$

La fonction $\tau_{(\Sigma,\theta)}$ transforme le mot séquentiellement de gauche à droite cependant, pour expliquer le principe du coloriage, nous allons uniquement considérer les couleurs associées à l'alphabet $\{a, b\}$. La première lettre a recevra le compteur $\{a, b, 0\}$, la deuxième lettre un compteur $\{a, b, 1\}$, le c ne reçoit pas de compteur pour l'alphabet $\{a, b\}$, le a suivant reçoit le compteur du b qui le précède, puisque $(a, b) \in \theta$ et $(b, a) \notin \theta$, le dernier b va recevoir un compteur $\{a, b, 0\}$. En effet, entre le premier a qui a reçu un compteur nul et le b en question, nous trouvons la lettre c or, acb est un chemin dans le graphe de non commutation de θ



le premier a et le dernier b ne pourront jamais commuter dans une dérivation de u par θ, il est donc acceptable et même souhaitable que les deux lettres correspondant dans le mot coloré ne puissent commuter. L'emploi du compteur $\{a, b, 0\}$ pour le b n'entrave en rien les commutations nécessaires et de plus, la réutilisation d'une valeur permet de limiter le nombre de valeur utilisées.

5.3.2 Calcul de la clôture par une semi-commutation grâce à une commutation partielle

Nous montrons donc à présent que la construction proposée permet effectivement de calculer la clôture par une semi-commutation.

Lemme 5.32 *Soient Σ un alphabet et θ une semi-commutation. Pour tout langage L de Σ^*, nous avons :*

$$\mathrm{f}_\theta(L) = \varphi_\Sigma(\mathrm{f}_{\varrho\theta}(\tau_{(\Sigma,\theta)}(L))).$$

Preuve. Clairement, il suffit de montrer l'égalité pour tout mot de Σ^*.

Soit v appartenant à $\mathrm{f}_\theta(u)$. Montrons par récurrence sur la longueur d'une dérivation minimale de u à v que v appartient à $\varphi_\Sigma(\mathrm{f}_{\varrho\theta}(\tau_{(\Sigma,\theta)}(u)))$, ce qui est clairement vrai pour une dérivation de longueur nulle. Considérons la dérivation :

$$u \xrightarrow[\theta]{n} w = w_1xyw_2 \xrightarrow[\theta]{} w_1yxw_2 = v.$$

Par hypothèse de récurrence, nous avons

$$\tau_{(\Sigma,\theta)}(u) \xrightarrow[\varrho\theta]{*} w_1'(x, E)(y, F)w_2'$$

avec $\varphi_\Sigma(w_1')$ et $\varphi_\Sigma(w_2')$ respectivement égaux à w_1 et w_2. Si $((x,E)(y,F))$ n'appartient pas à ϱ_θ alors (y,x) n'appartient pas à θ et $E \cap F \neq \emptyset$. Comme nous considérons une dérivation minimale (Lemme 1.25), les deux occurrences de lettres que nous considérons sont dans cet ordre dans u donc, par définition du coloriage, le facteur entre ces deux occurrences (x,E) et (y,F) dans $\tau_{(\Sigma,\theta)}(u)$ contient un sous-mot qui étiquette un chemin fortement connexe entre ces deux lettres d'où contradiction : si $((x,E)(y,F))$ n'appartient pas à ϱ_θ alors ces deux lettres ne peuvent se trouver côte à côte. En conséquence, donc, $((x,E),(y,F))$ appartient à ϱ_θ et

$$\tau_{(\Sigma,\theta)}(u) \xrightarrow[\varrho_\theta]{*} w_1'(y,F)(x,E)w_2' \in \varphi_\Sigma^{-1}(v).$$

En conséquence, nous avons $\mathrm{f}_\theta(u) \subseteq \varphi_\Sigma(\mathrm{f}_{\varrho_\theta}(\tau_{(\Sigma,\theta)}(u)))$.

Soit v appartenant à $\varphi_\Sigma(\mathrm{f}_{\varrho_\theta}(\tau_{(\Sigma,\theta)}(u)))$. Il existe un mot v' appartenant à $\mathrm{f}_{\varrho_\theta}(\tau_{(\Sigma,\theta)}(u))$ tel que $\varphi_\Sigma(v') = v$. Montrons par récurrence sur la longueur d'une dérivation minimale de $\tau_{(\Sigma,\theta)}(u)$ à v' que v appartient à $\mathrm{f}_\theta(u)$, ce qui est clairement vrai pour une dérivation de longueur nulle. Considérons la dérivation :

$$\tau_{(\Sigma,\theta)}(u) \xrightarrow[\varrho_\theta]{*} w' = w_1'(x,E)(y,F)w_2' \xrightarrow[\varrho_\theta]{} w_1'(y,F)(x,E)w_2' = v'.$$

Par hypothèse de récurrence, nous avons :

$$u \xrightarrow[\theta]{*} w = w_1xyw_2 \text{ avec } \varphi_\Sigma(w_1') = w_1 \text{ et } \varphi_\Sigma(w_2') = w_2.$$

Si (x,y) appartient à θ, nous avons immédiatement

$$u \xrightarrow[\theta]{*} w = w_1xyw_2 \xrightarrow[\theta]{} v = \varphi_\Sigma(v').$$

Nous allons montrer que le cas contraire ne peut survenir. Nous savons que $((x,E),(y,F))$ appartient à ϱ_θ et que (x,y) n'appartient pas à θ donc, par définition, $E \cap F = \emptyset$. Comme la longueur de la dérivation est minimale, les occurrences (x,E) et (y,F) que nous considérons sont dans cet ordre dans $\tau_{(\Sigma,\theta)}(u)$; or, par construction, il existe un découpage :

$$\tau_{(\Sigma,\theta)}(u) = u_1(x,E)u_2(x,F')u_3(y,F)u_4$$

avec $F \cap F' \neq \emptyset$. En effet, par définition du coloriage, comme (y,x) appartient à θ, (y,F) possède la même « couleur » que la lettre correspondant à x qui la précède. Comme E et F sont disjoints, il existe un (x,F') précédant (y,F) tel que $F \cap F' = \{\{x,y,\gamma\}\}$. Dans tout mot de $\mathrm{f}_{\varrho_\theta}(\tau_{(\Sigma,\theta)}(u))$, les deux occurrences (x,E) et (y,F) seront toujours séparées par un facteur contenant (x,F'). D'où contradiction : le couple (x,y) appartient forcément à θ. Nous avons donc $v \in \mathrm{f}_\theta(u)$ et, en conclusion, $\varphi_\Sigma(\mathrm{f}_{\varrho_\theta}(\tau_{(\Sigma,\theta)}(u))) \subseteq \mathrm{f}_\theta(u)$. □

Le coloriage est une fonction, cependant le nombre de valeurs différentes pour les compteurs peut s'avérer infini. S'il peut être utile de passer d'une commutation partielle à une semi-commutation dans certains cas, il est indispensable de conserver un alphabet fini. Nous allons voir que, sous certaines conditions, un langage peut être coloré par une fonction rationnelle. Ceci nous permet en particulier d'obtenir des résultats intéressants dans le cas des langages réguliers.

Lemme 5.33 *Soient Σ un alphabet, θ une semi-commutation et L un langage régulier sur Σ. Si pour tout facteur itérant u de L, toute composante connexe de $(\mathrm{alph}(u), \bar{\theta})$ est fortement connexe, alors $\tau_{(\Sigma,\theta)}(L)$ peut être calculé par une fonction rationnelle.*

Preuve. Soient Σ un alphabet, θ une semi-commutation et L un langage régulier sur Σ tel que pour tout facteur itérant u de L, toute composante connexe de $(\mathrm{alph}(u), \bar{\theta})$ est fortement connexe. Pour montrer que $\tau_{(\Sigma,\theta)}(L)$ peut être calculé par une fonction rationnelle, il suffit de montrer que l'alphabet de ce langage est fini.

Soit n le nombre d'états de l'automate déterministe minimal reconnaissant L.

Assertion 1. Soit un langage L reconnu par un automate de n états. Si l'on considère un facteur u de L ayant $(ab)^n$ ou $(ba)^n$ pour sous-mot, alors il existe un facteur de u contenant au moins un a et un b qui est un facteur itérant de L.

Assertion 2. Soit u un mot de Σ^*. S'il existe un découpage $\tau_{(\Sigma,\theta)}(u) = u_1(x,E)u_2$ avec $\{a,b,\gamma\} \in E$ et $\gamma > 0$ alors $\varphi_\Sigma(u_1)$ possède $(ab)^{\gamma-1}$ ou $(ba)^{\gamma-1}$ pour sous-mot.

En effet, par définition du coloriage, on n'utilise le compteur γ que si les valeurs de 0 à $\gamma - 1$ sont utilisées. De plus, si (a,b) appartient à θ alors une nouvelle valeur de compteur ne peut être introduite que pour un b suivant un a (et vice-versa si (b,a) appartient à θ). Si (a,b) appartient à θ, les plus petits mots nécessitant l'utilisation du compteur $\{a,b,\gamma\}$ sont $(ab)^\gamma$, $a(ba)^\gamma$, $(ba)^{\gamma-1}b$ ou $(ba)^\gamma$ - selon qu'ils commencent et se terminent par a ou b. Si (b,a) appartient à θ, on obtient $(ba)^\gamma$, $b(ab)^\gamma$, $(ab)^{\gamma-1}a$ ou $(ab)^\gamma$. Si $\{a,b,\gamma\}$ appartient à E alors le mot $\varphi_\Sigma(u_1)$ possède l'un de ces plus petits mots pour sous-mot et donc au moins $(ab)^{\gamma-1}$ ou $(ba)^{\gamma-1}$. Ceci achève la preuve de l'assertion.

Soit u un mot de L. Nous allons montrer par récurrence sur la longueur de u que toute lettre (x,E) de l'alphabet de $\tau_{(\Sigma,\theta)}(u)$ est telle que pour tout $\{x,y,\gamma\}$ appartenant à E, γ est inférieur ou égal à $n \times (t_{xy} - 1) \times \alpha_{xy}$, où α_{xy} est le nombre d'alphabets différents contenant x et y possibles pour les facteurs itérants de L et où t_{xy} est la taille du plus grand de ces alphabets. Ceci est clairement le cas pour un mot u de longueur nulle.

Considérons un mot u de longueur $k+1$ et posons $u = vb$, b appartenant à Σ. Par définition, $\tau_{(\Sigma,\theta)}(u) = \tau_{(\Sigma,\theta)}(v)(b,E)$ et, par hypothèse de récurrence, toute lettre (x,F) de l'alphabet de $\tau_{(\Sigma,\theta)}(v)$ est telle que pour tout $\{x,y,\gamma\}$ appartenant à F, γ est inférieur à $n \times (t_{xy} - 1) \times \alpha_{xy}$.

Supposons que $\{a,b,n \times (t_{ab} - 1) \times \alpha_{ab} + 1\}$ appartienne à E (dans ce cas, (a,b) appartient à θ et pas à θ^{-1}). D'après la deuxième assertion, il existe un découpage de v, $v = v_1v_2$ tel que $\Pi_{\{a,b\}}(v_2)$ appartient à $(b^+a^+)^{(n\times(t_{ab}-1)\times\alpha_{ab})}$ ou à $(a^+b^+)^{(n\times(t_{ab}-1)\times\alpha_{ab})}$. D'après la première assertion, v_2 possède au moins $(t_{ab} - 1) \times \alpha_{ab}$ facteurs (qui ne se recouvrent pas) qui sont des facteurs itérants de L contenant au moins un a et un b. Si l'on note $w_1, \ldots, w_{(t_{ab}-1)\times\alpha_{ab}}$ ces facteurs, nous obtenons le découpage de v_2 :

$$v_2 = u_0 w_1 u_1 \ldots w_{(t_{ab}-1)\times\alpha_{ab}} u_{(t_{ab}-1)\times\alpha_{ab}}.$$

Comme par hypothèse toutes les composantes connexes du graphe de non commutation d'un facteur itérant de L sont fortement connexes et comme nous sommes sûrs qu'au moins $t_{ab} - 1$ des facteurs $w_1, \ldots, w_{(t_{ab}-1)\times\alpha_{ab}}$ ont le même alphabet, il existe entre la dernière occurrence a du mot w_1 et la dernière lettre de u (qui est b), un sous-mot étiquetant un chemin de a à b dans le graphe de non commutation de θ. Par définition de $\tau_{(\Sigma,\theta)}$, la dernière lettre de u reçoit un compteur $\{a,b,\gamma_1\}$ qui, dans le pire des cas, est obtenu par :

$$\gamma_1 = \min(\mathbb{N} \setminus \{\gamma' \mid \exists (y,G) \in \mathrm{alph}(w') \text{ avec } \{a,b,\gamma'\} \in G\})$$

si w' est le facteur droit de longueur $|u_1 \ldots w_{(t_{ab}-1)\times\alpha_{ab}} u_{(t_{ab}-1)\times\alpha_{ab}}|$ de $\tau_{(\Sigma,\theta)}(v)$. Donc, dans le pire des cas, γ_1 est le premier entier disponible sachant que $n \times (t_{ab} - 1) \times \alpha_{ab}$ valeurs sont

utilisées, c'est-à-dire que γ_1 vaut au plus $n \times (t_{ab} - 1) \times \alpha_{ab}$ ce qui est en contradiction avec l'hypothèse.

Les valeurs de compteurs utilisées pour $\tau_{(\Sigma,\theta)}(L)$ sont donc bornées par

$$\max(\{n \times (t_{ab} - 1) \times \alpha_{ab} \mid a, b \in \text{alph}(L), (a, b) \in \theta, (b, a) \notin \theta\}).$$

□

Nous avons donc une condition suffisante pour qu'un langage régulier puisse être coloré par une fonction rationnelle, nous allons voir que cette condition est également nécessaire.

Lemme 5.34 *Soient Σ un alphabet, θ une semi-commutation et L un langage régulier sur Σ. S'il existe un facteur itérant u de L tel qu'une composante connexe de $(\text{alph}(u), \bar{\theta})$ n'est pas fortement connexe alors l'alphabet de $\tau_{(\Sigma,\theta)}(L)$ est infini.*

Preuve. Supposons qu'il existe un facteur itérant u de L dont le graphe de non commutation pour θ contient une composante connexe qui n'est pas fortement connexe. Par définition, il existe u_1 et u_2 tels que pour tout entier n, $u_1 u^n u_2$ appartient à L. Il existe deux lettres x et y de l'alphabet de u telles que (x, y) appartient à θ et pas à θ^{-1} et telles que, pour tout entier n, le mot u^n ne contient aucun sous-mot étiquetant un chemin de x à y dans le graphe de non commutation de θ. Par définition du coloriage, deux occurrences quelconques de x dans u^n séparées par au moins une occurrence de y se verront attribuer des valeurs de compteurs distinctes pour l'alphabet $\{x, y\}$, il faut donc au minimum n couleurs pour colorier la partie correspondant à u^n : nous obtenons un alphabet Σ_c infini.

□

Exemple 5.35 Soient $L = (ab)^*$ et $\theta = \{(a, b)\}$. Pour tout entier n, nous obtenons :

$$\begin{aligned}\tau_{(\Sigma,\theta)}((ab)^n) = \; & (a, \{\{a, b, 0\}\})(b, \{\{a, b, 1\}\})(a, \{\{a, b, 1\}\}) \dots \\ & \dots (a, \{\{a, b, n-1\}\})(b, \{\{a, b, n\}\}).\end{aligned}$$

L'alphabet de $\tau_{(\Sigma,\theta)}(L)$ est infini, il contient, pour tout entier n, les lettres $(a, \{\{a, b, n\}\})$ et $(b, \{\{a, b, n\}\})$.

Les deux lemmes précédents nous permettent d'énoncer la proposition :

Proposition 5.36 *Soient Σ un alphabet, θ une semi-commutation et L un langage régulier sur Σ. L'alphabet de $\tau_{(\Sigma,\theta)}(L)$ est fini et ce langage peut être calculé par une fonction rationnelle si et seulement si toutes les composantes connexes des graphes de non commutation pour θ des facteurs itérants de L sont fortement connexes.*

Cette proposition permet de faire le lien entre certains résultats existant pour les commutations partielles et les résultats similaires pour les semi-commutations. Prenons l'exemple de la condition suffisante du Théorème 1.29 : si tous les facteurs itérants d'un langage régulier L sont connexes pour une commutation partielle ϱ alors la clôture de L par ϱ est régulière. Le Théorème 1.30 donne une condition similaire dans le cas des semi-commutations, ce résultat se déduit du premier en passant par un langage coloré.

Preuve du Théorème 1.30. Soient θ une semi-commutation sur un alphabet Σ et L un langage régulier de Σ^* dont tous les facteurs itérants sont fortement connexes pour θ. D'après la Proposition 5.36, le langage $\tau_{(\Sigma,\theta)}(L)$ est régulier. Il suffit donc de montrer que ses facteurs itérants sont connexes pour la commutation partielle ϱ_θ pour obtenir le résultat.

Soit u un facteur itérant de $\tau_{(\Sigma,\theta)}(L)$. Remarquons tout d'abord que pour tout x et tout y de $\mathrm{alph}(u)$ tels que $\varphi_\Sigma(x) = \varphi_\Sigma(y)$, (x, y) n'appartient pas à ϱ_θ, donc les ensembles de lettres ayant la même image par φ_Σ forment des cliques du graphe $(\mathrm{alph}(u), \bar{\varrho}_\theta)$. Comme u est un facteur itérant de $\tau_{(\Sigma,\theta)}(L)$, $\varphi_\Sigma(u)$ est un facteur itérant de L, donc pour tout couple de lettres x et y de $\mathrm{alph}(u)$, il existe un chemin de $\varphi_\Sigma(x)$ à $\varphi_\Sigma(y)$ dans le graphe $(\mathrm{alph}(\varphi_\Sigma(u)), \bar{\theta})$. Notons $\varphi_\Sigma(x) = z_0, z_1, \ldots, z_{n-1}, z_n = \varphi_\Sigma(y)$ les sommets de ce chemin. Il nous reste à montrer que pour tout z_i et tout z_{i+1} il existe dans le graphe $(\mathrm{alph}(u), \bar{\varrho}_\theta)$ un arc entre un z'_i et un z'_{i+1} tels que $\varphi_\Sigma(z'_i) = z_i$ et $\varphi_\Sigma(z'_{i+1}) = z_{i+1}$.

Comme il existe un arc entre z_i et z_{i+1}, deux cas sont possibles. Si (z_{i+1}, z_i) n'appartient pas à la semi-commutation θ alors, par définition du coloriage, pour tout (z_i, E) ou tout (z_{i+1}, F) de l'alphabet de u, $\{z_i, z_{i+1}, 0\}$ appartient à E et F et nous avons un arc entre (z_i, E) et (z_{i+1}, F). Si (z_{i+1}, z_i) appartient à θ alors il suffit de montrer qu'il exite un (z_i, E) et un (z_{i+1}, F) dans l'alphabet de u tels que $E \cap F \neq \emptyset$. Ceci est clairement vrai : comme u est un facteur itérant de $\tau_{(\Sigma,\theta)}(L)$, nous pouvons raisonner sur u^2. Dans ce mot, pour toute lettre (z_{i+1}, F) de l'alphabet de u, au moins l'une de ces occurrences est précédée d'un (z_i, E) (et aucune lettre correspondant à z_i ne les sépare). Dans ce cas, par définition du coloriage, $E \cap F \neq \emptyset$ et donc nous avons un arc entre ces deux lettres. Pour tout couple de lettres x et y de $\mathrm{alph}(u)$, il existe un chemin de x à y dans $(\mathrm{alph}(u), \bar{\varrho}_\theta)$, le mot u est donc connexe.

Comme tout facteur itérant de $\tau_{(\Sigma,\theta)}(L)$ est connexe pour ϱ_θ d'après le Théorème 1.29, $\mathrm{f}_{\varrho_\theta}(\tau_{(\Sigma,\theta)}(L))$ est un langage régulier. D'après le Lemme 5.32, $\mathrm{f}_\theta(L)$ est l'image par le morphisme strictement alphabétique φ_Σ de $\mathrm{f}_{\varrho_\theta}(\tau_{(\Sigma,\theta)}(L))$, ce langage est donc régulier. □

Nous avons, dans ce chapitre, donné un premier lien entre familles de langages clos par des systèmes de réécriture et familles de langages construits à partir d'expressions de synchronisation. Nous avons à présent les inclusions données par la Figure 5.4.

Comme nous l'avons expliqué au Chapitre 4, nous conjecturons que le langage

$$\mathrm{f}_\theta(a_s(b_s(b_t b_s(a_t a_s)^+ b_t b_s)^* a_t b_t)),$$

st-langage régulier clos par θ, n'est pas un langage de synchronisation généralisé. Cependant, quelle que soit la réponse à la Conjecture III de Salomaa et Yu, la question peut à présent être formulée sous la forme : « la famille des langages de synchronisation généralisés est-elle close par st-morphisme? » Cette formulation peut être intéressante, en effet, jusqu'à présent, il s'est toujours avéré beaucoup plus aisé de montrer les propriétés propres d'une famille que de comparer une famille construite à partir des expressions de synchronisation et une famille de langages clos par un système de réécriture.

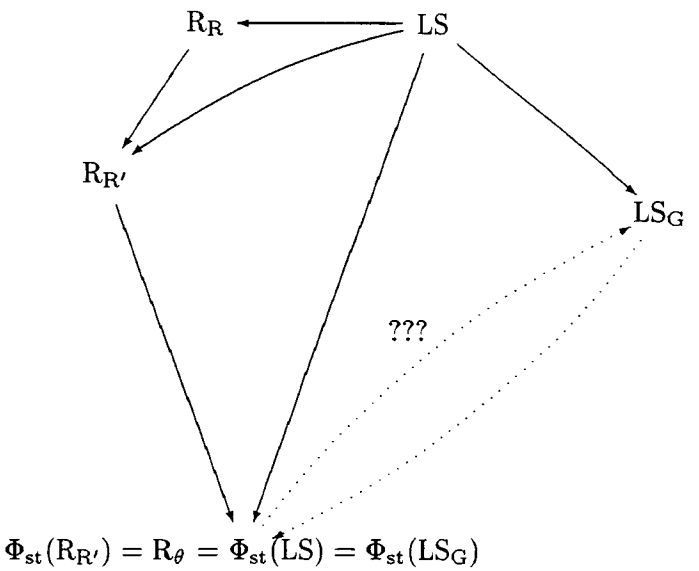


FIG. 5.4 - *Récapitulation*

Conclusion

Dans une première partie de ce travail, nous avons considéré la possibilité de construire une expression de synchronisation, à partir d'un langage donné, par le biais d'une caractérisation des langages de synchronisation en termes de langages clos par un système de réécriture. Nous avons vu, dans un premier temps, que le système R, introduit par Guo, Salomaa et Yu, permet de caractériser les langages de synchronisation définis sur des alphabets de deux actions et, dans ce cas, nous proposons une construction pour retrouver une expression de synchronisation associée à un langage donné. L'extension du système R que nous donnons permet de répondre à la question dans le cas des langages finis et, là aussi, nous proposons un algorithme pour construire une expression de synchronisation associée à un langage donné. Cependant, nous montrons que, dans le cas général, les langages de synchronisation ne peuvent être caractérisés par un système de réécriture.

Nous nous sommes ensuite intéressés aux langages de synchronisation généralisés. Ces langages ont des propriétés intéressantes et leur pouvoir d'expression dépasse largement celui des langages de synchronisation. Cependant, en introduisant une classe de fonctions rationnelles qui préservent les st-langages, les st-morphismes, nous avons montré que les familles des images des langages de synchronisation et des images des langages de synchronisation généralisés sont égales, mettant ainsi en évidence un lien relativement fort entre les deux familles de langages de synchronisation. De plus, la semi-commutation θ introduite par Salomaa et Yu caractérise les images par st-morphismes des langages de synchronisation (généralisés ou non) : un st-langage régulier est l'image par un st-morphisme d'un langage de synchronisation si et seulement s'il est clos par θ. Cette caractérisation est intéressante puisqu'elle est décidable. Ceci ne répond pas à la conjecture de Salomaa et Yu cependant, la question peut désormais être posée sous sa forme originelle :

Les langages de synchronisation généralisés sont-ils caractérisés par θ ?

ou sous la forme :

La famille des langages de synchronisation généralisés est-elle close par st-morphisme ?

La deuxième formulation semble plus intéressante dans la mesure où la question devrait pouvoir être résolue sans nécessiter la construction d'une expression de synchronisation généralisée associée à un langage donné. Cette étape s'avère en effet souvent être la plus délicate.

Fournir, pour un langage, une expression de synchronisation associée est l'un des deux points essentiels pour pouvoir utiliser les expressions de synchronisation pour l'analyse de traces d'exécution. Le deuxième point, que nous n'avons pas traité ici, est la possibilité de simplifier des expressions de synchronisation. Il faudrait pour cela définir une forme normale

pour les expressions – ou au moins de faire des choix parmi les différentes expressions qui décrivent le même comportement si l'on ne peut définir de forme normale unique. Ensuite, il est naturellement nécessaire de disposer d'un algorithme permettant de réduire une expression de synchronisation à sa forme normale de façon à la rendre la plus compréhensible possible. Il s'agit ici de trouver une présentation conviviale des expressions de synchronisation calculées de façon automatique. Le problème essentiel étant de définir une forme « conviviale » qui n'est pas forcément la plus courte possible.

Il serait également intéressant d'étudier les possibilités d'extension du modèle. Nous avons mentionné, par exemple, qu'il est impossible pour le moment d'attribuer des priorités à certaines actions comme dans le cas du problème classique des lecteurs/rédacteurs. S'il est relativement simple d'envisager l'ajout d'opérateurs de priorités pour les expressions de synchronisation, qu'en est-il pour les langages qui leur sont associés?

Bibliographie

[1] ARNOLD, A. *Systèmes de Transitions Finis et Sémantique des Processus Communicants.* eri. Masson, Paris, 1992.

[2] BAUGET, S., AND GASTIN, P. On congruences and partial orders. In *Proc. 20th International Symposium on Mathematical Foundations of Computer Science (MFCS'95)* (1995), vol. 969 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 434–443.

[3] BERNSTEIN, P. A., AND GOODMAN, N. Timestamp-based algorithms for concurrency control in distributed database systems. In *Proc. International Conference on Very Large Data Bases* (1980), pp. 285–300.

[4] BIERMANN, I. *Extensions Structurelles de Traces de Commutations.* PhD thesis, Université de Paris-Sud, France, 1995.

[5] BIERMANN, I., AND ROZOY, B. Context traces and transition systems. In *Proc. 9th International Symposium on Computer and Information Science (ICSIS'94)* (Bogazici University Printhouse, Turkey, 1994), S. Kuru, M. U. Caglayan, E. Gelembe, H. L. Ãkin, and C. Ersoy, Eds., pp. 301–309.

[6] BRINCH HANSEN, P. Structured multiprogramming. *Communications of the ACM 15*, 7 (1972), 574–578.

[7] BRINCH HANSEN, P. *Operating systems Principles.* Prentice-Hall, Enlewood Cliffs, N. J., 1973.

[8] CAMPBELL, R. H., AND HABERMANN, A. N. The specification of process synchronization by path expressions. vol. 16 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin.

[9] CARTIER, P., AND FOATA, D. *Problèmes combinatoires de commutation et réarrangements*, vol. 85 of *Lecture Notes in Mathematics.* Springer-Verlag, Berlin, 1969.

[10] CASTELLANO, L., MICHELIS, G. D., AND POMELLO, L. Concurrency versus interleaving: an instructive example. *Bulletin of the European Association for Theoretical Computer Science (EATCS) 31* (1987), 12–15.

[11] CHANDRA, A. K., KOREN, D. C., AND STOCKMEYER, L. J. Alternation. *Journal of the Association of Computing Machinery 28* (1981), 114–133.

[12] CLERBOUT, M. *Commutations Partielles et Familles de Langages.* PhD thesis, Université des Sciences et Technologies de Lille, France, 1984.

[13] CLERBOUT, M., AND LATTEUX, M. Semi-commutations. *Information and Computation 73*, 1 (1987), 59–74.

[14] CLERBOUT, M., LATTEUX, M., AND ROOS, Y. Semi-commutations. In *The Book of Traces*, V. Diekert and G. Rozenberg, Eds. World Scientific, Singapore, 1995, ch. 12, pp. 487–551.

[15] CLERBOUT, M., ROOS, Y., AND RYL, I. Synchronization languages. *Theoretical Computer Science.* to appear.

[16] CLERBOUT, M., ROOS, Y., AND RYL, I. Generalized synchronization languages and commutations. Tech. Rep. IT-98-315, Université des Sciences et Technologies de Lille, 1998.

[17] CLERBOUT, M., ROOS, Y., AND RYL, I. Langages de synchronisation et systèmes de réécriture. Tech. Rep. IT-98-311, Université des Sciences et Technologies de Lille, 1998.

[18] CORI, R., AND PERRIN, D. Automates et commutations partielles. *R.A.I.R.O. — Theoretical Informatics and Applications 19*, 1 (1985), 21–32.

[19] CORI, R., SOPENA, E., LATTEUX, M., AND ROOS, Y. 2-asynchronous automata. *Theoretical Computer Science 61*, 1 (1988), 93–102.

[20] DE SIMONE, R. Langages infinitaires et produit de mixage. *Theoretical Computer Science 31*, 1-2 (1984), 83–100.

[21] DIEKERT, V., AND EBINGER, W., Eds. *Infinite Traces. Proceedings of a workshop of the ESPRIT Basic Research Action 3166: Algebraic and Syntactic Methods in Computer Science (ASMICS)* (Tübingen, Germany, 1992), Bericht 4/92, Universität Stuttgart, Fakultät Informatik.

[22] DIEKERT, V., AND MÉTIVIER, Y. Partial commutation and traces. In *Handbook of Formal Languages*, G. Rozenberg and A. Salomaa, Eds., vol. 3. Springer-Verlag, Berlin, 1997, ch. 8, pp. 457–533.

[23] DIEKERT, V., AND ROZENBERG, G., Eds. *The Book of Traces.* World Scientific, Singapore, 1995.

[24] DIJKSTRA, E. W. Cooperating sequential processes. In *Programming Languages*, F. Genuys, Ed. Academic Press, London, England, 1968, pp. 43–112.

[25] DUBOC, C. *Commutations dans les Monoides libres: un Cadre Théorique pour l'Étude du Parallélisme.* PhD thesis, Université de Rouen, France, 1986.

[26] FELLAH, A., JÜRGENSEN, H., AND YU, S. Constructions for alternating finite automata. *International Journal of Computer Mathematics 35*, 3-4 (1990), 117–132.

[27] GASTIN, P., AND PETIT, A. Infinite traces. In *The Book of Traces*, V. Diekert and G. Rozenberg, Eds. World Scientific, Singapore, 1995, ch. 11, pp. 393–486.

[28] GINSBURG, S., AND SPANIER, E. H. Bounded regular sets. *Proceedings of the American Mathematical Society 17* (1966), 1043–1049.

[29] GONZALEZ, D. *Décomposition de Semi-Commutations.* PhD thesis, Université des Sciences et Technologies de Lille, France, 1993.

[30] GOVINDARAJAN, R., GUO, L., YU, S., AND WANG, P. ParC project: Practical constructs for parallel programming languages. In *Proc. IEEE 15th Annual Internationnal Computer Software & Applications Conference* (1991), pp. 183–189.

[31] GRAY, J. N., MCJONES, P. R., AND BLASGEN, M. The recovery manager of the system r database manager. *ACM ComputingSurveys 13*, 2 (1981), 223–242.

[32] GUO, L. *Synchronization Expressions in Parallel Programming Languages.* PhD thesis, University of Western Ontario, London, Canada, 1995.

[33] GUO, L., SALOMAA, K., AND YU, S. Synchronization expressions and languages. In *Proc. 6th Symposium on Parallel and Distributed Processing* (1994), pp. 257–264.

[34] GUO, L., SALOMAA, K., AND YU, S. On synchronization languages. *Fundamenta Informaticae 25* (1996), 423–436.

[35] HOARE, C. A. R. Towards a theory of parallel programming. In *Operating Systems Techniques*, C. A. R. Hoare and R. H. Perrot, Eds. Acdemic Press, New York, 1972, pp. 61–71.

[36] HOARE, C. A. R. Monitors: an operating system structuring concept (erratum in vol. 18, num. 2, p. 95). *Communications of the ACM 17*, 10 (1974), 549–557.

[37] HOARE, C. A. R. Communicating sequential processes. *Communications of the ACM 21*, 8 (1978), 666–677.

[38] HOARE, C. A. R. *Communicating Sequential Processes.* Prentice-Hall, Englewood Cliffs, 1985.

[39] HUNG, D. V., AND KNUTH, E. Semi-commutations and petri nets. *Theoretical Computer Science 64*, 1 (1989), 67–81.

[40] HUSSON, J. F. *Modélisation de la Causalité pas des Relations d'Indépendance.* PhD thesis, Université Paul Sabatier, Toulouse, France, 1996.

[41] KOSARAJU, S. R. Limitations of dijkstra's semaphore primitives and petri nets. *Operating Systems Review 7*, 4 (1973), 122–126.

[42] MATEESCU, A., ROZENBERG, G., AND SALOMAA, A. Shuffle on trajectories: syntactic constraints. *Theoretical Computer Science 197*, 1-2 (May 1998), 1–56.

[43] MAZURKIEWICZ, A. Concurrent program schemes and their interpretations. DAIMI Rep. PB 78, Aarhus University, 1977.

[44] MÉTIVIER, Y. On recognizable subsets of free partially commutative monoids. In *Proc. 13th International Colloquium on Automata, Languages and Programming (ICALP'86)* (Rennes, France, 1986), L. Kott, Ed., vol. 226 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 254–264.

[45] MÉTIVIER, Y. Une condition suffisante de reconnaissabilité dans un monoïde partiellement commutatif. *R.A.I.R.O. — Theoretical Informatics and Applications 20* (1986), 121–127.

[46] MÉTIVIER, Y. Contribution à l'étude des monoïdes de commutations. Thèse d'état, Université de Bordeaux I, France, 1987.

[47] MÉTIVIER, Y. On recognizable subsets of free partially comutative monoids. *Theoretical Computer Science 58*, 1–3 (1988), 201–208.

[48] MÉTIVIER, Y., AND WACRENIER, G. R. P. A. Computing the closure of sets of words under partial commutations. In *Proc. 22nd International Colloquium on Automata, Languages and Programming (ICALP'95)* (Szeged, Hungary, 1995), Z. Fülöp and F. Gècseg, Eds., vol. 944 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 75–86.

[49] MILNER, R. *A Calculus of Communicating Systems*, vol. 92 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1980.

[50] NIELSEN, M., PLOTKIN, G., AND WINSKEL, G. Petri nets, event structures and domains, part i. *Theoretical Computer Science 13*, 1 (1881), 85–108.

[51] OCHMAŃSKI, E. Regular behaviour of concurrent systems. *Bulletin of the European Association for Theoretical Computer Science (EATCS) 27* (1985), 56–67.

[52] OCHMAŃSKI, E. Semi-commutationes for place/transition systems. *Bulletin of the European Association for Theoretical Computer Science (EATCS) 38* (1989), 191–198.

[53] OCHMAŃSKI, E. Modelling concurrency with semi-commutations. In *Proc. 17th International Symposium on Mathematical Foundations of Computer Science (MFCS'92)* (Prague, Czechoslovakia, 1992), I. M. Havel and V. Koubek, Eds., vol. 629 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 412–420.

[54] ROOS, Y. *Contribution à l'Étude des Fonctions de Commutation Partielle.* PhD thesis, Université des Sciences et Technologies de Lille, France, 1989.

[55] RYL, I., ROOS, Y., AND CLERBOUT, M. Partial characterization of synchronization languages. In *Proc. 22nd International Symposium on Mathematical Foundations of Computer Science (MFCS'97)* (Bratislava, Slovakia, 1997), I. Prívara and P. Ružička, Eds., vol. 1295 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 209–218.

[56] RYL, I., ROOS, Y., AND CLERBOUT, M. About synchronization languages. In *Proc. 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS'98)* (Brno, Czech Republic, 1998), L. Brim, J. Gruska, and J. Zlatuska, Eds., vol. 1450 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin.

[57] SALOMAA, K., AND YU, S. Rewriting rules for synchronization languages. In *Structures in Logic and Computer Science, a Selection of Essays in Honor of A. Ehrenfeucht*, J. Mycielski, G. Rozenberg, and A. Salomaa, Eds., vol. 1261 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1997, pp. 322–338.

[58] SASSONE, V., NIELSEN, M., AND WINSKEL, G. Models for concurrency: Towards a classification. *Theoretical Computer Science 170*, 1-2 (1996), 297–348.

[59] VAN GLABBECK, R. J., AND VAANDRAGER, F. The difference between splitting in n and $n+1$. *Information and Computation 136*, 2 (1997), 109–142.

[60] ZIELONKA, W. Notes on finite asynchronous automata. *R.A.I.R.O. — Theoretical Informatics and Applications 21*, 2 (1987), 99–135.

Index