ÉCOLE NORMALE SUPÉRIEURE DE CACHAN

THÈSE

présentée
pour obtenir

LE GRADE DE DOCTEUR EN SCIENCES
DE L'ÉCOLE NORMALE SUPÉRIEURE DE CACHAN
Spécialité INFORMATIQUE

PAR

Grégoire SUTRE

Abstraction et accélération de systèmes infinis

Soutenue le 12 octobre 2000 devant le jury composé de :

MM.	Nicolas HALBWACHS	Président
	André ARNOLD	Rapporteur
	Ahmed BOUAJJANI	Rapporteur
	Petr JANČAR	Rapporteur
	Alain FINKEL	Directeur de thèse
	Laurent FRIBOURG	Examinateur

Laboratoire **S**pécification et **V**érification – CNRS UMR 8643 ;
ENS de Cachan ; 61, Avenue du Président Wilson ; 94235 CACHAN Cedex

Remerciements

Je remercie Nicolas Halbwachs de m'avoir fait l'honneur de présider ce jury.

Je remercie particulièrement André Arnold, Ahmed Bouajjani et Petr Jančar d'avoir accepté d'être les rapporteurs de ma thèse.

Je remercie Laurent Fribourg pour ses remarques très pertinentes, et pour ses nombreuses questions !

Ma profonde gratitude va à mon directeur de thèse, Alain Finkel, qui m'a soutenu et guidé tout au long de ma thèse. Je suis très reconnaissant à Alain pour la confiance (parfois effrayante !) qu'il a su m'accorder, pour son enthousiasme communicatif vis-à-vis de la recherche, pour sa grande disponibilité et pour m'avoir fait largement bénéficier de ses fortes compétences scientifiques.

Je tiens également à remercier tous les membres (permanents et non permanents) du LSV pour l'ambiance chaleureuse de travail qu'ils m'ont fait partager. Je remercie les membres permanents pour leur soutien et leurs conseils précieux. Je remercie les thésards et post-doctorants pour nos pauses, souvent explosives, mais parfois bien méritées !

Je remercie Alain Deutsch, Philippe Granger et Benoît Rottembourg de m'avoir initié à la recherche et de m'avoir encouragé à poursuivre dans cette voie.

Je remercie ma famille, et en particulier mes parents, de m'avoir soutenu durant toutes mes études.

Enfin, je ne saurai jamais comment te remercier assez, Barbara, de m'avoir supporté tant moralement que matériellement, d'avoir accepté de relire maintes fois ce document parsemé de mots et de symboles barbares, et d'avoir organisé le plus important : le pot !

Préface à la version finale

Peu après la soutenance de cette thèse, HABERMEHL et MAYR nous ont fait part [HM00a] d'une erreur dans un énoncé de théorème de [Yu97] que nous avions utilisé. Il est en effet écrit dans [Yu97] que le problème d'équivalence entre une expression régulière dénotant un langage borné et une expression régulière générale est NP–complet. Or ce problème est en réalité coNP–complet [HRS76] (et n'est donc pas NP–complet à moins que NP = coNP). Cette erreur nous avait conduit à une fausse formulation du théorème 4.5.2 de la page 144, où nous affirmions (a priori à tort) que les problèmes d'inclusion et d'équivalence entre deux SLRE sont dans $\mathrm{NP} \cap \mathrm{coNP}$.

HABERMEHL et MAYR ont également observé que [SM73] contient en fait une preuve de la NP–difficulté du problème d'inéquivalence entre deux SLRE [HM00a], ce qui permet finalement de conclure que les problèmes d'inclusion et d'équivalence entre deux SLRE sont coNP–complets (comme indiqué dans la nouvelle formulation du théorème 4.5.2 page 144).

Cette version finale[1] tient compte des remarques et corrections de HABERMEHL et MAYR décrites ci-dessus.

[1]Date de dernière modification : 21 mai 2002.

Table des matières

Introduction

La fiabilité des systèmes informatiques est un problème crucial de nos jours, car l'informatique est devenue un outil incontournable pour de nombreux secteurs d'activité. Un très large public a même été sensibilisé au problème de la certification des logiciels lors du "passage à l'an 2000". Les erreurs dans les logiciels non critiques (par exemple les logiciels de bureautique) sont fréquentes, mais elles ne sont pas dramatiques. En revanche, un défaut d'exécution dans un *système critique* (par exemple un système de contrôle-commande de centrale nucléaire, un système d'assistance au pilotage d'avion, ou encore un système de transactions financières) peut avoir des conséquences catastrophiques, tant sur le plan humain que sur le plan économique. Il est donc indispensable de s'assurer, au préalable, que de tels systèmes vont se comporter correctement, même (et surtout !) dans les cas imprévus. C'est le but des *méthodes formelles* de spécification et de validation du logiciel.

La phase préliminaire à toute vérification formelle d'un système consiste à traduire le système et les propriétés attendues de ce système dans des modèles mathématiques. Cette étape est cruciale car la vérification sera réalisée sur ces modèles mathématiques, aussi la traduction doit-elle être fidèle. Le choix des modèles mathématiques est également capital car l'automatisation de la vérification dépend de la puissance d'expression des modèles choisis. D'une manière générale, plus le modèle est expressif, plus la traduction peut être fidèle et moins la vérification peut être automatisée.

Les modèles mathématiques utilisés pour décrire les systèmes sont généralement des sous-classes particulières de *systèmes de transitions étiquetés*. Intuitivement, un système de transitions étiqueté est un graphe dirigé potentiellement infini, dont les noeuds représentent les *états* du système, et dont les arêtes, qui sont *étiquetées*, décrivent les comportements possibles du système. Quant aux propriétés, leur expression formelle passe bien souvent par l'utilisation de logiques. Les logiques temporelles LTL, CTL et CTL* [Pnu77, Eme90] sont particulièrement bien adaptées à la spécification des systèmes *réactifs*.

Les systèmes les plus simples pourront être modélisés par des systèmes de transitions étiquetés finis, dont la vérification par *model-checking* est entièrement automatique. Cependant, on est souvent confronté en pratique au problème de l'*explosion combinatoire du nombre d'états*. L'utilisation de techniques *symboliques*, les BDD [Bry86, McM93], a permis d'atténuer ce problème et de vérifier des systèmes finis comportant jusqu'à 10^{20} états [BCM+92].

Un système complexe, en revanche, pourra comporter des variables à valeurs dans des domaines de données a priori *non bornés*, telles que des compteurs, des files, des piles, des horloges... Même si en pratique, une fois implantés, de tels systèmes ne disposent que d'une mémoire finie pour toutes ces variables, on souhaiterait les vérifier en tant que systèmes de transitions étiquetés infinis, car :

- on connaît rarement la borne utilisée en pratique sur les domaines de données. On aurait du mal, par exemple, à évaluer la capacité des canaux d'un protocole de communication sur Internet !

- en bornant par des valeurs arbitraires les domaines de données, on est très vite confronté au problème de l'explosion du nombre d'états. Par exemple, en bornant à 20 la taille des canaux de communication d'un protocole utilisant 10 messages, on obtient déjà plus de 10^{20} contenus possibles pour un canal !

- lorsqu'une propriété de *sûreté* est satisfaite par le système infini, on peut généralement en déduire que la propriété sera encore satisfaite en bornant les domaines de données, quelle que soit la borne choisie. On obtient ainsi une *vérification paramétrée.*

Malheureusement, la vérification des systèmes de transitions étiquetés infinis se heurte à un obstacle théorique : l'*indécidabilité.*

Le problème de la vérification des systèmes infinis a fait l'objet de recherches intensives au cours de la dernière décennie. C'est dans ce cadre, et par des méthodes d'*abstraction* et d'*accélération*, que se situe la contribution de cette thèse.

Etat de l'art de la vérification des systèmes infinis

Trois approches principales se dégagent des différents travaux sur l'analyse des systèmes infinis. La première approche consiste à décrire le système ainsi que les propriétés à vérifier dans un assistant de preuve, comme COQ [BBC$^+$97] ou PVS [ORS92]. L'utilisateur devra ensuite prouver formellement, au sein de l'assistant de preuve, que la propriété est satisfaite par le système. Cette approche demande une interaction constante avec l'utilisateur, car c'est lui qui devra fournir les étapes importantes de la preuve, les déductions triviales étant automatisées par l'assistant. Par ailleurs, lorsque la propriété n'est pas satisfaite par le système, il est difficile de s'en rendre compte en essayant de la prouver ! Aussi cette méthode est-elle maintenant utilisée conjointement avec le model-checking, par le biais d'abstractions finies du système, afin de détecter au plus tôt les erreurs les plus grossières et de réduire la vérification du système au model-checking par raffinements successifs [SS99].

La seconde approche réside dans la conception de classes de systèmes infinis, où l'on restreint l'expressivité afin d'obtenir des résultats de décidabilité pour la vérification. De nombreuses classes restreintes de systèmes infinis ont été ainsi définies et analysées, comme par exemple :

– les *réseaux de Petri* [Pet62] ou de manière équivalente *systèmes d'addition de vecteurs* [KM69], pour lesquels le problème de l'accessibilité est décidable [Kos82, May84]. De plus, un graphe de couverture est calculable [KM69, Fin93], ce qui permet de déterminer les places non bornées du réseau, de décider si un état de blocage est accessible, etc. La logique temporelle LTL est décidable pour les réseaux de Petri [Esp97]. Les outils Design-CPN [CJK97], PAPETRI [BJP90] et PEP [Gra97] sont basés sur le modèle des réseaux de Petri.

– les *systèmes d'addition de vecteurs avec états de dimension* 2 (ou de manière équivalente les réseaux de Petri avec deux places non bornées), pour lesquels l'ensemble des états accessibles est semilinéaire et calculable [HP79].

– les *réseaux de Petri étendus* qui forment une hiérarchie très complète d'extensions de réseaux de Petri définie et analysée dans [Duf98]. Parmi les résultats obtenus dans [Duf98, DFS98, DJS99], on retiendra notamment que les problèmes de la terminaison et de la couverture sont décidables pour les réseaux de Petri avec reset et transfert. Le caractère borné est décidable pour les réseaux de Petri avec transfert, ainsi que pour les réseaux de Petri avec deux arcs de reset. En revanche, le caractère borné est indécidable pour les réseaux de Petri avec trois arcs de reset.

– les *réseaux de Petri avec test-à-zéro et avec perte* pour lesquels le model-checking de certaines logiques temporelles (par exemple CTL* interprétée sur des chemins finis) est décidable [BM99]. En revanche, le model-checking de LTL est indécidable pour cette classe.

– les *automates à compteurs plats* [Jur99] pour lesquels la relation binaire d'accessibilité est calculable, ce qui permet de vérifier des propriétés très fines de sûreté.

– les *automates à pile* dont l'ensemble des états accessibles est reconnaissable et calculable [FWW97, BEM97]. Le model-checking des formules de CTL* est décidable pour cette classe. Un algorithme efficace pour le model-checking de LTL a été récemment présenté dans [EHRS00], en vue d'une analyse “inter-procédurale” des programmes.

– les *automates à files linéaires* [GGLR87, CF87, FR88], pour lesquels le problème de l'accessibilité est décidable.

– les *automates à files quasi-stables* dont l'ensemble des états accessibles est reconnaissable et calculable [CF97], ce qui permet de vérifier de nombreuses propriétés sur ces modèles.

– les *automates à files avec perte* et les *automates à files avec insertion* pour lesquels le problème de l'accessibilité est décidable [AJ96b, CFP96]. L'ensemble des états accessibles est même effectivement reconnaissable pour les automates à files avec insertion.

– les *automates temporisés* [AD94] pour lesquels le vide est décidable, ainsi que certaines logiques temporelles temporisées. Les outils KRONOS [Yov97] et UPPAAL [LPY97] sont basés sur des extensions de ce modèle.

– l'*algèbre de processus PA* pour laquelle l'ensemble des termes accessibles est effectivement reconnaissable par automate d'arbres [LS00].

Enfin la dernière approche se propose de développer des techniques d'analyse incomplètes, mais entièrement automatiques, sur des modèles très expressifs (et donc indécidables). L'incomplétude de ces techniques peut provenir de deux facteurs :

– la non-terminaison éventuelle des algorithmes employés. C'est le cas dans de nombreux travaux où l'on cherche à "semi-calculer" par des méthodes *symboliques* l'ensemble des états accessibles de systèmes de transitions étiquetés infinis [BW94, CJ98, BGWW97, BH99, AAB99, HM00b]. L'outil HYTECH [HHWT97] de vérification des *systèmes hybrides* est ainsi basé sur le semi-calcul de l'ensemble des états accessibles.

– la non-exactitude de l'analyse réalisée. L'*interprétation asbraite*, ainsi que d'autres méthodes de *génération statique d'invariants*, reposent sur le calcul d'une approximation supérieure de l'ensemble des états accessibles dans le système [CC77, CH78, JN95, Cou00]. Cette approximation supérieure peut permettre, lorsqu'elle est suffisamment précise, de vérifier que le système considéré satisfait une propriété donnée [Hal93, LMR$^+$98].

Comment *bien* semi-calculer l'ensemble des états accessibles ?

Les résultats de décidabilité de la vérification dans des classes restreintes sont intéressants du point de vue théorique, mais ont en général peu d'impact pratique, car il est souvent difficile de modéliser fidèlement un système complexe dans de telles classes. En revanche, le succès de l'outil HYTECH montre qu'une méthode semi-algorithmique de calcul de l'ensemble des états accessibles post^* (ou de son dual pre^*) peut s'avérer efficace en pratique. Ce constat n'est pas très surprenant, car la puissance d'expression des modèles indécidables est rarement utilisée intégralement dans les applications pratiques.

Cependant, tous les semi-algorithmes ne sont pas efficaces en pratique, bien au contraire. Par exemple, un semi-algorithme naïf de calcul de post^*, consistant à explorer l'arbre d'accessibilité en espérant qu'au plus un nombre fini d'états seront accessibles, ne terminera pas dès que post^* sera infini, ce qui est assez fréquent !

Afin de concevoir un *bon* semi-algorithme pour le calcul de l'ensemble des états accessibles, on devra préalablement déterminer une *représentation symbolique*, c'est-à-dire un moyen de manipuler des ensembles infinis d'états. On aura également parfois besoin d'accélérer le calcul, c'est-à-dire de pouvoir collecter en une étape (mais de complexité raisonnable) un ensemble infini d'états accessibles. Ensuite, afin de mesurer l'efficacité du semi-algorithme, on pourra le tester sur les exemples classiques ainsi que sur les classes décidables. Rappelons brièvement les différents travaux dans ce domaine :

- les QDD [BGWW97] et les CQDD [BH99] sont des représentations symboliques pour les automates à files FIFO. L'effet de l'itération d'un circuit quelconque de l'automate à files est représentable et calculable avec les CQDD, alors que ce résultat est limité aux circuits "qui ne comptent pas" dans les cas des QDD.

- les SRE [ABJ98] forment une représentation symbolique pour les automates à files FIFO avec perte. L'effet de l'itération d'un circuit quelconque de l'automate est représentable et calculable avec les SRE, avec de plus une faible complexité.

- les ensembles périodiques permettent de représenter symboliquement des ensembles d'états de programmes avec variables entières [BW94]. Les opérations considérées sont exprimées sous forme matricielle, et il est montré que l'effet de l'itération d'une opération est représentable et calculable avec les ensembles périodiques.

- les formules de Presburger sont utilisées dans [CJ98] afin de représenter symboliquement des ensembles d'états d'automates à compteurs. La relation binaire d'accessibilité décrivant l'effet de l'itération d'un circuit quelconque de l'automate à compteurs est exprimable par une formule de Pressburger, et une telle formule est calculable.

- les polyèdres convexes [CH78, Hal93] sont utilisés dans HYTECH afin de représenter finiment des ensembles d'états de systèmes hybrides [AHH96].

Ces travaux sont en fait basés sur des concepts similaires : les hypothèses sur les représentations symboliques sont semblables (clôture effective par union, décidabilité de l'inclusion, etc.) et l'accélération du calcul passe par l'itération de certains circuits. Néanmoins ces concepts n'ont pas, à notre connaissance, été formalisés dans un cadre général permettant la conception et la comparaison de représentations symboliques et d'accélérations. Les notions d'accélération (ou de *meta-transition*) proposées ne sont pas identiques, et leurs liens avec l'*élargissement* de l'interprétation abstraite, même s'ils sont parfois évoqués, n'ont pas été approfondis. De plus, les semi-algorithmes proposés n'utilisent pas leur stratégie d'accélération de la même manière. Or la définition d'un cadre uniforme et d'un semi-algorithme générique sont à notre avis des étapes indispensables à la conception d'un outil utilisant toutes ces techniques.

Contributions de la thèse

Comme nous le verrons en détail au chapitre 2, le cadre général de l'interprétation abstraite ne permet pas de capturer les différents travaux cités plus haut sur les représentations symboliques, essentiellement parce que l'élargissement *n'est pas la bonne notion d'accélération* pour englober les résultats de calcul symbolique *exact* de l'effet de l'itération d'un circuit. En effet, la fonction d'élargissement permet d'accélérer la convergence, mais ne tient pas suffisamment compte du système de transitions étiqueté sous-jacent pour pouvoir être exacte en général.

Aussi définissons-nous dans cette thèse un nouveau cadre d'*abstraction* et d'*accélération* des systèmes de transitions étiquetés. Nos définitions sont basées sur un ensemble "minimal" d'hypothèses, ce qui nous permettra de capturer les différents travaux sur les représentations symboliques. Nous proposons aussi un semi-algorithme générique pour le calcul *exact* de l'ensemble des états accessibles, ainsi qu'une accélération générique. Ces notions sont ensuite appliquées à l'analyse des systèmes plats, où la notion de platitude sera ici exprimée par une condition sur l'ensemble des traces du système, plus générale que les notions structurelles de platitude utilisées dans [CJ98, CJ99, CC00]. Nous donnons un algorithme calculant l'ensemble des états accessibles d'un système de transitions étiqueté plat. Nous verrons également que les travaux sur les *systèmes de transitions bien structurés* [Fin90, AČJ00, EN98, FS00a] peuvent s'exprimer en termes d'abstractions et d'accélérations. Nous donnons une méthode générale pour construire une abstraction et une "bonne" accélération dans le cadre des *automates étendus*.

Nous appliquons ensuite ces idées à l'analyse de plusieurs modèles de systèmes infinis : les *systèmes d'addition de vecteurs avec états étendus*, les *automates à files* FIFO et les *automates à file réactifs*. Nous définissons des abstractions simples, et les accélérations associées, basées sur des représentations symboliques par ensembles semilinéaires, par expressions régulières (ou QDD), par CQDD et par expressions régulières semilinéaires. Il est intéressant de noter que les expressions régulières semilinéaires (SLRE) apparaissent à plusieurs endroits dans la thèse : dans la définition des systèmes plats, dans la construction d'une accélération générique pour les automates étendus, et comme représentation symbolique de contenus de files FIFO.

Nous nous intéresserons plus particulièrement aux SAVE étendus de dimension 2. Cette classe est intéressante car d'une part les SAVE de dimension 2 étendus par des test-à-zéro ont la puissance des machines de Turing, et d'autre part les SAVE (non étendus) de dimension 2 ont un ensemble d'états accessibles effectivement semilinéraire [HP79]. Nous introduisons une variante du semi-algorithme générique calculant post*, ainsi que des accélérations adaptées qui garantiront la terminaison de ce semi-algorithme pour certaines classes de SAVE étendus de dimension 2, généralisant ainsi le résultat d'HOPCROFT et PANSIOT. Nous obtenons que pre* est effectivement semilinéaire, et que post* est semilinéaire, pour la classe des SAVE de dimension 2 étendus par des tests-à-zéro sur le premier compteur et par des reset sur le second compteur. A l'aide d'une technique récente de JANČAR [DJS99], les *schéma de chemin réguliers*, nous montrons que post* et pre* sont effectivement semilinéaires pour la classe des SAVE de dimension 2 avec reset et transfert. Nous décrirons ainsi une hiérarchie exhaustive des SAVE étendus de dimension 2, complète vis-à-vis de la décidabilité de l'accessibilité et presque complète (il reste un problème ouvert) vis-à-vis de l'effective semilinéarité de post* et de pre*.

Nous définissons une nouvelle représentation symbolique de contenus de files FIFO basée sur les SLRE. Nous comparons les abstractions et les accélérations basées sur les QDD, les CQDD et les SLRE. Nous verrons que les SLRE sont capables d'itérer les mêmes circuits que les QDD et que les SLRE sont précisément à l'intersection des QDD et des CQDD. Nous analyserons ensuite la complexité de l'inclusion pour les SLRE, et nous montrons que ce problème est coNP–complet. Cette comparaison nous permet donc de nous rendre compte que les SLRE semblent *suffisants* pour représenter symboliquement des contenus de files FIFO et pour itérer des séquences d'opérations, et de plus ils ont une *meilleure complexité*. Nous terminons notre analyse des automates à files FIFO en rappelant les résultats connus sur la variante avec perte de ce modèle et en montrant comment ils s'inscrivent naturellement dans notre cadre d'abstraction et d'accélération.

Les automates à file réactifs permettent de modéliser des systèmes réactifs avec mémorisation d'événements [CR95, SFRC99]. Les occurrences d'événements ne pouvant pas être traitées dans l'état courant du système sont mémorisés dans une file afin d'être traités ultérieurement. Cependant, la file ne respecte pas la sémantique FIFO standard. En effet, afin de réagir au plus vite sur une occurrence d'événement mémorisée, la priorité est donnée aux "post-traitements" à partir de la file, et on parle alors de sémantique FIFFO (pour *First In First Fireable Out*). Le modèle initial introduit dans [CR95, SFRC99] comportait des contraintes structurelles fortes, correspondant au langage réactif ELECTRE dont il était issu, qui nous ont permis d'obtenir de nombreux résultats de décidabilité [SFRC99]. Ces résultats de décidabilité sont néanmoins surprenants car ce modèle est proche de celui des automates à une seule file FIFO, où le problème de l'accessibilité est indécidable. Nous isolerons dans cette thèse les contraintes structurelles d'une part, et les différentes sémantiques entre la sémantique FIFO et la sémantique FIFFO d'autre part. Nous analyserons ainsi exhaustivement les différents modèles obtenus, du point de vue de la décidabilité de l'accessibilité, ainsi que de l'effective reconnaissabilité de post* et de pre*.

Plan de la thèse

Au *chapitre* 1, nous rappellons quelques notions bien connues du domaine de la vérification des systèmes de transitions étiquetés. Nous y présentons également certaines notations qui servirons tout au long de la thèse.

Au *chapitre* 2, nous introduisons notre schéma d'abstraction et d'accélération des systèmes de transitions étiquetés, après une discussion sur les avantages et les inconvénients de l'interprétation abstraite. Un semi-algorithme générique semi-calculant l'ensemble des états accessibles sera proposé. Nous appliquons ensuite cette approche au calcul de l'ensemble des états accessibles d'un système plat, et à l'analyse des systèmes pré-ordonnés. Nous conclurons ce chapitre par une méthode générale d'abstraction et d'accélération des automates étendus.

Au *chapitre* 3, nous appliquons les notions d'abstraction et d'accélération à l'analyse des systèmes d'addition de vecteurs avec états étendus. Nous montrons comment les résultats sur le calcul d'un arbre de couverture [KM69, Fin93] peuvent de voir en termes d'abstraction et d'accélération. Nous définissons une abstraction et une accélération basées sur les sous-ensembles semilinéaires de vecteurs d'entiers. Nous nous intéressons ensuite aux SAVE étendus de dimension 2, et nous obtenons une hiérarchie exhaustive des SAVE étendus de dimension 2, complète vis-à-vis de la décidabilité de l'accessibilité et presque complète vis-à-vis de l'effective semilinéarité de post^* et de pre^*. Nous montrerons enfin comment appliquer ces résultats au calcul d'approximations supérieures et inférieures de post^* et de pre^* pour les SAVE étendus de dimension quelconque.

Au *chapitre* 4, nous appliquons les notions d'abstraction et d'accélération à l'analyse des automates à files FIFO. Nous définissons une nouvelle représentation symbolique de contenus de files FIFO basée sur les SLRE. Nous ensuite comparons les abstractions et les accélérations basées sur les QDD, les CQDD et les SLRE. Nous conclurons ce chapitre par l'analyse des automates à files FIFO non fiables.

Au *chapitre* 5, nous étudions les automates à file réactifs, du point de vue de la décidabilité de l'accessibilité, ainsi que de l'effective reconnaissabilité de post^* et de pre^*.

Nous conclurons cette thèse par un résumé des idées et des résultats présentés, puis par quelques perspectives.

Chapitre 1

Vérification des systèmes de transitions étiquetés

Lorsqu'on souhaite vérifier un programme ou un protocole `P`, il faut exprimer formellement l'ensemble des propriétés à vérifier sur `P`, mais il faut aussi donner une sémantique précise à `P`. Nous utiliserons dans cette thèse la *sémantique opérationnelle des petits pas*, associant à `P` un système de transitions étiqueté. Les propriétés que nous voudrons vérifier, quant à elles, seront des propriétés basiques de sûreté et s'exprimeront sous la forme d'ensembles d'états accessibles.

Ce chapitre se veut un rappel de quelques notions bien connues du domaine de la vérification des systèmes de transitions étiquetés, mais nous y présentons également certaines notations dont nous nous servirons tout au long de la thèse.

Après le rappel de notions préliminaires en section 1.1, nous présenterons le modèle des systèmes de transitions étiquetés en section 1.2. Le problème de la vérification des systèmes de transitions étiquetés sera abordé dans la section 1.3.

1.1 Préliminaires

Cette section fixe les notions et les notations de base qui seront utilisées dans toute la suite du document.

1.1.1 Ensembles, vecteurs, suites

On désigne par $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$ et $\mathbb{R}$ les ensembles usuels de nombres. Si $X \subseteq \mathbb{R}$, on note X^+ (resp. X^*, X^{+*}) l'ensemble des éléments positifs ou nuls (resp. non nuls, strictement positifs) de X. On écrit $\leq$ l'ordre habituel sur les nombres. On utilise les notations usuelles pour les intervalles :

- les intervalles de réels sont notés $[a, b]$, $[a, d[$, $]c, b]$ et $]c, d[$, avec $a, b \in \mathbb{R}$ et $c, d, \in \mathbb{R} \cup \{\infty\}$,
- les intervalles d'entiers sont notés $[k\,..\,l]$, $[k\,..{+\infty}[$ ou $]{-\infty}\,..\,k]$, avec $k, l \in \mathbb{Z}$.

On note $\mathcal{P}(X)$ (resp. $\mathcal{P}_f(X)$) l'ensemble des parties (resp. parties finies) d'un ensemble X. Si X est un ensemble fini, $|X|$ désigne son cardinal. Un élément

$x \in X$ sera parfois confondu avec le singleton $\{x\}$. Ainsi, lorsqu'une notion sera définie pour un ensemble X, on l'appliquera parfois directement à un élément $x \in X$ en le confondant avec $\{x\}$.

Etant donné deux ensembles X et X', on écrit $f : X \to X'$ lorsque f est une fonction partielle de X dans X'. Le domaine de définition de f s'écrit $\mathrm{Dom}(f)$. Si Y est un sous-ensemble de X, on écrit $f(Y)$ pour le sous-ensemble de X' défini par $f(Y) = \{f(y) \;/\; y \in Y \cap \mathrm{Dom}(f)\}$. On désigne classiquement par $\circ$ la composition des fonctions. Lorsque f est *injective* (i.e. $f(x) = f(y) \Rightarrow x = y$), nous notons f^{-1} la fonction partielle de X' dans X inverse de f, c'est-à-dire de domaine de définition $f(X)$ et telle que $f^{-1} \circ f(x) = x$ pour tout $x \in \mathrm{Dom}(f)$.

Les fonctions que nous considèrerons dans toute la suite de la thèse seront *implicitement totales* (i.e. $\mathrm{Dom}(f) = X$), et nous préciserons explicitement lorsqu'une fonction sera *partielle*.

La fonction identité sur X est notée Id_X. La puissance $n^{ème}$ d'une fonction $f : X \to X$ est définie récursivement par $f^0 = Id_X$ et $f^n = (f^{n-1}) \circ f$. Une fonction $\oplus : X \times X \to X$ est aussi appelée *opérateur* et on écrira simplement $x \oplus y$ pour $\oplus(x, y)$. Une *loi de composition interne* sur X est un opérateur commutatif et associatif sur X.

Soit X un ensemble et I un ensemble fini d'indices. On note X^I l'ensemble des *vecteurs* indexés par I d'éléments de X. Pour tout indice $i \in I$, on écrit $\mathsf{x}[i]$ la valeur d'un vecteur x à l'indice i. Etant donné un vecteur $\mathsf{x} \in X^I$, un élément $y \in X$ et un indice $i \in I$, on note $\mathsf{x}[i := y]$ le vecteur x' défini par $\mathsf{x}'[i] = y$ et $\mathsf{x}'[j] = \mathsf{x}[j]$ pour tout $j \neq i$. Cette définition est étendue récursivement aux suites d'indices comme suit :

$$\mathsf{x}[i_1 := y_1] \cdots [i_{k-1} := y_{k-1}][i_k := y_k] = (\mathsf{x}[i_1 := y_1] \cdots [i_{k-1} := y_{k-1}])[i_k := y_k]$$

Un vecteur $\mathsf{x} \in X^I$ sera aussi noté anonymement $\langle i \mapsto \mathsf{x}[i] \rangle$, l'ensemble des indices I se comprenant d'après le contexte. Pour tout $n \in \mathbb{N}^*$, on utilise la notation concise X^n pour l'ensemble de vecteurs $X^{[1\,..\,n]}$. Un vecteur $\mathsf{x} \in X^n$ pourra aussi être donné (anonymement) en extension sous la forme du n-uplet $\mathsf{x} = (\mathsf{x}[1], \mathsf{x}[2], \ldots, \mathsf{x}[n])$.

Une *suite* d'éléments de X est une fonction $x : I \to X$, également notée $(x_i)_{i \in I}$ ou encore plus simplement (x_i), où $I \subseteq \mathbb{N}$ est un intervalle d'entiers. On utilise les notations suivantes :

- si I est fini, alors I est de la forme $I = [k\,..\,l]$ et on dit que $(x_i)_{i \in I}$ est *finie*. De plus, la suite $(x_i)_{i \in I}$ s'écrit également $(x_i)_{k \leq i \leq l}$.
- sinon, I est de la forme $I = [k\,..\,+\infty[$, avec $k \in \mathbb{N}$ et on dit que $(x_i)_{i \in I}$ est *infinie*. De plus, la suite $(x_i)_{i \in I}$ s'écrit également $(x_i)_{i \geq k}$.

Afin de traiter d'une façon uniforme les suites finies et infinies, on écrira parfois abusivement une suite finie $(x_i)_{k \leq i \leq l}$ sous la forme simplifiée $(x_i)_{i \geq k}$.

On note ε la *suite vide* $\varepsilon : \emptyset \to \emptyset$. Pour tout ensemble X, la suite vide ε est une suite d'éléments de X. On remarque que si $(x_i)_{k \leq i \leq l}$ est une suite finie, alors deux cas se présentent :

– si $k > l$ alors (x_i) est la suite vide, et on dit que (x_i) est de *longueur* nulle,

– sinon, on a nécessairement $0 \leq k \leq l$, et on dit que (x_i) est de *longueur* $l - k + 1$.

Une *sous-suite* d'une suite $(x_i)_{i \in I}$ est une suite $(x_{\varphi(j)})_{j \in J}$ où $\varphi : J \to I$ est une *extraction*, c'est-à-dire une fonction satisfaisant $\varphi(j + 1) \geq \varphi(j) + 1$. Un *décalage* d'une suite $(x_i)_{i \in I}$ est une suite $(x_{\varphi(j)})_{j \in J}$ où $\varphi : J \to I$ est une bijection telle que $\varphi(j + 1) = \varphi(j) + 1$. On identifie une suite et ses décalages, ce qui nous permet :

– lorsqu'on se donne une suite $(x_i)_{i \in I}$, de supposer sans perte de généralité que I est de la forme $[0 \mathinner{..} k]$ ou $[0 \mathinner{..} +\infty[$,

– d'écrire une suite $(x_i)_{i \geq 0}$ en extension, sous la forme $x_0, x_1, \ldots, x_i, \ldots$.

1.1.2 Notions d'ordre

Un *pré-ordre* $\preceq$ sur un ensemble X est une relation binaire réflexive et transitive sur X. Un *ordre partiel* sur X est un pré-ordre antisymétrique sur X. Un *ordre total* $\preceq$ sur X est un ordre partiel sur X tel que pour tout $x, y \in X$ on a $x \preceq y$ ou $y \preceq x$. Si $\preceq$ est un pré-ordre (resp. ordre partiel, ordre total) sur X, alors on dit que le couple $(X, \preceq)$ est un *ensemble pré-ordonné* (resp. *ensemble partiellement ordonné*, *ensemble totalement ordonné*). Etant donné un pré-ordre (resp. ordre partiel, ordre total) $\preceq$ sur X, on note $\succeq$ le pré-ordre (resp. ordre partiel, ordre total) dual sur X défini par $x \succeq y$ si $y \preceq x$. De plus, on écrit $x \prec y$ (et $y \succ x$) lorsqu'on a $x \preceq y$ et $y \not\preceq x$.

Une fonction $f : X \to X'$ entre deux ensembles pré-ordonnés $(X, \preceq)$ et $(X', \preceq')$ est *croissante* (resp. *strictement croissante*) si $x \preceq y \;\Rightarrow\; f(x) \preceq' f(y)$ (resp. $x \prec y \;\Rightarrow\; f(x) \prec' f(y)$) pour tout $x, y \in X$. Les notions de fonctions *décroissantes* et de fonctions *strictement décroissantes* sont duales (elles sont obtenues à partir des définitions de fonctions croissantes et de fonctions strictement croissantes ci-dessus en remplaçant le pré-ordre $\preceq'$ par son dual $\succeq'$).

Un pré-ordre $\preceq$ sur X est *bien fondé* s'il n'admet pas de suite infinie strictement décroissante. Un *beau pré-ordre* sur un ensemble X est pré-ordre $\preceq$ sur X tel que toute suite infinie $(x_i)_{i \in \mathbb{N}}$ d'éléments de X admet une sous-suite infinie croissante $x_{i_0} \preceq x_{i_1} \preceq x_{i_2} \cdots x_{i_j} \preceq x_{i_{j+1}} \cdots$. Observons qu'un beau pré-ordre est nécessairement bien fondé. Un ensemble pré-ordonné $(X, \preceq)$ satisfait la *condition de chaîne ascendante* s'il n'existe pas de suite infinie strictement croissante d'éléments de X. On remarque que $(X, \preceq)$ satisfait la condition de chaîne ascendante si et seulement si $\succeq$ est bien fondé.

Un pré-ordre $\preceq$ sur un ensemble X induit naturellement une relation d'équivalence $\approx$ sur X (i.e. un pré-ordre symétrique sur X), définie par $x \approx y$ si $x \preceq y$ et $y \preceq x$. De plus, le pré-ordre $\preceq$ donne lieu à un ordre partiel sur l'ensemble quotient $X_{/\approx}$ des classes d'équivalence de X pour $\approx$.

Etant donné un pré-ordre $\preceq$ sur X et un sous-ensemble $Y \subseteq X$, un *élément minimal* (resp. un *élément maximal*) de Y est un élément $m \in Y$ tel que pour tout $y \in Y$, si $y \preceq m$ alors $y \approx m$ (resp. si $y \succeq m$ alors $y \approx m$). On dénote par $\mathrm{Min}(Y, \preceq)$ (resp. $\mathrm{Max}(Y, \preceq)$) l'ensemble des éléments minimaux (resp. maximaux) de Y pour le pré-ordre $\preceq$. On note plus simplement $\mathrm{Min}(Y)$ et $\mathrm{Max}(Y)$ lorsque le pré-ordre $\preceq$ se comprend d'après le contexte. On rappelle que lorsque $\preceq$ est un beau pré-ordre, $\mathrm{Min}(Y)$ et $\mathrm{Max}(Y)$ contiennent chacun un nombre fini d'éléments non équivalents, pour tout $Y \subseteq X$.

Etant donné un ensemble pré-ordonné $(X, \preceq)$ et un sous-ensemble $Y \subseteq X$, la *clôture inférieure* (resp. *clôture supérieure*) de Y par le pré-ordre $\preceq$, notée $\downarrow(Y, \preceq)$ (resp. $\uparrow(Y, \preceq)$), est définie par $\downarrow(Y, \preceq) = \{x \in X \mid \exists\, y \in Y, x \preceq y\}$ (resp. $\uparrow(Y, \preceq) = \{x \in X \mid \exists\, y \in Y, x \succeq y\}$). On note plus simplement $\downarrow Y$ et $\uparrow Y$ lorsque le pré-ordre $\preceq$ se comprend d'après le contexte. Un sous-ensemble $Y \subseteq X$ est *clos inférieurement* (resp. *clos supérieurement*) s'il est égal à sa clôture inférieure (resp. clôture supérieure). On observe que le complémentaire d'un sous-ensemble clos inférieurement (resp. supérieurement) est clos supérieurement (resp. inférieurement). Une *base* d'un sous-ensemble Y clos inférieurement (resp. supérieurement) est un ensemble $B \subseteq X$ tel que $Y = \downarrow B$ (resp. $Y = \uparrow B$). On rappelle que lorsque $\preceq$ est bien fondé, $\mathrm{Min}(Y)$ est une base de Y, pour tout $Y \subseteq X$ clos supérieurement.

1.1.3 Mots, langages

Soit Σ un *alphabet* (un ensemble fini de *symboles*). Un *mot fini* (resp. *mot infini*) w sur Σ est une suite finie (resp. infinie) d'éléments de Σ. On note Σ^* (resp. Σ^ω) l'ensemble des mots finis (resp. mots infinis) sur Σ. On désigne par Σ^∞ l'ensemble des *mots* finis ou infinis sur Σ, c'est-à-dire $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. Un mot fini $w = (w_i)_{0 \leq i \leq k} \in \Sigma^*$ est plus simplement noté $w = w_0 w_1 \cdots w_k$. Un mot fini ou infini $w = (w_i)_{i \geq 0} \in \Sigma^\infty$ est plus simplement noté $w = w_0 w_1 \cdots w_i \cdots$. Le mot vide s'écrit toujours ε. On désigne par Σ^+ l'ensemble des mots finis non vides, c'est-à-dire $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$.

La *longueur* d'un mot fini $w \in \Sigma^*$ est notée $|w|$. Soit deux mots $w \in \Sigma^*$ et $w' \in \Sigma^\infty$, donnés par $w = w_0 w_1 \cdots w_k$ et $w' = w'_0 w'_1 \cdots w'_i \cdots$. La *concaténation* de w et de w', notée $w \cdot w'$ (ou plus simplement ww'), est le mot défini par $w \cdot w' = w_0 w_1 \cdots w_k w'_0 w'_1 \cdots w'_i \cdots$. On rappelle que ε est élément neutre pour la concaténation.

Un *langage* L est un ensemble de mots $L \subseteq \Sigma^\infty$. On étend la concaténation aux langages par $L \cdot L' = \{w \cdot w' \mid w \in L, w' \in L'\}$ où $L \subseteq \Sigma^*$ et $L' \subseteq \Sigma^\infty$. La *puissance $n^{\text{ème}}$* d'un langage $L \subseteq \Sigma^*$ est définie récursivement par $L^0 = \{\varepsilon\}$ et $L^n = L \cdot (L^{n-1})$. La *fermeture de Kleene* d'un langage $L \subseteq \Sigma^*$ est $L^* = \bigcup_{i \in \mathbb{N}} L^i$. Un mot est *primitif* s'il n'est pas la puissance d'un autre mot.

Nous nous servirons de quelques notions bien connues de théorie des langages [HU79, Ber79]. Un langage $L \subseteq \Sigma^*$ est *rationnel* s'il peut être reconnu par un automate fini. Nous notons $REG(\Sigma)$ l'ensemble des *expressions régulières* ρ sur un alphabet Σ. On désigne par $[\![\rho]\!]$ le langage (rationnel) associé à une expression régulière ρ, et on écrit $\emptyset$ pour l'expression régulière dénotant l'ensemble vide de mots.

Un sous-ensemble $L \subseteq (\Sigma^*)^I$ de vecteurs de mots est *reconnaissable* s'il peut s'écrire comme une union finie de produits de langages rationnels.

Nous utiliserons deux ordres partiels sur les mots. Un mot $w \in \Sigma^*$ est un *facteur gauche* d'un mot $w' \in \Sigma^*$, noté $w \leq w'$, s'il existe un mot $z \in \Sigma^*$ tel que $w' = w \cdot z$; de plus, on écrit alors $z = w^{-1}w'$. La relation facteur gauche $\leq$ est un ordre partiel sur Σ^*. Pour tout langage $L \subseteq \Sigma^*$, on désigne par $\mathrm{FG}(L)$ l'ensemble des facteurs gauches d'éléments de L, ou plus formellement $\mathrm{FG}(L) = \downarrow(L, \leq)$. Le *dérivatif* $w^{-1}L$ d'un langage $L \subseteq \Sigma^*$ par un mot $w \in \Sigma^*$ est le langage $\{z \;/\; w \cdot z \in L\}$.

Un mot $w \in \Sigma^*$ est un *sous-mot* d'un mot $w' \in \Sigma^*$, noté $w \,|\, w'$, si w est une sous-suite de w'. On remarque que $w \,|\, w'$ si et seulement s'il existe des mots $x_0, x_1, \ldots, x_k, x'_0, x'_1, \ldots, x'_{k+1} \in \Sigma^*$ tels que $w = x_0 x_1 \cdots x_k$ et $w' = x'_0 x_0 x'_1 x_1 \cdots x'_k x_k x'_{k+1}$. La relation sous-mot $|$ est un ordre partiel sur Σ^*.

Proposition 1.1.1 ([Hig52]) *L'ordre partiel sous-mot est un beau pré-ordre sur Σ^*.*

On rappelle que la clôture supérieure d'un mot $w = w_0 w_1 \cdots w_k$ pour l'ordre partiel sous-mot est le langage rationnel $\Sigma^* w_0 \Sigma^* w_1 \Sigma^* \cdots \Sigma^* w_k \Sigma^*$. Par conséquent, on déduit de la proposition 1.1.1 que tout langage clos supérieurement (resp. inférieurement) pour l'ordre partiel sous-mot est rationnel.

1.2 Systèmes de transitions étiquetés

Nous rappelons dans cette section le modèle des systèmes de transitions étiquetés, sur lequel est basé cette thèse, ainsi que plusieurs notions associées. On pourra consulter [Arn92] pour une description plus détaillée des systèmes de transitions étiquetés.

Définition 1.2.1 *Un* système de transitions étiqueté *est un triplet $\mathcal{LTS} = (S, \Sigma, \rightarrow)$ où S est un ensemble d'*états*, Σ est un alphabet d'*étiquettes *et le sous-ensemble $\rightarrow \subseteq S \times \Sigma \times S$ est une* relation de transitions étiquetée.

Lorsque l'ensemble S des états est fini (resp. infini), on dit que le système de transitions étiqueté est fini (resp. infini). Un système de transitions étiqueté est *déterministe* si pour tout état $s \in S$ et pour toute étiquette $\alpha \in \Sigma$ il existe un unique état $s' \in S$ tel que $s \xrightarrow{\alpha} s'$.

Exemple 1.2.2 La figure 1.1 décrit la *structure de contrôle* EC d'un système à deux compteurs c et d. Cette structure de contrôle est formellement définie comme le système de transitions étiqueté fini $\mathsf{EC} = (Q, \Sigma, \rightarrow_{\mathsf{EC}})$, où :

- $Q = \{q_0, q_1, q_2\}$ est l'ensemble fini des *états de contrôle*, et
- $\Sigma = \{c = 0, c \leftarrow c + 2, c \leftarrow c - 7, (c \leftarrow c + 1) \wedge (d \leftarrow d + 3)\}$ est l'alphabet d'étiquettes, et

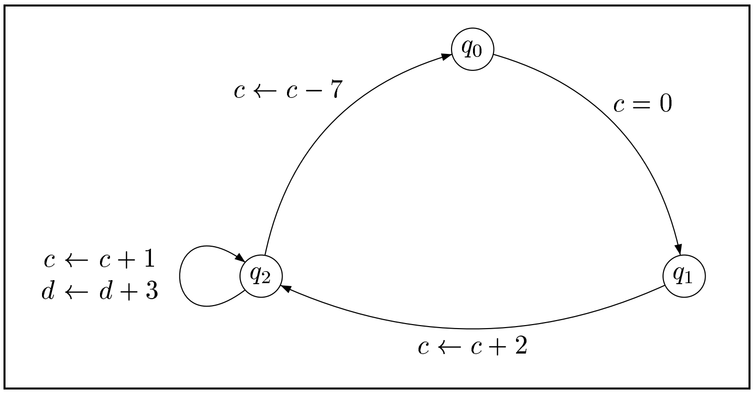


FIG. 1.1 – Exemple de système à deux compteurs

– la relation de transitions étiquetée $\rightarrow_{\mathsf{EC}}$ est donnée par :

$$\rightarrow_{\mathsf{EC}} \; = \left\{ \begin{array}{lcl} (q_0, & c = 0 & , q_1)\,, \\ (q_1, & c \leftarrow c + 2 & , q_2)\,, \\ (q_2, & c \leftarrow c - 7 & , q_0)\,, \\ (q_2, (c \leftarrow c+1) \wedge (d \leftarrow d+3) & & , q_2) \end{array} \right\}$$

Exemple 1.2.3 La structure de contrôle $\mathsf{EC} = (Q, \Sigma, \rightarrow_{\mathsf{EC}})$ ne décrit pas comment les "opérations" de Σ influent sur les compteurs c et d. Intuitivement, lorsque le système de la figure 1.1 passe de l'état de contrôle q_1 à l'état de contrôle q_2, on s'attend à ce que la valeur du compteur c soit incrémentée par 2. Cette idée intuitive est formalisée par la sémantique opérationnelle qu'on associe à EC, donnée sous la forme d'un système de transitions étiqueté infini $\mathcal{EC}$. Il est formellement défini par $\mathcal{EC} = (S, \Sigma, \rightarrow_{\mathcal{EC}})$ où :

– $S = Q \times \mathbb{N}^{\{c,d\}}$ est l'ensemble des états $s = (q, \mathsf{x})$ composés d'un *état de contrôle* q et d'un vecteur x représentant la valeur des variables entières c et d, et

– Σ est le même alphabet d'étiquettes que celui de EC, et

– la relation de transitions étiquetée $\rightarrow_{\mathcal{EC}}$ est le plus petit sous-ensemble $\rightarrow_{\mathcal{EC}}$ de $\subseteq S \times \Sigma \times S$ tel que pour tout $\mathsf{x} \in \mathbb{N}^{\{c,d\}}$:

  1. si $\mathsf{x}[c] = 0$ alors $(q_0, \mathsf{x}) \xrightarrow{c=0}_{\mathcal{EC}} (q_1, \mathsf{x})$, et
  2. on a $(q_1, \mathsf{x}) \xrightarrow{c \leftarrow c+2}_{\mathcal{EC}} (q_2, \mathsf{x}[c := \mathsf{x}[c] + 2])$, et
  3. si $\mathsf{x}[c] \geq 7$ alors $(q_2, \mathsf{x}) \xrightarrow{c \leftarrow c-7}_{\mathcal{EC}} (q_0, \mathsf{x}[c := \mathsf{x}[c] - 7])$, et
  4. on a $(q_2, \mathsf{x}) \xrightarrow{(c \leftarrow c+1) \wedge (d \leftarrow d+3)}_{\mathcal{EC}} (q_2, \mathsf{x}[c := \mathsf{x}[c] + 1][d := \mathsf{x}[d] + 3])$,

Cette thèse s'inscrit dans le cadre de l'analyse des systèmes de transitions étiquetés. Cependant, certaines considérations dans ce chapitre ne tiendront pas compte de l'étiquetage des transitions, et s'appliqueront donc aussi bien aux systèmes de transitions non étiquetés. Rappelons qu'un *système de transitions* (non étiqueté) est classiquement défini comme un couple $\mathcal{TS} = (S, \rightarrow)$ où S est un ensemble d'états et $\rightarrow \subseteq S \times S$ est une relation de transitions (non étiquetée).

Remarque. Un système de transitions étiqueté $\mathcal{LTS} = (S, \Sigma, \rightarrow_{\mathcal{LTS}})$ définit implicitement un système de transitions (non étiqueté) $\mathcal{TS} = (S, \rightarrow_{\mathcal{TS}})$, où la relation de transitions (non étiquetée) $\rightarrow_{\mathcal{TS}}$ est définie par $s \rightarrow_{\mathcal{TS}} s'$ s'il existe une étiquette $\alpha \in \Sigma$ telle que $s \xrightarrow{\alpha}_{\mathcal{LTS}} s'$. Observons également que les système de transitions (non étiquetés) sont isomorphes aux systèmes de transitions étiquetés dont l'alphabet d'étiquettes est un singleton.

1.2.1 Simulation

Etant donné deux systèmes de transitions étiquetés $\mathcal{LTS}_1 = (S_1, \Sigma, \rightarrow_1)$ et $\mathcal{LTS}_2 = (S_2, \Sigma, \rightarrow_2)$ avec états initiaux respectifs $X_1 \subseteq S_1$ et $X_2 \subseteq S_2$, une *relation de simulation* entre $(\mathcal{LTS}_1, X_1)$ et $(\mathcal{LTS}_2, X_2)$ est une relation binaire $R \subseteq S_1 \times S_2$ satisfaisant les deux conditions suivantes :

1. on a $X_1 \times X_2 \subseteq R$,
2. pour tout $(s_1, s_2) \in R$ et $\alpha \in \Sigma$, si $s_1 \xrightarrow{\alpha}_1 s'_1$ alors il existe s'_2 tel que $(s'_1, s'_2) \in R$ et $s_2 \xrightarrow{\alpha}_2 s'_2$.

On dit que $(\mathcal{LTS}_1, X_1)$ *simule* $(\mathcal{LTS}_2, X_2)$ s'il existe une relation de simulation entre $(\mathcal{LTS}_1, X_1)$ et $(\mathcal{LTS}_2, X_2)$.

1.2.2 Chemins

Soit $\mathcal{LTS} = (S, \Sigma, \rightarrow)$ un système de transitions étiqueté. Une suite d'étiquettes $\sigma \in \Sigma^*$ est aussi appelée une *séquence d'exécution*. Une suite de transitions $(s_i \xrightarrow{\alpha_i} s'_i)_{i \in I}$ est *contiguë* si $s'_i = s_{i+1}$ pour tout $i \in I$ tel que $i + 1 \in I$.

Définition 1.2.4 *Un* chemin *dans un système de transitions étiqueté $\mathcal{LTS} = (S, \Sigma, \rightarrow)$ est un couple $\pi = (s, (s_i \xrightarrow{\alpha_i} s'_i)_{i \in I})$ où :*

1. *$s \in S$ est l'origine du chemin π, et*
2. *$(s_i \xrightarrow{\alpha_i} s'_i)_{i \in I}$ est une suite contiguë de transitions, et*
3. *si I est non vide alors $s_m = s$ où m est l'unique élément minimal de I.*

Si $(s_i \xrightarrow{\alpha_i} s'_i)_{i \in I}$ est une suite finie (resp. infinie), alors on dit que π est un *chemin fini* (resp. *chemin infini*). Si π est fini, alors sa *longueur* est la longueur de la suite de transitions $(s_i \xrightarrow{\alpha_i} s'_i)_{i \in I}$.

Un chemin fini ou infini $\pi = (s, (s_i \xrightarrow{\alpha_i} s'_i)_{i \geq 0})$ est plus simplement noté $\pi = s \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} s_2 \cdots s_{i-1} \xrightarrow{\alpha_{i-1}} s_i \cdots$. Un chemin fini $\pi = (s, (s_i \xrightarrow{\alpha_i} s'_i)_{0 \leq i \leq k})$ se note également $\pi = s \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} s_2 \cdots s_{k-1} \xrightarrow{\alpha_{k-1}} s_k$, ou encore $\pi = s \xrightarrow{\sigma} s_k$ avec $\sigma = \alpha_0 \alpha_1 \cdots \alpha_{k-1}$. Ainsi, pour tout $s \in S$, le chemin fini $\pi = (s, \varepsilon)$ de longueur nulle se note également $\pi = s$ ou encore $\pi = s \xrightarrow{\varepsilon} s$. On écrit aussi

$\pi = s \xrightarrow{*} s'$ (resp. $\pi = s \xrightarrow{+} s'$) lorsque π est un chemin fini (resp. de longueur non nulle) de la forme $\pi = s \xrightarrow{\sigma} s'$.

Un *circuit* dans un système de transitions étiqueté $\mathcal{LTS}$ est un chemin fini de longueur non nulle $\pi = s \xrightarrow{+} s'$ tel que $s = s'$, et on dit alors que π est un *circuit sur* s. Un circuit π est *élémentaire* lorsqu'il ne contient pas de sous-circuit strict (i.e. π ne peut pas s'écrire sous la forme $s \xrightarrow{+} s' \xrightarrow{+} s' \xrightarrow{*} s$).

Exemple 1.2.5 Dans la structure de contrôle EC, on a par exemple le chemin suivant : $q_1 \xrightarrow{c \leftarrow c+2} q_2 \xrightarrow{(c \leftarrow c+1) \wedge (d \leftarrow d+3)} q_2 \xrightarrow{c \leftarrow c-7} q_0$. Ce chemin n'est pas un circuit, mais en revanche $q_1 \xrightarrow{c \leftarrow c+2} q_2 \xrightarrow{(c \leftarrow c+1) \wedge (d \leftarrow d+3)} q_2 \xrightarrow{c \leftarrow c-7} q_0 \xrightarrow{c=0} q_1$ est un circuit sur q_1.

Exemple 1.2.6 Lorsque $\mathsf{x}[c] \neq 0$, le seul chemin dans le système de transitions étiqueté $\mathcal{EC}$ d'origine (q_0, x) est le chemin de longueur nulle (q_0, x).

Etant donné une séquence d'exécution $\sigma \in \Sigma^*$, on écrit $s \xrightarrow{\sigma} s'$ (resp. $s \xrightarrow{*} s'$, $s \xrightarrow{+} s'$) lorsqu'il existe un chemin $\pi = s \xrightarrow{\sigma} s'$ (resp. $\pi = s \xrightarrow{*} s'$, $\pi = s \xrightarrow{+} s'$). Observons que la relation binaire $\xrightarrow{*} \subseteq S \times S$ (resp. $\xrightarrow{+} \subseteq S \times S$) ainsi définie est la fermeture réflexive et transitive (resp. fermeture transitive) de $\rightarrow$. Lorsqu'on a $s \xrightarrow{*} s'$ (resp. $s \xrightarrow{\sigma} s'$), alors on dit que s' est *accessible* à partir de s (resp. par la séquence d'exécution σ). La relation binaire $\xrightarrow{*}$ est aussi appelée *relation binaire d'accessibilité*.

1.2.3 Ensemble d'accessibilité

La relation de transition étiquetée capture le comportement du système lorsqu'il passe d'un état à un autre. Le problème central de cette thèse concerne le calcul des états accessibles à partir d'un ensemble d'états initiaux donnés. On associe donc à chaque système de transitions étiqueté $\mathcal{LTS} = (S, \Sigma, \rightarrow)$ quatre fonctions post, pre, post* et pre* de $\mathcal{P}(S)$ dans $\mathcal{P}(S)$ définies par :

$$\begin{array}{rcl} \mathrm{post}(\mathcal{LTS}, X) & = & \{s' \in S \ / \ \exists\, s \in X,\, s \rightarrow s'\} \\ \mathrm{pre}(\mathcal{LTS}, X) & = & \{s \in S \ / \ \exists\, s' \in X, s \rightarrow s'\} \\ \mathrm{post}^*(\mathcal{LTS}, X) & = & \{s' \in S \ / \ \exists\, s \in X,\, s \xrightarrow{*} s'\} \\ \mathrm{pre}^*(\mathcal{LTS}, X) & = & \{s \in S \ / \ \exists\, s' \in X, s \xrightarrow{*} s'\} \end{array}$$

Les quatres fonctions ci-dessus ne prennent pas en compte l'étiquetage des transitions, aussi nous associons à $\mathcal{LTS}$ encore deux fonctions (également appelées) post et pre, de $\mathcal{P}(S) \times \Sigma^*$ dans $\mathcal{P}(S)$, et définies par :

$$\begin{array}{rcl} \mathrm{post}(\mathcal{LTS}, X, \sigma) & = & \{s' \in S \ / \ \exists\, s \in X,\, s \xrightarrow{\sigma} s'\} \\ \mathrm{pre}(\mathcal{LTS}, X, \sigma) & = & \{s \in S \ / \ \exists\, s' \in X, s \xrightarrow{\sigma} s'\} \end{array}$$

On note plus simplement $\mathrm{post}(X, \sigma)$, $\mathrm{pre}(X, \sigma)$, $\mathrm{post}(X)$, $\mathrm{pre}(X)$, $\mathrm{post}^*(X)$ et $\mathrm{pre}^*(X)$ quand le système de transitions étiqueté $\mathcal{LTS}$ sous-jacent se comprend d'après le contexte.

Exemple 1.2.7 Sur notre système de transitions étiqueté d'exemple $\mathcal{EC}$, en partant de l'état initial $s_0 = (q_0, \mathsf{x}_0)$, avec $\mathsf{x}_0[c] = \mathsf{x}_0[d] = 0$, on vérifie qu'on a par exemple :

$$\begin{array}{rcl} \mathrm{pre}(s_0) & = & \{(q_2, \mathsf{x}_0[c := 7])\} \\ \mathrm{post}(s_0, c = 0 \cdot c \leftarrow c + 2) & = & \{(q_2, \mathsf{x}_0[c := 2])\} \end{array}$$

De plus, on vérifie aisément qu'étant donné un vecteur $\mathsf{x} \in \mathbb{N}^{\{c,d\}}$, l'état (q_1, x) est accessible à partir de s_0 (i.e. $(q_1, \mathsf{x}) \in \mathrm{post}^*(s_0)$) si et seulement si $\mathsf{x}[c] = 0$ et $\mathsf{x}[d]$ est un multiple de 15.

Remarque. Etant donné un système de transitions étiqueté $\mathcal{LTS} = (S, \Sigma, \rightarrow)$, les six fonctions ci-dessus vérifient les relations suivantes :

$$\begin{array}{rcllcl} \mathrm{post}(X) & = & \bigcup_{\alpha \in \Sigma} & \mathrm{post}(X, \alpha) & & \\ \mathrm{pre}(X) & = & \bigcup_{\alpha \in \Sigma} & \mathrm{pre}(X, \alpha) & & \\ \mathrm{post}^*(X) & = & \bigcup_{i \in \mathbb{N}} & \mathrm{post}^i(X) & = & \bigcup_{\sigma \in \Sigma^*} \mathrm{post}(X, \sigma) \\ \mathrm{pre}^*(X) & = & \bigcup_{i \in \mathbb{N}} & \mathrm{pre}^i(X) & = & \bigcup_{\sigma \in \Sigma^*} \mathrm{pre}(X, \sigma) \end{array}$$

Remarque. Les systèmes de transitions étiquetés que nous considérons ne sont pas nécessairement à branchement fini. Autrement dit, il peut exister un état $s \in S$ et une étiquette $\alpha \in \Sigma$ tels que $\mathrm{post}(s, \alpha)$ soit infini.

1.2.4 Dual

Nous utiliserons la notion de dual d'un système de transitions étiqueté, afin de nous concentrer sur le calcul de post*, et d'obtenir par dualité des résultats sur le calcul de pre*.

Définition 1.2.8 *Le* dual *d'un système de transitions étiqueté* $\mathcal{LTS} = (S, \Sigma, \rightarrow_{\mathcal{LTS}})$ *est le système de transitions étiqueté* $\overline{\mathcal{LTS}} = (S, \overline{\Sigma}, \rightarrow_{\overline{\mathcal{LTS}}})$ *d'alphabet d'étiquettes* $\overline{\Sigma} = \{\overline{\alpha} \;/\; \alpha \in \Sigma\}$*, et de relation de transitions étiquetée définie par* $s \xrightarrow{\overline{\alpha}}_{\overline{\mathcal{LTS}}} s'$ *si* $s' \xrightarrow{\alpha}_{\mathcal{LTS}} s$*.*

Remarque. Etant donné un système de transitions étiqueté $\mathcal{LTS} = (S, \Sigma, \rightarrow)$, on a $\overline{\overline{\mathcal{LTS}}} = \mathcal{LTS}$. De plus les fonctions post* et pre* vérifient les relations suivantes :

$$\begin{array}{rcl} \mathrm{post}^*(\overline{\mathcal{LTS}}, X) & = & \mathrm{pre}^*(\mathcal{LTS}, X) \\ \mathrm{pre}^*(\overline{\mathcal{LTS}}, X) & = & \mathrm{post}^*(\mathcal{LTS}, X) \end{array}$$

Notons enfin que de nombreuses classes de systèmes de transitions étiquetés sont closes par dualité, comme par exemple les automates à file, les automates à pile, les réseaux de Petri, les automates à compteurs...

1.3 Vérification

Nous précisons dans cette section ce que nous entendons par *vérification* des systèmes de transitions étiquetés. Nous commencerons par rappeler des notions d'effectivité usuelles, puis nous donnerons quelques problèmes de décision classiques du domaine de la vérification. Nous montrerons enfin l'intérêt d'un calcul "symbolique" de post*.

1.3.1 Notions d'effectivité

Une fonction partielle $f : \{0,1\}^* \to \{0,1\}^*$ est *récursive* (ou *calculable*) s'il existe une machine de Turing s'arrêtant pour tout $w \in \mathrm{Dom}(f)$ et répondant $f(w)$. Un langage $L \subseteq \{0,1\}^*$ est *récursif* (ou *décidable*) si sa fonction caractéristique χ_L est récursive, où χ_L est la fonction (totale) $\chi_L : \{0,1\}^* \to \{0,1\}$ définie par $\chi_L(w) = 1$ si $w \in L$ et $\chi_L(w) = 0$ sinon.

Remarque. En pratique, lorsqu'on veut "programmer" une fonction récursive f, on a besoin de connaître précisément une machine de Turing calculant f. Aussi, pour des raisons d'effectivité,

– lorsque nous nous *donnerons* une fonction partielle récursive f, nous *supposerons connue* (implicitement) une machine de Turing calculant f, et

– lorsque nous *montrerons* qu'une fonction partielle f est récursive, nous *donnerons* explicitement (sous la forme d'un algorithme) une machine de Turing calculant f.

Observons que $\{0,1\}^*$ est un ensemble *dénombrable* (i.e. en bijection avec $\mathbb{N}$). Ces notions de récursivité s'étendent naturellement aux ensembles finis ou dénombrables, quitte à *encoder* chaque élément de l'ensemble par un mot de $\{0,1\}^*$. Formellement, un *encodage* d'un ensemble fini ou dénombrable X est une injection enc_X de X dans $\{0,1\}^*$ telle que $enc_X(X)$ est récursif, et on dit alors que (X, enc_X) est un *ensemble encodé.*

Remarque. Un ensemble infini dénombrable X admet une infinité d'encodages, et les notions de récursivité sur X dépendent de l'encodage utilisé. Nous n'utiliserons dans la suite que des encodages "raisonnables", comme par exemple :

– un encodage "raisonnable" de $\{0,1\}^*$ est la fonction identité sur $\{0,1\}^*$.

– un encodage "raisonnable" de $\mathbb{N}$ consiste à encoder chaque entier par sa décomposition binaire.

– un encodage "raisonnable" de l'ensemble des langages récursifs consiste à encoder chaque langage récursif $L \subseteq \{0,1\}^*$ par le code d'une machine de Turing calculant la fonction caractéristique χ_L.

– Si (X, enc_X) et $(X', enc_{X'})$ sont deux ensembles encodés, un encodage "raisonnable" du produit $X \times X'$ est l'encodage $enc_{X\times X'} : X \times X' \to \{0,1\}^*$ défini par $enc_{X\times X'}(x, x') = \varphi(enc_X(x)) \cdot 01 \cdot \varphi(enc_{X'}(x'))$, où la fonction $\varphi : \{0,1\}^* \to \{0,1\}^*$ est le morphisme défini par $\varphi(0) = 00$ et $\varphi(1) = 11$.

– Si (X, enc_X) est un ensemble encodé, un encodage "raisonnable" de l'ensemble des suites finies d'éléments de X est l'encodage enc défini par $enc(x_0, x_1, \ldots, x_k) = \varphi(enc_X(x_0)) \cdot 01 \cdot \varphi(enc_X(x_1)) \cdot 01 \cdots 01 \cdot \varphi(enc_X(x_k))$, où φ est le morphisme défini ci-dessus.

Etant donné deux ensembles encodés (X, enc_X) et $(X', enc_{X'})$, une fonction partielle $f : X \to X'$ est *récursive* si la fonction partielle $g : \{0,1\}^* \to \{0,1\}^*$ définie ci-après est récursive :

$$g(w) = \begin{cases} enc_{X'}(f(enc_X^{-1}(w))) & \text{si } w \in enc_X(X) \text{ et } enc_X^{-1}(w) \in \mathrm{Dom}(f) \\ \text{n'est pas défini} & \text{sinon} \end{cases}$$

Un sous-ensemble $Y \subseteq X$ est *récursif* (ou *décidable*) si le langage $enc_X(Y)$ est récursif (ou de manière équivalente si la fonction caractéristique $\chi_Y : X \to \{0,1\}$ de Y est récursive).

Remarque. On vérifie aisément qu'avec ces définitions, pour toute fonction partielle $f : X \to X'$ telle que $\mathrm{Dom}(f)$ est récursif, f est récursive si et seulement si le sous-ensemble $\{(x, f(x)) \ / \ x \in \mathrm{Dom}(f)\}$ de $X \times X'$ est récursif.

Lorsque X' est infini, l'ensemble $\mathcal{P}(X')$ des parties de X' est infini non dénombrable, et par conséquent il ne peut pas être encodé. Cependant, une fonction partielle $f : X \to \mathcal{P}(X')$ peut se voir de manière équivalente comme le sous-ensemble $E_f = \{(x, x') \ / \ x \in \mathrm{Dom}(f) \text{ et } x' \in f(x)\}$ de $X \times X'$. On dira alors que f est *récursive* si le sous-ensemble E_f est récursif.

Remarque. Observons que si $f : X \to \mathcal{P}(X')$ est récursive alors pour tout $x \in X$, le sous-ensemble $f(x)$ de X' est récursif. En revanche, la réciproque est fausse (i.e. $f : X \to \mathcal{P}(X')$ peut ne pas être récursive, même si pour tout $x \in X$, le sous-ensemble $f(x)$ de X' est récursif).

Remarque. On réalise que la notion de récursivité de post* peut être ambiguë, puisque cette fonction prend a priori en argument un système de transitions étiqueté, mais elle peut aussi être définie pour un système de transitions étiqueté fixé. Lorsque nous parlerons de récursivité de post*, pour une classe $\mathfrak{C}$ de systèmes de transitions étiquetés, nous verrons implicitement post* comme la fonction prenant en argument un système de transitions étiqueté $\mathcal{LTS} \in \mathfrak{C}$ et un état s de $\mathcal{LTS}$.

Notation. Nous ne manipulerons plus désormais d'encodage. Nous dirons simplement qu'un ensemble fini ou dénombrable X est *encodé* lorsqu'il admet un encodage “raisonnable”.

Nous serons amenés dans les chapitres suivants à définir différents algorithmes. Afin de simplifier la présentation de ces algorithmes, nous les présenterons parfois dans un cadre très général (notamment dans le chapitre 2), sans hypothèse d'effectivité. Par exemple, un algorithme pourra utiliser une affectation de la forme $a \leftarrow f(b)$ où :

- a et b sont des variables à valeurs dans des domaines non nécessairement encodés, et où
- f est une fonction non nécessairement récursive.

Nous préciserons alors les hypothèses d'effectivité permettant de *mettre en œuvre* l'algorithme. D'une façon générale, un algorithme `Alg` sera dit *effectif* lorsque les deux conditions suivantes seront satisfaites :

i) toutes les variables utilisées dans `Alg` sont à valeurs dans des domaines de données encodés,

ii) toutes les fonctions utilisées dans `Alg` sont récursives.

Une machine de Turing pourra alors être associée à `Alg`, en remplaçant les données et les fonctions de `Alg` par leur encodage.

1.3.2 Problèmes de décision pour la vérification

Nous rappelons ici quelques problèmes de décision classiques du domaine de la vérification des systèmes de transitions étiquetés. Ces problèmes sont généralement formulés pour une classe $\mathfrak{C}$ *encodée* de systèmes de transitions étiquetés, dont chaque élément $\mathcal{LTS}$ a un ensemble d'états *encodé* ; comme par exemple, les réseaux de Petri, les systèmes d'addition de vecteurs avec états étendus, les automates à files FIFO, les automates à pile, les automates à file réactifs... Nous donnons pour chaque problème l'ensemble le définissant :

1. le problème de l'*accessibilité* est l'ensemble des triplets $(\mathcal{LTS}, s, s')$, où $\mathcal{LTS} \in \mathfrak{C}$ et s, s' sont deux états de $\mathcal{LTS}$, et tels que $s \xrightarrow{*}_{\mathcal{LTS}} s'$.

2. le problème du *caractère borné* est l'ensemble des couples $(\mathcal{LTS}, s_0)$, où $\mathcal{LTS} \in \mathfrak{C}$ et s_0 est un état de $\mathcal{LTS}$, et tels que $\mathrm{post}^*(\mathcal{LTS}, s_0)$ est fini.

3. le problème de la *terminaison* est l'ensemble des couples $(\mathcal{LTS}, s_0)$, où $\mathcal{LTS} \in \mathfrak{C}$ et s_0 est un état de $\mathcal{LTS}$, et tels que tout chemin d'origine s_0 dans $\mathcal{LTS}$ est fini.

4. le problème du *blocage* est l'ensemble des couples $(\mathcal{LTS}, s_0)$, où $\mathcal{LTS} \in \mathfrak{C}$ et s_0 est un état de $\mathcal{LTS}$, et tels qu'il existe $s \in \mathrm{post}^*(\mathcal{LTS}, s_0)$ satisfaisant $\mathrm{post}(\mathcal{LTS}, s) = \emptyset$.

5. le problème du *model-checking*, pour une classe Φ encodée de formules, est l'ensemble des triplets $(\mathcal{LTS}, s_0, \varphi)$, où $\mathcal{LTS} \in \mathfrak{C}$, s_0 est un état de $\mathcal{LTS}$ et $\varphi \in \Phi$, et tels que le couple $(\mathcal{LTS}, s_0)$ satisfait la formule φ.

Remarque. Le problème de l'accessibilité est décidable si et seulement si la fonction post^* est récursive.

Ces problèmes, qui sont tous décidables pour les systèmes de transitions étiquetés finis[1], font partie des problèmes de décision cruciaux pour la vérification des systèmes de transitions étiquetés infinis.

[1]Le problème du model-checking est décidable, dans le cadre des systèmes de transitions étiquetés finis, pour des classes Φ de formules "raisonnables", comme par exemple le μ-calcul propositionnel [Koz83], qui englobe les logiques temporelles classiques LTL, CTL et CTL* [VW86, CES86, Eme90, Dam94].

Une hypothèse naturelle d'effectivité sur la classe $\mathfrak{C}$ consiste à supposer que chaque $\mathcal{LTS} \in \mathfrak{C}$ est à branchement *effectivement* fini, c'est-à-dire que la fonction qui à tout $\mathcal{LTS} \in \mathfrak{C}$ et à tout état s de $\mathcal{LTS}$ associe l'ensemble fini $\mathrm{post}(s)$ est récursive. Observons que cette hypothèse de branchement fini effectif est satisfaite dans les classes les plus courantes de systèmes de transitions étiquetés.

Si la classe $\mathfrak{C}$ vérifie cette hypothèse d'effectivité, alors le problème du caractère borné est "plus difficile" que le problème de la terminaison. En effet, considérons un arbre d'accessibilité réduit de $(\mathcal{LTS}, s_0)$, où un même état n'apparaît jamais deux fois sur une même branche. On voit bien que lorsque cet arbre est fini, il peut être construit puisque $\mathcal{LTS}$ est à branchement effectivement fini, et il permet ainsi de décider le problème de la terminaison. Réciproquement, si cet arbre est infini alors d'après le lemme de KONIG, il y a une branche infinie dans l'arbre et par conséquent $(\mathcal{LTS}, s_0)$ ne termine pas. Or il est clair que cet arbre d'accessibilité réduit de $(\mathcal{LTS}, s_0)$ est fini si et seulement si $\mathrm{post}^*(s_0)$ est fini. On en déduit donc que le problème de la terminaison se réduit pour la décidabilité au problème du caractère borné (i.e. si le problème du caractère borné est décidable alors le problème de la terminaison est décidable).

1.3.3 Pourquoi calculer post^* ?

En pratique, les propriétés les plus courantes qu'on souhaite vérifier sur un système de transitions étiqueté sont les propriétés de *sûreté*, qui expriment qu'un "mauvais" comportement ne peut pas se produire. Une propriété de sûreté φ peut souvent se traduire par un ensemble X de "mauvais" états. Vérifier que φ est satisfaite par un système de transitions étiqueté $\mathcal{LTS}$ muni d'un état initial s_0 revient alors à déterminer si $\mathrm{post}^*(s_0) \cap X = \emptyset$.

Remarque. D'une manière équivalente, on peut aussi déterminer si $\mathrm{pre}^*(X) \cap \{s_0\} = \emptyset$. Nous serons donc également concernés par le calcul de pre^*, mais comme nous l'avons déjà remarqué, calculer pre^* revient à calculer post^* dans le dual $\overline{\mathcal{LTS}}$.

Lorsque X est un singleton $X = \{s\}$, on a $\mathrm{post}^*(s_0) \cap X = \emptyset$ si et seulement si s n'est pas accessible à partir de s_0, et nous sommes donc ramenés au problème de l'accessibilité. Cette méthode s'étend facilement au cas où X est fini. Nous pouvons alors nous contenter d'une hypothèse de récursivité de post^* pour décider si la propriété φ est satisfaite.

Cependant, X peut être infini car il peut y avoir une infinité de "mauvais" états. Dans un tel cas, on ne peut pas directement se ramener au problème de l'accessibilité, et il faut donc trouver un moyen de :

– représenter finiment le sous-ensemble infini d'états X, et de

– calculer une représentation finie de $\mathrm{post}^*(s_0)$, et de

– décider du vide de l'intersection de ces deux représentations finies.

L'objectif principal de cette thèse est la mise en œuvre de cette démarche informelle, afin de pouvoir calculer "efficacement" $\mathrm{post}^*(s_0)$.

Observons qu'une machine de Turing permet de représenter finiment un sous-ensemble infini d'états. De plus, les sous-ensembles d'états récursifs sont effectivement clos par intersection (i.e. la fonction qui à deux sous-ensembles d'états récursifs associe leur intersection est récursive). Malheureusement, le vide d'un sous-ensemble d'états n'est pas décidable.

Nous allons donc chercher des représentations finies de sous-ensembles infinis d'états, mais avec de bonnes propriétés de clôture et d'effectivité. De telles représentations sont souvent appelées *représentations symboliques* et ont été utilisées dans [BW94, CJ98, BGWW97, BH99, ABJ98] pour analyser des automates étendus avec des variables numériques ou avec des files. Nous donnerons dans le prochain chapitre un ensemble "minimal" d'hypothèses devant être satisfaites par ces *abstractions* (ou représentations symboliques) pour permettre le calcul de $\mathrm{post}^*(s_0)$.

Exemple 1.3.1 Rappelons que sur le système de transitions étiqueté $\mathcal{EC}$, en partant de l'état initial $s_0 = (q_0, \mathsf{x}_0)$, un état de la forme (q_1, x) est accessible à partir de s_0 si et seulement si $\mathsf{x}[c] = 0$ et $\mathsf{x}[d]$ est un multiple de 15. En d'autres termes :

$$\mathrm{post}^*(s_0) \cap \left(\{q_1\} \times \mathbb{N}^{\{c,d\}}\right) \; = \; \{q_1\} \times \{\mathsf{x} \in \mathbb{N}^{\{c,d\}} \;/\; \mathsf{x}[c] = 0 \text{ et } \frac{\mathsf{x}[d]}{15} \in \mathbb{N}\}$$

Comme nous le verrons au chapitre 3, ce sous-ensemble infini d'états est semilinéaire, et peut donc être décrit finiment. La classe des sous-ensembles semilinéaires a de bonnes propriétés de clôture et d'effectivité (par exemple, l'inclusion est décidable pour cette classe).

D'un point de vue plus théorique, le calcul effectif d'une représentation symbolique de $\mathrm{post}^*(s_0)$ permet bien souvent — cela dépend en fait des propriétés de clôture et d'effectivité de la représentation symbolique utilisée — de décider de nombreux problèmes relatifs à la vérification.

Nous allons dans le prochain chapitre donner un cadre théorique, pour le calcul efficace de $\mathrm{post}^*(s_0)$, basé sur les notions d'*abstraction* et d'*accélération*. Nous appliquerons ensuite ce schéma d'abstraction et d'accélération à la vérification des systèmes d'addition de vecteurs avec états étendus, des automates à files FIFO et des automates à file réactifs.

Chapitre 2

Systèmes de transitions bien abstraits

Le succès des méthodes formelles d'analyse des systèmes finis [BCM+92] laisse penser que des techniques d'analyse des systèmes infinis pourraient également être développées. Cependant, les systèmes infinis ont en général la puissance des machines de Turing. Par conséquent, une méthode permettant de les analyser est nécessairement incomplète. Parmi les différentes techniques connues d'analyse des systèmes infinis, que nous avons rappelées en introduction, on peut distinguer au moins deux approches :

- le semi-calcul de l'ensemble des états accessibles par des méthodes *symboliques* [BW94, CJ98, BGWW97, BH99, AAB99, HM00b], et
- le calcul d'une approximation supérieure de l'ensemble des états accessibles [CC77, PP91].

Ces deux approches utilisent des "représentations symboliques", c'est-à-dire des descriptions finies d'ensembles infinis, pour manipuler effectivement des sous-ensembles infinis d'états. De plus, des techniques appelées *accélérations* ou *meta-transitions* ont été introduites afin de calculer en une étape un ensemble infini d'états accessibles. Nous proposons dans ce chapitre un schéma d'*abstraction* et d'*accélération* proche de l'interprétation abstraite [CC77], permettant de décrire formellement les hypothèses devant être satisfaites par ces représentations symboliques. Nous montrons ensuite comment appliquer ce schéma d'*abstraction* et d'*accélération* au calcul de l'ensemble des états accessibles d'un système plat et à l'analyse des systèmes structurés. Ces notions seront également appliquées dans les chapitres suivants à l'analyse des systèmes d'addition de vecteurs avec états étendus, des automates à files FIFO et des automates à file réactifs.

Nous commencerons par introduire notre schéma d'abstraction et d'accélération en section 2.1, puis nous l'appliquerons au calcul de l'ensemble des états accessibles d'un système plat en section 2.1, et à l'analyse des systèmes préordonnés en section 2.3. Nous conclurons ce chapitre par une méthode générale d'abstraction et d'accélération des automates étendus dans la section 2.4.

2.1 Abstraction et accélération

Cette section est dédiée à l'abstraction et à l'accélération des systèmes de transitions étiquetés. Nous rappellerons dans un premier temps le cadre général de l'interprétation abstraite, qui permet un calcul *approché symbolique* de post^*. Après une discussion sur les avantages et les inconvénients de cette approche, nous introduirons notre propre schéma d'abstraction, puis d'accélération, des systèmes de transitions étiquetés.

2.1.1 Calcul approché de post^* par interprétation abstraite

Nous décrivons ici brièvement le cadre général de l'analyse approchée des systèmes de transitions (étiquetés) par *interprétation abstraite* [CC77]. Nous n'entrerons pas dans tous les détails de ce mécanisme d'abstraction, le lecteur pouvant les trouver par exemple dans [CC79, CC92a, CC92b].

Notions de treillis et de point fixe

Définissons en premier lieu la notion de treillis, sur laquelle est basée l'interprétation abstraite. Soit $(X, \preceq)$ un ensemble partiellement ordonné et soit Y un sous-ensemble de X. Un *élément inférieur* (resp. *élément supérieur*) de Y est un élément $x \in X$ tel que $x \preceq y$ (resp. $x \succeq y$) pour tout $y \in Y$. On dit qu'un élément inférieur (resp. élément supérieur) $x \in X$ est une *borne inférieure* (resp. une *borne supérieure*) de Y si on a $x \succeq x'$ (resp. $x \preceq x'$) pour tout élément inférieur (resp. élément supérieur) x' de Y. La borne inférieure (resp. borne supérieure) d'un sous-ensemble $Y \subseteq X$ est unique lorsqu'elle existe, et elle est notée $\wedge Y$ (resp. $\vee Y$).

Définition 2.1.1 *Un* treillis complet *est un ensemble partiellement ordonné $(X, \preceq)$ tel que tout sous-ensemble $Y \subseteq X$ admet une borne inférieure et une borne supérieure.*

Un treillis complet $(X, \preceq)$ admet un plus petit élément $\bot = \wedge X = \vee\emptyset$ et un plus grand élément $\top = \vee X = \wedge\emptyset$.

Exemple 2.1.2 Pour tout ensemble X, l'ensemble des parties de X muni de l'inclusion $(\mathcal{P}(X), \subseteq)$ forme un treillis complet, de borne inférieure $\cap$ et de borne supérieure $\cup$.

Une fonction $f : X \to X'$ entre deux treillis complets X et X' est *continue* si $f(\bigvee_{i\in\mathbb{N}} x_i) = \bigvee_{i\in\mathbb{N}} f(x_i)$ pour toute suite infinie croissante $(x_i)_{i\in\mathbb{N}}$ d'éléments de X. Observons qu'une fonction continue est nécessairement croissante, et qu'une fonction croissante sur un treillis complet satisfaisant la condition de chaîne ascendante est nécessairement continue.

Un point fixe d'une fonction $f : X \to X$ est un élément $x \in X$ tel que $f(x) = x$. Un plus petit point fixe d'une fonction $f : X \to X$ est un point fixe x de f tel que $x \preceq y$ pour tout point fixe y de f. Le plus petit point fixe d'une fonction f est unique lorsqu'il existe, et on le note $\mathrm{lfp}(f)$ (pour *Least Fix Point*).

Théorème 2.1.3 (Tarski) *Toute fonction croissante $f : X \to X$ sur un treillis complet $(X, \preceq)$ admet un plus petit point fixe.*

Théorème 2.1.4 (Kleene) *Le plus petit point fixe* $\mathrm{lfp}(f)$ *d'une fonction continue $f : X \to X$ sur un treillis complet $(X, \preceq)$ est égal à $\bigvee_{i\in\mathbb{N}} f^i(\perp)$.*

Une fonction $f : X \to X'$ entre deux treillis complets X et X' *préserve la borne inférieure* (resp. *la borne supérieure*) si elle satisfait $f(\wedge Y) = \wedge f(Y)$ (resp. $f(\vee Y) = \vee f(Y)$) pour tout sous-ensemble $Y \subseteq X$.

Interprétation abstraite

Soit $\mathcal{LTS} = (S, \Sigma, \to)$ un système de transitions étiqueté. Etant donné un ensemble S_0 d'états initiaux, on remarque que $\mathrm{post}^*(S_0)$ est le *plus petit point fixe* de la fonction croissante $f : \mathcal{P}(S) \to \mathcal{P}(S)$ définie par $f(X) = S_0 \cup \mathrm{post}(X)$. Comme f est continue, on obtient que $\mathrm{post}^*(S_0)$ est la limite $\bigcup_{i\in\mathbb{N}} F_i$ de la suite infinie croissante $(F_i)_{i\in\mathbb{N}}$ définie par :

$$\begin{cases} F_0 &= \emptyset \\ F_{i+1} &= F_i \cup f(F_i) \end{cases}$$

On remarque que la suite $(F_i)_{i\in\mathbb{N}}$ atteint sa limite, c'est-à-dire $F_k = \bigcup_{i\in\mathbb{N}} F_i$, dès qu'on a $F_{k+1} = F_k$. Cependant, comme $\mathcal{P}(S)$ est infini lorsque $\mathcal{LTS}$ est infini, la suite $(F_i)_{i\in\mathbb{N}}$ peut être strictement croissante et ainsi ne jamais atteindre sa limite. Cette méthode ne permet donc pas a priori de calculer $\mathrm{post}^*(S_0)$.

Pour résoudre ce problème, le cadre général de l'interprétation abstraite propose de ne calculer qu'une *approximation supérieure* de $\mathrm{post}^*(S_0)$, en se donnant un treillis complet approché $(A, \preceq)$ et en calculant le plus petit point fixe d'une fonction approchée $f'_A : A \to A$ "simulant" f. Idéalement, le treillis complet approché $(A, \preceq)$ satisfera la condition de chaîne ascendante afin de garantir un calcul effectif du plus petit point fixe de f'_A. Dans le cas contraire, un *élargissement* pourra être utilisé afin d'accélérer la convergence.

Décrivons maintenant plus précisément cette approche. Pour mettre en œuvre ce mécanisme d'abstraction, il faut commencer par se donner un treillis complet approché $(A, \preceq)$ et une fonction préservant la borne inférieure $\gamma : A \to \mathcal{P}(S)$ permettant de donner un "sens" aux éléments de A. Intuitivement,

- l'ensemble A correspond à une famille $\gamma(A)$ de sous-ensembles S' (potentiellement infinis) de S, chaque sous ensemble $S' \in \gamma(A)$ admettant une représentation finie donnée par un élément de A. On impose parfois que chaque sous ensemble $S' \in \gamma(A)$ admet une représentation canonique dans A, ce qui revient à supposer que γ est injective. Les éléments de A sont appelés *états abstraits* par opposition aux éléments de S qui sont appelés *états concrets.*

- la fonction de *concrétisation* $\gamma : A \to \mathcal{P}(S)$ donne l'interprétation d'un état abstrait, en lui associant un ensemble (potentiellement infini) d'états concrets. La condition sur γ de préservation de la borne inférieure revient en fait à supposer que la famille $\gamma(A)$ de sous-ensembles de S est *complètement fermée pour l'intersection*, c'est-à-dire $\cap X \in \gamma(A)$ pour tout $X \subseteq \gamma(A)$ (on parle aussi de *famille de Moore*).

– L'ordre partiel $\preceq$ sur A indique la *précision* des états abstraits ; lorsque $a \preceq b$, on dit que b est *moins précis* que a. Notons que γ est croissante, ce qui correspond à l'idée intuitive qu'un état abstrait moins précis dénote un ensemble plus grand d'états concrets. Lorsque γ est injective, on a même $a \preceq b \Leftrightarrow \gamma(a) \subseteq \gamma(b)$.

Comme $(A, \preceq)$ est un treillis complet, une fonction d'*abstraction* croissante $\alpha : \mathcal{P}(S) \to A$ peut être définie par :

$$\alpha(X) = \wedge\{a \in A \ / \ X \subseteq \gamma(a)\}$$

Intuitivement, la fonction d'abstraction associe à un ensemble d'états concrets X l'état abstrait permettant d'approximer supérieurement X le plus précisément possible. On dit de la paire (α, γ) qu'elle forme une *connexion de Galois*.

Afin de calculer une approximation supérieure de $\mathrm{post}^*(S_0)$ dans le treillis complet approché $(A, \preceq)$, on utilise une fonction de "post abstrait" $\mathrm{post}_A : A \to A$ croissante, $\mathrm{post}_A = \alpha \circ \mathrm{post} \circ \gamma$. On définit la fonction $\mathrm{post}_A^* : A \to A$ comme on s'y attend par :

$$\mathrm{post}_A^*(a) = \vee\{\mathrm{post}_A^i(a) \ / \ i \in \mathbb{N}\}$$

On vérifie que pour tout état abstrait $a \in A$, on a $\gamma(\mathrm{post}_A^*(a)) \supseteq \mathrm{post}^*(\gamma(a))$, et ainsi $\mathrm{post}_A^*(a)$ est une approximation supérieure de $\mathrm{post}^*(\gamma(a))$. De plus, étant donné $a_0 = \alpha(S_0)$, l'état abstrait $\mathrm{post}_A^*(a_0)$ peut lui aussi s'exprimer comme plus petit point fixe de la fonction croissante $f_A : A \to A$ définie par $f_A(a) = a_0 \vee \mathrm{post}_A(a)$. En pratique, on ne sait pas nécessairement calculer post_A, et on en calculera donc plutôt une approximation supérieure post'_A, elle aussi croissante. Le plus petit point fixe de la fonction croissante associée $f'_A : A \to A$, définie par $f'_A(a) = a_0 \vee \mathrm{post}'_A(a)$, nous donnera enfin l'approximation supérieure de $\mathrm{post}^*(S_0)$ attendue.

Si le treillis complet approché $(A, \preceq)$ satisfait la condition de chaîne ascendante, alors le plus petit point fixe $\mathrm{lfp}(f'_A)$ de f'_A est calculable par itérations croissantes, comme limite $\bigvee_{i \in \mathbb{N}} a_i$ de la suite infinie croissante ultimement stationnaire $(a_i)_{i \in \mathbb{N}}$ définie par :

$$\left\{ \begin{array}{rcl} a_0 & = & \bot, \text{ où } \bot = \wedge A \\ a_{i+1} & = & a_i \vee f'_A(a_i) \end{array} \right.$$

Dans le cas contraire, on peut calculer une approximation supérieure de $\mathrm{lfp}(f'_A)$ en utilisant un *élargissement* ∇, c'est-à-dire une fonction $\nabla : A \times A \to A$ telle que :

i) pour tout $a, b \in A$, on a $a \vee b \ \preceq \ a \,\nabla\, b$, et

ii) pour toute suite infinie croissante $(a_i)_{i \in \mathbb{N}}$, la suite infinie $(b_i)_{i \in \mathbb{N}}$ définie par $b_0 = a_0$ et $b_{i+1} = b_i \,\nabla\, a_{i+1}$ est non strictement croissante.

Comme indiqué dans [CC92b], la suite infinie croissante avec élargissement $(\hat{a}_i)_{i\in\mathbb{N}}$ définie par :

$$\begin{cases} \hat{a}_0 &= \bot & \text{où } \bot = \wedge A \\ \hat{a}_{i+1} &= \hat{a}_i & \text{si } f'_A(\hat{a}_i) \preceq \hat{a}_i \\ \hat{a}_{i+1} &= \hat{a}_i \,\nabla\, f'_A(\hat{a}_i) & \text{sinon} \end{cases}$$

est ultimement stationnaire et sa limite est une approximation supérieure de $\mathrm{lfp}(f'_A)$.

Avantages et inconvénients de l'approche

L'interprétation abstraite est un mécanisme d'abstraction qui a déjà très largement fait ses preuves et démontré son utilité, tant du point de vue théorique que du point de vue pratique [JN95, LMR+98, Cou00]. Pour notre étude, nous retiendrons que l'avantage principal de cette approche est qu'elle donne un cadre uniforme pour le calcul “symbolique” (ou abstrait) d'une approximation supérieure de post^*. Ce cadre est basé sur les notions très générales de systèmes de transitions (étiquetés) et de treillis complets.

Cependant, notre objectif principal dans cette thèse est le calcul *exact* de post^* pour les systèmes de transitions infinis. De plus, le cadre général d'abstraction dans lequel nous voulons nous placer doit capturer les représentations symboliques par ensembles rationnels ou semilinéaires. Aussi, l'approche ci-dessus ne nous satisfait pas complètement, car :

- elle ne permet dans toute sa généralité qu'un calcul approché de post^*, même si on n'utilise pas d'élargissement. Afin d'obtenir un calcul exact de post^*, il faudrait faire les deux suppositions supplémentaires suivantes :

  1. la fonction de concrétisation γ *préserve la borne supérieure*. Cette condition exprime que la borne supérieure $\vee$ dans le treillis complet approché $(A, \preceq)$ correspond précisément à l'union.
  2. la fonction de “post abstrait” post_A est *exacte*, dans le sens où on a $\gamma(\mathrm{post}_A(a)) \subseteq \mathrm{post}^*(\gamma(a))$ pour tout $a \in A$.

  En effet, lorsque les deux assertions ci-dessus sont satisfaites, $\mathrm{post}^*_A(a)$ est une représentation exacte de $\mathrm{post}^*(\gamma(a))$, et non pas seulement une approximation supérieure, car on a alors :

  $$\begin{aligned} \mathrm{post}^*(\gamma(a)) \subseteq \gamma(\mathrm{post}^*_A(a)) &= \gamma(\vee\{\mathrm{post}^i_A(a) \ / \ i \in \mathbb{N}\}) \\ &= \bigcup_{i\in\mathbb{N}} \{\gamma(\mathrm{post}^i_A(a))\} \\ &\subseteq \mathrm{post}^*(\gamma(a)) \end{aligned}$$

- la fonction d'élargissement permet d'accélérer la convergence, mais *ne tient pas suffisamment compte du système de transitions étiqueté $\mathcal{LTS}$* sous-jacent. En effet, le treillis complet approché n'est défini qu'en fonction de l'ensemble des états S de $\mathcal{LTS}$, et un même treillis complet approché peut servir à différents systèmes de transitions ayant le même ensemble d'états. Par exemple, un même treillis approché de $(\mathcal{P}(\mathbb{N}^k), \subseteq)$ peut servir à deux réseaux de Petri avec k places, l'un sans arc étendu (i.e. dont les arcs sont étiquetés par des entiers), et l'autre étendu par des test-à-zéro, des resets et des transferts[1]. On voit donc bien que la fonction d'élargissement ne peut pas prendre en compte la relation de transition de $\mathcal{LTS}$. Cette dernière est en fait "codée" dans la fonction de "post abstrait" post_A. Une fonction d'élargissement qui accélère, c'est-à-dire différente de la borne supérieure, ne peut donc pas en général être exacte.

Cette constatation montre que l'élargissement *n'est pas la bonne notion d'accélération* pour notre étude, car elle ne capture pas les résultats connus d'accélération *exacte* de circuits dans les automates à files FIFO [BGWW97, BH99] et dans les automates à compteurs [CJ98].

- la condition de préservation de la borne inférieure sur γ implique que la famille $\gamma(A)$ de parties de $\mathcal{P}(S)$ est complètement fermée pour l'intersection. Ceci entraîne que $(\gamma(A), \subseteq)$ est un sous-treillis complet de $(\mathcal{P}(S), \subseteq)$, de borne inférieure $\cap$ et de borne supérieure $\vee Y = \cap\{X \in \gamma(A) \ / \ Y \subseteq X\}$. Cependant, nous utiliserons dans les chapitres suivants des représentations symboliques, pour décrire des contenus de compteurs ou de files, ne satisfaisant pas ces conditions de complétude.

  Considérons par exemple l'ensemble Rat, ordonné par l'inclusion, des sous-ensembles rationnels de Σ^*. Une intersection quelconque de rationnels n'est pas nécessairement rationnelle, donc Rat n'est pas complètement fermé pour l'intersection. Par ailleurs, on sait que Rat est clos par union, donc tout sous-ensemble fini de Rat admet une borne supérieure dans Rat. Cependant, étant donné $a, b \in \Sigma$ tels que $a \neq b$ — on suppose que Σ contient au moins deux éléments — le sous-ensemble infini $\{\{a^i b^i\} \ / \ i \in \mathbb{N}\}$ de Rat n'admet pas de borne supérieure dans Rat.

Le cadre de l'interprétation abstraite étant à la fois *trop général* et *trop particulier* pour notre étude, nous proposons un nouveau schéma d'abstraction et d'accélération, inspiré de l'interprétation abstraite, mais permettant l'analyse *exacte* des systèmes de transitions étiquetés.

2.1.2 Abstraction d'un système de transitions étiqueté

Nous introduisons maintenant la notion d'abstraction d'un système de transitions étiqueté. Notre définition d'abstraction est basée sur un ensemble "minimal" d'hypothèses, qui nous donnera les moyens de concevoir et de comparer des représentations symboliques d'une manière uniforme. D'autre part, cette définition prend en compte l'étiquetage des systèmes de transitions, ce qui permettra de définir des accélérations exactes, car prenant en compte la séquence d'exécution à accélérer. Formellement,

[1] On pourra consulter [Duf98] pour une description et une analyse détaillées des réseaux de Petri étendus.

Définition 2.1.5 *Une* abstraction $\mathcal{A}_{\mathcal{LTS}}$ *d'un système de transitions étiqueté* $\mathcal{LTS} = (S, \Sigma, \rightarrow)$ *est un* 5*-uplet* $\mathcal{A}_{\mathcal{LTS}} = (A, \text{post}_{\mathcal{A}}, \gamma, \sqcup, \Sigma)$ *où* A *est un ensemble d'*états abstraits*,* $\text{post}_{\mathcal{A}} : A \times \Sigma \rightarrow A$*,* $\gamma : A \rightarrow \mathcal{P}(S)$ *et* $\sqcup : A \times A \rightarrow A$ *sont des fonctions satisfaisant pour tout* $a, b \in A$ *et* $\alpha \in \Sigma$ *:*

$$\gamma(a \sqcup b) = \gamma(a) \cup \gamma(b), \text{ et}, \tag{2.1}$$

$$\text{post}(\gamma(a), \alpha) \subseteq \gamma(\text{post}_{\mathcal{A}}(a, \alpha)) \subseteq \text{post}^*(\gamma(a)) \tag{2.2}$$

Expliquons cette définition. Un état abstrait a dénote un ensemble potentiellement infini $\gamma(a)$ d'états concrets. La fonction $\sqcup$ est en fait une opération d'union abstraite, comme exprimé par la condition (2.1). La condition (2.2) permet de s'assurer qu'un calcul exact abstrait de post* peut être effectué, en utilisant la fonction de "post abstrait" $\text{post}_{\mathcal{A}}$. Nous étendons naturellement $\text{post}_{\mathcal{A}}$ aux séquences d'exécutions par :

$$\begin{cases} \text{post}_{\mathcal{A}}(a, \varepsilon) & = & a \\ \text{post}_{\mathcal{A}}(a, \alpha \cdot \sigma) & = & \text{post}_{\mathcal{A}}(\text{post}_{\mathcal{A}}(a, \alpha), \sigma) \end{cases}$$

Observons qu'un système de transitions étiqueté peut avoir plusieurs abstractions. Dans la suite de la thèse, on écrira $\mathcal{A}_{\mathcal{LTS}}$ lorsque $\mathcal{A}_{\mathcal{LTS}}$ est une abstraction d'un système de transitions étiqueté $\mathcal{LTS}$.

Définition 2.1.6 *Une abstraction* $\mathcal{A}_{\mathcal{LTS}} = (A, \text{post}_{\mathcal{A}}, \gamma, \sqcup, \Sigma)$ *est* effective *si d'une part* A *est un ensemble encodé, et d'autre part les fonctions* $\text{post}_{\mathcal{A}}$ *et* $\sqcup$ *sont récursives.*

Remarque. La condition (2.1) permet essentiellement de s'assurer que l'ensemble $\gamma(A)$ est effectivement clos par union finie. Cette hypothèse n'est pas très restrictive, car on peut toujours clore effectivement un ensemble X par union finie, en considérant l'ensemble des parties finies de X.

Exemple 2.1.7 Pour tout système de transitions étiqueté $\mathcal{LTS} = (S, \Sigma, \rightarrow)$, le 5-uplet $(\mathcal{P}(S), \text{post}, Id_{\mathcal{P}(S)}, \cup, \Sigma)$ est clairement une abstraction de $\mathcal{LTS}$. Cependant, cette abstraction triviale *n'est pas effective* lorsque S est infini, car $\mathcal{P}(S)$ est alors un ensemble infini non dénombrable, et il ne peut donc pas être encodé.

Une abstraction comporte un pré-ordre naturel $\sqsubseteq$ induit par la fonction γ. De plus, ce pré-ordre induit lui-même une relation d'équivalence $\cong$. Ils sont définis formellement comme suit :

Définition 2.1.8 *Etant donné une abstraction* $\mathcal{A}_{\mathcal{LTS}} = (A, \text{post}_{\mathcal{A}}, \gamma, \sqcup, \Sigma)$*, on définit un pré-ordre* $\sqsubseteq$ *et une relation d'équivalence* $\cong$ *sur* A *par :*

- $a \sqsubseteq b$ *si* $\gamma(a) \subseteq \gamma(b)$*, et*
- $a \cong b$ *si* $a \sqsubseteq b$ *et* $b \sqsubseteq a$*.*

On remarque que deux états abstraits sont équivalents si et seulement s'ils dénotent le même ensemble d'états concrets ($a \cong b \Leftrightarrow \gamma(a) = \gamma(b)$). La fonction γ se prolonge donc naturellement sur l'ensemble $A_{/\cong}$ des classes d'équivalence de A pour $\cong$. L'intérêt de la relation d'équivalence $\cong$ est que l'opération d'union abstraite $\sqcup$ est commutative et associative modulo cette équivalence :

Proposition 2.1.9 *Etant donné des états abstraits $a, b, c \in A$, on a :*

i) $a \sqcup b \;\cong\; b \sqcup a$, *et*

ii) $(a \sqcup b) \sqcup c \;\cong\; a \sqcup (b \sqcup c)$

Ainsi, pour tout sous-ensemble fini X de A, $\sqcup X$ est bien défini modulo $\cong$ et de plus on a $\gamma(\sqcup X) = \bigcup \gamma(X)$.

Preuve. Ces deux propriétés découlent immédiatement de la condition (2.1). ∎

Remarque. Rappelons que la relation d'équivalence $\cong$ est une congruence pour γ car on a $a \cong b \Rightarrow \gamma(a) = \gamma(b)$. On remarque que $\cong$ est aussi une congruence pour l'opération d'union abstraite $\sqcup$, puisque pour tout $a, b, c, d \in A$ tels que $a \cong b$ et $c \cong d$, on a $a \sqcup b \;\cong\; c \sqcup d$. Par conséquent, l'opération d'union abstraite $\sqcup$ se prolonge aussi naturellement sur $A_{/\cong}$ tout en restant commutative et associative. Observons que :

- l'ensemble quotient $A_{/\cong}$ est naturellement partiellement ordonné par $\sqsubseteq$ et on vérifie que pour tout sous-ensemble fini X de $A_{/\cong}$, $\sqcup X$ est la borne supérieure de X. Cependant, contrairement au cadre habituel d'interprétation abstraite, $A_{/\cong}$ *n'est pas* en général *complet pour la borne supérieure* (i.e. un sous-ensemble quelconque X de $A_{/\cong}$ n'admet pas nécessairement de borne supérieure dans $A_{/\cong}$).

- on serait tenté d'utiliser l'ensemble quotient $A_{/\cong}$ comme ensemble d'états abstraits à la place de A. Cependant, au contraire des fonctions γ et $\sqcup$, la relation d'équivalence $\cong$ n'est pas nécessairement une congruence pour la fonction de "post abstrait" $\mathrm{post}_{\mathcal{A}}$. En effet, si on se trouve dans le cas d'une abstraction par ensembles rationnels, représentés par expressions régulières, on peut très bien imaginer que la fonction de "post abstrait" donne des résultats différents sur deux expressions régulières *syntaxiquement différentes* mais dénotant le même ensemble rationnel.

  Lorsque $\cong$ est une congruence pour la fonction $\mathrm{post}_{\mathcal{A}}$, comme par exemple lorsqu'on a $\mathrm{post}(\gamma(a), \alpha) = \gamma(\mathrm{post}_{\mathcal{A}}(a, \alpha))$ pour tout $a \in A$, alors on peut remplacer A par l'ensemble quotient $A_{/\cong}$ et ainsi obtenir une abstraction plus simple où la fonction γ est injective, et où la fonction $\sqcup$ est la borne supérieure. Cependant, même dans ce cas particulier plus simple, on ne peut pas a priori définir, comme c'était en revanche le cas dans le cadre de l'interprétation abstraite, de fonction abstraite post_A^* sur $A_{/\cong}$, car $A_{/\cong}$ n'est pas en général complet pour la borne supérieure.

Notation. Si $\mathcal{A}_{\mathcal{LTS}} = (A, \mathrm{post}_{\mathcal{A}}, \gamma, \sqcup, \Sigma)$ est une abstraction, alors la fonction également notée $\mathrm{post}_{\mathcal{A}}$ de “post abstrait non étiqueté” $\mathrm{post}_{\mathcal{A}} : A \to A$ est définie de sorte que $\mathrm{post}_{\mathcal{A}}(a) \cong \bigsqcup_{\alpha \in \Sigma} \mathrm{post}_{\mathcal{A}}(a, \alpha)$, par exemple en ordonnant les éléments de Σ et en appliquant l'opération d'union abstraite dans cet ordre. La condition (2.2) entraîne que pour tout $a \in A$, on a :

$$\mathrm{post}(\gamma(a)) \subseteq \gamma(\mathrm{post}_{\mathcal{A}}(a)) \subseteq \mathrm{post}^*(\gamma(a)) \tag{2.3}$$

L'algorithme `EnsembleSymbolique` est une façon naturelle de calculer post* à l'aide uniquement de la fonction de “post abstrait” $\mathrm{post}_{\mathcal{A}}$. Il calcule post* d'une manière itérative, en appliquant l'opération d'union abstraite à chaque étape afin de ne maintenir toujours que deux états abstraits durant le calcul de post*. L'algorithme peut être stoppé à tout moment et donner alors une approximation inférieure de post*.

Algorithme 1 `EnsembleSymbolique`$(\mathcal{A}_{\mathcal{LTS}}, a_0)$

Entrée: une abstraction $\mathcal{A}_{\mathcal{LTS}} = (A, \mathrm{post}_{\mathcal{A}}, \gamma, \sqcup, \Sigma)$ et un état abstrait initial $a_0 \in A$.
1: $a \leftarrow a_0$
2: **répéter**
3: $\quad a' \leftarrow a$
4: $\quad a \leftarrow a' \sqcup \mathrm{post}_{\mathcal{A}}(a')$
5: **jusqu'à ce que** $a \sqsubseteq a'$
6: **retourner** a

Remarque. L'algorithme `EnsembleSymbolique` est implémentable si l'abstraction $\mathcal{A}_{\mathcal{LTS}}$ est effective, et si le pré-ordre $\sqsubseteq$ est décidable.

Proposition 2.1.10 *Soit $\mathcal{A}_{\mathcal{LTS}} = (A, \mathrm{post}_{\mathcal{A}}, \gamma, \sqcup)$ une abstraction et $a_0 \in A$ un état abstrait initial. Si l'algorithme* `EnsembleSymbolique` *termine sur $(\mathcal{A}_{\mathcal{LTS}}, a_0)$ alors on a :*

$$\gamma(\texttt{EnsembleSymbolique}(\mathcal{A}_{\mathcal{LTS}}, a_0)) = \mathrm{post}^*(\gamma(a_0))$$

Preuve. Supposons que l'algorithme `EnsembleSymbolique` termine après n itérations de la boucle **répéter**. Pour tout $i \in [1\,..\,n]$, notons a_i la valeur de la variable a à la fin de la i^{me} itération de la boucle **répéter**. Nous devons montrer que $\gamma(a_n) = \mathrm{post}^*(\gamma(a_0))$. D'après la définition de l'algorithme `EnsembleSymbolique`, on a :

$$\begin{array}{rcll} a_{i+1} & = & a_i \sqcup \mathrm{post}_{\mathcal{A}}(a_i) & \text{pour tout } i \in [0\,..\,n-1] \\ a_n & \sqsubseteq & a_{n-1} & \end{array}$$

On remarque que la suite $(a_i)_{0 \leq i \leq n}$ est croissante (pour le pré-ordre $\sqsubseteq$).

Montrons que $\gamma(a_n) \subseteq \mathrm{post}^*(\gamma(a_0))$. On prouve par récurrence sur i que $\gamma(a_i) \subseteq \mathrm{post}^*(\gamma(a_0))$ pour tout $i \in [0\,..\,n]$. La base est triviale et l'étape d'induction se montre en observant que :

- comme $a_{i+1} = a_i \sqcup \mathrm{post}_{\mathcal{A}}(a_i)$, on obtient d'après la condition (2.1) que $\gamma(a_{i+1}) = \gamma(a_i) \cup \gamma(\mathrm{post}_{\mathcal{A}}(a_i))$.
- d'après l'équation (2.3), on a $\gamma(\mathrm{post}_{\mathcal{A}}(a_i)) \subseteq \mathrm{post}^*(\gamma(a_i))$.

Montrons à présent que $\gamma(a_n) \supseteq \mathrm{post}^*(\gamma(a_0))$. Comme on a $a_n \sqsubseteq a_{n-1}$, il vient $a_{n-1} \sqcup \mathrm{post}_{\mathcal{A}}(a_{n-1}) \sqsubseteq a_{n-1}$. Par conséquent on obtient d'après la condition (2.1) que $\gamma(\mathrm{post}_{\mathcal{A}}(a_{n-1})) \subseteq \gamma(a_{n-1})$. On en déduit avec l'équation (2.3) qu'on a $\mathrm{post}(\gamma(a_{n-1})) \subseteq \gamma(a_{n-1})$. Ainsi, on aboutit à $\mathrm{post}^*(\gamma(a_{n-1})) \subseteq \gamma(a_{n-1})$. Puisque la suite $(a_i)_{0 \leq i \leq n}$ est croissante (pour $\sqsubseteq$), on obtient finalement que $\mathrm{post}^*(\gamma(a_0)) \subseteq \mathrm{post}^*(\gamma(a_{n-1})) \subseteq \gamma(a_{n-1}) \subseteq \gamma(a_n)$. ■

Remarque. L'algorithme `EnsembleSymbolique` ne termine pas si et seulement si la suite infinie $(a_i)_{i\in\mathbb{N}}$ qu'il calcule — voir la preuve de la proposition ci-dessus pour la définition de (a_i) — est strictement croissante. Par conséquent, si $(A, \sqsubseteq)$ satisfait la condition de chaîne ascendante alors on obtient que l'algorithme `EnsembleSymbolique` termine. Dans le cas contraire, on peut toujours choisir d'arrêter la boucle **répéter**, afin de forcer la terminaison de l'algorithme. La valeur correspondante a_k de la variable a, où k est le nombre d'itérations de la boucle **répéter**, satisfait alors :

$$\bigcup_{i=0}^{k} \mathrm{post}^i(\gamma(a_0)) \subseteq \gamma(a_k) \subseteq \mathrm{post}^*(\gamma(a_0))$$

Ainsi, une approximation inférieure de post^* est obtenue.

Une abstraction définit implicitement un système de transitions étiqueté déterministe $(A, \Sigma, \mathrm{post}_{\mathcal{A}})$. Comme conséquence immédiate de nos définitions, on obtient qu'une abstraction simule le système de transitions étiqueté qu'elle abstrait :

Proposition 2.1.11 *Soit $\mathcal{A}_{\mathcal{LTS}} = (A, \mathrm{post}_{\mathcal{A}}, \gamma, \sqcup, \Sigma)$ une abstraction d'un système de transitions étiqueté $\mathcal{LTS} = (S, \Sigma, \rightarrow)$. Pour tout $a_0 \in A$, le système de transitions étiqueté $(A, \Sigma, \mathrm{post}_{\mathcal{A}})$ avec état initial a_0 simule $(\mathcal{LTS}, \gamma(a_0))$.*

Preuve. Soit R la relation sur $S \times A$ définie par $R(s, a)$ si $s \in \gamma(a)$. Nous prouvons que R est une relation de simulation entre $((A, \Sigma, \mathrm{post}_{\mathcal{A}}), a_0)$ et $(\mathcal{LTS}, \gamma(a_0))$. On remarque que $\gamma(a_0) \times \{a_0\} \subseteq R$. Maintenant si $R(s, a)$ et $s \xrightarrow{\alpha} s'$ alors d'après la condition (2.2), on obtient que $R(s', a')$ où $a' = \mathrm{post}_{\mathcal{A}}(a, \alpha)$. ■

2.1.3 Accélération

Nous montrons à présent comment différentes stratégies appelées accélérations ou encore méta-transitions peuvent êtres capturées par nos définitions. Sans accélération, comme c'est le cas dans l'algorithme `EnsembleSymbolique`, l'état abstrait obtenu à chaque étape du calcul de $\mathrm{post}^*(\gamma(a_0))$ peut *ne représenter qu'une portion finie* de $\mathrm{post}^*(\gamma(a_0))$. Dans un tel cas, on ne bénéficie pas de l'avantage d'utiliser une abstraction (sauf, bien sûr, si le but poursuivi est la représentation compacte d'ensembles finis d'états, comme par exemple avec les BDD [Bry86, McM93]). La solution à ce dilemme est basée sur l'observation suivante : si une séquence d'exécution σ peut être itérée infiniment souvent à partir d'un état s en produisant un ensemble infini d'états $\bigcup_{i \in \mathbb{N}} \mathrm{post}(s, \sigma^i)$ qui peut être calculé symboliquement, alors une portion infinie de $\mathrm{post}^*(s)$ peut être capturée en une seule étape.

Exemple 2.1.12 Sur notre système de transitions étiqueté d'exemple $\mathcal{EC}$, en partant de l'état initial $s_0 = (q_2, \mathsf{x}_0)$, avec $\mathsf{x}_0[c] = \mathsf{x}_0[d] = 0$, l'itération du circuit de contrôle $q_2 \xrightarrow{(c \leftarrow c+1) \wedge (d \leftarrow d+3)} q_2$ produit le sous-ensemble infini d'états $\{q_2\} \times \{\mathsf{x}_0, \mathsf{x}_1, \ldots, \mathsf{x}_k, \ldots\}$, où $\mathsf{x}_k[c] = k$ et $\mathsf{x}_k[d] = 3k$ pour tout $k \in \mathbb{N}$. Comme nous le verrons au chapitre 3, ce sous-ensemble infini d'états est semilinéaire, et peut donc être décrit symboliquement.

Le but recherché est le même que dans le cas de l'*élargissement* en interprétation abstraite [CC77]. Cependant, un calcul abstrait avec élargissement produit une approximation supérieure de post^* ; il ne peut pas garantir un calcul exact de post^*, car il ne tient pas suffisamment compte du système de transitions étiqueté sous-jacent. Notre définition d'accélération tient compte de la séquence d'exécution σ à itérer, et permet donc un calcul exact abstrait de post^* lorsqu'elle est correcte.

Définition 2.1.13 *Une fonction $\nabla : A \times \Sigma^+ \to A$ est une* accélération *pour une abstraction $\mathcal{A}_{\mathcal{LTS}} = (A, \mathrm{post}_{\mathcal{A}}, \gamma, \sqcup, \Sigma)$ si elle vérifie la condition suivante pour tout $a \in A$ et $\sigma \in \Sigma^+$:*

$$\nabla(a, \sigma) \sqsupseteq a \tag{2.4}$$

Une accélération ∇ est correcte *si elle vérifie la condition suivante pour tout $a \in A$ et $\sigma \in \Sigma^+$:*

$$\gamma(\nabla(a, \sigma)) \subseteq \mathrm{post}^*(\gamma(a)) \tag{2.5}$$

Une accélération ∇ est complète *si elle vérifie la condition suivante pour tout $a \in A$ et $\sigma \in \Sigma^+$:*

$$\gamma(\nabla(a, \sigma)) \supseteq \bigcup_{i \in \mathbb{N}} \mathrm{post}(\gamma(a), \sigma^i) \tag{2.6}$$

Les trois conditions de la définition ci-dessus méritent quelques explications. La condition (2.4) permet de s'assurer qu'un état abstrait croît par application d'une accélération. Une accélération est correcte si les états qu'elle produit restent dans post^*, comme exprimé par la condition (2.5). De façon similaire, la condition (2.6) exprime qu'une accélération est complète si elle capture tous les états accessibles par itérations arbitraires de σ.

Etant donné une abstraction et une accélération, nous proposons de développer d'une façon uniforme un arbre symbolique d'accessibilité. Les différentes étapes de l'algorithme `ArbreSymbolique` se comprennent bien. L'accélération est appliquée autant que possible, en utilisant la séquence d'exécution étiquetant la branche en cours.

Notation. Nous définirons dans la suite de la thèse plusieurs algorithmes construisant des arbres symboliques d'accessibilité. Les notations suivantes seront utilisées tout au long du document :

- un *noeud* $\mathtt{t}$ étiqueté par une étiquette e sera noté $\mathtt{t}:e$,
- un *arc* étiqueté par α entre deux noeuds $\mathtt{t}:e$ et $\mathtt{t}':e'$ sera noté avec une double flèche $\mathtt{t}:e \stackrel{\alpha}{\Longrightarrow} \mathtt{t}':e'$.

Algorithme 2 $\mathtt{ArbreSymbolique}(\mathcal{A}_{\mathcal{LTS}}, a_0, \nabla)$

Entrée: une abstraction $\mathcal{A}_{\mathcal{LTS}} = (A, \mathrm{post}_{\mathcal{A}}, \gamma, \sqcup, \Sigma)$, un état abstrait initial $a_0 \in A$ et une accélération ∇ pour $\mathcal{A}_{\mathcal{LTS}}$.

créer la racine $\mathtt{r}$ étiquetée par a_0
tant que il reste des noeuds non marqués **faire**
  choisir un noeud non marqué $\mathtt{t}:a$
  si $a \sqsubseteq \bigsqcup\{b \;/\; \mathtt{t}':b \text{ est un noeud marqué}\}$ **alors**
    ne rien faire {le noeud $\mathtt{t}$ est dit *couvert*}
  sinon
    $b \leftarrow a$
    pour chaque ancêtre $\mathtt{t}'$ de $\mathtt{t}$ **faire**
      $b \leftarrow \nabla(b, \sigma)$ où $\mathtt{t}' \xrightarrow{\sigma} \mathtt{t}$
    pour chaque étiquette $\alpha \in \Sigma$ **faire**
      $a' \leftarrow \mathrm{post}_{\mathcal{A}}(b, \alpha)$
      construire un fils $\mathtt{t}':a'$ de $\mathtt{t}$ et étiqueter l'arc $\mathtt{t} \stackrel{\alpha}{\Longrightarrow} \mathtt{t}'$
  marquer $\mathtt{t}$

Remarque. L'algorithme `ArbreSymbolique` est implémentable si l'abstraction $\mathcal{A}_{\mathcal{LTS}}$ est effective, si le pré-ordre $\sqsubseteq$ est décidable et si l'accélération ∇ est récursive.

Nous montrons maintenant que l'arbre construit par l'algorithme `Arbre Symbolique` capture post*, comme on s'y attendait !

Proposition 2.1.14 *Soit $\mathcal{A}_{\mathcal{LTS}} = (A, post_{\mathcal{A}}, \gamma, \sqcup)$ une abstraction, $a_0 \in A$ un état abstrait initial et ∇ une accélération pour $\mathcal{A}_{\mathcal{LTS}}$. Si T désigne l'arbre potentiellement infini construit par l'algorithme* `ArbreSymbolique` *appliqué à $(\mathcal{A}_{\mathcal{LTS}}, a_0, \nabla)$, alors on a :*

$$\mathrm{post}^*(\gamma(a_0)) \subseteq \bigcup_{\mathtt{t}:a \in T} \gamma(a)$$

Si de plus ∇ est une accélération correcte, alors on a :

$$\mathrm{post}^*(\gamma(a_0)) = \bigcup_{\mathtt{t}:a \in T} \gamma(a)$$

Preuve. Pour la première assertion, on montre par récurrence sur i qu'on a $\mathrm{post}^i(\gamma(a_0)) \subseteq \bigcup_{\mathtt{t}:a\in T} \gamma(a)$ pour tout $i \in \mathbb{N}$. La base est triviale car la racine $\mathtt{r}$ de T est étiquetée par a_0. Pour montrer l'étape d'induction, supposons que l'hypothèse de récurrence est satisfaite pour un certain $i \in \mathbb{N}$. Soit $s' \in \mathrm{post}^{i+1}(\gamma(a_0))$.

Il existe $s \in \mathrm{post}^i(\gamma(a_0))$ et $\alpha \in \Sigma$ tels que $s \xrightarrow{\alpha} s'$. D'après l'hypothèse de récurrence, on sait qu'il existe un noeud $\mathtt{t}\!:\!a$ dans T tel que $s \in \gamma(a)$. Soit $\mathtt{t}\!:\!a$ le premier noeud lors de la construction de T tel que $s \in \gamma(a)$. Observons que $\mathtt{t}\!:\!a$ n'est pas couvert. Par conséquent, on obtient d'après les lignes 7–12 de l'algorithme **`ArbreSymbolique`** qu'il existe $b \in A$ et un fils $\mathtt{t}'\!:\!a'$ de $\mathtt{t}$ tel que $a \sqsubseteq b$ et $a' = \mathrm{post}_{\mathcal{A}}(b, \alpha)$. D'après la condition (2.2), il vient $\mathrm{post}(\gamma(b), \alpha) \subseteq \gamma(a')$. Comme $a \sqsubseteq b$, on aboutit à $\mathrm{post}(\gamma(a), \alpha) \subseteq \gamma(a')$. Rappelons que $s \xrightarrow{\alpha} s'$ et que $s \in \gamma(a)$. Ainsi, on en déduit que $s' \in \gamma(a')$ et par conséquent $s' \in \bigcup_{\mathtt{t}:a \in T} \gamma(a)$. Finalement, il vient $\mathrm{post}^{i+1}(\gamma(a_0)) \subseteq \bigcup_{\mathtt{t}:a \in T} \gamma(a)$, ce qui conclut la preuve de la première assertion.

Supposons que ∇ est une accélération correcte. La seconde assertion se montre en prouvant par récurrence sur la profondeur d'un noeud dans T qu'on a $\gamma(a) \subseteq \mathrm{post}^*(\gamma(a_0))$ pour tout noeud $\mathtt{t} : a \in T$. La base est triviale car la racine $\mathtt{r}$ de T est étiquetée par a_0. Pour montrer l'étape d'induction, supposons que l'hypothèse de récurrence est satisfaite pour un certain $i \in \mathbb{N}$. Soit $\mathtt{t}'\!:\!a'$ un noeud de profondeur $i+1$ dans T et soit $\mathtt{t} : a$ le père de $\mathtt{t}'$. D'après l'hypothèse de récurrence, on sait que $\gamma(a) \subseteq \mathrm{post}^*(\gamma(a_0))$. Examinons le traitement de $\mathtt{t}\!:\!a$. Comme ∇ est correcte, on déduit des lignes 7–9 de l'algorithme **`ArbreSymbolique`** que la valeur $b_{\mathtt{t}}$ de la variable b lors du traitement de $\mathtt{t}$ satisfait $\gamma(b_{\mathtt{t}}) \subseteq \mathrm{post}^*(\gamma(a))$. Or il existe $\alpha \in \Sigma$ tel que $a' = \mathrm{post}_{\mathcal{A}}(b_{\mathtt{t}}, \alpha)$. On aboutit en utilisant la condition (2.2) à $\gamma(a') \subseteq \mathrm{post}^*(\gamma(b_{\mathtt{t}}))$. Par conséquent, on obtient que $\gamma(a') \subseteq \mathrm{post}^*(\gamma(a)) \subseteq \mathrm{post}^*(\gamma(a_0))$, ce qui conclut la preuve de la seconde assertion. ∎

Remarque. Lorsque $(A, \sqsubseteq)$ satisfait la condition de chaîne ascendante, l'algorithme **`ArbreSymbolique`** termine quelle que soit l'accélération choisie. On peut donc alors choisir de ne pas accélérer, en définissant ∇ par $\nabla(a, \sigma) = a$. Mais dans ce cas l'algorithme **`ArbreSymbolique`** est moins performant que l'algorithme **`EnsembleSymbolique`**, car la construction de l'arbre symbolique et la conservation en mémoire de ses noeuds est alors inutile.

Accélération générique

Nous montrons maintenant comment définir une accélération générique correcte (et “presque complète”), lorsque l'abstraction satisfait les conditions *i*) et *ii*) ci-dessous. La démarche d'accélération que présentons ici permet de capturer les notions de “limite” utilisées dans [Fin90, EN98] pour la construction d'un arbre de couverture, dans le cadre plus restreint des systèmes de transitions bien structurés.

Supposons que $\mathcal{A}_{\mathcal{LTS}} = (A, \mathrm{post}_{\mathcal{A}}, \gamma, \sqcup, \Sigma)$ est une abstraction satisfaisant les deux conditions suivantes :

i) la fonction de “post abstrait” $\mathrm{post}_{\mathcal{A}}$ est croissante (pour le pré-ordre $\sqsubseteq$).

ii) pour toute suite infinie $(a_i)_{i \in \mathbb{N}}$ d'états abstraits croissante (pour le pré-ordre $\sqsubseteq$), il existe un état asbtrait $b \in A$ tel que $\gamma(b) = \bigcup_{i \in \mathbb{N}} \gamma(a_i)$.

Remarque. La fonction de "post abstrait" est également supposée croissante dans [Fin90, EN98]. La condition "de complétude" *ii*) que nous imposons sur l'ensemble pré-ordonné $(A, \sqsubseteq)$ est identique à celle utilisée dans [EN98] pour construire un arbre de couverture. Aucune condition n'est imposée dans [Fin90], mais l'ensemble des états asbtraits est complété en y ajoutant les limites de suites croissantes, et la fonction de "post abstrait" est prolongée sur cette complétion.

Comme la fonction de "post abstrait" $\mathrm{post}_{\mathcal{A}}$ est croissante, on en déduit que la relation d'équivalence $\cong$ associée à $\mathcal{A}_{\mathcal{LTS}}$ est ici une congruence pour $\mathrm{post}_{\mathcal{A}}$. Comme indiqué dans la sous-section 2.1.2, nous pouvons donc remplacer l'ensemble des états abstraits A par l'ensemble quotient $A' = A_{/\cong}$ et ainsi définir une nouvelle abstraction plus simple $\mathcal{A}'_{\mathcal{LTS}} = (A', \mathrm{post}_{\mathcal{A}'}, \gamma', \sqcup', \Sigma)$. Notons que le pré-ordre $\sqsubseteq'$ associé à $\mathcal{A}'_{\mathcal{LTS}}$ est en réalité l'ordre partiel induit par $\sqsubseteq$ sur l'ensemble quotient A'. La condition "de complétude" *ii*) implique que toute suite infinie croissante $(a'_i)_{i\in\mathbb{N}}$ d'éléments de A' admet une borne supérieure $\bigvee_{i\in\mathbb{N}} a'_i$ satisfaisant $\gamma'(\bigvee_{i\in\mathbb{N}} a'_i) = \bigcup_{i\in\mathbb{N}} \gamma'(a'_i)$. Nous allons en fait plutôt définir une accélération pour l'abstraction $\mathcal{A}'_{\mathcal{LTS}}$.

Remarque. Le passage à l'ensemble quotient pour définir une accélération est également effectué dans [Fin90, EN98].

On rappelle que la fonction $\mathrm{post}_{\mathcal{A}'}$ est croissante (pour l'ordre partiel $\sqsubseteq'$). Ainsi lorsqu'on a $\mathrm{post}_{\mathcal{A}'}(a', \sigma) \sqsupseteq' a'$, alors la suite infinie $(a'_i)_{i\in\mathbb{N}}$ d'éléments de A' définie par $a'_i = \mathrm{post}_{\mathcal{A}'}(a', \sigma^i)$ est croissante. Nous pouvons donc définir la fonction $\nabla' : A' \times \Sigma^+ \to A'$ par :

$$\nabla'(a', \sigma) = \begin{cases} \bigvee_{i\in\mathbb{N}} \mathrm{post}_{\mathcal{A}'}(a', \sigma^i) & \text{si } \mathrm{post}_{\mathcal{A}'}(a', \sigma) \sqsupseteq' a' \\ a' & \text{sinon} \end{cases}$$

On vérifie que ∇' est une accélération correcte pour l'abstraction $\mathcal{A}'_{\mathcal{LTS}}$. On remarque que ∇' n'est pas nécessairement complète, cependant la condition (2.6) de complétude de l'accélération est vérifiée lorsque $\mathrm{post}_{\mathcal{A}'}(a', \sigma) \sqsupseteq' a'$.

Observons enfin que l'algorithme `ArbreSymbolique` appliqué à $(\mathcal{A}'_{\mathcal{LTS}}, a'_0, \nabla')$ construit un arbre symbolique "de couverture" *très similaire* aux arbres de couverture construits dans [Fin90, EN98] pour les systèmes de transitions bien structurés.

2.2 Calcul de post* pour les systèmes plats

La platitude est une notion que l'on rencontre dans de nombreux travaux récents sur l'analyse des systèmes de transitions étiquetés infinis [BGWW97, FO97a, ABJ98, BH99, Jur99, CC00]. Dans cette section, nous décrirons tout d'abord les expressions régulières semilinéaires puis les langages plats, en vue d'introduire les systèmes de transitions étiquetés plats, et d'en faire un bref état de l'art. Nous montrerons ensuite comment utiliser notre schéma d'abstraction et d'accélération afin de calculer post^* pour cette classe particulière de systèmes de transitions étiquetés.

2.2.1 Expressions régulières semilinéaires

Soit Σ un alphabet. Nous utiliserons les sous-classes suivantes d'expressions régulières :

Définition 2.2.1 *Les expressions régulières* linéaires *et* semilinéaires *sont définies par :*

- *Une* expression régulière linéaire *(écrit* LRE*) est une expression régulière de la forme* $x_0 {y_0}^* x_1 {y_1}^* \cdots x_{k-1} {y_{k-1}}^* x_k$ *avec* $x_i \in \Sigma^*$ *et* $y_i \in \Sigma^+$.
- *Une* expression régulière semilinéaire *(écrit* SLRE*) est une somme finie de LRE (éventuellement la somme vide* $\emptyset$*).*

On note $LRE(\Sigma)$ (resp. $SLRE(\Sigma)$) l'ensemble des expressions régulières linéaires (resp. semilinéaires) sur Σ. Le langage $[\![\rho]\!]$ associé à un SLRE ρ est dit *SLRE-définissable.*

Nous nous intéresserons par la suite à la complexité de certaines opérations sur les SLRE, aussi nous associons une taille aux SLRE, définie ci-dessous. Cette taille nous permettra également de raisonner par récurrence sur les SLRE.

Définition 2.2.2 *La* taille *des LRE et des SLRE est définie comme suit :*

- *la taille* $|\rho|$ *d'un LRE* $\rho = x_0 {y_0}^* x_1 {y_1}^* \cdots x_{k-1} {y_{k-1}}^* x_k$ *est donnée par* $|\rho| = |x_k| + \Sigma_{i=0}^{k-1}(|x_i| + |y_i| + 1)$, *et*
- *la taille* $|\rho|$ *d'un SLRE* $\rho = \rho_0 + \rho_1 \cdots + \rho_k$ *est donnée par* $|\rho| = \Sigma_{i=0}^{k}|\rho_i|$.

Remarque. Nous avons choisi pour des raisons de simplicité de ne pas fermer les SLRE par concaténation. Par exemple, l'expression régulière $(a+b)\cdot(a+b)$ n'est pas semilinéaire. Comme nous le verrons en proposition 2.2.4, ce choix ne diminue pas la puissance d'expression de SLRE, car les langages SLRE-définissables sont clos par concaténation. Par exemple, le langage $[\![(a+b)\cdot(a+b)]\!] = [\![aa+ab+ba+bb]\!]$. En revanche, comme on le voit sur l'exemple, la représentation par un SLRE est bien moins compacte (d'un facteur potentiellement exponentiel !) que la représentation par une expression régulière quelconque.

Rappelons que la *fonction de densité* $\eta_L : \mathbb{N} \to \mathbb{N}$ d'un langage $L \subseteq \Sigma^*$ compte le nombre de mots de longueur $n \in \mathbb{N}$ dans L. Certaines propriétés des langages rationnels de densité polynomiale (i.e. de fonction de densité dans $O(n^k)$, $k \in \mathbb{N}$), ainsi que leurs liens avec les SLRE, sont décrites dans [Yu97, pages 80-82]. Il y est notamment indiqué que la classe des langages SLRE-définissables coïncide avec la classe des langages rationnels de densité polynomiale, mais est strictement incluse dans la classe des langages rationnels de hauteur d'étoile un.

Remarque. Lorsque l'alphabet Σ est réduit à un singleton, tout langage est de densité polynomiale, et par conséquent la classe des langages SLRE-définissables coïncide alors avec la classe des langages rationnels.

Exemple 2.2.3 Le langage $L_1 = \{(ab)^i \ / \ \exists \ k \in \mathbb{N}, i = 3 + 5k\}$ est SLRE-définissable, puisque $L_1 = [\![(ab)^3 \cdot ((ab)^5)^*]\!]$. Le langage $L_2 = \{a^i b^i \ / \ i \in \mathbb{N}\}$ est de densité polynomiale, mais il n'est pas rationnel et par conséquent, il n'est pas SLRE-définissable.

Les propositions suivantes expriment plusieurs propriétés de clôture des langages SLRE-définissables, dont nous nous servirons dans la suite de la thèse.

Proposition 2.2.4 *La classe des langages SLRE-définissables est close par morphisme, par union, par concaténation et par intersection avec des rationnels.*

Preuve. Les deux premières propriétés de clôture sont triviales. La clôture par concaténation se montre en remarquant d'une part que les LRE sont syntaxiquement clos par concaténation, et d'autre part que la concaténation se distribue par rapport à l'addition. Prouvons en détails la dernière propriété de clôture. Observons qu'il suffit de montrer que pour tout LRE ρ et pour langage rationnel $L \subseteq \Sigma^*$, le langage $[\![\rho]\!] \cap L$ est SLRE-définissable.

On prouve par récurrence sur la taille de ρ que pour tout LRE ρ et pour toute expression régulière r, il existe un SLRE ρ' tel que $[\![\rho]\!] \cap [\![r]\!] = [\![\rho']\!]$. La base est triviale car l'unique LRE de taille nulle est ε, et tout langage inclus dans $\{\varepsilon\}$ est SLRE-définissable. Pour montrer l'étape d'induction, supposons que pour un certain $k \in \mathbb{N}$, l'hypothèse de récurrence est satisfaite pour tous les LRE ρ de taille $|\rho| \leq k$. Soit ρ un LRE de taille $k+1$ et r une expression régulière. Deux cas se présentent :

- si $\rho = x \cdot \rho''$ avec $x \in \Sigma^+$, alors $[\![\rho]\!] \cap [\![r]\!] = x \cdot ([\![\rho'']\!] \cap x^{-1}[\![r]\!])$. Observons que $|\rho''| = |\rho| - |x| \leq k$. Comme $[\![r]\!]$ est un langage rationnel, $x^{-1}[\![r]\!]$ est rationnel et on obtient d'après l'hypothèse de récurrence qu'il existe un ρ' tel que $[\![\rho'']\!] \cap x^{-1}[\![r]\!] = [\![\rho']\!]$. Par conséquent $[\![\rho]\!] \cap [\![r]\!] = [\![x \cdot \rho'']\!]$.

- si $\rho = y^* \cdot \rho''$ avec $y \in \Sigma^+$, alors on a :

$$[\![y^* \cdot \rho'']\!] \cap [\![r]\!] = \bigcup_{i \in \mathbb{N}} y^i \cdot ([\![\rho'']\!] \cap (y^i)^{-1}[\![r]\!])$$

  Observons que $|\rho''| = |\rho| - |y| - 1 \leq k$. Comme $[\![r]\!]$ est un langage rationnel, $(y^i)^{-1}[\![r]\!]$ est rationnel pour tout $i \in \mathbb{N}$, et de plus il existe $h, k \in \mathbb{N}$ avec $h > 0$ tels que $(y^{k+h})^{-1}[\![r]\!] = (y^k)^{-1}[\![r]\!]$. D'après l'hypothèse de récurrence, on obtient que pour tout $i \in \mathbb{N}$, il existe un SLRE ρ_i tel que $[\![\rho'']\!] \cap (y^i)^{-1}[\![r]\!] = [\![\rho_i]\!]$. Ainsi, on en déduit que $[\![\rho_{i+jh}]\!] = [\![\rho_i]\!]$ pour tout

$i, j \in \mathbb{N}$ tels que $i \geq k$. Il vient donc :

$$\begin{aligned}
[\![\rho]\!] \cap [\![r]\!] &= \bigcup_{i \in \mathbb{N}} y^i \cdot [\![\rho_i]\!] \\
&= \bigcup_{i=0}^{k-1} y^i \cdot [\![\rho_i]\!] \quad \bigcup_{i=k}^{k+h-1} \quad \bigcup_{j \in \mathbb{N}} y^{i+jh} \cdot [\![\rho_{i+jh}]\!] \\
&= \bigcup_{i=0}^{k-1} y^i \cdot [\![\rho_i]\!] \quad \bigcup_{i=k}^{k+h-1} \; y^i (y^h)^* \cdot [\![\rho_i]\!] \\
&= \left[\!\!\left[\sum_{i=0}^{k-1} y^i \cdot \rho_i \quad + \quad \sum_{i=k}^{k+h-1} \; y^i (y^h)^* \cdot \rho_i \right]\!\!\right]
\end{aligned}$$

Ce qui conclut la preuve de la dernière propriété de clôture. ■

Observons que la preuve ci-dessus est constructive. Une preuve bien plus simple mais non constructive consisterait à remarquer que l'intersection d'un langage rationnel avec un langage SLRE-définissable est un langage rationnel de densité polynomiale, et par conséquent SLRE-définissable.

Remarque. La classe des langages SLRE-définissable n'est pas close par morphisme inverse. En effet, si $\varphi : \{a, b\}^* \to \{a, b\}^*$ est le morphisme défini par $\varphi(a) = a$ et $\varphi(b) = a$, alors l'image inverse par φ du langage SLRE-définissable a^* est le langage $\{a, b\}^*$ qui n'est pas SLRE-définissable.

La proposition suivante montre, en particulier, que la classe des langages SLRE-définissables est close par facteur gauche, mais qu'elle n'est pas close par complémentaire.

Proposition 2.2.5 *Pour tout langage $L \subseteq \Sigma^*$, les deux assertions suivantes sont satisfaites :*

i) L est SLRE-définissable si et seulement si L est rationnel et $\mathrm{FG}(L)$ est SLRE-définissable.

ii) si $|\Sigma| \geq 2$ et L est SLRE-définissable, alors $\Sigma^* \setminus L$ n'est pas SLRE-définissable.

Preuve. L'assertion *ii)* se montre en observant que si $|\Sigma| \geq 2$ et si L est SLRE-définissable alors $\Sigma^* \setminus L$ est de densité exponentielle. Le sens $\Leftarrow$ de l'assertion *i)* se montre en remarquant que si $\mathrm{FG}(L)$ est SLRE-définissable, alors L est de densité polynomiale. Afin de montrer le sens $\Rightarrow$ de l'assertion *i)*, on prouve par récurrence sur la taille de ρ que pour tout SLRE ρ, il existe un SLRE ρ' tel que $[\![\rho']\!] = \mathrm{FG}([\![\rho]\!])$. La plupart des cas se vérifient aisément, aussi nous nous contentons de détailler le dernier cas, lorsque ρ est un LRE de la forme $\rho = y^* \cdot \rho''$ avec $y = y_0 y_1 \cdots y_k \in \Sigma^+$. Supposons que l'hypothèse de récurrence est satisfaite pour ρ''. Il existe donc un SLRE ρ' tel que $[\![\rho']\!] = \mathrm{FG}([\![\rho'']\!])$. Ainsi, on obtient que :

$$\begin{aligned}
\mathrm{FG}([\![\rho]\!]) &= \mathrm{FG}(y^*) \quad \bigcup \quad y^* \cdot \mathrm{FG}([\![\rho'']\!]) \\
&= \bigcup_{i=0}^{k} y^* \cdot y_0 y_1 \cdots y_i \quad \bigcup \quad y^* \cdot [\![\rho']\!] \\
&= \left[\!\!\left[\sum_{i=0}^{k} y^* \cdot y_0 y_1 \cdots y_i \quad + \quad y^* \cdot \rho' \right]\!\!\right]
\end{aligned}$$

Ce qui conclut la preuve de la proposition. ■

Remarque. Un langage $L \subseteq \Sigma^*$ tel que $\mathrm{FG}(L)$ est SLRE-définissable n'est pas nécessairement rationnel, et par conséquent L n'est pas nécessairement SLRE-définissable. Il suffit de choisir par exemple $L = \{a^{2^i} \ / \ i \in \mathbb{N}\}$.

Remarquons enfin que la classe des langages SLRE-définissables n'est pas close par sous-mot. En effet, le langage $(ab)^*$ est SLRE-définissable, mais sa clôture inférieure pour l'ordre partiel sous-mot $\downarrow((ab)^*, |) = \{a, b\}^*$ n'est pas SLRE-définissable.

2.2.2 Langages plats

Nous introduisons maintenant les langages *plats* qui sont fortement liés aux SLRE, et qui nous permettrons de définir ensuite les systèmes de transitions étiquetés plats.

Définition 2.2.6 *Un langage L est* plat *s'il est inclus dans un langage SLRE-définissable.*

Observons que la classe des langages plats contient la classe des langages SLRE-définissables.

Remarque. En fait, la classe des langages plats est la clôture inférieure pour l'inclusion de la classe des langages SLRE-définissables. Ainsi, un langage plat est nécessairement de densité polynomiale.

Comme le montre l'exemple suivant, la classe des langages plats est incomparable avec la classe des langages rationnels.

Exemple 2.2.7 Le langage $L_2 = \{a^i b^i \ / \ i \in \mathbb{N}\}$ est plat, car $L_2 \subseteq [\![a^* b^*]\!]$. En revanche, le langage $L_3 = [\![(a + bc)^*]\!]$ n'est pas plat car il est de densité exponentielle.

Les propositions suivantes expriment plusieurs propriétés de clôture des langages plats, qui sont pour la plupart des conséquences des propriétés de clôture des langages SLRE-définissables.

Proposition 2.2.8 *La classe des langages plats est close par morphisme, par union, par concaténation et par intersection avec des langages quelconques.*

Preuve. Les trois premières propriétés de clôture proviennent de la clôture par morphisme, par union et par concaténation des langages SLRE-définissables. La dernière propriété de clôture se déduit du fait que la classe des langages plats est close inférieurement pour l'inclusion. ∎

Remarque. Tout comme la classe des langages SLRE-définissables, la classe des langages plats n'est pas close par morphisme inverse, et elle n'est pas close par sous-mot non plus.

Proposition 2.2.9 *Pour tout langage $L \subseteq \Sigma^*$, les deux assertions suivantes sont satisfaites :*

i) *L est plat si et seulement si* $\mathrm{FG}(L)$ *est plat.*

ii) *si $|\Sigma| \geq 2$ et L est plat, alors $\Sigma^* \setminus L$ n'est pas plat.*

iii) *L est plat et rationnel si et seulement si L est SLRE-définissable.*

iv) *L^* est plat si et seulement si il existe $w \in \Sigma^*$ tel que $L \subseteq w^*$.*

Preuve. L'assertion *ii)* se montre en observant que si $|\Sigma| \geq 2$ et si L est plat alors $\Sigma^* \setminus L$ est de densité exponentielle. Le sens $\Leftarrow$ de l'assertion *i)* vient de la clôture par facteur gauche des langages SLRE-définissables. Le sens $\Rightarrow$ de l'assertion *i)* se déduit du fait que la classe des langages plats est close inférieurement pour l'inclusion. Le sens $\Leftarrow$ de l'assertion *iii)* est trivial. Le sens $\Rightarrow$ de l'assertion *iii)* se montre en observant que si L est rationnel et inclus dans un langage SLRE-définissable L', alors on a $L = L \cap L'$ et par conséquent L est SLRE-définissable, puisque la classe des langages SLRE-définissables est close par intersection avec un rationnel.

Montrons à présent l'assertion *iv)*. Le sens $\Leftarrow$ de cette assertion est trivial. Prouvons donc le sens $\Rightarrow$ et supposons que L^* est plat. Soit P l'ensemble des mots primitifs $z \in \Sigma^+$ tels qu'il existe $w \in L$ satisfaisant $w \in z^+$. Montrons par l'absurde que l'ensemble P est vide ou réduit à un singleton, ce qui achèvera la preuve de l'assertion *iv)*. Supposons donc que P contient deux mots distincts z_1 et z_2. Observons tout d'abord que $z_1^\omega \neq z_2^\omega$, car z_1 et z_2 sont primitifs et distincts — ce résultat est une conséquence immédiate du *théorème du défaut* (voir par exemple [Per83]). D'après la définition de P, il existe $w_1, w_2 \in L$ tels que $w_1 \in z_1^+$ et $w_2 \in z_2^+$. On remarque que $w_1^\omega \neq w_2^\omega$, car $w_1^\omega = z_1^\omega$ et $w_2^\omega = z_2^\omega$. Par conséquent, on en déduit que $\{w_1, w_2\}$ est un *code* (voir par exemple [Per83]). Mais puisque $\{w_1, w_2\}$ est un code, le langage $L^* \supseteq \{w_1, w_2\}^*$ est de densité exponentielle, ce qui contredit l'hypothèse que L^* est plat. ∎

Remarque. On pourrait croire que la classe des langages plats coïncide avec la classe des langages de densité polynomiale, mais il n'en est rien (sauf bien sûr lorsque l'alphabet Σ est réduit à un singleton). En effet, le langage $L_4 =$

$\{w \cdot a^{2^{|w|}} \ / \ w \in \{a,b\}^*\}$ est de densité polynomiale, puisque le nombre de mots dans L_4 de longueur $n2^n$ est égal à 2^n. Cependant, $\mathrm{FG}(L_4) = \{a,b\}^*$ est de densité exponentielle, et par conséquent $\mathrm{FG}(L_4)$ n'est pas plat. On en déduit d'après l'assertion i) de la proposition précédente que L_4 n'est pas plat.

L'assertion iv) de la proposition 2.2.9 permet de déduire un algorithme pour décider la platitude d'un langage rationnel. Cet algorithme fonctionne par induction sur la structure d'une expression régulière r en observant que :

- si $r \in \{\emptyset, \varepsilon\}$, alors le langage rationnel $[\![r]\!]$ est plat.
- si $r = r_1 + r_2$, alors le langage rationnel $[\![r]\!]$ est plat si et seulement si les langages rationnels $[\![r_1]\!]$ et $[\![r_2]\!]$ sont tous deux plats.
- si $r = r_1 \cdot r_2$, alors le langage rationnel $[\![r]\!]$ est plat si et seulement si on se trouve dans l'une des deux situtations suivantes :
  - on a $[\![r]\!] = \emptyset$, ce qui est décidable, ou bien
  - les langages rationnels $[\![r_1]\!]$ et $[\![r_2]\!]$ sont tous deux plats et non vides.
- si $r = (r')^*$ alors le langage rationnel $[\![r]\!]$ est plat si et seulement s'il existe $w \in \Sigma^*$ tel que $[\![r']\!] \subseteq w^*$. De plus, si $[\![r']\!] \subseteq w^*$ alors la longueur de w est nécessairement plus petite que la longueur du plus petit mot non vide de $[\![r']\!]$. Par conséquent, la condition d'existence d'un mot $w \in \Sigma^*$ tel que $[\![r']\!] \subseteq w^*$ est décidable.

Nous venons donc de démontrer la proposition suivante.

Proposition 2.2.10 *La platitude d'un langage rationnel est décidable.*

Et comme un langage rationnel $L \subseteq \Sigma^*$ est plat si et seulement si L est SLRE-définissable, on en déduit que :

Corollaire 2.2.11 *La SLRE-définissabilité d'un langage rationnel est décidable.*

Rappelons qu'un langage L est *borné* s'il existe des mots $w_0, w_1, \ldots, w_k$ dans Σ^* tels que $L \subseteq [\![w_0^* w_1^* \cdots w_k^*]\!]$ [Gin66, HRS76]. La complexité des problèmes d'égalité et d'inclusion entre langages bornés a été étudiée dans [HRS76].

Remarque. Un langage est plat si et seulement si il est borné.

Nous utiliserons dans la suite la notion de langage ρ-plat, car on a généralement besoin de connaître un SLRE ρ tel que $L = [\![\rho]\!]$.

Définition 2.2.12 *Un language L est ρ-*plat *si $L \subseteq [\![\rho]\!]$, où ρ est un SLRE.*

2.2.3 Systèmes de transitions étiquetés plats

La notion de *platitude* d'un système de transitions étiqueté $\mathcal{LTS}$ s'exprime comme une condition sur l'ensemble des traces de $\mathcal{LTS}$. Rappelons que :

Définition 2.2.13 *L'ensemble* $Traces(\mathcal{LTS}, X)$ *des* traces *d'un système de transitions étiqueté* $\mathcal{LTS} = (S, \Sigma, \to)$ *depuis un ensemble* $X \subseteq S$ *d'états initiaux est défini par :*

$$Traces(\mathcal{LTS}, X) = \bigcup_{s \in X} \{\sigma \in \Sigma^* \ / \ \exists\ s' \in S, s \xrightarrow{\sigma} s'\}$$

Remarque. Une autre notion de platitude est définie dans [CJ98, CJ99, CC00], dans le cadre des *automates étendus à compteurs*, comme une contrainte structurelle sur l'automate, permettant de s'assurer que pour tout état de contrôle q, au plus un circuit élémentaire sur q dans la structure de contrôle peut modifier la valeur des compteurs. Notre définition est plus générale, car d'une part elle s'applique aux systèmes de transitions étiquetés, et d'autre part elle englobe plus d'automates étendus, comme par exemple l'automate étendu dont la structure de contrôle est donnée dans la figure 2.1 de la page 55, et qui a deux circuits de contrôle élémentaires sur q_0.

Les notions de langages plats, d'automates plats et formules plates apparaissent dans différents papiers sur les automates à file, les automates plats à compteurs, les automates temporisés, les logiques temporelles plates et l'utilisation des langages plats pour le calcul de la relation binaire d'accessibilité dans les réseaux de Petri. Survolons rapidement ces résultats :

– Les automates à files *monogènes* et *linéaires* sont tels que la projection des traces sur les émissions dans chaque canal est un langage plat particulier, plus précisément un langage ρ-plat où :

  – *monogène* : ρ est une somme finie de LRE de la forme x ou xy^*, avec $x \in \Sigma^*$ et $y \in \Sigma^+$.
  – *linéaire* : ρ est un LRE de la forme $a^*b^*c^* \cdots z^*$ où $a, b, c \ldots, z \in \Sigma$ sont des symboles.

  Le problème de l'accessibilité est décidable pour les automates à files monogènes [MF85] et pour les automates à files linéaires [GGLR87, CF87, FR88].

– Tout automate temporisé "à la Alur & Dill" [AD94] peut être traduit en un *automate à compteurs plat*, où la notion de platitude exprime ici que pour chaque état de contrôle q, il existe au plus un circuit élémentaire sur q dans la structure de contôle pouvant modifier la valeur des compteurs [CJ99]. Une extension de LTL permettant d'exprimer des propriétés sur les compteurs a été définie et étudiée dans [CC00]. Les formules de cette extension sont dites *plates* car sur la gauche d'un "until", seules des contraintes atomiques ou des formules propositionnelles de LTL sont autorisées. Les modèles de cette nouvelle logique sont reconnus par automates plats à compteurs.

- Les langages de la forme $w_0^* w_1^* \cdots w_k^*$ sont utilisés dans [FO97a, FO97b] — et ils y sont nommés plats ! — afin de calculer la relation binaire d'accessibilité dans les réseaux de Petri (avec arcs inhibiteurs). Les auteurs décomposent la relation binaire d'accessibilité sous une forme SLRE de type $s \xrightarrow{w_0^* w_1^* \cdots w_k^*} s'$. Lorsque le processus de décomposition termine, la relation binaire d'accessibilité ainsi calculée est semilinéraire, et il est démontré dans [FO97a] que le processus de décomposition termine pour les réseaux BPP.
- De nombreux papiers, portant sur l'analyse par des méthodes symboliques d'automates étendus avec des variables numériques ou avec des files, cherchent à calculer symboliquement en une étape l'effet de l'itération d'un circuit [BW94, CJ98, BGWW97, BH99, ABJ98]. Ces travaux sont souvent présentés dans un cadre général, sans restriction de platitude et sans garantie de terminaison. Comme nous allons nous en rendre compte dans la prochaine sous-section, la classe de systèmes de transitions étiquetés sur laquelle ces méthodes terminent naturellement est la classe des systèmes de transitions étiquetés *plats*.

Définition 2.2.14 *Un système de transitions étiqueté $\mathcal{LTS} = (S, \Sigma, \rightarrow)$ muni d'un ensemble $X \subseteq S$ d'états initiaux est ρ-*plat *si $Traces(LTS, X)$ est ρ-plat, où $\rho \in SLRE(\Sigma)$.*

Remarquons qu'un ensemble de traces est nécessairement clos par facteur gauche. Ainsi, il peut exister un SLRE ρ tel que $Traces(\mathcal{LTS}, X) \subseteq \mathrm{FG}([\![\rho]\!])$ alors que $Traces(\mathcal{LTS}, X)$ n'est pas ρ-plat. Cependant, puisque les langages SLRE-définissables sont clos par facteur gauche, il existerait alors nécessairement un autre SLRE ρ' tel que $[\![\rho']\!] = \mathrm{FG}([\![\rho]\!])$ et par conséquent $Traces(\mathcal{LTS}, X)$ serait ρ'-plat.

2.2.4 Calcul de post*

Soit $\mathcal{LTS}$ un système de transitions étiqueté muni d'un état initial s_0. Observons que même si $(\mathcal{LTS}, s_0)$ est ρ-plat, son ensemble de traces n'est pas nécessairement SLRE-définissable. Imaginons par exemple qu'on ait $Traces(\mathcal{LTS}, s_0) = \{\alpha^i \beta^j \ / \ j \leq i\}$. Comme $\{\alpha^i \beta^j \ / \ j \leq i\}$ est inclus dans $[\![\alpha^* \beta^*]\!]$, on obtient que $\mathcal{LTS}$ est $(\alpha^* \beta^*)$-plat. Cependant, $Traces(\mathcal{LTS}, s_0)$ n'est clairement pas rationnel, et par conséquent il n'est pas SLRE-définissable. Ce n'est pas dérangeant, car on peut tout de même utiliser la platitude de $\mathcal{LTS}$ pour calculer $\mathrm{post}^*(s_0)$ en itérant d'abord α puis β. Cette idée intuitive est mise en œuvre dans l'algorithme `EnsSymbPlat`.

Remarque. D'après sa définition récursive, l'algorithme `EnsembleSymbolique Plat` termine toujours. Cet algorithme est implémentable si l'abstraction $\mathcal{A}_{\mathcal{LTS}}$ est effective.

Algorithme 3 `EnsSymbPlat`$(\mathcal{A}_{\mathcal{LTS}}, a, \nabla, \rho)$

Entrée: une abstraction $\mathcal{A}_{\mathcal{LTS}} = (A, \mathrm{post}_{\mathcal{A}}, \gamma, \sqcup, \Sigma)$, un état abstrait $a \in A$, une accélération ∇ pour $\mathcal{A}_{\mathcal{LTS}}$ et une expression régulière $\rho \in SLRE(\Sigma)$, $\rho \neq \emptyset$.

1: **si** $\rho = \varepsilon$ **alors**
2:    **retourner** a
3: **sinon si** $\rho = \rho_1 + \rho_2$ **alors**
4:    **retourner** `EnsSymbPlat`$(\mathcal{A}_{\mathcal{LTS}}, a, \nabla, \rho_1) \sqcup$ `EnsSymbPlat`$(\mathcal{A}_{\mathcal{LTS}}, a, \nabla, \rho_2)$
5: **sinon si** $\rho = \alpha \cdot \rho'$ with $\alpha \in \Sigma$ **alors**
6:    **retourner** `EnsSymbPlat`$(\mathcal{A}_{\mathcal{LTS}}, \mathrm{post}_{\mathcal{A}}(a, \alpha), \nabla, \rho')$
7: **sinon si** $\rho = y^* \cdot \rho'$ with $y \in \Sigma^+$ **alors**
8:    **retourner** `EnsSymbPlat`$(\mathcal{A}_{\mathcal{LTS}}, \nabla(a, y), \nabla, \rho')$

La proposition suivante entraîne que lorsque ∇ est une accélération correcte et complète, l'algorithme `EnsSymbPlat` calcule exactement l'ensemble des états accessibles par une séquence d'exécution dans $[\![\rho]\!]$.

Proposition 2.2.15 *Soit $\mathcal{A}_{\mathcal{LTS}} = (A, \mathrm{post}_{\mathcal{A}}, \gamma, \sqcup)$ une abstraction, $a \in A$ un état abstrait et ∇ une accélération pour $\mathcal{A}_{\mathcal{LTS}}$. Pour tout $\rho \in SLRE(\Sigma)$, $\rho \neq \emptyset$, les deux assertions suivantes sont vraies :*

i) si ∇ est une accélération correcte, alors on a :

$$\gamma(\texttt{EnsSymbPlat}(\mathcal{A}_{\mathcal{LTS}}, a, \nabla, \rho)) \subseteq \mathrm{post}^*(\gamma(a))$$

ii) si ∇ est une accélération complète, alors on a :

$$\bigcup_{\sigma \in [\![\rho]\!]} \mathrm{post}(\gamma(a), \sigma) \subseteq \gamma(\texttt{EnsSymbPlat}(\mathcal{A}_{\mathcal{LTS}}, a, \nabla, \rho))$$

Preuve. Les deux assertions se prouvent par récurrence sur la taille du SLRE ρ. La plupart des cas se vérifient aisément, aussi nous nous contentons de détailler le dernier cas, lorsque $\rho = y^* \cdot \rho'$ avec $y \in \Sigma^+$. Supposons que les deux assertions sont satisfaites pour ρ' et montrons qu'elles le sont aussi pour ρ. D'après les lignes 7–8 de l'algorithme `EnsSymbPlat`, on a :

$$\gamma(\texttt{EnsSymbPlat}(\mathcal{A}_{\mathcal{LTS}}, a, \nabla, \rho)) = \texttt{EnsSymbPlat}(\mathcal{A}_{\mathcal{LTS}}, \nabla(a, y), \nabla, \rho'))$$

Puisque $i)$ est satisfaite pour ρ', il vient $\gamma(\texttt{EnsSymbPlat}(\mathcal{A}_{\mathcal{LTS}}, a, \nabla, \rho)) \subseteq \mathrm{post}^*(\gamma(\nabla(a, y)))$. Ainsi, si ∇ est correcte, on obtient que $\gamma(\nabla(a, y)) \subseteq \mathrm{post}^*(\gamma(a))$ et par conséquent on aboutit à :

$$\gamma(\texttt{EnsSymbPlat}(\mathcal{A}_{\mathcal{LTS}}, a, \nabla, \rho)) \subseteq \mathrm{post}^*(\gamma(a))$$

ce qui montre que $i)$ est satisfaite pour ρ.

Montrons à présent que $ii)$ est satisfaite pour ρ, et supposons que ∇ est complète. Puisque $\gamma(\nabla(a, y)) \supseteq \bigcup_{i \in \mathbb{N}} \mathrm{post}(\gamma(a), y^i)$, on obtient que pour tout $\sigma \in \Sigma^*$,

$$\bigcup_{i \in \mathbb{N}} \mathrm{post}(\gamma(a), y^i \cdot \sigma) \subseteq \mathrm{post}(\gamma(\nabla(a, y), \sigma))$$

Comme on a $[\![\rho]\!] = \{y^i \ / \ i \in \mathbb{N}\} \cdot [\![\rho']\!]$, il vient

$$\bigcup_{\sigma \in [\![\rho]\!]} \mathrm{post}(\gamma(a), \sigma) \quad = \bigcup_{\substack{\sigma \in [\![\rho']\!] \\ i \in \mathbb{N}}} \mathrm{post}(\gamma(a), y^i \cdot \sigma) \quad \subseteq \bigcup_{\sigma \in [\![\rho']\!]} \mathrm{post}(\gamma(\nabla(a, y), \sigma))$$

Rappelons que $ii)$ est satisfaite pour ρ', et on en déduit finalement que :

$$\begin{aligned} \bigcup_{\sigma \in [\![\rho]\!]} \mathrm{post}(\gamma(a), \sigma) &\subseteq \gamma(\mathtt{EnsSymbPlat}(\mathcal{A}_{\mathcal{LTS}}, \nabla(a, y), \nabla, \rho')) \\ &\subseteq \gamma(\mathtt{EnsSymbPlat}(\mathcal{A}_{\mathcal{LTS}}, a, \nabla, \rho)) \end{aligned}$$

ce qui montre que $ii)$ est satisfaite pour ρ. ∎

Dans le cas particulier des systèmes de transitions étiquetés plats, l'algorithme `EnsSymbPlat` calcule une approximation supérieure (resp. une représentation exact) de post^*, pourvu que l'accélération soit complète (resp. correcte et complète).

Théorème 2.2.16 *Soit $\mathcal{A}_{\mathcal{LTS}} = (A, \mathrm{post}_{\mathcal{A}}, \gamma, \sqcup)$ une abstraction et ∇ une accélération complète pour $\mathcal{A}_{\mathcal{LTS}}$. Pour tout état initial abstrait $a_0 \in A$, si $(\mathcal{LTS}, \gamma(a_0))$ est ρ-plat, alors on a :*

$$\mathrm{post}^*(\gamma(a_0)) \subseteq \gamma(\mathtt{EnsSymbPlat}(\mathcal{A}_{\mathcal{LTS}}, a_0, \nabla, \rho))$$

Si de plus ∇ est correcte, alors on a :

$$\mathrm{post}^*(\gamma(a_0)) = \gamma(\mathtt{EnsSymbPlat}(\mathcal{A}_{\mathcal{LTS}}, a, \nabla, \rho))$$

Preuve. Puisque $(\mathcal{LTS}, \gamma(a_0))$ est ρ-plat, il vient $Traces(\mathcal{LTS}, \gamma(a_0)) \subseteq [\![\rho]\!]$. Par conséquent, on obtient que $\mathrm{post}^*(\gamma(a_0)) = \bigcup_{\sigma \in [\![\rho]\!]} \mathrm{post}(\gamma(a_0), \sigma)$. On conclut avec la proposition 2.2.15. ∎

Une autre application de l'algorithme `EnsSymbPlat` consiste, étant donné un système de transitions étiqueté $\mathcal{LTS}$ et une formule plate φ (i.e. une formule représentant un langage SLRE-définissable de séquences d'exécution), à calculer l'ensemble des états de $\mathcal{LTS}$ accessibles par une séquence d'exécution satisfaisant φ.

2.3 Analyse des systèmes de transitions étiquetés préordonnés

Nous montrons dans cette section comment nos définitions d'abstraction et d'accélération englobent les travaux réalisés sur les *systèmes de transitions bien structurés* [Fin90, AČJ00, EN98, FS00a].

Notre présentation mettra en évidence le rôle de chaque hypothèse de la définition des systèmes de transitions bien structurés :

– la condition habituelle de *beau pré-ordre* sur l'ensemble des états permet d'obtenir des résultats de terminaison des semi-algorithmes, alors que

– la condition habituelle de *compatibilité* permet d'obtenir des résultats de *correction*.

Commençons par rappeler la définition des systèmes de transitions pré-ordonnés :

Définition 2.3.1 ([Fin90]) *Un* système de transitions étiqueté pré-ordonné *est un couple* $(\mathcal{LTS}, \preceq)$ *où* $\mathcal{LTS} = (S, \Sigma, \to)$ *est un système de transitions étiqueté, et* $\preceq$ *est un pré-ordre sur* S.

La structure de pré-ordre $\preceq$ sur l'ensemble des états conduit à la notion bien connue de *couverture*, où l'on cherche à calculer une approximation supérieure (pour l'inclusion) de post* en considérant sa clôture inférieure (pour le pré-ordre $\preceq$ sur l'ensemble des états).

La notion d'ensemble de couverture introduite dans [Fin90] était donnée en termes de limites de suites croissantes d'états accessibles. Nous en proposons une nouvelle définition bien plus simple mais aussi plus générale, car permettant de représenter l'ensemble de couverture autrement que par des limites de suites croissantes d'états.

Définition 2.3.2 *L'*ensemble de couverture *d'un système de transitions étiqueté pré-ordonné* $(\mathcal{LTS}, \preceq)$ *muni d'un ensemble* X *d'états initiaux, est l'ensemble* $\downarrow\mathrm{post}^*(X)$.

Afin de calculer la clôture inférieure et la clôture supérieure de $\mathrm{post}^*(X)$, on associe à $(\mathcal{LTS}, \preceq)$ deux systèmes de transitions $\downarrow(\mathcal{LTS}, \preceq)$ et $\uparrow(\mathcal{LTS}, \preceq)$ définis comme suit :

Définition 2.3.3 *La* clôture inférieure *(resp.* clôture supérieure*) d'un système de transitions étiqueté pré-ordonné* $(\mathcal{LTS}, \preceq)$, *avec* $\mathcal{LTS} = (S, \Sigma, \to)$, *est le système de transitions étiqueté, noté* $\downarrow(\mathcal{LTS}, \preceq)$ *(resp.* $\uparrow(\mathcal{LTS}, \preceq)$*),*

– *d'ensemble d'états* S*, et*

– *d'alphabet d'étiquettes* $\Sigma \cup \{\downarrow\}$ *(resp.* $\Sigma \cup \{\uparrow\}$*), et*

– *de relation de transition étiquetée égale à* $\to \cup \{s \xrightarrow{\downarrow} s' \ / \ s' \preceq s\}$ *(resp.* $\to \cup \{s \xrightarrow{\uparrow} s' \ / \ s' \succeq s\}$*).*

On note plus simplement $\downarrow\mathcal{LTS}$ et $\uparrow\mathcal{LTS}$ lorsque le pré-ordre $\preceq$ se comprend d'après le contexte.

Remarque. Comme on le verra dans les chapitres suivants, la clôture inférieure d'un système de transitions étiqueté pré-ordonné $(\mathcal{LTS}, \preceq)$ représente parfois une variante "avec perte" de $\mathcal{LTS}$, comme par exemple dans le cas des SAVE étendus avec perte [BM99], ou des automates à files avec perte [ABJ98].

Remarque. Un système de transitions étiqueté pré-ordonné $(\mathcal{LTS}, \preceq)$ satisfait les relations de dualité suivantes :

$$\begin{array}{rcl} \overline{\downarrow(\mathcal{LTS}, \preceq)} & = & \uparrow(\overline{\mathcal{LTS}}, \preceq) \\ \overline{\uparrow(\mathcal{LTS}, \preceq)} & = & \downarrow(\overline{\mathcal{LTS}}, \preceq) \end{array}$$

2.3.1 Abstraction et accélération de $\downarrow\mathcal{LTS}$ et de $\uparrow\mathcal{LTS}$

Nous décrivons maintenant comment définir une abstraction de $\downarrow\mathcal{LTS}$ (resp. $\uparrow\mathcal{LTS}$) et une accélération pour cette abstraction. Nous ne détaillons que le cas de la clôture inférieure de $\mathcal{LTS}$, le cas de la clôture supérieure de $\mathcal{LTS}$ s'obtenant par symétrie (ou par dualité).

Abstraction

Considérons un système de transitions étiqueté pré-ordonné $(\mathcal{LTS}, \preceq)$, avec $\mathcal{LTS} = (S, \Sigma, \rightarrow)$. Afin de définir une abstraction de $\downarrow\mathcal{LTS}$, nous avons besoin de la définition suivante.

Définition 2.3.4 ([AČJ00, FS00a]) *On dit que* $(\mathcal{LTS}, \preceq)$ *admet une* base inférieure post finie *s'il existe une fonction* $\mathrm{post}_\downarrow : \mathcal{P}_f(S) \times \Sigma \rightarrow \mathcal{P}_f(S)$ *telle que* $\mathrm{post}_\downarrow(X, \alpha)$ *est une base finie de* $\downarrow \mathrm{post}(\downarrow X, \alpha)$*, pour tout sous-ensemble fini* $X \subseteq S$ *et* $\alpha \in \Sigma$.

Lorsque $(\mathcal{LTS}, \preceq)$ satisfait la condition de base inférieure post finie, on peut définir une abstraction de $\downarrow\mathcal{LTS}$, par ensembles d'états clos inférieurement de base finie. Elle est formellement définie comme suit :

Définition 2.3.5 *Soit* $\mathcal{A}^b_{\downarrow\mathcal{LTS}} = (A, \mathrm{post}_\mathcal{A}, \gamma, \sqcup, \Sigma \cup \{\downarrow\})$ *le quintuplet où l'ensemble* A*, et les fonctions* $\mathrm{post}_\mathcal{A} : A \times \Sigma \rightarrow A$*,* $\gamma : A \rightarrow \mathcal{P}(S)$ *et* $\sqcup : A \times A \rightarrow A$ *sont définis par :*

$$\begin{array}{rcl} A & = & \mathcal{P}_f(S) \\ \mathrm{post}_\mathcal{A}(a, \downarrow) & = & a \\ \mathrm{post}_\mathcal{A}(a, \alpha) & = & \mathrm{post}_\downarrow(a, \alpha) \\ \gamma(a) & = & \downarrow a \\ a \sqcup b & = & a \cup b \end{array}$$

Proposition 2.3.6 $\mathcal{A}^b_{\downarrow\mathcal{LTS}}$ *est une abstraction de* $\downarrow\mathcal{LTS}$.

Remarque. Lorsque l'ensemble des états S de $\mathcal{LTS}$ est encodé, et si la fonction $\mathrm{post}_\downarrow$ est récursive, alors l'abstraction $\mathcal{A}^b_{\downarrow\mathcal{LTS}}$ est effective.

Remarque. L'abstraction $\mathcal{A}^b_{\downarrow\mathcal{LTS}}$ est basée sur les sous-ensembles d'états clos inférieurement et de base finie. Cependant il peut arriver que l'ensemble clos inférieurement des états accessibles de $\downarrow\mathcal{LTS}$ n'admette pas de base finie, auquel cas cette abstraction n'est pas suffisante. On cherchera alors une autre abstraction $\mathcal{A}_{\downarrow\mathcal{LTS}}$ de $\downarrow\mathcal{LTS}$, capturant si possible tous les ensembles d'états clos inférieurement, afin d'être certain de pouvoir représenter post*. Par exemple, les *contraintes closes inférieurement* de [BM99], ou les *expression régulières simples* de [ABJ98], permettent de représenter finiment n'importe quel ensemble d'états clos inférieurement d'un SAVE étendu, ou d'un automate à files FIFO avec pertes respectivement.

La notion de *base supérieure post finie* est définie symétriquement, et lorsque $(\mathcal{LTS}, \preceq)$ admet une base supérieure post finie, une abstraction $\mathcal{A}^b_{\uparrow\mathcal{LTS}}$ de $\uparrow\mathcal{LTS}$ est définie symétriquement. La remarque suivante exprime quelques propriétés supplémentaires satisfaites par cette abstraction lorsque $\preceq$ est un beau pré-ordre.

Remarque. Lorsque $\preceq$ est un beau pré-ordre sur S, tout sous-ensemble de S clos supérieurement admet une base finie, et par conséquent $(\mathcal{LTS}, \preceq)$ admet nécessairement une base supérieure post finie (mais pas obligatoirement récursive !). De plus, toute suite infinie croissante (pour l'inclusion) de sous-ensembles clos supérieurement est ultimement stationnaire. Par conséquent, le pré-ordre $\sqsubseteq$ associé à $\mathcal{A}^b_{\uparrow\mathcal{LTS}}$ satisfait la condition de chaîne ascendante, et ainsi l'algorithme `EnsembleSymbolique` appliqué à $\mathcal{A}^b_{\uparrow\mathcal{LTS}}$ termine. Cette instance particulière de l'algorithme `EnsembleSymbolique` correspond en fait à l'algorithme de chaînage arrière décrit dans [AČJ00, FS00a] pour le calcul de $\mathrm{pre}^*(\uparrow\{s_0\})$ dans le cadre des systèmes de transitions bien structurés.

On obtient donc le théorème suivant :

Théorème 2.3.7 ([AČJ00, FS00a]) *Soit $(\mathcal{LTS}, \preceq)$ un système de transitions étiqueté pré-ordonné admettant une base supérieure post finie. Si $\preceq$ est un beau pré-ordre sur l'ensemble des états de $\mathcal{LTS}$, alors pour tout sous-ensemble fini X d'états initiaux, l'algorithme* `EnsembleSymbolique` *appliqué à $(\mathcal{A}^b_{\uparrow\mathcal{LTS}}, X)$ termine et calcule une base (finie) de* $\mathrm{post}^*(\uparrow\mathcal{LTS}, X)$.

Accélération

Commençons par remarquer que la fonction de "post abstrait" $\mathrm{post}_{\mathcal{A}}$ de l'abstraction $\mathcal{A}^b_{\downarrow\mathcal{LTS}}$ est ici croissante (pour le pré-ordre $\sqsubseteq$ associé à $\mathcal{A}^b_{\downarrow\mathcal{LTS}}$). Si de plus on a : pour toute suite infinie $(X_i)_{i\in\mathbb{N}}$ de sous-ensembles finis de S croissante pour $\sqsubseteq$ (i.e. $\downarrow X_i \subseteq \downarrow X_{i+1}$), il existe un sous-ensemble fini Y tel que $\downarrow Y = \bigcup_{i\in\mathbb{N}} \downarrow X_i$, alors l'abstraction $\mathcal{A}^b_{\downarrow\mathcal{LTS}}$ satisfait les deux conditions permettant la définition d'une accélération générique, comme décrit à la fin de la sous-section 2.1.3.

Comme on l'a déjà remarqué, la faiblesse de l'abstraction $\mathcal{A}^b_{\downarrow\mathcal{LTS}}$ réside dans le fait qu'elle ne peut représenter que les ensembles d'états clos inférieurement et de base finie. Par conséquent, elle ne satisfait par nécessairement la condition de complétude ci-dessus. En revanche, une autre abstraction $\mathcal{A}_{\downarrow\mathcal{LTS}}$ de $\downarrow\mathcal{LTS}$ capturant tous les ensembles d'états clos inférieurement pourra satisfaire la condition de complétude ci-dessus. Ainsi, pourvu que sa fonction de "post abstrait" $\mathrm{post}_{\mathcal{A}}$ soit croissante, $\mathcal{A}_{\downarrow\mathcal{LTS}}$ satisfait alors les deux conditions permettant la définition d'une accélération générique.

2.3.2 Application à l'analyse des systèmes de transitions étiquetés bien structurés

Nous montrons à présent comment appliquer les résultats de la sous-section précédente au calcul de post* pour certaines classes de systèmes de transitions étiquetés pré-ordonnés. Intuitivement, les classes que nous considérons satisfont des propriétés permettant de relier $\mathrm{post}^*(\mathcal{LTS})$ à $\mathrm{post}^*(\downarrow \mathcal{LTS})$ et à $\mathrm{post}^*(\uparrow \mathcal{LTS})$.

Commençons par une première proposition dont nous nous servirons au chapitre 5 pour analyser les automates à file réactifs :

Proposition 2.3.8 *Etant donné un système de transitions étiqueté pré-ordonné* $(\mathcal{LTS}, \preceq)$ *et un ensemble* X *d'états initiaux, les deux assertions suivantes sont satisfaites :*

– *si* $\mathrm{post}^*(\mathcal{LTS}, X)$ *est clos inférieurement alors on a :*
$$\mathrm{post}^*(\mathcal{LTS}, X) = \mathrm{post}^*(\downarrow\mathcal{LTS}, X)$$
– *si* $\mathrm{post}^*(\mathcal{LTS}, X)$ *est clos supérieurement alors on a :*
$$\mathrm{post}^*(\mathcal{LTS}, X) = \mathrm{post}^*(\uparrow\mathcal{LTS}, X)$$

Preuve. Nous ne prouvons que la première assertion, la preuve de la seconde assertion s'obtenant par symétrie. D'après la définition de $\downarrow\mathcal{LTS}$, on obtient que $\mathrm{post}^*(\mathcal{LTS}, X) \subseteq \mathrm{post}^*(\downarrow\mathcal{LTS}, X)$. Supposons maintenant que $\mathrm{post}^*(\mathcal{LTS}, X)$ est clos inférieurement et montrons l'inclusion réciproque. On montre par récurrence sur i qu'on a $\mathrm{post}^i(\downarrow\mathcal{LTS}, X) \subseteq \mathrm{post}^*(\mathcal{LTS}, X)$ pour tout $i \in \mathbb{N}$. La base est triviale et l'étape d'induction se prouve en remarquant que pour tout $i \in \mathbb{N}$, $\mathrm{post}^{i+1}(\downarrow\mathcal{LTS}, X) = \mathrm{post}(\mathcal{LTS}, \mathrm{post}^i(\downarrow\mathcal{LTS}, X)) \cup \downarrow\mathrm{post}^i(\downarrow\mathcal{LTS}, X)$. ■

Nous nous intéressons maintenant aux systèmes de transitions étiquetés pré-ordonnés dont la relation de transitions étiquetée $\rightarrow$ est "compatible" avec le pré-ordre $\preceq$. Formellement,

Définition 2.3.9 ([FS00a]) *Un système de transitions étiqueté pré-ordonné* $(\mathcal{LTS}, \preceq)$ *est* compatible en avant *(resp.* compatible en arrière*) si pour toute transition* $s_1 \xrightarrow{\alpha} s_1'$ *et pour tout état* $s_2 \succeq s_1$ *il existe un état* $s_2' \succeq s_1'$ *tel que* $s_2 \xrightarrow{*} s_2'$ *(resp. pour toute transition* $s_1 \xrightarrow{\alpha} s_1'$ *et pour tout état* $s_2' \succeq s_1'$ *il existe un état* $s_2 \succeq s_1$ *tel que* $s_2 \xrightarrow{*} s_2'$*).*

Remarque. Un système de transitions étiqueté pré-ordonné $(\mathcal{LTS}, \preceq)$ est compatible en avant si et seulement si son dual $(\overline{\mathcal{LTS}}, \preceq)$ est compatible en arrière.

La proposition suivante montre que le calcul de la clôture inférieure (resp. clôture supérieure) de post* dans un système de transitions étiqueté pré-ordonné compatible en avant (resp. en arrière) revient au calcul de post* dans sa clôture inférieure (resp. supérieure).

Proposition 2.3.10 *Etant donné un système de transitions étiqueté pré-ordonné* $(\mathcal{LTS}, \preceq)$, *les deux assertions suivantes sont satisfaites :*

- *si* $(\mathcal{LTS}, \preceq)$ *est compatible en avant, alors pour tout ensemble* X *d'états initiaux, on a :*

$$\begin{aligned} \downarrow \mathrm{post}^*(\mathcal{LTS}, X) &= \downarrow \mathrm{post}^*(\mathcal{LTS}, \downarrow X) \\ &= \mathrm{post}^*(\downarrow \mathcal{LTS}, \downarrow X) = \mathrm{post}^*(\downarrow \mathcal{LTS}, X) \end{aligned}$$

- *si* $(\mathcal{LTS}, \preceq)$ *est compatible en arrière, alors pour tout ensemble* X *d'états initiaux, on a :*

$$\begin{aligned} \mathrm{post}^*(\mathcal{LTS}, \uparrow X) &= \uparrow \mathrm{post}^*(\mathcal{LTS}, \uparrow X) \\ &= \mathrm{post}^*(\uparrow \mathcal{LTS}, \uparrow X) = \mathrm{post}^*(\uparrow \mathcal{LTS}, X) \end{aligned}$$

Preuve. Observons que par dualité, il suffit de prouver l'assertion suivante : si $(\mathcal{LTS}, \preceq)$ est compatible en avant, alors pour tout ensemble X d'états initiaux, les deux égalités suivantes sont satisfaites :

$$\begin{aligned} \downarrow \mathrm{post}^*(\mathcal{LTS}, X) &= \mathrm{post}^*(\downarrow \mathcal{LTS}, X) \\ \mathrm{pre}^*(\mathcal{LTS}, \uparrow X) &= \mathrm{pre}^*(\downarrow \mathcal{LTS}, X) \end{aligned}$$

Ces deux égalités s'obtiennent, lorsque $(\mathcal{LTS}, \preceq)$ est compatible en avant, en remarquant que pour toute paire d'états s, s', on a :

$$s \xrightarrow{*}_{\downarrow \mathcal{LTS}} s' \text{ si et seulement s'il existe un état } s'' \succeq s' \text{ tel que } s \xrightarrow{*}_{\mathcal{LTS}} s''$$

Supposons que $(\mathcal{LTS}, \preceq)$ est compatible en avant, et montrons l'équivalence ci-dessus. Soit s et s' deux états.

S'il existe un état $s'' \succeq s'$ tel que $s \xrightarrow{*}_{\mathcal{LTS}} s''$, alors il vient $s'' \xrightarrow{\downarrow}_{\downarrow \mathcal{LTS}} s'$ et par conséquent $s \xrightarrow{*}_{\downarrow \mathcal{LTS}} s'$, ce qui prouve le sens $\Leftarrow$.

Montrons le sens $\Rightarrow$ par récurrence sur la longueur d'un chemin $s \xrightarrow{*}_{\downarrow \mathcal{LTS}} s'$. La base est triviale, car si le chemin est de longueur nulle, alors $s = s'$. Pour l'étape d'induction, supposons que $s \xrightarrow{*}_{\downarrow \mathcal{LTS}} s_1 \xrightarrow{\alpha}_{\downarrow \mathcal{LTS}} s'$ et qu'il existe $s_2 \succeq s_1$ tel que $s \xrightarrow{*}_{\mathcal{LTS}} s_2$. Deux cas se présentent :

- si $\alpha = \downarrow$ alors on a $s_1 \succeq s'$ et par conséquent $s_2 \succeq s_1 \succeq s'$.
- sinon, par compatibilité en avant de $(\mathcal{LTS}, \preceq)$, on sait qu'il existe un état $s'' \succeq s'$ tel que $s_2 \xrightarrow{*}_{\mathcal{LTS}} s''$, et par conséquent $s \xrightarrow{*}_{\mathcal{LTS}} s_2 \xrightarrow{*}_{\mathcal{LTS}} s''$.

Dans les deux cas, on obtient qu'il existe un état $s'' \succeq s'$ tel que $s \xrightarrow{*}_{\mathcal{LTS}} s''$. ■

Corollaire 2.3.11 *Etant donné un système de transitions étiqueté pré-ordonné* $(\mathcal{LTS}, \preceq)$, *si* $(\mathcal{LTS}, \preceq)$ *est compatible en avant, alors pour tout ensemble* X *d'états initiaux, on a :*

$$\mathrm{pre}^*(\mathcal{LTS}, \uparrow X) = \uparrow \mathrm{pre}^*(\mathcal{LTS}, \uparrow X) = \mathrm{post}^*(\uparrow \overline{\mathcal{LTS}}, X)$$

Nous pouvons maintenant appliquer les résultats de la sous-section précédente au calcul de pre* dans les systèmes de transitions étiquetés bien structurés. Rappelons tout d'abord la définition des systèmes de transitions bien structurés :

Définition 2.3.12 ([AČJ00, FS00a]) *Un système de transitions étiqueté pré-ordonné* $(\mathcal{LTS}, \preceq)$ *est* bien structuré *si d'une part* $(\mathcal{LTS}, \preceq)$ *est compatible en avant, et d'autre part* $\preceq$ *est un beau pré-ordre sur l'ensemble des états de* $\mathcal{LTS}$*.*

En utilisant le théorème 2.3.7 et le corollaire 2.3.11, on retrouve le résultat de [AČJ00, FS00a] sur le calcul de pre* par ensembles clos supérieurement.

Théorème 2.3.13 ([AČJ00, FS00a]) *Soit* $(\mathcal{LTS}, \preceq)$ *un système de transitions étiqueté bien structuré, et supposons que son dual* $\overline{\mathcal{LTS}}$ *admet une base supérieure post finie. Pour tout sous-ensemble fini* X *d'états initiaux, l'algorithme* `EnsembleSymbolique` *appliqué à* $(\mathcal{A}_{\uparrow\overline{\mathcal{LTS}}}, X)$ *termine et calcule une base (finie) de* $\mathrm{pre}^*(\mathcal{LTS}, \uparrow X)$.

2.4 Abstraction et accélération des automates étendus

En pratique, les systèmes de transitions étiquetés à analyser ne seront pas donnés en extension, car ils pourront être infinis, et même lorsqu'ils seront finis, une description finie en extension pourrait être trop grande. Ils seront plutôt exprimés sous la forme d'une sémantique opérationnelle associée à une *structure de contrôle* C finie. Les *automates étendus* que nous allons étudier dans la suite de cette thèse, à savoir les systèmes d'addition de vecteurs avec états étendus, les automates à files FIFO et les automates à file réactifs, suivent cette approche.

Nous précisons dans cette section la notion de *représentation symbolique*, puis nous expliquons comment définir, à partir d'une représentation symbolique, une abstraction d'un automate étendu et une accélération pour cette abstraction.

2.4.1 Automates étendus

Commençons par définir les automates étendus. Nos définitions seront suffisamment générales pour pouvoir englober les systèmes d'addition de vecteurs avec états étendus, les automates à files FIFO et les automates à file réactifs, mais aussi les automates à compteurs, les automates temporisés, les automates à pile...

Définition 2.4.1 *Un* contrôle d'automate étendu *est un système de transitions étiqueté fini* $\mathsf{E} = (Q, \mathtt{Op}, \rightarrow)$.

Ainsi, un contrôle d'automate étendu $\mathsf{E} = (Q, \mathtt{Op}, \rightarrow)$ est essentiellement un graphe (de contrôle) fini, et on dit que :

- Q est l'ensemble fini des *états de contrôle*, et
- $\mathtt{Op}$ est l'alphabet d'*opérations*, et

– la relation de transitions étiquetée $\rightarrow \subseteq Q \times \mathtt{Op} \times Q$ est l'ensemble fini des *transitions de contrôle.*

Exemple 2.4.2 La structure de contrôle EC décrite sur la figure 1.1 de la page 14, et définie dans l'exemple 1.2.2, est un contrôle d'automate étendu. Elle a pour ensemble d'opérations $\{c = 0, c \leftarrow c + 2, c \leftarrow c - 7, (c \leftarrow c + 1) \wedge (d \leftarrow d + 3)\}$.

On s'attend à ce qu'une opération tienne compte des variables, afin :

– d'en modifier la valeur, comme par exemple l'opération $c \leftarrow c + 2$, et afin

– d'activer ou de désactiver une transition de contrôle, comme par exemple l'opération $c = 0$. L'opération joue alors un rôle de *garde*.

Aussi utiliserons-nous la définition suivante pour donner une signification aux opérations :

Définition 2.4.3 *Une* sémantique *d'un alphabet d'opérations* $\mathtt{Op}$ *est un triplet* $\Delta = (V, \mathbb{D}, \delta)$*, où :*

– *V est un ensemble fini de variables, et*

– *$\mathbb{D}$ est un domaine d'interprétation pour les variables, et*

– *δ est une fonction de $\mathtt{Op}$ dans $\mathcal{P}(\mathbb{D}^V \times \mathbb{D}^V)$.*

La fonction δ associe à chaque opération une relation sur $\mathbb{D}^V$, permettant de décrire l'effet d'une opération sur les variables. Observons que le domaine d'interprétation des variables est le même pour toutes les variables. La notion de sémantique d'un alphabet d'opérations peut en fait s'étendre facilement au cas où les variables ont des domaines d'interprétation différents. Cependant, les automates étendus que nous allons étudier par la suite n'auront que des variables du même type.

Nous étendons naturellement la fonction $\delta : \mathtt{Op} \rightarrow \mathcal{P}(\mathbb{D}^V \times \mathbb{D}^V)$ aux séquences d'opérations $\sigma \in \mathtt{Op}^*$, par la définition récursive suivante :

$$\begin{cases} \delta(\varepsilon) &= \{(\mathsf{d}, \mathsf{d}) \ / \ \mathsf{d} \in \mathbb{D}^V\} \\ \delta(\mathsf{op} \cdot \sigma) &= \{(\mathsf{d}, \mathsf{d}') \ / \ \exists\ \mathsf{d}'' \in \mathbb{D}^V, (\mathsf{d}, \mathsf{d}'') \in \delta(\mathsf{op}) \text{ et } (\mathsf{d}'', \mathsf{d}') \in \delta(\sigma)\} \end{cases}$$

Exemple 2.4.4 L'ensemble d'opérations $\mathtt{Op} = \{c = 0, c \leftarrow c + 2, c \leftarrow c - 7, (c \leftarrow c + 1) \wedge (d \leftarrow d + 3)\}$ du système à deux compteurs de la figure 1.1 a pour sémantique naturelle $(\mathbb{N}, \{c, d\}, \delta)$, où la fonction $\delta : \mathtt{Op} \rightarrow \mathcal{P}(\mathbb{N}^{\{c,d\}} \times \mathbb{N}^{\{c,d\}})$ est définie par :

$$\begin{array}{lrll} (\mathsf{x}, \mathsf{x}') \in & \delta(c = 0) & \text{si} & \mathsf{x}[c] = 0 \ \text{ et } \ \mathsf{x}' = \mathsf{x} \\ (\mathsf{x}, \mathsf{x}') \in & \delta(c \leftarrow c + 2) & \text{si} & \mathsf{x}' = \mathsf{x}[c := \mathsf{x}[c] + 2] \\ (\mathsf{x}, \mathsf{x}') \in & \delta(c \leftarrow c - 7) & \text{si} & \mathsf{x}[c] \geq 7 \ \text{ et } \ \mathsf{x}' = \mathsf{x}[c := \mathsf{x}[c] - 7] \\ (\mathsf{x}, \mathsf{x}') \in & \delta((c \leftarrow c + 1) \wedge (d \leftarrow d + 3)) & \text{si} & \mathsf{x}' = \mathsf{x}[c := \mathsf{x}[c] + 1][d := \mathsf{x}[d] + 3] \end{array}$$

Un contrôle d'automate étendu E muni d'une sémantique Δ pour son alphabet d'opérations permet de définir un système de transitions étiqueté (potentiellement infini), donnant la sémantique opérationnelle associée à E, conformément à la sémantique Δ. Formellement,

Définition 2.4.5 *Un contrôle d'automate étendu* $\mathsf{E} = (Q, \mathtt{Op}, \rightarrow_{\mathsf{E}})$*, muni d'une sémantique* $\Delta = (V, \mathbb{D}, \delta)$ *de* $\mathtt{Op}$*, définit le système de transitions étiqueté* $\mathcal{E}_\Delta = (S, \Sigma, \rightarrow_{\mathcal{E}})$*, appelé* automate étendu*, comme suit :*

- $S = Q \times \mathbb{D}^V$ *est l'ensemble des états, et*
- $\Sigma = \mathtt{Op}$ *est l'alphabet d'étiquettes (ou d'opérations), et*
- *la relation de transitions étiquetée* $\rightarrow_{\mathcal{E}}$ *est le plus petit sous-ensemble* $\rightarrow_{\mathcal{E}}$ *de* $S \times \Sigma \times S$ *satisfaisant pour tout* $\mathsf{d}, \mathsf{d}' \in \mathbb{D}^V$ *la condition suivante :*

$$si \quad q \xrightarrow{\mathtt{op}}_{\mathsf{E}} q' \;\; et \;\; (\mathsf{d}, \mathsf{d}') \in \delta(\mathtt{op}) \quad alors \quad (q, \mathsf{d}) \xrightarrow{\mathtt{op}}_{\mathcal{E}} (q', \mathsf{d}')$$

On dit que Δ est la sémantique de l'automate étendu $\mathcal{E}_\Delta$, qu'on note plus simplement $\mathcal{E}$ lorsque Δ se comprend d'après le contexte.

Exemple 2.4.6 Le système de transitions étiqueté $\mathcal{EC}$ associé au contrôle d'automate étendu EC dans l'exemple 1.2.3 est en fait l'automate étendu associé à EC pour la sémantique naturelle donnée dans l'exemple 2.4.4.

Remarque. On voudra parfois rendre prioritaires certaines opérations "plus urgentes", comme ce sera le cas pour les automates à file réactifs. On associera alors à un contrôle d'automate étendu E un autre automate étendu $\mathcal{E}'$, dont la relation de transitions étiquetée tiendra compte de ces priorités.

Observons que si un contrôle d'automate étendu E muni d'un état de contrôle initial $q_0 \in Q$ est ρ-plat, alors pour toute sémantique $\Delta = (V, \mathbb{D}, \delta)$ de $\mathtt{Op}$ et pour tout ensemble $X \subseteq \{q_0\} \times \mathbb{D}^V$ d'états initiaux de $\mathcal{E}$, on a $Traces(\mathcal{E}, X) \subseteq Traces(\mathsf{E}, q_0) \subseteq [\![\rho]\!]$ et par conséquent $Traces(\mathcal{E}, X)$ est ρ-plat. Cette observation nous donne donc une condition suffisante décidable de platitude pour les automates étendus.

Remarque. Nous avions choisi dans [FIS00] d'étiqueter les transitions d'un automate étendu par ses transitions de contrôle, c'est-à-dire à la place d'une transition $(q, \mathsf{d}) \xrightarrow{\mathtt{op}} (q', \mathsf{d}')$, nous avions la transition $(q, \mathsf{d}) \xrightarrow{(q, \mathtt{op}, q')} (q', \mathsf{d}')$. Notons $\widetilde{\mathcal{E}}$ le système de transitions étiqueté ainsi obtenu. L'intérêt de $\widetilde{\mathcal{E}}$ est que les seules séquences d'exécutions itérables au moins deux fois de suite sont en fait les circuits de contrôle, ce qui permet de simplifier la définition de l'accélération. Cependant, outre le fait qu'il n'est pas habituel, l'étiquetage de $\widetilde{\mathcal{E}}$ souffre de plusieurs inconvénients :

- $\widetilde{\mathcal{E}}$ est moins souvent plat que $\mathcal{E}$. En effet, lorsque $(\widetilde{\mathcal{E}}, X)$ est plat, alors $(\mathcal{E}, X)$ est nécessairement plat car d'une part $Traces(\mathcal{E}, X)$ est l'image

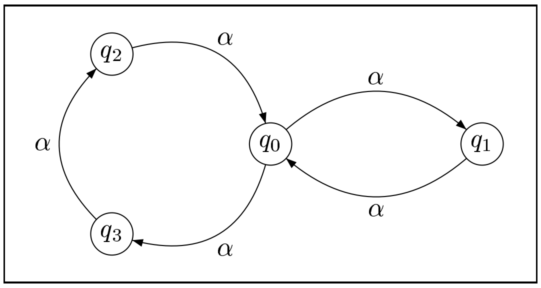


FIG. 2.1 – Contrôle d'automate étendu EP montrant l'importance de l'étiquetage pour la platitude

par un morphisme de $Traces(\widetilde{\mathcal{E}}, X)$, et d'autre part les langages plats sont stables par morphisme.

Cependant même lorsque (E, q_0) est plat, $(\widetilde{\mathcal{E}}, \{q_0\} \times \mathbb{D}^V)$ peut ne pas être plat. Il suffit pour s'en convaincre de considérer le contrôle d'automate étendu EP donné en figure 2.1, muni par exemple d'une sémantique triviale donnée par $\delta(\alpha) = \mathbb{D}^V \times \mathbb{D}^V$. Il est clair que (EP, q_0) est plat et donc $(\mathcal{EP}, \{q_0\} \times \mathbb{D}^V)$ est lui aussi plat. Cependant, on obtient que $Traces(\widetilde{\mathcal{EP}}, \{q_0\} \times \mathbb{D}^V) = [\![((q_0, \alpha, q_1) \cdot (q_1, \alpha, q_0) + (q_0, \alpha, q_3) \cdot (q_3, \alpha, q_2) \cdot (q_2, \alpha, q_0))^*]\!]$, et par conséquent $(\widetilde{\mathcal{EP}}, \{q_0\} \times \mathbb{D}^V)$ n'est pas plat. Cette différence entre $\mathcal{EP}$ et $\widetilde{\mathcal{EP}}$ provient en fait de la non clôture des langages plats par morphisme inverse.

- Le fait de choisir $\widetilde{\mathcal{E}}$ à la place de $\mathcal{E}$ multiplie la taille de l'alphabet d'étiquetage (dans le cas le pire) par $|Q|^2$, ce qui augmente d'autant le degré de l'arbre symbolique construit par l'algorithme **`ArbreSymbolique`** de la page 34. Observons qu'en pratique, le nombre d'opérations est bien plus petit que le nombre d'états de contrôle, aussi cet accroissement du degré de l'arbre symbolique est énorme. Le nombre de noeuds de profondeur n est ainsi (dans le cas le pire) multiplié par $|Q|^{2n}$, ce qui limite l'intérêt pratique de l'algorithme.

- Une accélération complète pour une abstraction de $\mathcal{E}$ ne se contente pas d'itérer les circuits de contrôle ; elle accélère donc plus qu'une accélération complète pour une abstraction de $\mathcal{E}'$, qui se limitera, elle, à l'itération des circuits de contrôle (ce qui la rend par ailleurs plus simple).

- Un état abstrait a, obtenu par application de la fonction de "post abstrait" sur une étiquette α de la forme $\alpha = (q, \mathtt{op}, q')$, dénote nécessairement un sous-ensemble de $\{q'\} \times \mathbb{D}^V$, alors que si $\alpha \in \mathtt{Op}$, a peut dénoter un ensemble bien plus grand d'états.

Nous avons donc choisi de garder dans cette thèse l'étiquetage habituel des automates étendus. Avec les mêmes hypothèses que dans [FIS00] pour l'abstraction de $\widetilde{\mathcal{E}}$ et l'accélération $\widetilde{\nabla}$ de cette abstraction, nous définissons dans la suite de cette section une abstraction de $\mathcal{E}$ et une accélération correcte ∇ pour cette abstraction. De plus, ∇ est complète sous les mêmes hypothèses que celles assurant la complétude $\widetilde{\nabla}$ dans [FIS00].

Nous considérons maintenant, pour le reste de cette section, un alphabet d'opérations fixé $\mathtt{Op}$ muni d'une sémantique fixée $\Delta = (V, \mathbb{D}, \delta)$.

2.4.2 Abstraction

L'ensemble des états de contrôle étant fini, un automate étendu $\mathcal{E}$ de sémantique Δ est infini (i.e. son ensemble d'états est infini) lorsque le domaine d'interprétation des variables $\mathbb{D}$ est infini. On cherchera donc à représenter symboliquement (de manière finie) des sous-ensembles infinis de $\mathbb{D}^V$. De plus, afin de pouvoir définir une fonction de "post abstrait", cette représentation symbolique devra comporter des "opérations symboliques" correspondant aux opérations de E. Formellement,

Définition 2.4.7 *Une* représentation symbolique pour la sémantique Δ *est un quintuplet* $\mathcal{R}_\Delta = (R, \delta_\mathcal{R}, [\![\cdot]\!], +, \bot)$ *où :*

- *R est un ensemble de représentants symboliques, et $+$ est une loi de composition interne sur R d'élément neutre $\bot$, et*
- *$\delta_\mathcal{R} : \mathtt{Op} \times R \to R$ et $[\![\cdot]\!] : R \to \mathcal{P}(\mathbb{D}^V)$ sont deux fonctions satisfaisant pour tout $r, r' \in R$ et $\mathsf{op} \in \mathtt{Op}$:*

$$\begin{aligned} [\![\bot]\!] &= \emptyset && (2.7)\\ [\![\delta_\mathcal{R}(\mathsf{op}, r)]\!] &= \{\mathsf{d}' \in \mathbb{D}^V \ / \ \exists\ \mathsf{d} \in [\![r]\!], (\mathsf{d}, \mathsf{d}') \in \delta(\mathsf{op})\} && (2.8)\\ [\![r + r']\!] &= [\![r]\!] \cup [\![r']\!] && (2.9) \end{aligned}$$

On note plus simplement $\mathcal{R}$ au lieu de $\mathcal{R}_\Delta$ lorsque la sémantique Δ se comprend d'après le contexte. Observons que puisque $+$ est une loi de composition interne sur R, d'élément neutre $\bot$, la notation $\sum_{r \in I} r$ est licite pour tout $I \subseteq R$, avec $\sum_{r \in I} r = \bot$ si $I = \emptyset$. Un sous-ensemble $X \subseteq \mathbb{D}^V$ est dit $\mathcal{R}$*–représentable* s'il existe un représentant symbolique $r \in R$ tel que $X = [\![r]\!]$.

Nous étendons aussi naturellement la fonction $\delta_\mathcal{R} : \mathtt{Op} \times R \to R$ aux séquences d'opérations $\sigma \in \mathtt{Op}^*$, par la définition récursive suivante :

$$\left\{ \begin{aligned} \delta_\mathcal{R}(\varepsilon, r) &= r \\ \delta_\mathcal{R}(\mathsf{op} \cdot \sigma, r) &= \delta_\mathcal{R}(\delta_\mathcal{R}(\mathsf{op}, r), \sigma) \end{aligned} \right.$$

Remarque. La condition (2.8) est naturellement préservée par cette extension aux séquences d'opérations. En effet, pour tout représentant symbolique $r \in R$ et pour toute séquence d'opérations $\sigma \in \mathtt{Op}^*$, on a :

$$[\![\delta_\mathcal{R}(\sigma, r)]\!] = \{\mathsf{d}' \in \mathbb{D}^V \ / \ \exists\ \mathsf{d} \in [\![r]\!], (\mathsf{d}, \mathsf{d}') \in \delta(\sigma)\}$$

Remarque. L'opération de composition interne $+$ ainsi que son élément neutre $\perp$ ne sont pas cruciaux, mais permettront par la suite de simplifier les notations. En effet, étant donné un triplet $\mathcal{R} = (R, \delta_{\mathcal{R}}, [\![\cdot]\!])$ satisfaisant la condition (2.8) ci-dessus, on peut construire une représentation symbolique pour Δ en considérant $R' = \mathcal{P}_f(R)$, et en étendant les fonctions $\delta_{\mathcal{R}}$ et $[\![\cdot]\!]$ sur R' comme on s'y attend par $\delta_{\mathcal{R}}(\mathtt{op}, r') = \{\delta_{\mathcal{R}}(\mathtt{op}, r) \ / \ r \in r'\}$, et par $[\![r']\!] = \bigcup_{r \in r'} [\![r]\!]$. Le quintuplet $\mathcal{R}' = (R', \delta_{\mathcal{R}}, [\![\cdot]\!], \cup, \emptyset)$ est alors une représentation symbolique pour Δ.

Remarque. Nous avons choisi d'intégrer l'opération $+$ dans la description d'une représentation symbolique, car il est souvent possible de définir une opération $+$ adaptée à l'ensemble des représentants symboliques, plutôt que de choisir l'opération naïve d'union décrite dans la remarque précédente.

Définition 2.4.8 *Une représentation symbolique* $\mathcal{R} = (R, \delta_{\mathcal{R}}, [\![\cdot]\!], +, \perp)$ *est* effective *si d'une part* R *est un ensemble encodé, et d'autre part les fonctions* $\delta_{\mathcal{R}}$ *et* $+$ *sont récursives.*

Exemple 2.4.9 Considérons l'ensemble d'opérations $\mathtt{Op} = \{c \leftarrow c+1\}$ de sémantique naturelle $(\mathbb{N}, \{c\}, \delta)$, où la fonction $\delta : \mathtt{Op} \to \mathcal{P}(\mathbb{N}^{\{c\}} \times \mathbb{N}^{\{c\}})$ est définie par :

$$(\mathsf{x}, \mathsf{x}') \in \delta(c \leftarrow c+1) \quad \text{si} \quad \mathsf{x}' = \mathsf{x}[c := \mathsf{x}[c] + 1]$$

Nous pouvons définir une représentation symbolique effective, par sous-ensemble clos supérieurement de $\mathbb{N}$, pour cette sémantique. Elle est formellement définie par $\mathcal{R} = (\mathbb{N} \cup \{\perp\}, \delta_{\mathcal{R}}, [\![\cdot]\!], +, \perp)$ avec pour tout $x, x' \in N$:

$$\begin{array}{rcl}
[\![\perp]\!] & = & \emptyset \\
[\![x]\!] & = & \uparrow x \\
\delta_{\mathcal{R}}(c \leftarrow c+1, \perp) & = & \perp \\
\delta_{\mathcal{R}}(c \leftarrow c+1, x) & = & x+1 \\
x + x' & = & \min(x, x') \\
x + \perp & = & x \\
\perp + x & = & x
\end{array}$$

où $\min(x, x')$ désigne l'unique élément minimal, pour l'ordre total habituel $\leq$ sur les entiers, de $\{x, x'\}$.

Une représentation symbolique (resp. représentation symbolique effective) $\mathcal{R}$ pour la sémantique Δ permet de définir une abstraction (resp. abstraction effective) basée sur $\mathcal{R}$ d'un automate étendu quelconque $\mathcal{E} = (S, \mathtt{Op}, \to)$ d'états de contrôle Q et de sémantique Δ, comme suit :

Définition 2.4.10 *Soit* $\mathcal{A}_{\mathcal{E}}^{\mathcal{R}} = (A, \mathrm{post}_{\mathcal{A}}, \gamma, \sqcup, \mathtt{Op})$ *le quintuplet où l'ensemble* A*, et les fonctions* $\mathrm{post}_{\mathcal{A}} : A \times \mathtt{Op} \to A$*,* $\gamma : A \to \mathcal{P}(S)$ *et* $\sqcup : A \times A \to A$ *sont* *définis par :*

$$\begin{array}{rcl} A & = & R^Q \\ \mathrm{post}_{\mathcal{A}}(\mathsf{a}, \mathsf{op}) & = & \left\langle q \mapsto \sum_{q' \xrightarrow{\mathsf{op}} q} \delta_{\mathcal{R}}(\mathsf{op}, \mathsf{a}[q']) \right\rangle \\ \gamma(\mathsf{a}) & = & \bigcup_{q \in Q} \{q\} \times [\![\mathsf{a}[q]]\!] \\ \mathsf{a} \sqcup \mathsf{b} & = & \langle q \mapsto \mathsf{a}[q] + \mathsf{b}[q] \rangle \end{array}$$

Proposition 2.4.11 $\mathcal{A}_{\mathcal{E}}^{\mathcal{R}}$ *est une abstraction de* $\mathcal{E}$.

Remarque. On se rend compte que l'abstraction $\mathcal{A}_{\mathcal{E}}^{\mathcal{R}}$ définie ci-dessus est "fidèle", car la fonction de "post abstrait" vérifie pour tout $\mathsf{a} \in A$ et $\mathsf{op} \in \mathsf{Op}$:

$$\mathrm{post}(\gamma(\mathsf{a}), \mathsf{op}) \;=\; \gamma(\mathrm{post}_{\mathcal{A}}(\mathsf{a}, \mathsf{op})) \tag{2.10}$$

Remarque. Lorsqu'on s'intéresse à des sous-classes d'automates étendus, définies par exemple par une contrainte structurelle (i.e. une contrainte sur la structure de contrôle), il est parfois plus simple de construire une abstraction sans passer par une représentation symbolique. Supposons par exemple qu'une certaine opération τ (par exemple une opération "de perte") soit exécutable en boucle sur chaque état de contrôle, c'est-à-dire que pour tout état de contrôle q, il existe une transition de contrôle $q \xrightarrow{\tau} q$. On voudra alors peut-être choisir un ensemble d'états abstraits plus simple, dont chaque élément est "clos par application de τ" (par exemple clos inférieurement). L'abstraction correspondante n'aurait pas nécessairement pu être obtenue à partir d'une représentation symbolique, car elle ne satisfera pas obligatoirement la condition de "fidélité" (2.10). Par exemple, dans le cas des automates à files FIFO avec perte, l'application d'une opération d'émission à partir d'un ensemble d'états clos inférieurement conduit à un ensemble d'états qui n'est pas clos inférieurement.

2.4.3 Accélération

Une accélération complète pour l'abstraction $\mathcal{A}_{\mathcal{E}}^{\mathcal{R}}$ définie ci-dessus cherchera naturellement à itérer une séquence d'opérations σ de manière à capturer l'ensemble d'états $\bigcup_{i \in \mathbb{N}} \mathrm{post}(\gamma(a), \sigma^i)$. Cependant, l'effet de l'itération de σ n'est pas nécessairement représentable par un état abstrait. Nous montrons maintenant comment définir une accélération ∇ correcte pour l'abstraction $\mathcal{A}_{\mathcal{E}}^{\mathcal{R}}$, et telle que ∇ itère "le mieux possible" une séquence d'opérations σ.

Nous considérons maintenant, pour le reste de cette section, une représentation symbolique fixée $\mathcal{R}$ pour la sémantique Δ. Tous les automates étendus seront implicitement de sémantique Δ.

Pour toute séquence d'opérations $\sigma \in \mathsf{Op}^+$, nous définissons deux fonctions (également notées) σ et σ^* de $\mathcal{P}(\mathbb{D}^V)$ dans $\mathcal{P}(\mathbb{D}^V)$ comme suit :

$$\begin{array}{rcl} \sigma(X) & = & \{\mathsf{d}' \in \mathbb{D}^V \;/\; \exists\; \mathsf{d} \in X, (\mathsf{d}, \mathsf{d}') \in \delta(\sigma)\} \\ \sigma^*(X) & = & \bigcup_{i \in \mathbb{N}} \sigma^i(X) \end{array}$$

Remarque. Soit $\mathcal{E}$ un automate étendu. Si son contrôle E est restreint à un circuit de contrôle $q \xrightarrow{\sigma} q$, alors pour tout $X \subseteq \mathbb{D}^V$, on a les deux égalités suivantes :

$$\begin{array}{rcl} \{q\} \times \sigma(X) & = & \mathrm{post}(\{q\} \times X, \sigma) \\ \{q\} \times \sigma^*(X) & = & \bigcup_{i \in \mathbb{N}} \mathrm{post}(\{q\} \times X, \sigma^i) \end{array}$$

Définition 2.4.12 *Etant donné une séquence d'opérations $\sigma \in \mathtt{Op}^+$, on dit que σ^* est* $\mathcal{R}$–représentable *lorsqu'il existe une fonction $\sigma^*_{\mathcal{R}} : R \to R$ telle que $[\![\sigma^*_{\mathcal{R}}(r)]\!] = \sigma^*([\![r]\!])$ pour tout $r \in R$.*

Remarque. Si $\pi = q \xrightarrow{\sigma} q$ est un circuit de contrôle d'un automate étendu $\mathcal{E}$, alors σ^* est $\mathcal{R}$–représentable si et seulement si pour tout $r \in R$, l'effet de l'itération du circuit π depuis $\{q\} \times [\![r]\!]$ est égal à $\{q\} \times [\![r']\!]$ pour un représentant $r' \in R$. La notion de séquence d'opérations $\mathcal{R}$–représentable est donc fortement liée à la notion de circuits de contrôle dont l'effet itéré est "$\mathcal{R}$–représentable".

Lorsqu'on a un circuit de contrôle $\pi = q \xrightarrow{\sigma} q$ se décomposant sous la forme $\pi = q \xrightarrow{\sigma'} q' \xrightarrow{\sigma''} q$, il semblerait naturel que si l'effet du circuit π est "$\mathcal{R}$–représentable", alors l'effet de la permutation $q' \xrightarrow{\sigma''} q \xrightarrow{\sigma'} q'$ (qui est également un circuit de contrôle) est encore "$\mathcal{R}$–représentable". Cette idée intuitive est formalisée par la proposition suivante :

Proposition 2.4.13 *Pour tout $\sigma'\sigma'' \in \mathtt{Op}^+$, si $(\sigma'\sigma'')^*$ est $\mathcal{R}$–représentable alors $(\sigma''\sigma')^*$ est $\mathcal{R}$–représentable.*

Preuve. Pour tout sous-ensemble $X \subseteq \mathbb{D}^V$, on a :

$$\begin{array}{rcl} (\sigma''\sigma')^*(X) & = & \bigcup_{i \in \mathbb{N}} (\sigma''\sigma')^i(X) \\ & = & X \quad \bigcup \quad \bigcup_{i \in \mathbb{N}} \sigma''((\sigma'\sigma'')^i(\sigma'(X))) \\ & = & X \quad \bigcup \quad \sigma'' \left(\bigcup_{i \in \mathbb{N}} (\sigma'\sigma'')^i(\sigma'(X)) \right) \\ & = & X \quad \bigcup \quad \sigma''((\sigma'\sigma'')^*(\sigma'(X))) \end{array}$$

Or pour tout $r \in R$, on a $[\![\delta_{\mathcal{R}}(\sigma', r)]\!] = \sigma'([\![r]\!])$ et $[\![\delta_{\mathcal{R}}(\sigma'', r)]\!] = \sigma''([\![r]\!])$. Par conséquent, on en déduit que si $(\sigma'\sigma'')^*$ est $\mathcal{R}$–représentable alors $(\sigma''\sigma')^*$ est $\mathcal{R}$–représentable. ∎

Lorsqu'une séquence d'opérations σ est telle que σ^* n'est pas $\mathcal{R}$–représentable, alors, afin de définir une accélération ∇ itérant "le mieux possible", on pourrait essayer d'itérer σ^k, avec $k \in \mathbb{N}^*$ fixé, plutôt que σ, en espérant que $(\sigma^k)^*$ sera $\mathcal{R}$–représentable. Cette tentative serait en fait inutile, car la proposition suivante montre que lorsque σ^* n'est pas $\mathcal{R}$–représentable, alors pour tout $k \in \mathbb{N}^*$, $(\sigma^k)^*$ n'est pas $\mathcal{R}$–représentable non plus.

Proposition 2.4.14 *Pour tout $\sigma \in \mathsf{Op}^+$ et $k \in \mathbb{N}^*$, si $(\sigma^k)^*$ est $\mathcal{R}$–représentable alors σ^* est $\mathcal{R}$–représentable.*

Preuve. Pour tout sous-ensemble $X \subseteq \mathbb{D}^V$, on a :

$$\begin{aligned}
\sigma^*(X) &= \bigcup_{i\in\mathbb{N}} \sigma^i(X) \\
&= \bigcup_{h=0}^{k-1} \bigcup_{j\in\mathbb{N}} \sigma^{h+kj}(X) \\
&= \bigcup_{h=0}^{k-1} \bigcup_{j\in\mathbb{N}} (\sigma^k)^j(\sigma^h(X)) \\
&= \bigcup_{h=0}^{k-1} (\sigma^k)^*(\sigma^h(X))
\end{aligned}$$

Or pour tout $r \in R$ et $h \in [0\,..\,k-1]$, on a $[\![\delta_{\mathcal{R}}(\sigma^h, r)]\!] = \sigma^h([\![r]\!])$. Par conséquent, on en déduit que si $(\sigma^k)^*$ est $\mathcal{R}$–représentable alors σ^* est $\mathcal{R}$–représentable. ■

Comme conséquence immédiate de la proposition, on obtient que si $(\sigma^k)^*$ est $\mathcal{R}$–représentable alors pour tout diviseur d de k, $(\sigma^d)^*$ est $\mathcal{R}$–représentable.

Remarque. La réciproque de la proposition n'est pas vraie. Reprenons la représentation symbolique $\mathcal{R}$ par sous-ensemble clos supérieurement de $\mathbb{N}$ de l'exemple 2.4.9. Choisissons la séquence d'opérations $\sigma = (c \leftarrow c + 1)$. On voit bien que σ^* est $\mathcal{R}$–représentable, car pour tout $x \in \mathbb{N}$, $\sigma^*([\![x]\!]) = [\![x]\!]$. En revanche, l'ensemble $(\sigma^2)^*([\![0]\!]) = \{2k \ / \ k \in \mathbb{N}\}$ n'est pas clos supérieurement, et par conséquent $(\sigma^2)^*$ n'est pas $\mathcal{R}$–représentable.

Rappelons qu'un état abstrait est donné sous la forme d'un vecteur $\mathsf{a} \in R^Q$. L'accélération que nous allons définir va essayer, pour chaque état de contrôle $q \in Q$, de calculer l'ensemble des états accessibles depuis $\{q\} \times [\![\mathsf{a}[q]]\!]$, par un chemin étiqueté par une puissance de σ. Afin de calculer cet ensemble d'états accessibles, nous devrons distinguer, pour chaque état de contrôle q', l'ensemble des états accessibles *d'état de contrôle* q', depuis $\{q\} \times [\![\mathsf{a}[q]]\!]$ et par un chemin étiqueté par une puissance de σ (et aboutissant nécessairement en q'). Ceci nous amène à la définition suivante :

Définition 2.4.15 *Le* langage $L(\mathsf{E}, q_0, q_f)$ *d'un contrôle d'automate étendu* $\mathsf{E} = (Q, \mathsf{Op}, \rightarrow)$ *avec état de contrôle initial* $q_0 \in Q$ *et état de contrôle final* $q_f \in Q$ *est donné par* $L(\mathsf{E}, q_0, q_f) = \{\sigma \in \mathsf{Op}^* \ / \ q_0 \xrightarrow{\sigma} q_f\}$.

Comme expliqué auparavant, nous sommes plus particulièrement intéressés par le langage $L(\mathsf{E}, q, q') \cap \sigma^*$, décrivant l'ensemble des puissance de σ étiquetant un chemin de contrôle d'origne q et aboutissant en q'. Cet ensemble de puissances de σ sera plus simplement réprésenté par le sous-ensemble d'entiers $SL(\mathsf{E}, q, q', \sigma)$ défini comme suit :

$$SL(\mathsf{E}, q_0, q_f, \sigma) = \{k \in \mathbb{N} \ / \ \sigma^k \in L(\mathsf{E}, q_0, q_f)\}$$

On remarque qu'on a bien évidemment :

$$L(\mathsf{E}, q, q') \cap \sigma^* = \{\sigma^k \ / \ k \in SL(\mathsf{E}, q, q', \sigma)\}$$

aussi allons-nous chercher une description "optimale" de $SL(\mathsf{E}, q, q', \sigma)$, ce qui nous permettra ensuite de définir une accélération ∇ itérant "le mieux possible".

Observons que si un entier $k \in \mathbb{N}^*$ est tel que $jk \in SL(\mathsf{E}, q, q', \sigma)$ pour tout $j \in \mathbb{N}$, alors on a :

$$\{q\} \times (\sigma^k)^* (\llbracket \mathsf{a}[q'] \rrbracket) \ \subseteq \ \bigcup_{i \in \mathbb{N}} \mathrm{post}(\gamma(a), \sigma^i)$$

Par conséquent, on aimerait ajouter $(\sigma^k)^* (\llbracket \mathsf{a}[q'] \rrbracket)$ à $\mathsf{a}[q]$ lors de l'accélération, pourvu que $(\sigma^k)^*$ soit $\mathcal{R}$–représentable. Or nous avons remarqué qu'étant donné une puisssance σ^k de σ, $(\sigma^k)^*$ a d'autant plus de chances d'être $\mathcal{R}$–représentable que k est petit pour l'ordre de la divisibilité. Nous avons même donné un exemple de séquence d'opérations σ telle que σ^* est $\mathcal{R}$–représentable alors que $(\sigma^2)^*$ n'est pas $\mathcal{R}$–représentable. Par conséquent, nous allons chercher une description de $SL(\mathsf{E}, q, q', \sigma)$ faisant apparaître les plus petits entiers k (pour l'ordre partiel de la divisibilité) tels que $jk \in SL(\mathsf{E}, q, q', \sigma)$ pour tout $j \in \mathbb{N}$.

Description canonique d'un semilinéaire de $\mathbb{N}$

Comme nous allons le voir dans la suite de cette section, le sous-ensemble $SL(\mathsf{E}, q, q', \sigma)$ est un sous-ensemble semilinéaire de $\mathbb{N}$. Nous établissons maintenant un théorème de description canonique des sous-ensembles semilinéaires de $\mathbb{N}$, faisant apparaître les périodes les plus petites pour l'ordre partiel de la divisibilité.

Commençons par rappeler brièvement les notions de sous-ensembles linéaires et semilinéaires de $\mathbb{N}$. Les sous-ensembles linéaires et semilinéaires de $\mathbb{N}^C$ (i.e. de dimension quelconque) seront définis et utilisés dans le chapitre 3 pour analyser les systèmes d'addition de vecteurs avec états étendus, mais nous nous limitons ici aux sous-ensembles semilinéaires de $\mathbb{N}$.

Soit $P = \{p_1, p_2, \ldots, p_k\}$ un sous-ensemble fini (de *périodes*) de $\mathbb{N}$. Pour tout sous-ensemble $B \subseteq \mathbb{N}$ (appelé *base*), on utilise la notation suivante :

$$(B + P^*) = \{b + \Sigma_{i=1}^k c_i \, p_i \ / \ b \in B \text{ et } \forall\, i \in [1 \,..\, k], c_i \in \mathbb{N}\}$$

Lorsque B est un singleton, l'ensemble $(B+P^*)$ est dit *linéaire*. Un linéaire de la forme $(0 + P^*)$ s'écrit aussi plus simplement P^*. Un sous-ensemble *semilinéaire* de $\mathbb{N}$ est une union finie de sous-ensembles linéaires de $\mathbb{N}$. Une *description finie* d'un sous-ensemble semilinéaire $SL \subseteq \mathbb{N}$ est une famille finie $(b_i, P_i)_{i \in I}$, où $b_i \in \mathbb{N}$ et $P_i \in \mathcal{P}_f(\mathbb{N})$, et telle que $SL = \bigcup_{i \in I} (b_i + P_i{}^*)$. On remarque que les sous-ensembles semilinéaires de $\mathbb{N}$ sont isomorphes aux sous-ensembles SLRE-définissables de $\{1\}^*$.

Pour tout $p \in \mathbb{N}$, on définit la relation binaire $\leq_p$ sur $\mathbb{N}$ par $x \leq_p y$ si $y \in (x + p^*)$, et on vérifie aisément que cette relation est un ordre partiel sur $\mathbb{N}$.

Définition 2.4.16 *Une* description mono-périodique *d'un sous-ensemble semi-linéaire $SL \subseteq \mathbb{N}$ est un couple $(X, (B_i, p_i)_{i \in I})$ où :*

- *$X \subseteq \mathbb{N}$ est un sous-ensemble fini,*
- *$(B_i, p_i)_{i \in I}$ est une famille finie de couples (B_i, p_i) où $B_i \subseteq \mathbb{N}$ est un sous-ensemble fini non vide et $p_i \in \mathbb{N}^*$,*

et tel que $SL = X \bigcup \bigcup_{i \in I} (B_i + {p_i}^*)$

Définition 2.4.17 *Une* description canonique *d'un sous-ensemble semilinéaire $SL \subseteq \mathbb{N}$ est une description mono-périodique $(X, (B_i, p_i)_{i \in I})$ de SL satisfaisant les quatre conditions suivantes :*

i) on a $X \bigcap \left(\bigcup_{i \in I} (B_i + {p_i}^*) \right) = \emptyset,$

ii) pour tout $i \in I$, on a $B_i = \mathrm{Min}(\{b \in \mathbb{N} \ / \ (b + {p_i}^*) \subseteq SL\}, \leq_{p_i})$,

iii) pour tout $x \in \mathbb{N}$ et $p \in \mathbb{N}^*$, si $(x + p^*) \subseteq SL$ alors il existe $y \in (x + p^*)$, $i \in I$ et $b_i \in B_i$ tels que $(y + p^*) \subseteq (b_i + {p_i}^*)$.

iv) pour tout $i \in I$, l'ensemble $(B_i + {p_i}^*) \setminus \left(\bigcup_{p_j \mid p_i, j \neq i} (B_j + {p_j}^*) \right)$ *est infini.*

Remarque. Pour tout $x, x', p, p' \in \mathbb{N}$, si $(x + p^*) \subseteq (x' + p'^*)$ alors p est un multiple de p'.

Cette définition mérite quelques explications :

- La condition *i)* implique que X est l'ensemble fini des éléments de SL qui ne sont pas déjà capturés par l'un des $(B_i + {p_i}^*)$.
- La condition *ii)* exprime que chaque B_i est une base "minimale" pour la période p_i.
- La condition *iii)* permet de s'assurer que si $(x + p^*) \subseteq SL$, comme par exemple si $(x + p^*)$ apparaît dans une autre décomposition de SL, alors $(x + p^*)$ est capturé par l'un des $(b_i + {p_i}^*)$, excepté pour un nombre fini d'entiers. Observons que cette condition implique que p_i est un diviseur de p.
- Enfin, la dernière condition exprime qu'on ne garde une période p_i dans la décomposition canonique que si sa "contribution" $(B_i + {p_i}^*)$ apporte un nombre infini d'éléments n'étant pas déjà dans l'un des $(B_j + {p_j}^*)$ pour un diviseur strict p_j de p_i. Intuitivement, une période dont la "contribution" serait finie pourrait être supprimée de la décomposition quitte à compléter l'ensemble X.

L'intérêt de cette description canonique des sous-ensembles semilinéaires est qu'elle fait apparaître, par la condition *iii*), *les périodes les plus petites pour l'ordre partiel de la divisibilité* $|$. Les trois autres conditions permettent essentiellement d'assurer l'unicité de la description canonique.

Exemple 2.4.18 Considérons le semilinéaire SL donné par :

$$SL \;=\; (2+\{4,6\}^*) \cup (\{7,11\}+8^*) \cup (\{8,11\}+5^*) \cup \{3,13,14,15,16,17\}$$

On vérifie facilement que le couple $(\{2,17\},\{(6,2),(3,4),(\{8,11\},5)\})$ est une description canonique de SL, et ce dernier s'écrit donc :

$$SL \;=\; \{2,17\} \cup (6+2^*) \cup (3+4^*) \cup (\{8,11\}+5^*)$$

Remarque. Une telle représentation canonique des sous-ensembles semilinéaires de $\mathbb{N}$, faisant apparaître les périodes les plus petites pour l'ordre partiel de la divisibilité, a peut-être déjà été définie, mais nous n'en avons pas connaissance.

Nous allons maintenant montrer que tout sous-ensemble semilinéaire de $\mathbb{N}$ admet une unique représentation canonique. Nous aurons besoin pour y parvenir des trois lemmes suivants.

Lemme 2.4.19 *Tout sous-ensemble semilinéaire SL de $\mathbb{N}$ admet au plus une description canonique.*

Preuve. Supposons que $(X,(B_i,p_i)_{i\in I})$ et $(X',(B'_j,p'_j)_{j\in J})$ sont deux descriptions canoniques de SL. On note que d'après la condition *ii*), les ensembles B_i (resp. B'_j) sont déterminés de façon uniques par les p_i (resp. p'_j). De plus, d'après la condition *i*), l'ensemble X (resp. X') est lui aussi déterminé de façon unique par la famille $(B_i,p_i)_{i\in I}$ (resp. $(B'_j,p'_j)_{j\in J}$). On en déduit qu'il suffit de montrer que $\{p_i \;/\; i\in I\} = \{p'_j \;/\; j\in J\}$.

Par l'absurde, supposons qu'il existe $k\in I$ tel que $p_k \neq p'_j$ pour tout $j\in J$. Nous allons montrer que la condition *iv*) n'est pas respectée pour k.

Soit $b_k\in B_k$. Comme $(b_k+{p_k}^*)\subseteq SL$, on obtient d'après la condition *iii*) qu'il existe $y_k\in(b_k+{p_k}^*)$, $j\in J$ et $b'_j\in B'_j$ tels que $(y_k+{p_k}^*)\subseteq(b'_j+{p'_j}^*)$. Réciproquement, comme $(y_k+{p'_j}^*)\subseteq(b'_j+{p'_j}^*)\subseteq SL$, on obtient d'après la condition *iii*) qu'il existe $y'_j\in(y_k+{p'_j}^*)$, $i\in I$ et $b_i\in B_i$ tels que $(y'_j+{p'_j}^*)\subseteq(b_i+{p_i}^*)$. On obtient ainsi que :

- Comme $(y_k+{p_k}^*)\subseteq(b'_j+{p'_j}^*)$, il vient $p'_j\,|\,p_k$, et comme $(y'_j+{p'_j}^*)\subseteq(b_i+{p_i}^*)$, il vient $p_i\,|\,p'_j$. Rappelons que $p'_j\neq p_k$, et par conséquent on obtient que $p_i\,|\,p_k$ et $p_i\neq p_k$.

- Comme $y'_j\in(y_k+{p'_j}^*)$ et $p'_j\,|\,p_k$, on peut supposer sans perte de généralité que $y'_j\in(y_k+{p_k}^*)$, et donc on a $y'_j\in(b_k+{p_k}^*)$. De plus, comme on a $(y'_j+{p'_j}^*)\subseteq(b_i+{p_i}^*)$, on obtient que $(y'_j+{p_k}^*)\subseteq(b_i+{p_i}^*)$, car $p'_j\,|\,p_k$. Ainsi, l'ensemble $(b_k+{p_k}^*)\setminus(b_i+{p_i}^*)$ est fini.

On vient donc de démontrer que pour tout $b_k \in B_k$, il existe $i \in I$ tel que d'une part $p_i \,|\, p_k$ et $p_i \neq p_k$, et d'autre part $(b_k + p_k{}^*) \setminus (b_i + p_i{}^*)$ est fini. On en conclut que l'ensemble :

$$(B_k + p_k{}^*) \setminus \left(\bigcup_{p_i \,|\, p_k,\, i \neq k} (B_i + p_i{}^*) \right)$$

est fini, ce qui contredit la condition *iv*). ■

Notation. Tout sous-ensemble fini X de $\mathbb{N}$, totalement ordonné par l'ordre habituel $\leq$ sur les entiers, admet un unique élément maximal, noté $\max(X)$, et un unique élément minimal, noté $\min(X)$. On a donc $\mathrm{Max}(X) = \{\max(X)\}$ et $\mathrm{Min}(X) = \{\min(X)\}$.

Lemme 2.4.20 *Tout sous-ensemble semilinéaire SL de $\mathbb{N}$ peut s'écrire sous la forme $SL = Y \cup (B + m^*)$, où d'une part $Y, B \subseteq \mathbb{N}$ sont finis et $m \in \mathbb{N}^*$, et d'autre part si $B \neq \emptyset$ alors $\max(Y) < \min(B)$ et $(\max(B) - \min(B)) < m$.*

Preuve. Le cas où SL est fini est trivial, supposons donc que SL est infini. Soit M l'automate minimal pour le langage rationnel $\{1^k \;/\; k \in SL\}$. Rappelons que M est déterministe, avec un unique état initial q_0. Comme l'alphabet est réduit à un seul symbole, on en déduit que M est de la forme $q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_2 \cdots q_{k-1} \xrightarrow{1} q_k$ avec $q_k = q_h$ pour un certain $h \in [0 \mathinner{..} k-1]$. L'ensemble Y se déduit de M à partir des états finaux q_i tels que $i \in [0 \mathinner{..} h-1]$, l'entier m est précisément $k-h$, et l'ensemble B se déduit de M à partir des états finaux q_i tels que $i \in [h \mathinner{..} k]$. Les conditions $\max(Y) < \min(B)$ et $(\max(B) - \min(B)) < m$ sont satisfaites par construction. ■

Lemme 2.4.21 *Soit $SL = (B + m^*)$ avec $m \in \mathbb{N}^*$ et $B \subseteq \mathbb{N}$. Supposons que B est fini, non vide, et qu'il vérifie $(\max(B) - \min(B)) < m$. Pour tout $x \in \mathbb{N}$ et $p \in \mathbb{N}^*$, si $(x + p^*) \subseteq SL$ alors on a $(x + d^*) \subseteq SL$ où $d = \mathrm{pgcd}(m, p)$.*

Preuve. Soit $x \in \mathbb{N}$ et $p \in \mathbb{N}^*$, et supposons que $(x + p^*) \subseteq SL$. Quitte à remplacer $\min(B)$ par $\min(B) + m$ dans B, et à itérer ce processus de remplacement, on peut supposer que $\min(B) \geq x$. On peut aussi, quitte à soustraire x de chaque élément de B, supposer que $x = 0$. Comme $x \in (B + m^*)$, on obtient que $0 = \min(B)$. Soit $d = \mathrm{pgcd}(m, p)$ et posons $k = \frac{m}{d}$. Il reste à prouver que $d^* \subseteq SL$.

Puisque $p^* \subseteq (B + m^*)$, et comme $\max(B) < m$, on en déduit que pour tout $c \in \mathbb{N}$, il existe un unique élément $b_c \in B$ tel que $cp \in (b_c + m^*)$. Observons que $kp = \mathrm{ppcm}(m, p)$, et donc k est le plus petit entier non nul tel que kp soit un multiple de m. Ainsi on obtient d'une part que $b_k = 0$, et d'autre part que les b_c, pour $c \in [0 \mathinner{..} k-1]$, sont tous distincts. Il est clair que la “distance” $|b_{c+1} - b_c|$ est constante et par conséquent, il vient $|b_{c+1} - b_c| = d$ pour tout $c \in [0 \mathinner{..} k-1]$. On conclut donc que $d^* \subseteq SL$. ■

Théorème 2.4.22 *Tout sous-ensemble semilinéaire de $\mathbb{N}$ admet une unique représentation canonique.*

Preuve. L'unicité de la représentation canonique, sous réserve d'existence, est déjà démontrée par le lemme 2.4.19. Soit SL un sous-ensemble semilinéaire de $\mathbb{N}$, et montrons que SL admet une représentation canonique. Le cas où SL est fini est trivial, supposons donc que SL est infini. D'après le lemme 2.4.20, SL peut s'écrire sous la forme $SL = Y \cup (B + m^*)$, où $Y, B \subseteq \mathbb{N}$ sont finis, $B \neq \emptyset$ et $m \in \mathbb{N}^*$, et tels que $\max(Y) < \min(B)$ et $(\max(B) - \min(B)) < m$.

Pour tout $p \in \mathbb{N}^*$, soit CB_p l'ensemble des "candidats de base pour p" défini par :

$$CB_p = \{b \in \mathbb{N} \ / \ (b + p^*) \subseteq SL\}$$

Remarquons que :

- si $p, p' \in \mathbb{N}^*$ sont tels que $p' \,|\, p$, alors on a $CB_{p'} \subseteq CB_p$.
- pour tout $p \in \mathbb{N}^*$, on a $CB_d \subseteq CB_p \subseteq (Y \cup CB_d)$ où $d = \mathrm{pgcd}(m, p)$. En effet, la première inclusion $CB_d \subseteq CB_p$ se montre en notant que $d \,|\, p$. Montrons la deuxième inclusion et supposons que $b \in CB_p \setminus Y$. Comme $\max(Y) < \min(B)$ et $b \in (B + m^*)$, on en déduit que $b > \max(Y)$. Par conséquent, $(b + p^*) \subseteq (B + m^*)$ et on déduit du lemme 2.4.21 qu'on a $(b + d^*) \subseteq (B + m^*) \subseteq SL$, d'où $b \in CB_d$.

Pour tout $p \in \mathbb{N}^*$, soit $\widetilde{CB}_p$ le sous-ensemble des candidats de base pour p défini par :

$$\widetilde{CB}_p = CB_p \setminus \left(\bigcup_{p' \,|\, p,\, p' \neq p} CB_{p'} \right)$$

On observe que pour tout $p \in \mathbb{N}^*$, on a $CB_p = \bigcup_{p' \,|\, p} \widetilde{CB}_{p'}$.

Posons $P = \{p \in \mathbb{N}^* \ / \ \widetilde{CB}_p \text{ est infini}\}$. Pour tout $p \in \mathbb{N}^*$, si p ne divise pas m alors $CB_p \subseteq (Y \cup CB_d)$ où $d = \mathrm{pgcd}(m, p)$, et puisque $d \neq p$ on obtient que $\widetilde{CB}_p \subseteq Y$ est fini. Ainsi, P est un ensemble fini, car c'est un sous-ensemble de l'ensemble des diviseurs de m. En posant $Y' = \bigcup_{p \in (\mathbb{N}^* \setminus P)} \widetilde{CB}_p$, on aboutit à :

$$\begin{array}{rcccc} Y' & = & \displaystyle\bigcup_{p \not| m} \widetilde{CB}_p & \bigcup & \displaystyle\bigcup_{p \,|\, m, p \notin P} \widetilde{CB}_p \\ & \subseteq & Y & \bigcup & \displaystyle\bigcup_{p \,|\, m, p \notin P} \widetilde{CB}_p \end{array}$$

et on déduit que Y' est fini, puisque d'une part $\widetilde{CB}_p$ est fini pour $p \notin P$, et d'autre part m admet un nombre fini de diviseurs. Ainsi, on obtient que :

$$\begin{array}{rcl} SL & = & Y \quad \bigcup \quad (B + m^*) \\ & = & Y \quad \bigcup \quad CB_m \\ & = & Y \quad \bigcup \quad \bigcup_{p \in \mathbb{N}^*} CB_p \\ & = & Y \quad \bigcup \quad \bigcup_{p \in \mathbb{N}^*} \widetilde{CB}_p \\ & = & (Y \cup Y') \quad \bigcup \quad \bigcup_{p \in P} \widetilde{CB}_p \\ & = & (Y \cup Y') \quad \bigcup \quad \bigcup_{p \in P} CB_p \end{array}$$

Pour tout $p \in P$, on pose $B_p = \mathrm{Min}(CB_p, \leq_p)$. Observons que B_p est non vide, puisque CB_p est infini. De plus, B_p est nécessairement fini car il admet au plus un élément modulo p. On remarque que pour tout $p \in P$, CB_p est clos supérieurement pour l'ordre partiel $\leq_p$, et par conséquent, on a $CB_p = (B_p + p^*)$.

On définit enfin X par $X = SL \setminus \left(\bigcup_{p \in P} CB_p\right)$. Remarquons que X est fini puisque $X \subseteq (Y \cup Y')$. De plus, on a :

$$\begin{array}{rcl} SL \setminus X & = & \bigcup_{p \in P} CB_p \\ & = & \bigcup_{p \in P} (B_p + p^*) \end{array}$$

Il reste maintenant à vérifier que le couple $(X, (B_p, p)_{p \in P})$ satisfait les trois conditions $ii)$, $iii)$ et $iv)$ de la définition d'une représentation canonique. La condition $ii)$ est satisfaite de par la définition de B_p. La condition $iv)$ est vérifiée puisque pour tout $p \in P$, l'ensemble $\widetilde{CB}_p$ est infini. Montrons à présent que la condition $iii)$ est également satisfaite.

Soit $x \in \mathbb{N}$ et $p \in \mathbb{N}^*$ tels que $(x + p^*) \subseteq SL$. On note que $(x + p^*) \subseteq CB_p$, et par conséquent $(x + p^*) \subseteq \bigcup_{p' \mid p} \widetilde{CB}_{p'}$. Comme $(x + p^*)$ est infini, on en déduit qu'il existe un diviseur $p' \mid p$ tel que $(x + p^*) \cap \widetilde{CB}_{p'}$ est infini. Ainsi, on obtient d'une part que $\widetilde{CB}_{p'}$ est infini et donc $p' \in P$, et d'autre part qu'il existe $y \in (x + p^*)$ tel que $y \in \widetilde{CB}_{p'}$. Puisque $y \in \widetilde{CB}_{p'}$, il vient $y \in CB_{p'}$. Or on a $CB_{p'} = (B_{p'} + p'^*)$, donc on en déduit qu'il existe $b \in B_{p'}$ tel que $y \in (b + p'^*)$. Enfin, comme $p' \mid p$, on aboutit à $(y + p^*) \subseteq (b + p'^*)$. ■

Remarque. La preuve du théorème ci-dessus permet de déduire un algorithme calculant une description canonique d'un sous-ensemble semilinéaire SL de $\mathbb{N}$ (où SL est donné par une description finie). Décrivons brièvement cet algorithme. On commence par calculer une description finie de SL de la forme $SL = Y \cup (B + m^*)$, donnée par l'automate minimal du langage rationnel

$\{1^k \ / \ k \in SL\}$, comme dans la preuve du lemme 2.4.20. Ensuite, pour tout entier p divisant m, on calcule l'ensemble CB_p, représenté finiment par sa base "minimale" $B_p = \mathrm{Min}(CB_p, \leq_p)$. On décide ensuite pour tout diviseur p de m si l'ensemble $\widetilde{CB_p}$ est fini, afin d'en déduire l'ensemble P. On observe que l'ensemble $\widetilde{CB_p}$ est fini si et seulement si pour tout $b \in B_p$, il existe un diviseur $p' \mid p$ et un élément $b' \in B_{p'}$ tels que b et b' sont équivalents modulo p'. Par conséquent, on peut décider pour tout diviseur p de m si l'ensemble $\widetilde{CB_p}$ est fini, et on peut donc ainsi calculer P. On calcule enfin l'ensemble $X = SL \setminus \left(\bigcup_{p \in P} (B_p + p^*)\right)$.

Définition de l'accélération

Nous allons maintenant pouvoir définir l'accélération ∇ pour l'abstraction $\mathcal{A}_{\mathcal{E}}^{\mathcal{R}}$ définie précédemment. Commençons par établir le résultat de semilinéarité de $SL(\mathsf{E}, q, q', \sigma)$, justifiant l'étude précédente des sous-ensembles semilinéaires de $\mathbb{N}$.

Proposition 2.4.23 *Le sous-ensemble $SL(\mathsf{E}, q, q', \sigma)$ de $\mathbb{N}$ est semilinéaire pour tout contrôle d'automate étendu $\mathsf{E} = (Q, \mathtt{Op}, \rightarrow)$, état de contrôle initial $q_0 \in Q$, état de contrôle final $q_f \in Q$ et séquence d'opérations $\sigma \in \mathtt{Op}^+$.*

Preuve. Rappelons que d'après la proposition 2.2.4, la classe des langages SLRE-définissables est close par intersection avec des rationnels. Or σ^* est clairement un langage SLRE-définissable. Donc on obtient qu'il existe un SLRE ρ tel que $L(\mathsf{E}, q, q') \cap \sigma^* = [\![\rho]\!]$. On note que l'ensemble $\{|w| \ / \ w \in [\![\rho]\!]\}$ est un sous-ensemble semilinéaire de $\mathbb{N}$. Or $[\![\rho]\!] \subseteq \sigma^*$, et par conséquent $|w|$ est un multiple de $|\sigma|$ pour tout $w \in [\![\rho]\!]$. On en déduit que le sous-ensemble :

$$SL(\mathsf{E}, q, q', \sigma) \ = \ \left\{ \frac{|w|}{|\sigma|} \ / \ w \in [\![\rho]\!] \right\}$$

est un sous-ensemble semilinéaire de $\mathbb{N}$. ■

Comme on l'a vu précédemment, un sous-ensemble infini semilinéaire de $\mathbb{N}$ admet une infinité de décompositions sous la forme d'unions de sous-ensembles linéaires. Cependant, lorsqu'une période p intervient dans la décomposition de $SL(\mathsf{E}, q, q', \sigma)$, la proposition 2.4.14 incite à choisir p le plus petit possible pour l'ordre partiel de la divisibilité $\mid$. Ceci motive l'intérêt de la décomposition canonique donnée dans la définition 2.4.17.

L'accélération générique essaie essentiellement d'itérer les séquences d'opérations correspondant aux périodes de la décomposition canonique :

Définition 2.4.24 *Etant donné un automate étendu $\mathcal{E}$ de sémantique Δ, soit $\nabla_{\mathcal{E}}^{\mathcal{R}} : R^Q \times \mathtt{Op}^+ \rightarrow R^Q$ la fonction définie par :*

$$\nabla_{\mathcal{E}}^{\mathcal{R}}(\mathsf{a}, \sigma) = \left\langle q \mapsto \sum_{q' \in Q} iter(q', q, \sigma, \mathsf{a}[q']) \right\rangle$$

où pour tout $q, q' \in Q$, $\sigma \in \mathtt{Op}^+$ et $r \in R$, on définit $iter(q, q', \sigma, r)$ par :

$$iter(q,q',\sigma,r) = \sum_{x \in X} \delta_{\mathcal{R}}(\sigma^x, r) \quad + \quad \sum_{i \in I} star(\sigma^{p_i}, \sum_{b \in B_i} \delta_{\mathcal{R}}(\sigma^b, r))$$

avec $(X, (B_i, p_i)_{i \in I})$ description canonique de $SL(\mathsf{E}, q, q', \sigma)$, et où pour tout $\sigma \in \mathtt{Op}^+$ et $r \in R$, le représentant symbolique $star(\sigma, r)$ est défini par :

$$star(\sigma, r) = \begin{cases} \sigma^*_{\mathcal{R}}(r) & \text{si } \sigma^* \text{ est } \mathcal{R}\text{–représentable} \\ r & \text{sinon} \end{cases}$$

Remarque. On peut également se contenter d'une description non canonique du sous-ensemble semilinéaire $SL(\mathsf{E}, q, q', \sigma)$, comme celle donnée par la preuve de la proposition 2.4.23, mais l'accélération obtenue dans ce cas sera "moins complète".

Remarque. La fonction $\nabla^{\mathcal{R}}_{\mathcal{E}}$ est récursive si d'une part, on peut décider si σ^* est $\mathcal{R}$–représentable pour toute séquence d'opérations σ, et si d'autre part, lorsque σ^* est $\mathcal{R}$–représentable, la fonction $\sigma^*_{\mathcal{R}}$ est récursive.

Nous montrons maintenant que la fonction $\nabla^{\mathcal{R}}_{\mathcal{E}}$ est bien une accélération :

Proposition 2.4.25 *Etant donné un automate étendu $\mathcal{E}$ de sémantique Δ, les deux assertions suivantes sont satisfaites :*

1. *la fonction $\nabla^{\mathcal{R}}_{\mathcal{E}}$ est une accélération correcte pour l'abstraction $\mathcal{A}^{\mathcal{R}}_{\mathcal{E}}$.*
2. *Si σ^* est $\mathcal{R}$–représentable pour tout circuit de contrôle $q \xrightarrow{\sigma} q$, alors l'accélération $\nabla^{\mathcal{R}}_{\mathcal{E}}$ est complète.*

Preuve. De par sa définition, il est clair que $\nabla^{\mathcal{R}}_{\mathcal{E}}$ est une accélération correcte pour l'abstraction $\mathcal{A}^{\mathcal{R}}_{\mathcal{E}}$. Supposons maintenant que σ est $\mathcal{R}$–représentable pour tout circuit de contrôle $q \xrightarrow{\sigma} q$. Soit $\sigma \in \mathtt{Op}^+$ une séquence d'opérations et montrons que $\gamma(\nabla(\mathsf{a}, \sigma)) \supseteq \bigcup_{i \in \mathbb{N}} \mathrm{post}(\gamma(\mathsf{a}), \sigma^i)$ pour tout état abstrait $\mathsf{a} \in R^Q$. On remarque qu'il suffit de montrer que pour toute paire d'états de contrôle q, q', et pour représentant symbolique $r \in R$, on a :

$$[\![iter(q,q',\sigma,r)]\!] \quad = \bigcup_{\sigma \in L(\mathsf{E},q,q') \cap \sigma^*} \sigma([\![r]\!])$$

et comme $L(\mathsf{E}, q, q') \cap \sigma^* = \{\sigma^k \ / \ k \in SL(\mathsf{E}, q, q', \sigma)\}$, on obtient qu'il suffit en fait de montrer que :

$$[\![iter(q,q',\sigma,r)]\!] \quad = \bigcup_{k \in SL(\mathsf{E},q,q',\sigma)} (\sigma^k)([\![r]\!])$$

Soit $(X, (B_i, p_i)_{i \in I})$ la description canonique de $SL(\mathsf{E}, q, q', \sigma)$. D'après la définition de $iter(q, q', \sigma, r)$, afin de montrer l'égalité ci-dessus, il suffit de prouver que $(\sigma^{p_i})^*$ est $\mathcal{R}$–représentable pour tout $i \in I$.

Nous démontrons à présent que $(\sigma^{p_i})^*$ est $\mathcal{R}$–représentable pour tout $i \in I$. Le cas où $SL(\mathsf{E}, q, q', \sigma)$ est fini est trivial, car on a alors $I = \emptyset$. Supposons maintenant que $SL(\mathsf{E}, q, q', \sigma)$ est infini. On en déduit qu'il existe nécessairement un circuit de contrôle $q'' \xrightarrow{\sigma^h} q''$ étiqueté par une puissance de σ. En notant que $q'' \xrightarrow{\sigma^{kh}} q''$ est toujours un circuit de contrôle pour tout $k \in \mathbb{N}^*$, et que par conséquent $(\sigma^{kh})^*$ est $\mathcal{R}$–représentable, on déduit de la proposition 2.4.14 que σ^k est $\mathcal{R}$–représentable pour tout $k \in \mathbb{N}^*$. Ainsi, on obtient que $(\sigma^{p_i})^*$ est $\mathcal{R}$–représentable pour tout $i \in I$, ce qui termine la preuve. ■

L'approche décrite au cours de cette section sera entièrement appliquée dans le cadre des automates à files FIFO au chapitre 4, ce qui nous permettra de comparer différentes représentations symboliques FIFO. En revanche, le chapitre suivant utilise des techniques plus complexes d'accélération, afin d'obtenir des résultats de terminaison du calcul symbolique de post* et de pre* sur les systèmes d'addition de vecteurs avec états étendus.

Chapitre 3

Analyse des systèmes d'addition de vecteurs avec états étendus

Nous étudions dans ce chapitre les programmes avec variables entières à valeurs dans $\mathbb{N}$. Ces variables entières, aussi appelées *compteurs*, sont très puissantes malgré leur apparente simplicité. Il suffit en effet de deux compteurs, munis du test-à-zéro, de l'incrément et du décrément, pour permettre à un programme (ici une machine de Minsky) de simuler n'importe quelle machine de Turing.

Les *systèmes d'addition de vecteurs avec états étendus* forment une classe d'automates à compteurs où les opérations sur les compteurs sont les suivantes : incrément, décrément (sous réserve de rester positif ou nul), test-à-zéro, remise-à-zéro et transfert (d'un compteur vers un autre compteur). Cette classe contient les machines de Minsky, et par conséquent le problème de l'accessibilité est indécidable pour cette classe. Nous allons cependant chercher à semi-calculer post^* en utilisant une représentation symbolique basée sur les *sous-ensembles semilinéaires* de vecteurs d'entiers.

Nous allons également nous intéresser plus particulièrement à des sous-classes de systèmes d'addition de vecteurs avec états étendus. Il en effet intéressant de noter que d'une part les SAVE de dimension 2 étendus par des test-à-zéro (i.e. les machines de Minsky) ont la puissance des machines de Turing, et d'autre part les SAVE (non étendus) de dimension 2 ont un ensemble d'états accessibles effectivement semilinéraire [HP79]. Nous décrirons une hiérarchie exhaustive des SAVE étendus de dimension 2, complète vis-à-vis de la décidabilité de l'accessibilité et presque complète (il reste un problème ouvert) vis-à-vis de l'effective semilinéarité de post^* et de pre^*.

Nous commencerons par introduire le modèle des systèmes d'addition de vecteurs avec états étendus en section 3.1, puis nous présenterons le cadre d'abstraction et d'accélération de ce modèle en section 3.2. Nous décrirons un nouvel algorithme calculant un arbre d'accessibilité symbolique dans la section 3.3, puis nous définirons les sous-classes de SAVE étendus de dimension 2 en section 3.4. Les sections 3.5, 3.6, 3.7 et 3.8 seront dévolues au calcul de post^* et de pre^*

pour ces sous-classes. Nous conclurons ce chapitre par des applications de nos résultats à la vérification des SAVE étendus en section 3.9.

3.1 Systèmes d'addition de vecteurs avec états étendus

Nous présentons dans cette section le modèle des systèmes d'addition de vecteurs avec états étendus, que nous allons analyser en détails dans la suite du chapitre.

3.1.1 Vecteurs de nombres

Soit I un ensemble fini d'indices. Nous notons $\mathsf{0}$ le vecteur de $\mathbb{R}^I$ valant zéro à chaque indice (i.e. $\mathsf{0}[i] = 0$ pour tout $i \in I$). Les opérations $+$ et $-$ sur les vecteurs de nombres sont les extensions composante par composante de leur version scalaire. Formellement, si $\oplus \in \{+, -\}$ alors pour tout $\mathsf{x}, \mathsf{y} \in \mathbb{R}^I$, $\mathsf{x} \oplus \mathsf{y}$ est le vecteur défini par $(\mathsf{x} \oplus \mathsf{y})[i] = \mathsf{x}[i] \oplus \mathsf{y}[i]$ pour tout $i \in I$. Ces opérations s'étendent naturellement sur les ensembles de vecteurs de nombres de la façon suivante : si $\oplus \in \{+, -\}$ alors pour tout $X, Y \subseteq \mathbb{R}^I$, $X \oplus Y$ est l'ensemble $\{\mathsf{x} \oplus \mathsf{y} \ / \ \mathsf{x} \in X, \mathsf{y} \in Y\}$. Le produit d'un scalaire $\rho \in \mathbb{R}$ et d'un vecteur x est le vecteur $\rho\mathsf{x}$ défini par $(\rho\mathsf{x})[i] = \rho\mathsf{x}[i]$ pour tout $i \in I$.

La relation binaire $\leq$ sur $\mathbb{R}^I$ est définie par $\mathsf{x} \leq \mathsf{y}$ si pour tout $i \in I$ on a $\mathsf{x}[i] \leq \mathsf{y}[i]$. Cette relation est un ordre partiel sur $\mathbb{R}^I$.

Proposition 3.1.1 ([Dic13]) *L'ordre partiel $\leq$ est un beau pré-ordre sur $\mathbb{N}^I$.*

Ovservons que $\leq$ n'est pas bien fondé sur $\mathbb{Z}^I$, et ce n'est donc pas un beau pré-ordre sur $\mathbb{Z}^I$.

3.1.2 SAVE étendus

Les *systèmes d'addition de vecteurs avec états étendus* permettent de modéliser des systèmes avec *variables entières* à valeur dans $\mathbb{N}$, aussi appelées *compteurs*. Comme nous allons le voir dans la suite de cette section, l'exemple de système à deux compteurs décrit dans la figure 1.1 de la page 14, et dont nous nous sommes servis pour illustrer le premier chapitre, est un système d'addition de vecteurs avec états étendu.

Le modèle de base, les *systèmes d'addition de vecteurs* [KM69] est équivalent au modèle bien connu des *réseaux de Petri* [Pet62]. Ces modèles ne comportent pas (directement) d'états de contrôle, et les seules opérations disponibles sont l'incrément et le décrément (sous réserve de rester positif ou nul) des compteurs. Une variante équivalente obtenue en ajoutant une structure de contrôle finie, les *systèmes d'addition de vecteurs avec états* (*SAVE*), a ensuite été introduite dans [HP79]. De nombreuses restrictions/extensions de ces modèles ont été analysées depuis l'introduction des réseaux de Petri. Une hiérarchie très complète d'extensions de réseaux de Petri a été définie et analysée dans [Duf98].

Nous étudions dans ce chapitre une extension des SAVE, obtenue en rajoutant la possibilité de tester à zéro un compteur, de remettre un compteur à zéro (on parle aussi de *reset*) et de transférer le contenu d'un compteur dans un autre. Cette extension est donc une sous-classe de la classe des *G-réseaux* de [Duf98], cependant nous allons ici plus particulièrement étudier les SAVE étendus à deux compteurs.

Les SAVE étendus forment une sous-classe de la classe des *automates étendus*, définie et analysée dans la section 2.4 du premier chapitre. Un SAVE étendu $\mathcal{V}$ est donc défini comme une sémantique opérationnelle (i.e. un système de transitions étiqueté) associée à une structure de contrôle V finie, cette dernière décrivant les opérations à effectuer sur les compteurs lors du passage d'un état de contrôle à l'autre.

Etant donné un ensemble fini C de compteurs, les opérations que nous considérons se décomposent de la façon suivante :

– les opérations usuelles d'*addition* : $\mathtt{A}(C) = \{\mathtt{add}(\mathsf{v}) \;/\; \mathsf{v} \in \mathbb{Z}^C\}$. Les opérations d'addition permettent d'incrémenter et de décrémenter les compteurs. Plusieurs compteurs peuvent être affectés par une unique opération d'addition.

– les *opérations étendues* :

$$\begin{array}{rcll} \mathtt{T}(C) & = & \{\mathtt{test}(c) \;/\; c \in C\} & (\textit{test-à-zéro}) \\ \mathtt{WT}(C) & = & \{\mathtt{weaktest}(c) \;/\; c \in C\} & (\textit{test-à-zéro faible}) \\ \mathtt{R}(C) & = & \{\mathtt{reset}(c) \;/\; c \in C\} & (\textit{remise-à-zéro}) \\ \mathtt{Tr}(C) & = & \{\mathtt{transfer}(c \to c') \;/\; c, c' \in C, c \neq c'\} & (\textit{transfert}) \end{array}$$

Comme on le voit d'après leur définition, chaque opération étendue agit sur un seul compteur, excepté les opérations de transfert qui prennent en argument deux compteurs distincts.

Notation. Pour tout ensemble fini C de compteurs, on note $\mathtt{TOp}(C)$ l'ensemble de toutes les opérations disponibles sur les compteurs de C :

$$\mathtt{TOp}(C) = \mathtt{A}(C) \cup \mathtt{T}(C) \cup \mathtt{WT}(C) \cup \mathtt{R}(C) \cup \mathtt{Tr}(C)$$

La structure de contrôle d'un SAVE étendu est définie comme suit :

Définition 3.1.2 *Un* contrôle de système d'addition de vecteurs avec états étendu *est un quadruplet* $\mathsf{V} = (Q, C, \mathtt{Op}, \to)$*, où :*

– Q *est un ensemble fini d'*états de contrôle*, et*

– C *est un ensemble fini de* compteurs*, et*

– *le sous-ensemble fini* $\mathtt{Op} \subseteq \mathtt{TOp}(C)$ *est l'alphabet d'*opérations*, et*

– *l'ensemble* $\to \subseteq Q \times \mathtt{Op} \times Q$ *est un ensemble fini de* transitions de contrôle.

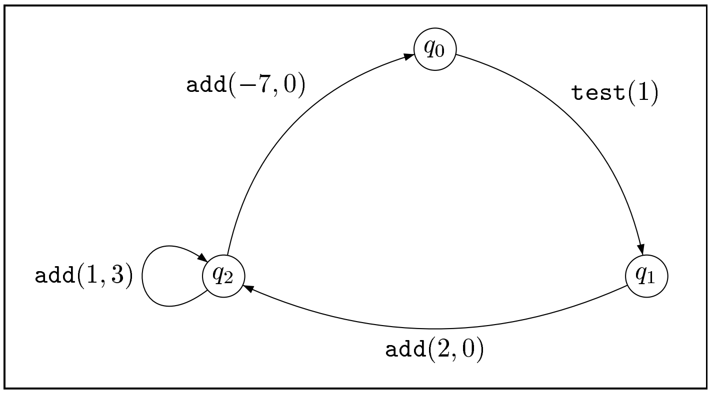


FIG. 3.1 – Le contrôle de SAVE étendu EV

Observons qu'un contrôle de SAVE étendu $\mathsf{V} = (Q, C, \mathtt{Op}, \rightarrow)$ définit implicitement le contrôle d'automate étendu $(Q, \mathtt{Op}, \rightarrow)$.

Exemple 3.1.3 Nous avons traduit en figure 3.1 la structure de contrôle de l'exemple de système à deux compteurs décrit dans la figure 1.1 de la page 14. Le contrôle de SAVE étendu EV décrit en figure 3.1 est formellement défini par $\mathsf{EV} = (Q, C, \mathtt{Op}, \rightarrow_{\mathsf{EV}})$, où :

- $Q = \{q_0, q_1, q_2\}$ est l'ensemble des états de contrôle, et
- $C = \{1, 2\}$ est l'ensemble des compteurs, et
- $\mathtt{Op} = \{\mathtt{test}(1), \mathtt{add}(2,0), \mathtt{add}(-7,0), \mathtt{add}(1,3)\}$ est l'alphabet d'opérations, et
- l'ensemble $\rightarrow_{\mathsf{EV}} \subseteq Q \times (C \times \{?, !\} \times M) \times Q$ des transitions de contrôle est donné par :

$$\rightarrow_{\mathsf{EV}} \; = \left\{ \begin{array}{l} (q_0,\ \mathtt{test}(1)\ , q_1)\,, \\ (q_1,\ \mathtt{add}(2,0)\ , q_2)\,, \\ (q_2, \mathtt{add}(-7,0)\,, q_0)\,, \\ (q_2,\ \mathtt{add}(1,3)\ , q_2) \end{array} \right\}$$

Remarque. Dans toute la suite de ce chapitre, nous désignerons par C un ensemble fini de compteurs. L'alphabet d'opérations $\mathtt{Op}$ sera toujours un sous-ensemble fini de $\mathtt{TOp}(C)$, et les séquences d'opérations que nous considérerons seront dans $\mathtt{Op}^*$.

Nous supposons que les compteurs contiennent des valeurs entières positives ou nulles. La signification formelle des différentes opérations sur l'ensemble C des compteurs est donnée par la définition suivante :

Définition 3.1.4 *La* sémantique de compteurs*, notée $\Delta_{\mathbb{N}}$, d'un alphabet d'opérations $\mathtt{Op} \subseteq \mathtt{TOp}(C)$ est le triplet $\Delta_{\mathbb{N}} = (C, \mathbb{N}, \delta_{\mathbb{N}})$, où la fonction $\delta_{\mathbb{N}}$ de $\mathtt{Op}$ dans*

$\mathcal{P}(\mathbb{N}^C \times \mathbb{N}^C)$ *est définie par :*

$$\begin{array}{llll}
(\mathsf{x},\mathsf{x}') & \in \delta_{\mathbb{N}}(\mathtt{add}(\mathsf{v})) & si & \mathsf{x}' = \mathsf{x}+\mathsf{v} \\
(\mathsf{x},\mathsf{x}') & \in \delta_{\mathbb{N}}(\mathtt{test}(c)) & si & \mathsf{x}[c] = 0 \ \ et\ \mathsf{x}' = \mathsf{x} \\
(\mathsf{x},\mathsf{x}') & \in \delta_{\mathbb{N}}(\mathtt{weaktest}(c)) & si & \mathsf{x}[c] = 0 \ \ et\ \exists\ y \in \mathbb{N}, \mathsf{x}' = \mathsf{x}[c := y] \\
(\mathsf{x},\mathsf{x}') & \in \delta_{\mathbb{N}}(\mathtt{reset}(c)) & si & \mathsf{x}' = \mathsf{x}[c := 0] \\
(\mathsf{x},\mathsf{x}') & \in \delta_{\mathbb{N}}(\mathtt{transfer}(c \to c')) & si & \mathsf{x}' = \mathsf{x}[c := 0][c' := (\mathsf{x}[c]+\mathsf{x}[c'])]
\end{array}$$

Le sens intuitif de chaque opération se déduit d'après leurs noms respectifs, et la définition de la fonction $\delta_{\mathbb{N}}$ correspond à cette idée intuitive. Obervons tout de même que si $(\mathsf{x},\mathsf{x}') \in \delta_{\mathbb{N}}(\mathtt{add}(\mathsf{v}))$ alors on a obligatoirement $\mathsf{x}+\mathsf{v} \geq \mathsf{0}$. Ainsi, une opération d'addition permet de tester si un compteur est plus grand qu'un vecteur de nombres donné.

Remarque. Dans toute la suite de ce chapitre, la sémantique utilisée sera la sémantique de compteurs.

Un contrôle de système d'addition de vecteurs avec états étendu V, vu comme un contrôle d'automate étendu, définit l'automate étendu $\mathcal{V}$ conformément à la sémantique $\Delta_{\mathbb{N}}$ (voir la définition 2.4.5 de la page 54). L'automate étendu $\mathcal{V}$, qui est aussi appelé *système d'addition de vecteurs avec états étendu*, décrit ainsi la sémantique opérationnelle du contrôle de SAVE étendu V. Redonnons la définition de l'automate étendu $\mathcal{V}$ dans le contexte des système d'addition de vecteurs avec états étendus :

Définition 3.1.5 *Un contrôle de SAVE étendu* $\mathsf{V} = (Q, C, \mathtt{Op}, \to_{\mathsf{V}})$ *définit le système de transitions étiqueté* $\mathcal{V} = (S, \Sigma, \to_{\mathcal{V}})$*, appelé* système d'addition de vecteurs avec états étendu*, comme suit :*

– $S = Q \times \mathbb{N}^C$ *est l'ensemble des états, et*

– $\Sigma = \mathtt{Op}$ *est l'alphabet d'opérations, et*

– *la relation de transitions étiquetée* $\to_{\mathcal{V}}$ *est le plus petit sous-ensemble* $\to_{\mathcal{V}}$

de $\subseteq S \times \Sigma \times S$ *satisfaisant pour tout* $\mathsf{x} \in \mathbb{N}^C$ *les cinq conditions suivantes :*

$$
\begin{array}{llll}
i) & si & q \xrightarrow{\mathtt{add}(\mathsf{v})}_{\mathsf{V}} q' & et \quad \mathsf{x}+\mathsf{v} \geq \mathsf{0} \\
 & alors & (q,\mathsf{x}) \xrightarrow{\mathtt{add}(\mathsf{v})}_{\mathcal{V}} (q',\mathsf{x}+\mathsf{v}) & \\[1em]
ii) & si & q \xrightarrow{\mathtt{test}(c)}_{\mathsf{V}} q' & et \quad \mathsf{x}[c]=0 \\
 & alors & (q,\mathsf{x}) \xrightarrow{\mathtt{test}(c)}_{\mathcal{V}} (q',\mathsf{x}) & \\[1em]
iii) & si & q \xrightarrow{\mathtt{weaktest}(c)}_{\mathsf{V}} q' & et \quad \mathsf{x}[c]=0 \\
 & alors & (q,\mathsf{x}) \xrightarrow{\mathtt{weaktest}(c)}_{\mathcal{V}} (q',\mathsf{x}[c:=y]) \ \textit{pour tout}\ y \in \mathbb{N} & \\[1em]
iv) & si & q \xrightarrow{\mathtt{reset}(c)}_{\mathsf{V}} q' & \\
 & alors & (q,\mathsf{x}) \xrightarrow{\mathtt{reset}(c)}_{\mathcal{V}} (q',\mathsf{x}[c:=0]) & \\[1em]
v) & si & q \xrightarrow{\mathtt{transfer}(c\to c')}_{\mathsf{V}} q' & \\
 & alors & (q,\mathsf{x}) \xrightarrow{\mathtt{transfer}(c\to c')}_{\mathcal{V}} (q',\mathsf{x}[c:=0][c':=(\mathsf{x}[c]+\mathsf{x}[c'])]) &
\end{array}
$$

Ainsi, un état (q,x) d'un SAVE étendu est composé d'un *état de contrôle* q et d'un vecteur d'entiers $\mathsf{x} \in \mathbb{N}^C$ représentant le contenu des compteurs. Les deux conditions de franchissement d'une transition suivent la sémantique de compteurs. Observons qu'une transition étiquetée par $\mathtt{add}(\mathsf{0})$ se comporte comme une "ε-transition", car elle est toujours franchissable et ne modifie pas la valeur des compteurs. Un *système d'addition de vecteurs avec états* (non étendu) est un SAVE étendu, dont l'alphabet d'opérations est inclus dans $\mathtt{A}(C)$.

Exemple 3.1.6 Dans le SAVE étendu $\mathcal{EV}$ associé au contrôle de SAVE étendu EV, on a par exemple le chemin suivant :

$$
\begin{array}{ccccccc}
(q_0,(0,0)) & \xrightarrow{\mathtt{test}(1)} & (q_1,(0,0)) & \xrightarrow{\mathtt{add}(2,0)} & (q_2,(2,0)) & \xrightarrow{\mathtt{add}(1,3)} & (q_2,(3,3)) \\
 & & & & & & \big\downarrow{\scriptstyle (\mathtt{add}(1,3))^4} \\
 & & (q_1,(0,15)) & \xleftarrow{\mathtt{test}(1)} & (q_0,(0,15)) & \xleftarrow{\mathtt{add}(-7,0)} & (q_2,(7,15))
\end{array}
$$

Remarque. L'opération $\mathtt{weaktest}(c)$ est "le dual" de l'opération $\mathtt{reset}(c)$ dans la mesure où $(q,\mathsf{x}) \xrightarrow{\mathtt{weaktest}(c)} (q',\mathsf{x}')$ si et seulement si $(q',\mathsf{x}') \xrightarrow{\mathtt{reset}(c)} (q,\mathsf{x})$. Nous utiliserons les opérations de test-à-zéro faibles dans la section 3.6 afin de calculer pre^* dans les SAVE étendus par des remises-à-zéro.

Notation. La *dimension* d'un contrôle de SAVE étendu $\mathsf{V} = (Q, C, \mathtt{Op}, \to)$ est $|C|$, c'est-à-dire le nombre de compteurs de V. La *dimension* d'un SAVE étendu est la dimension de son contrôle.

3.2 Abstraction et accélération des SAVE étendus

3.2.1 Calcul de l'ensemble de couverture

Nous montrons dans cette sous-section comment appliquer aux SAVE étendus le cadre général d'abstraction et d'accélération de la clôture inférieure d'un système de transitions étiqueté pré-ordonné, décrit dans la sous-section 2.3.1 du premier chapitre.

L'ensemble des états $S = Q \times \mathbb{N}^C$ d'un système d'addition de vecteurs avec états étendu $\mathcal{V}$ est naturellement partiellement ordonné par la relation binaire $\preceq \subseteq S \times S$ définie par $(q, \mathsf{x}) \preceq (q', \mathsf{x}')$ si $q = q'$ et $\mathsf{x} \leq \mathsf{x}'$. Le couple $(\mathcal{V}, \preceq)$ est ainsi un système de transitions étiqueté pré-ordonné. Comme $\leq$ est un beau pré-ordre sur $\mathbb{N}^C$, l'ordre partiel $\preceq$ sur S est un beau pré-ordre sur S.

Rappelons que l'ensemble de couverture d'un SAVE étendu $\mathcal{V}$ muni d'un ensemble X d'états initiaux est $\downarrow\mathrm{post}^*(\mathcal{V}, X)$. Nous avons proposé en section 2.3 un cadre général d'abstraction et d'accélération pour le calcul de l'ensemble de couverture. L'idée consiste à considérer plutôt la clôture inférieure $\downarrow\mathcal{V}$ et à calculer $\mathrm{post}^*(\downarrow\mathcal{V}, X)$, ce qui donne une approximation supérieure de l'ensemble de couverture. Lorsque le SAVE étendu $\mathcal{V}$ est compatible en avant, on a même l'égalité $\downarrow\mathrm{post}^*(\mathcal{V}, X) = \mathrm{post}^*(\downarrow\mathcal{V}, X)$. Remarquons que la clôture inférieure $\downarrow\mathcal{V}$ d'un SAVE étendu $\mathcal{V}$ correspond précisément à la sémantique "avec pertes" de $\mathcal{V}$, introduite dans [BM99].

Remarque. Observons qu'un état s appartient à $\downarrow\mathrm{post}^*(\mathcal{V}, X)$ si et seulement si $X \cap \mathrm{pre}^*(\mathcal{V}, \uparrow s) \neq \emptyset$. Ainsi, pour décider si un état s appartient à $\downarrow\mathrm{post}^*(\mathcal{V}, X)$, on peut aussi choisir de calculer $\mathrm{pre}^*(\mathcal{V}, \uparrow s)$ afin de tester s'il a une intersection vide avec X. On considère alors plutôt la clôture supérieure $\uparrow\overline{\mathcal{V}}$ du dual de $\mathcal{V}$, et on calcule $\mathrm{post}^*(\uparrow\overline{\mathcal{V}}, s) = \mathrm{pre}^*(\downarrow\mathcal{V}, s)$, ce qui donne une approximation supérieure de $\mathrm{pre}^*(\mathcal{V}, \uparrow s)$. Lorsque le SAVE étendu $\mathcal{V}$ est compatible en avant, on a même l'égalité $\mathrm{pre}^*(\mathcal{V}, \uparrow s) = \mathrm{post}^*(\uparrow\overline{\mathcal{V}}, s)$. Observons que $\overline{\mathcal{V}}$ admet clairement une base supérieure post finie. Comme $\preceq$ est un beau pré-ordre sur l'ensemble des états, on déduit du théorème 2.3.7 qu'une base (finie) de $\mathrm{post}^*(\uparrow\overline{\mathcal{V}}, s)$ est calculable.

Représentation des sous-ensembles d'états clos inférieurement

L'abstraction $\mathcal{A}^b_{\downarrow\mathcal{V}}$ que nous avons présentée en sous-section 2.3.1 est basée sur les sous-ensembles d'états clos inférieurement et de base finie. Cependant, l'ensemble de couverture d'un SAVE étendu n'admet pas nécessairement de base finie. Nous allons donc utiliser les *contraintes closes inférieurement* de [BM99], sous une forme légèrement différente, afin de représenter finiment n'importe quel sous-ensemble d'états clos inférieurement d'un SAVE étendu.

On désigne par $\mathbb{N}_\omega$ (resp. $\mathbb{Z}_\omega$) la fermeture classique $\mathbb{N} \cup \{\omega\}$ (resp. $\mathbb{Z} \cup \{\omega\}$) de l'ensemble $\mathbb{N}$ (resp. $\mathbb{Z}$), ou $v < \omega$ pour tout $v \in \mathbb{Z}$. L'addition est étendue sur $\mathbb{Z}_\omega$ par $v + \omega = \omega + v = \omega$ pour $v \in \mathbb{Z}_\omega$. L'addition sur les vecteurs dans $\mathbb{Z}_\omega^C$ est l'extension composante par composante de l'addition dans $\mathbb{Z}_\omega$. L'ordre partiel $\leq$ sur $\mathbb{Z}^C$ (resp. $\preceq$ sur $Q \times \mathbb{N}^C$) se prolonge naturellement sur $\mathbb{Z}_\omega^C$ (resp. sur $Q \times \mathbb{N}_\omega^C$), tout en restant un ordre partiel.

Dans différents papiers traitant de la couverture des SAVE [KM69, Fin93], l'élément ω est vu comme une limite de suite croissante d'entiers. Cette démarche oblige alors à voir l'ensemble de couverture comme l'ensemble des limites de suites croissantes d'états. Notre définition d'ensemble de couverture est bien plus simple, et le fait qu'un ensemble de couverture soit clos inférieurement incite à voir l'élément ω plutôt comme l'ensemble $\mathbb{N}$ (clos inférieurement) de tous les entiers positifs ou nuls. Formellement,

Définition 3.2.1 *La fonction* $[\![\cdot]\!] : \mathbb{N}_\omega \to \mathcal{P}(\mathbb{N})$ *est définie par :*

$$[\![x]\!] = \begin{cases} \downarrow x & si\ x \neq \omega \\ \mathbb{N} & si\ x = \omega \end{cases}$$

Nous étendons cette fonction sur $\mathbb{N}_\omega^C$ *et sur* $Q \times \mathbb{N}_\omega^C$ *comme suit :*

$$\begin{aligned} [\![\mathsf{x}]\!] &= \prod_{c \in C} [\![\mathsf{x}[c]]\!] \\ [\![(q, \mathsf{x})]\!] &= \{q\} \times [\![\mathsf{x}]\!] \end{aligned}$$

On remarque que pour tout $(q, \mathsf{x}) \in Q \times \mathbb{N}_\omega^C$, le sous-ensemble d'états $[\![(q, \mathsf{x})]\!]$ est clos inférieurement. Nous savons d'après [BM99] que tout sous-ensemble clos inférieurement d'états s'écrit comme une union finie de sous-ensembles de la forme $[\![(q, \mathsf{x})]\!]$. Ceci nous conduit à la définition et à la proposition suivantes :

Définition 3.2.2 *Une* ω-base *d'un sous-ensemble* $X \subseteq Q \times \mathbb{N}^C$ *clos inférieurement est un sous-ensemble fini* $B \subseteq Q \times \mathbb{N}_\omega^C$ *telle que* $X = \bigcup_{b \in B} [\![b]\!]$. *On dit que* B *est une* ω-base minimale *de* X *si aucun sous-ensemble strict de* B *n'est une* ω*-base de* X.

Proposition 3.2.3 *Tout sous-ensemble d'états clos inférieurement admet une unique* ω*-base minimale.*

Preuve. L'existence d'une ω-base se déduit de [BM99]. Une ω-base peut être minimisée en ne gardant que ses éléments maximaux (pour l'ordre partiel $\preceq$ sur $Q \times \mathbb{N}_\omega^C$). Montrons maintenant l'unicité d'une ω-base minimale. Supposons que B et B' sont deux ω-bases minimales d'un même sous-ensemble $X \subseteq Q \times \mathbb{N}^C$ clos inférieurement.

Montrons tout d'abord que pour tout $b \in B$ il existe $b' \in B'$ tel que $b \preceq b'$. Soit donc $b \in B$ et soit $k \in \mathbb{N}$ un entier strictement supérieur à tous les entiers apparaissant dans les vecteurs de B'. Notons b_k le vecteur d'entiers obtenu à partir de b en remplaçant tous les ω par k. Comme $b_k \in [\![b]\!]$, on en déduit que $b_k \in X$ et par conséquent il existe $b' \in B'$ tel que $b_k \in [\![b']\!]$. Comme k est strictement supérieur à tous les entiers apparaissant dans b', on en déduit que pour tout $c \in C$, si $b_k[c] = k$ alors $b'[c] = \omega$. On en conclut donc que $b \preceq b'$.

Par l'absurde, supposons maintenant que $B \neq B'$. Alors sans perte de généralité, on peut supposer qu'il existe $b \in B$ tel que $b \notin B'$. D'après le raisonnement ci-dessus, on en déduit qu'il existe $b' \in B'$ tel que $b \preceq b'$. Par symétrie, on obtient qu'il existe $b'' \in B$ tel que $b' \preceq b''$. Comme $b \neq b'$, il vient $b \prec b''$, ce qui contredit l'hypothèse de minimalité de B. ∎

Remarque. La notion d'ω-base minimale est précisément ce qu'il y a derrière la notion d'ensemble de couverture minimal dans [Fin90, Fin93]. Plus exactement, étant donné un sous-ensemble d'états $X \subseteq Q \times \mathbb{N}^C$ clos inférieurement, notons $\lim(X)$ l'ensemble des limites (dans $Q \times \mathbb{N}_\omega^C$) de suites croissantes (pour $\preceq$) d'éléments de X. Avec cette notation, on vérifie que $\mathrm{Max}(\lim(X))$ est l'unique ω-base minimale de X. Par conséquent, l'ensemble des éléments maximaux de $\lim(\downarrow\mathrm{post}^*(\mathcal{V}, X))$, qui est appelé *ensemble de couverture minimal* dans [Fin90, Fin93], est aussi l'unique ω-base minimale de $\downarrow\mathrm{post}^*(\mathcal{V}, X)$.

Exemple 3.2.4 Considérons le SAVE étendu d'exemple $\mathcal{EV}$ muni d'un unique état initial $s_0 = (q_0, (0, 0))$. L'ensemble de couverture $\downarrow\mathrm{post}^*(\mathcal{EV}, s_0)$ a pour ω-base minimale $\{(q_0, (\omega, \omega)), (q_1, (0, \omega)), (q_2, (\omega, \omega))\}$.

Abstraction et accélération de $\downarrow\mathcal{V}$

Maintenant que tout sous-ensemble clos inférieurement peut être représenté finiment par une ω-base, nous pouvons définir une abstraction, basée sur les ω-bases, de la clôture inférieure $\downarrow\mathcal{V}$ d'un SAVE étendu $\mathcal{V}$. L'intérêt de cette abstraction est qu'elle capture tous les sous-ensembles d'états clos inférieurement. Pour simplifier la présentation, nous ne détaillons que le cas où l'ensemble d'opérations de $\mathcal{V}$ est inclus dans $\mathtt{A}(C) \cup \mathtt{T}(C)$.

Soit $\mathcal{V}$ un SAVE étendu de contrôle $\mathsf{V} = (Q, C, \mathtt{Op}, \rightarrow)$ d'ensemble d'opérations $\mathtt{Op} \subseteq \mathtt{A}(C) \cup \mathtt{T}(C)$. L'abstraction de $\mathcal{V}$ construite à partir des ω-bases est définie comme suit :

Définition 3.2.5 *Soit* $\mathcal{A}^\omega_{\downarrow\mathcal{V}} = (A, \mathrm{post}_{\mathcal{A}^\omega}, \gamma, \sqcup, \Sigma \cup \{\downarrow\})$ *le quintuplet où l'ensemble* A*, et les fonctions* $\mathrm{post}_{\mathcal{A}^\omega} : A \times \Sigma \rightarrow A$*,* $\gamma : A \rightarrow \mathcal{P}(Q \times \mathbb{N}^C)$ *et* $\sqcup : A \times A \rightarrow A$ *sont définis par :*

$$
\begin{array}{rcl}
A & = & \mathcal{P}_f(\mathbb{N}_\omega^C)^Q \\
\mathrm{post}_{\mathcal{A}^\omega}(\mathsf{a}, \downarrow) & = & \mathsf{a} \\
\mathrm{post}_{\mathcal{A}^\omega}(\mathsf{a}, \mathtt{add}(\mathsf{v})) & = & \left\langle q \mapsto \displaystyle\bigcup_{q' \xrightarrow{\mathtt{add}(\mathsf{v})} q} \{\mathsf{x} + \mathsf{v} \ / \ \mathsf{x} \in \mathsf{a}[q'], \mathsf{x} + \mathsf{v} \geq 0\} \right\rangle \\
\mathrm{post}_{\mathcal{A}^\omega}(\mathsf{a}, \mathtt{test}(c)) & = & \left\langle q \mapsto \displaystyle\bigcup_{q' \xrightarrow{\mathtt{test}(c)} q} \{\mathsf{x}[c := 0] \ / \ \mathsf{x} \in \mathsf{a}[q']\} \right\rangle \\
\gamma(\mathsf{a}) & = & \displaystyle\bigcup_{q \in Q} \bigcup_{\mathsf{x} \in \mathsf{a}[q]} [\![(q, \mathsf{x})]\!] \\
\mathsf{a} \sqcup \mathsf{b} & = & \langle q \mapsto \mathsf{a}[q] \cup \mathsf{b}[q] \rangle
\end{array}
$$

Proposition 3.2.6 *$\mathcal{A}^{\omega}_{\downarrow\mathcal{V}}$ est une abstraction effective de $\downarrow\mathcal{V}$.*

Observons que la fonction de “post abstrait” $\mathrm{post}_{\mathcal{A}^\omega}$ est croissante (pour le pré-ordre $\sqsubseteq$ associé à $\mathcal{A}^{\omega}_{\downarrow\mathcal{V}}$). On en déduit que l'abstraction $\mathcal{A}^{\omega}_{\downarrow\mathcal{V}}$ satisfait les deux conditions permettant la définition d'une accélération générique, comme décrit à la fin de la sous-section 2.1.3. On réalise facilement que cette accélération générique, dans le cas des SAVE (non étendus), correspond exactement à la stratégie de Karp et Miller pour le calcul d'un arbre de couverture [KM69].

3.2.2 Abstraction et accélération par ensembles semilinéaires

Ensembles semilinéaires, cônes et pré-ordres semilinéaires

Soit $P = \{\mathsf{p}_1, \mathsf{p}_2, \cdots, \mathsf{p}_k\}$ un sous-ensemble fini (de *périodes*) de $\mathbb{N}^C$. Nous utiliserons les notations suivantes :

$$\begin{array}{rcl} P^* & = & \left\{\sum_{i=1}^{k} m_i\, \mathsf{p}_i \ / \ \forall\, i \in [1\,..\,k], m_i \in \mathbb{N}\right\} \\ \widehat{P} & = & \left\{\sum_{i=1}^{k} \rho_i\, \mathsf{p}_i \ / \ \forall\, i \in [1\,..\,k], \rho_i \in \mathbb{Q}^+\right\} \cap \mathbb{N}^C \end{array}$$

Pour toute *base* $\mathsf{x} \in \mathbb{N}^C$, on appelle $(\mathsf{x} + P^*)$ un sous-ensemble *linéaire*, et on dit que $(\mathsf{x} + \widehat{P})$ est un *cône*. Un sous-ensemble *semilinéaire* de $\mathbb{N}^C$ est une union finie de sous-ensembles linéaires de $\mathbb{N}^C$. Une *description finie* d'un ensemble semilinéaire SL est une famille finie $(\mathsf{x}_i, P_i)_{i\in I}$, où $\mathsf{x}_i \in \mathbb{N}^C$ et $P_i \in \mathcal{P}_f(\mathbb{N}^C)$, et telle que $SL = \bigcup_{i\in I}(\mathsf{x}_i + P_i{}^*)$. On dit qu'un ensemble semilinéaire SL est *constructible* lorsqu'on peut donner *explicitement* un algorithme calculant une description finie de SL.

Observons que l'ensemble $SL = ((\mathsf{x} + P^*) + \mathsf{v}) \cap \mathbb{N}^C$ est semilinéaire pour tout $\mathsf{x} \in \mathbb{N}^C$ et $\mathsf{v} \in \mathbb{Z}^C$, car il peut s'écrire sous la forme $SL = (B + P^*)$ où B est l'ensemble fini donné par $B = \mathrm{Min}(((\mathsf{x} + P^*) + \mathsf{v}) \cap \mathbb{N}^C)$. De plus, on peut montrer que B est calculable à partir de x, P et v [HP79].

Nous définissons deux relations binaires, $\leq_P$ sur $\mathbb{N}^C$ et $\sqsubseteq_P$ sur $\mathcal{P}(\mathbb{N}^C)$, comme suit :

$$\begin{array}{rcl} \mathsf{x} \leq_P \mathsf{y} & \text{si} & \mathsf{y} \in (\mathsf{x} + P^*) \\ X \sqsubseteq_P Y & \text{si} & \forall\, \mathsf{y} \in Y, \exists\, \mathsf{x} \in X, \mathsf{x} \leq_P \mathsf{y} \end{array}$$

On se rend aisément compte que $\leq_P$ (resp. $\sqsubseteq_P$) est un pré-ordre sur $\mathbb{N}^C$ (resp. sur $\mathcal{P}(\mathbb{N}^C)$). D'après la définition de $\sqsubseteq_P$, on obtient la caractérisation suivante : pour tout $X, Y \in \mathcal{P}(\mathbb{N}^C)$, on a $X \sqsubseteq_P Y$ si et seulement si $Y \subseteq (X + P^*)$.

Proposition 3.2.7 *Le pré-ordre $\sqsubseteq_P$ est un beau pré-ordre sur l'ensemble des parties de $\widehat{P}$.*

Preuve. Soit B le sous-ensemble fini de $\mathbb{N}^C$ défini par :

$$B = \left\{\sum_{i=1}^{k} \rho_i\, \mathsf{p}_i \ / \ \forall\, i \in [1\,..\,k], \rho_i \in \mathbb{Q}^+, \rho_i < 1\right\} \cap \mathbb{N}^C$$

Observons que pour tout $\mathsf{x} \in \widehat{P}$, il existe $\mathsf{b} \in B$ et $(m_1, m_2, \cdots m_k) \in \mathbb{N}^k$ tels que $\mathsf{x} = \mathsf{b} + \sum_{i=1}^{k} m_i \mathsf{p}_i$. Par conséquent, il existe une fonction $f : \widehat{P} \to B \times \mathbb{N}^k$ satisfaisant la condition suivante : pour tout $\mathsf{x} \in \widehat{P}$, on a $\mathsf{x} = \mathsf{b} + \sum_{i=1}^{k} \mathsf{m}[i]\mathsf{p}_i$ où $f(\mathsf{x}) = (\mathsf{b}, \mathsf{m})$.

On étend sur $B \times \mathbb{N}^k$ le pré-ordre habituel $\leq$ sur $\mathbb{N}^k$ comme suit : pour tout $(\mathsf{b}, \mathsf{m}), (\mathsf{b}', \mathsf{m}') \in B \times \mathbb{N}^k$, on a $(\mathsf{b}, \mathsf{m}) \leq (\mathsf{b}', \mathsf{m}')$ si $\mathsf{b} = \mathsf{b}'$ et $\mathsf{m} \leq \mathsf{m}'$. On se rend compte que pour tout $\mathsf{x}, \mathsf{y} \in \widehat{P}$, si $f(\mathsf{x}) \leq f(\mathsf{y})$ alors $\mathsf{x} \leq_P \mathsf{y}$.

On définit également un pré-ordre $\sqsubseteq$ sur $\mathcal{P}(B \times \mathbb{N}^k)$ comme suit : pour tout $Z, Z' \subseteq B \times \mathbb{N}^k$, on a $Z \sqsubseteq Z'$ si $\forall\, (\mathsf{b}', \mathsf{m}') \in Z', \exists\, (\mathsf{b}, \mathsf{m}) \in Z, (\mathsf{b}, \mathsf{m}) \leq (\mathsf{b}', \mathsf{m}')$. Remarquons que pour tout $X, Y \subseteq \widehat{P}$, si $f(X) \sqsubseteq f(Y)$ alors $X \sqsubseteq_P Y$.

Comme B est fini et puisque $\leq$ est un ordre total sur $\mathbb{N}$, on déduit des résultats de [Jan99] que $\sqsubseteq$ est un beau pré-ordre sur $\mathcal{P}(B \times \mathbb{N}^k)$. Maintenant, considérons une suite infinie $(X_i)_{i \in \mathbb{N}}$ de sous-ensembles de $\widehat{P}$. Comme $\sqsubseteq$ est un beau pré-ordre sur $\mathcal{P}(B \times \mathbb{N}^k)$, il existe une sous-suite infinie $(X_{i_j})_{j \in \mathbb{N}}$ telle que $(f(X_{i_j}))_{j \in \mathbb{N}}$ est croissante pour $\sqsubseteq$. Par conséquent, on obtient que $(X_{i_j})_{j \in \mathbb{N}}$ est croissante pour $\sqsubseteq_P$ et on en conclut donc que $\sqsubseteq_P$ est un beau pré-ordre sur l'ensemble des parties de $\widehat{P}$. ■

Corollaire 3.2.8 *Le pré-ordre $\leq_P$ est un beau pré-ordre sur $\widehat{P}$.*

Abstraction et accélération

Les sous-ensembles semilinéaires de $\mathbb{N}^C$ forment une classe très riche de sous-ensembles de $\mathbb{N}^C$, avec de bonnes propriétés de clôture. Par exemple, tout sous-ensemble clos inférieurement de $\mathbb{N}^C$ est semilinéaire. Les sous-ensembles semilinéaires de $\mathbb{N}^C$ sont également les ensembles définissables dans l'arithmétique de Pressburger, et on peut aussi les voir comme les sous-ensembles rationnels de $\mathbb{N}^C$. Il est naturel de les utiliser pour représenter symboliquement des ensembles infinis de contenus de compteurs [HP79, BW94, FO97a, CJ98], car les sous-ensembles semilinéaires sont effectivement clos par la plupart des opérations considérées habituellement sur les compteurs.

Notation. Nous étendons naturellement les notions de sous-ensembles linéaires et semilinéaires de $\mathbb{N}^C$ aux sous-ensembles d'états de la façon suivante. Un sous-ensemble $X \subseteq Q \times \mathbb{N}^C$ d'états d'un SAVE étendu est dit *linéaire* lorsqu'il existe un état de contrôle q et un sous-ensemble linéaire L de $\mathbb{N}^C$ tels que $X = \{q\} \times L$. Un sous-ensemble *semilinéaire* d'états est une union finie de sous-ensembles linéaires d'états.

Exemple 3.2.9 Pour le SAVE étendu d'exemple $\mathcal{EV}$, en partant de l'état initial $(q_0, (0,0))$, l'ensemble des états accessibles est semilinéaire car il s'écrit :

$$\begin{array}{rcl} \mathrm{post}^*(q_0,(0,0)) = & & \{(q_0,(0,0))\} \\ & \bigcup & \{q_0\} \times ((0,15) + \{(0,15),(1,3)\}^*) \\ & \bigcup & \{q_1\} \times ((0,0) + \{(0,15)\}^*) \\ & \bigcup & \{q_2\} \times ((2,0) + \{(0,15),(1,3)\}^*) \end{array}$$

Nous montrons maintenant comment définir une abstraction des systèmes d'addition de vecteurs avec états étendus, en suivant la démarche que nous avons proposée en section 2.4 pour l'abstraction et l'accélération des automates étendus. Commençons par définir la *représentation symbolique* basée sur les sous-ensembles semilinéaires :

Définition 3.2.10 *Soit* $\mathcal{SL} = (\mathcal{P}_f(\mathbb{N}^C \times \mathcal{P}_f(\mathbb{N}^C)), \delta_{\mathcal{SL}}, [\![\cdot]\!], \cup, \emptyset)$*, où les fonctions* $\delta_{\mathcal{SL}}$ *et* $[\![\cdot]\!]$ *sont définies par :*

$$\begin{array}{rcl}
\delta_{\mathcal{SL}}(\mathtt{add}(\mathsf{v}), \chi) & = & \displaystyle\bigcup_{(\mathsf{x},P)\in\chi} \{(\mathsf{b}, P) \ / \ \mathsf{b} \in \mathrm{Min}(((\mathsf{x} + P^*) + \mathsf{v}) \cap \mathbb{N}^C)\} \\
\delta_{\mathcal{SL}}(\mathtt{test}(c), \chi) & = & \displaystyle\bigcup_{(\mathsf{x},P)\in\chi, \mathsf{x}[c]=0} \{\mathsf{x}\} \times \{\mathsf{p} \ / \ \mathsf{p} \in P, \mathsf{p}[c] = 0\} \\
\delta_{\mathcal{SL}}(\mathtt{weaktest}(c), \chi) & = & \{(\mathsf{x}, P \cup \{0[c := 1]\}) \ / \ (\mathsf{x}, P) \in \delta_{\mathcal{SL}}(\mathtt{test}(c), \chi)\} \\
\delta_{\mathcal{SL}}(\mathtt{reset}(c), \chi) & = & \displaystyle\bigcup_{(\mathsf{x},P)\in\chi} \{\mathsf{x}[c := 0]\} \times \{\mathsf{p}[c := 0] \ / \ \mathsf{p} \in P\} \\
\delta_{\mathcal{SL}}(\mathtt{transfer}(c \to c'), \chi) & = & \displaystyle\bigcup_{(\mathsf{x},P)\in\chi} \{tr(\mathsf{x}, c, c')\} \times \{tr(\mathsf{p}, c, c') \ / \ \mathsf{p} \in P\} \\
[\![\chi]\!] & = & \displaystyle\bigcup_{(\mathsf{x},P)\in\chi} (\mathsf{x} + P^*)
\end{array}$$

où pour tout $\mathsf{x} \in \mathbb{N}^C$ *et* $c, c' \in C$*, avec* $c \neq c'$*, on désigne par* $tr(\mathsf{x}, c, c')$ *le vecteur* $\mathsf{x}[c := 0][c' := (\mathsf{x}[c] + \mathsf{x}[c'])]$.

Proposition 3.2.11 $\mathcal{SL}$ *est une représentation symbolique effective pour la sémantique de compteurs* $\Delta_{\mathbb{N}}$.

Par conséquent, pour tout SAVE étendu $\mathcal{V}$, $\mathcal{A}_{\mathcal{V}}^{\mathcal{SL}}$ est une abstraction effective de $\mathcal{V}$. De plus, la fonction $\nabla_{\mathcal{V}}^{\mathcal{SL}}$, basée sur l'itération des séquences d'opérations σ dans $\mathtt{TOp}(C)^+$ telles que σ^* est $\mathcal{SL}$–représentable, est une accélération correcte pour l'abstraction $\mathcal{A}_{\mathcal{V}}^{\mathcal{SL}}$.

Rappelons que la relation binaire d'accessibilité décrivant l'effet d'un circuit quelconque d'un *automate à compteurs* peut être décrite par une formule de l'arithmétique de Presburger [CJ98]. Les automates à compteurs définis dans [CJ98] englobent clairement les SAVE étendus dont l'ensemble d'opérations ne contient aucun transfert, et on en déduit donc le théorème suivant :

Théorème 3.2.12 ([CJ98]) *Pour toute séquence d'opérations* σ*, si on a* $\sigma \in (\mathtt{TOp}(C) \setminus \mathtt{Tr}(C))^+$*, alors* σ^* *est* $\mathcal{SL}$*–représentable.*

Comme conséquence immédiate du théorème, dont la preuve est constructive, on obtient que si l'ensemble d'opérations d'un SAVE étendu $\mathcal{V}$ ne contient aucun transfert, alors la fonction $\nabla_{\mathcal{V}}^{\mathcal{SL}}$ est une accélération complète récursive pour l'abstraction $\mathcal{A}_{\mathcal{V}}^{\mathcal{SL}}$.

Remarque. Nous ne connaissons pas de caractérisation effective des séquences d'opérations $\sigma \in \mathtt{TOp}(C)^+$ (avec transfert) telles que σ^* est $\mathcal{SL}$–représentable.

Vers un calcul effectif de post^* **?**

Hopcroft et Pansiot ont donné dans [HP79] un algorithme calculant l'ensemble semilinéaire des états accessibles d'un SAVE (non étendu) quelconque de dimension 2. Cet algorithme construit un arbre symbolique d'accessibilité semblable à notre algorithme `ArbreSymbolique` de la page 34. Nous avons généralisé dans [FS00c, FS00b] le résultat d'Hopcroft et Pansiot à diverses classes de SAVE étendus de dimension 2, comme par exemple la classe des SAVE de dimension 2 étendus par des test-à-zéro sur un seul des deux compteurs, dont le SAVE étendu $\mathcal{EV}$ fait partie. Pour y parvenir, nous avons généralisé l'algorithme d'Hopcroft et Pansiot, et défini des accélérations adaptées permettant d'assurer la terminaison de cet algorithme généralisé. Avant de présenter nos travaux, commençons par montrer pourquoi l'algorithme `ArbreSymbolique` n'était pas suffisant pour obtenir ces résultats de calcul effectif de post^*.

Considérons la classe $\mathfrak{T}_1$ des SAVE de dimension 2 étendus par des test-à-zéro sur un seul des deux compteurs. Pour tout SAVE étendu $\mathcal{V}$ dans cette classe, $\mathcal{A}_{\mathcal{V}}^{\mathcal{SL}}$ est une abstraction de $\mathcal{V}$ et la fonction $\nabla_{\mathcal{V}}^{\mathcal{SL}}$ est une accélération correcte et complète pour l'abstraction $\mathcal{A}_{\mathcal{V}}^{\mathcal{SL}}$. L'algorithme `ArbreSymbolique` appliqué à $\mathcal{A}_{\mathcal{V}}^{\mathcal{SL}}$ et à $\nabla_{\mathcal{V}}^{\mathcal{SL}}$, pour un ensemble semilinéaire d'états initiaux, calcule un arbre symbolique d'accessibilité potentiellement infini. Observons que lorsque cet algorithme termine, il calcule l'ensemble des états accessibles (qui est alors nécessairement semilinéaire).

Comme l'accélération $\nabla_{\mathcal{V}}^{\mathcal{SL}}$ est correcte et complète, il semblerait naturel que cet algorithme termine dès que l'algorithme d'Hopcroft et Pansiot généralisé termine. Cependant, cette idée intuitive est fausse, car si on considère le SAVE étendu $\mathcal{EV}$ muni d'un unique état initial $(q_0, (0,0))$, il est assez facile de se rendre compte que l'algorithme `ArbreSymbolique` ne termine pas.

Nous avons décrit en figure 3.2 le début de l'arbre symbolique calculé (en largeur d'abord) par l'algorithme `ArbreSymbolique`, appliqué à $(\mathcal{A}_{\mathcal{EV}}^{\mathcal{SL}}, a_0, \nabla_{\mathcal{EV}}^{\mathcal{SL}})$, où l'état abstrait a_0 dénote l'unique état initial $(q_0, (0,0))$. Rappelons que chaque noeud de l'arbre est étiqueté par un état abstrait, qui est ici un élément de $\left(\mathcal{P}_f(\mathbb{N}^C \times \mathcal{P}_f(\mathbb{N}^C))\right)^Q$, c'est-à-dire un vecteur donnant pour chaque état de contrôle q une description finie du semilinéaire décrivant l'ensemble des contenus de compteurs en q. Comme nous partons d'un unique état initial, et comme le contrôle V est déterministe, les états abstraits étiquetant les noeuds de l'arbre ont une seule composant "non vide", aussi nous utilisons une notation simplifiée dans la figure 3.2. Le noeud étiqueté par $(q_2, (3,3))$ est le seul noeud "accéléré" du début de l'arbre représenté, et nous avons noté juste en dessous (entouré en pointillé) l'état abstrait obtenu "après accélération" (lignes 7–9 de l'algorithme `ArbreSymbolique`). Pour tous les autres noeuds, l'état abstrait obtenu "après accélération" est le même que celui "avant accélération". Les noeud entourés deux fois sont les noeuds couverts par leurs ancêtres.

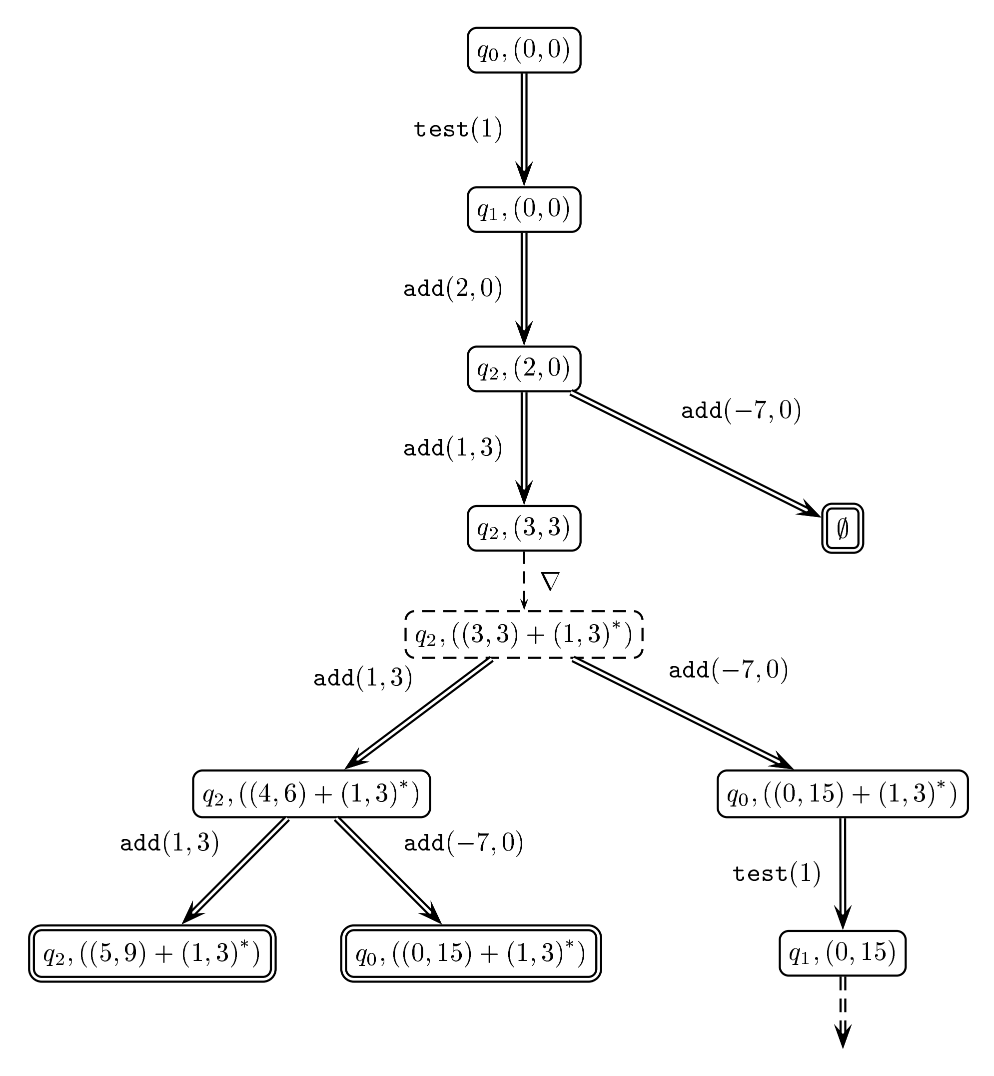


FIG. 3.2 – Arbre symbolique pour le SAVE étendu $\mathcal{EV}$

On voit bien que l'arbre symbolique construit n'est pas fini sur cet exemple, car le chemin de l'arbre entre le noeud étiqueté par $(q_1, (0,0))$ et le noeud étiqueté par $(q_1, (0,15))$ va se répéter indéfiniment, donnant lieu à des noeuds étiquetés par $(q_1, (0,30)), (q_1, (0,45)), \ldots$ tous non couverts. En fait, pour obtenir la période $(0,15)$ dans la description semilinéaire de l'ensemble des états accessibles, il faudrait itérer, par exemple à partir de q_1, la séquence d'opérations $\mathtt{add}(2,0)(\mathtt{add}(1,3))^5\mathtt{add}(-7,0)\mathtt{test}(1)$. Cependant, cette séquence d'opérations ne peut pas apparaître sur une branche de l'arbre symbolique, car la transition $q_2 \xrightarrow{\mathtt{add}(1,3)} q_2$ ne sera jamais franchie 5 fois de suite. En effet, si $\mathtt{t}:a$ est un noeud non couvert obtenu après le franchissement de cette transition, alors la période $(1,3)$ sera capturée par l'étape d'accélération et se trouvera donc dans la décomposition semilinéaire du fils $\mathtt{t}':a'$ de $\mathtt{t}:a$ obtenu après le franchissement de cette même transition. Si $\mathtt{t}' : a'$ n'est pas couvert, alors son fils $\mathtt{t}'' : a''$, obtenu toujours après le franchissement de $q_2 \xrightarrow{\mathtt{add}(1,3)} q_2$, sera, lui, nécessairement couvert par son père $\mathtt{t}' : a'$.

En revanche, l'algorithme que nous allons définir dans la suite, muni d'une accélération adaptée, terminera nécessairement sur cet exemple. Intuitivement, nous exploiterons la donnée des ancêtres, et la façon dont ils sont accélérés, afin d'accélérer encore mieux ! Ici par exemple, on se rend compte que le chemin dans l'arbre symbolique de la figure 3.2 entre le noeud étiqueté par $(q_1, (0, 0))$ et le noeud étiqueté par $(q_1, (0, 15))$ satisfait les deux conditions suivantes :

- les étiquettes des arcs le long de ce chemin sont toutes des opérations d'addition, et
- l'accélération du noeud $(q_2, (3, 3))$, permettant d'obtenir la période $(1, 3)$, n'a utilisé que des opérations d'additions.

Par conséquent, il existe un chemin $\pi = (q_1, (0, 0)) \xrightarrow{\sigma} (q_1, (0, 15))$ dans $\mathcal{EV}$, où σ ne contient que des opérations d'addition et des $\mathtt{test}(1)$. On en déduit que $(q_1, (0, 0)) \xrightarrow{\sigma^k} (q_1, (0, 15k))$ pour tout $k \in \mathbb{N}$, ce qui permet donc d'ajouter $(0, 15)$ dans l'ensemble des périodes.

3.3 Calcul d'un arbre d'accessibilité symbolique

Considérons un SAVE étendu $\mathcal{V}$ d'ensemble de compteurs C. L'algorithme que nous allons développer est basé sur la construction d'un arbre symbolique d'accessibilité, étiqueté par des triplets (q, x, P) où q est un état de contrôle de $\mathcal{V}$, x est un vecteur de $\mathbb{N}^C$ et P est un sous-ensemble fini de $\mathbb{N}^C$. Intuitivement, l'*étiquette* $l = (q, \mathsf{x}, P)$ d'un *noeud* t (noté $\mathtt{t} : l$) représente l'ensemble d'états $[\![l]\!] = \{q\} \times (x + P^*)$ de $\mathcal{V}$.

Notation. Pour tout état de contrôle q de $\mathcal{V}$, pour tout $\mathsf{x} \in \mathbb{N}^C$ et pour tout sous-ensemble fini P de $\mathbb{N}^C$, on désigne par $[\![q, \mathsf{x}, P]\!]$ l'ensemble d'états $[\![q, \mathsf{x}, P]\!] = \{q\} \times (x + P^*)$ de $\mathcal{V}$.

3.3.1 Chemins symboliques et arbres symboliques d'accessibilité

Une *étiquette d'accélération* Γ est soit l'élément $\perp$, soit un sous-ensemble d'opérations $\Gamma \subseteq \mathtt{TOp}(C)$. Intuitivement, une étiquette d'accélération indique quelles opérations ont été utilisées pour accélérer le calcul. Par exemple, l'étiquette d'accélération $\emptyset$ indique que le calcul n'a pas été accéléré. L'étiquette d'accélération particulière $\perp$ indique qu'on ne sait pas comment le calcul a été accéléré.

Soit $\mathtt{t} : l$ and $\mathtt{t}' : l'$ deux noeuds étiquetés par $l = (q, \mathsf{x}, P)$ et $l' = (q', \mathsf{x}', P')$. Etant donné une étiquette d'accélération Γ et une opération α, on dit que le quadruplet $(\mathtt{t} : l, \Gamma, \alpha, \mathtt{t}' : l')$ est un *arc symbolique* pour $\mathcal{V}$, noté $\mathtt{t} : l \xRightarrow{\Gamma, \alpha} \mathtt{t}' : l'$

(ou plus simplement $t \xRightarrow{\Gamma,\alpha} t'$ ou encore $t \Rightarrow t'$), pourvu que les deux conditions suivantes soient satisfaites :

1. le triplet $q \xrightarrow{\alpha} q'$ est une transition de contrôle dans V, et,
2. si $\Gamma \neq \bot$ alors on a $[\![q', \mathsf{x}', P']\!] \subseteq \bigcup_{\sigma \in \Gamma^*} \mathrm{post}([\![q, \mathsf{x}, P]\!], \sigma\alpha)$.

Une suite d'arcs symboliques $(\mathtt{t_i}\!:\!l_i \xRightarrow{\Gamma_i,\alpha_i} \mathtt{t'_i}\!:\!l'_i)_{i\in I}$ est *contiguë* si $t'_i = t_{i+1}$ pour tout $i \in I$ tel que $i+1 \in I$.

Définition 3.3.1 *Un* chemin symbolique *pour un SAVE étendu $\mathcal{V}$ est un couple $\pi = (\mathtt{t}\!:\!l, (\mathtt{t_i}\!:\!l_i \xRightarrow{\Gamma_i,\alpha_i} \mathtt{t'_i}\!:\!l'_i)_{i\in I})$ où :*

1. *le noeud $\mathtt{t}\!:\!l$ est l'origine du chemin π, et*
2. *$(\mathtt{t_i}\!:\!l_i \xRightarrow{\Gamma_i,\alpha_i} \mathtt{t'_i}\!:\!l'_i)_{i\in I}$ est une suite contiguë d'arcs symboliques, et*
3. *si I est non vide alors $t_m = t$ où m est l'unique élément minimal de I.*
4. *on a $[\![l'_i]\!] \subseteq \mathrm{post}(\mathrm{post}^*([\![l]\!]), \alpha_i)$ pour tout $i \in I$ tel que $\Gamma_i = \bot$.*

De la même façon que pour les chemins d'un système de transitions étiqueté, on dit que π est *fini* (resp. *infini*) si l'intervalle d'entiers I est fini (resp. infini). Si π est fini, alors sa *longueur* est la longueur de la suite d'arcs symboliques $(\mathtt{t_i}\!:\!l_i \xRightarrow{\Gamma_i,\alpha_i} \mathtt{t'_i}\!:\!l'_i)_{i\in I}$.

Un chemin symbolique fini ou infini $\pi = (\mathtt{t}:l, (\mathtt{t_i}\!:\!l_i \xRightarrow{\Gamma_i,\alpha_i} \mathtt{t'_i}\!:\!l'_i)_{i\geq 0})$ est plus simplement noté $\pi = t \xRightarrow{\Gamma_0,\alpha_0} t_1 \xRightarrow{\Gamma_1,\alpha_1} t_2 \cdots t_i \xRightarrow{\Gamma_i,\alpha_i} t_{i+1} \cdots$. Un chemin symbolique fini $\pi = (\mathtt{t}:l, (\mathtt{t_i}\!:\!l_i \xRightarrow{\Gamma_i,\alpha_i} \mathtt{t'_i}\!:\!l'_i)_{0\leq i \leq k})$ s'écrit également $\pi = t \xRightarrow{\Gamma_0,\alpha_0} t_1 \xRightarrow{\Gamma_1,\alpha_1} t_2 \cdots t_{k-1} \xRightarrow{\Gamma_i,\alpha_{k-1}} t_k$, ou encore $\pi = t \overset{*}{\Rightarrow} t_k$. Si de plus $\Gamma_i \neq \bot$ pour tout $i \in [0\,..\,k-1]$, alors on écrit aussi $\pi = t_0 \xRightarrow{\Gamma^*} t_k$ pour tout sous-ensemble d'opérations Γ tel que $\Gamma \supseteq \bigcup_{i=0}^{k-1} \Gamma_i \cup \{\alpha_i\}$. Intuitivement, Γ^* signifie que les opérations de Γ sont suffisantes pour étiqueter un chemin allant d'un état de $[\![l_0]\!]$ à un état de $[\![l_k]\!]$.

Pour tout noeud $\mathtt{t}\!:\!l$, le chemin symbolique fini $\pi = (\mathtt{t}\!:\!l, \varepsilon)$ de longueur nulle se note également $\pi = \mathtt{t}\!:\!l$ ou encore $\pi = \mathtt{t}\!:\!l \overset{\varepsilon}{\Rightarrow} \mathtt{t}\!:\!l$. Un chemin symbolique fini $\pi = t \overset{*}{\Rightarrow} t'$ de longueur non nulle s'écrit également $\pi = t \overset{+}{\Rightarrow} t'$.

Remarque. Etant donné un chemin symbolique $\pi = t_0 \Rightarrow t_1 \Rightarrow t_2 \cdots t_i \Rightarrow t_{i+1} \cdots$, d'étiquettes $t_i : l_i$, on a $[\![l_i]\!] \subseteq \mathrm{post}^*([\![l_0]\!])$ pour tout $i \geq 0$. Etant donné chemin symbolique $\pi = t_0 \xRightarrow{\Gamma^*} t_k$, d'étiquette $t_i : l_i$, on a $[\![l_k]\!] \subseteq \bigcup_{\sigma\in\Gamma^*} \mathrm{post}([\![l_0]\!], \sigma)$.

Définition 3.3.2 *U noeud $\mathtt{t}\!:\!l$ est* couvert *par un noeud $\mathtt{t'}\!:\!l'$ si $[\![l]\!] \subseteq [\![l']\!]$. Un chemin symbolique $\pi = r \overset{*}{\Rightarrow} t$ est* auto-couvrant *si π peut s'écrire sous la forme $\pi = r \overset{*}{\Rightarrow} t' \overset{+}{\Rightarrow} t$ où t' un noeud intermédiaire couvrant t.*

La proposition suivante sera utile pour prouver la terminaison de l'algorithme `ArbreSymbolique` que nous allons définir.

Proposition 3.3.3 *Soit* $\pi = t_0 \Rightarrow t_1 \Rightarrow t_2 \cdots t_i \Rightarrow t_{i+1} \cdots$ *un chemin symbolique infini, d'étiquettes* $t_i : (q_i, \mathsf{x}_i, P_i)$. *S'il existe un ensemble fini* P *et une sous-suite infinie* $t_{i_0} \overset{*}{\Rightarrow} t_{i_1} \overset{*}{\Rightarrow} t_{i_2} \cdots t_{i_j} \overset{*}{\Rightarrow} t_{i_{j+1}} \cdots$ *telle que pour tout* $j \in \mathbb{N}$, $q_{i_j} = q_{i_0}$, $P \subseteq P_{i_j} \subseteq \widehat{P}$ *et* $\mathsf{x}_{i_j} \in (\mathsf{x}_{i_0} + \widehat{P})$, *alors un préfixe fini* $t_0 \overset{*}{\Rightarrow} t_k$ *de* π *est auto-couvrant.*

Preuve. Nous savons d'après la proposition 3.2.7 et le corollaire 3.2.8 que $\sqsubseteq_P$ et $\leq_P$ sont de beaux pré-ordres. Par conséquent, il existe $h < k$ tel que $P_{i_h} \sqsubseteq_P P_{i_k}$ et $\mathsf{x}_{i_h} \leq_P \mathsf{x}_{i_k}$. Comme $P_{i_h} \sqsubseteq_P P_{i_k}$ et $P \subseteq P_{i_h}$, on obtient que $P_{i_k} \subseteq (P_{i_h} + P^*) \subseteq {P_{i_h}}^*$. Maintenant, puisque $\mathsf{x}_{i_h} \leq_P \mathsf{x}_{i_k}$, on aboutit à $\mathsf{x}_{i_k} \in (\mathsf{x}_{i_h} + P^*)$ et donc $\mathsf{x}_{i_k} \in (\mathsf{x}_{i_h} + {P_{i_h}}^*)$. On obtient finalement que $(\mathsf{x}_{i_k} + {P_{i_k}}^*) \subseteq (\mathsf{x}_{i_h} + {P_{i_h}}^*)$, et par conséquent un préfixe fini $t_0 \overset{*}{\Rightarrow} t_k$ de π est auto-couvrant. ■

Nous pouvons maintenant définir une notion d'arbre symbolique d'accessibilité comme suit.

Définition 3.3.4 *Un* arbre symbolique d'accessibilité *pour un SAVE étendu* $\mathcal{V}$ *muni d'un ensemble* S_0 *d'états initiaux, est un arbre dirigé enraciné* T, *étiqueté par des triplets* (q, x, P) *où* q *est un état de contrôle de* $\mathcal{V}$, $\mathsf{x} \in \mathbb{N}^C$ *et* P *est un sous-ensemble fini de* $\mathbb{N}^C$, *et satisfaisant les deux conditions suivantes :*

1. *on a* $\mathrm{post}^*(S_0) = \displaystyle\bigcup_{\mathtt{t}:l \in T} [\![l]\!]$, *et,*

2. *toute branche*[1] *de* T *est un chemin symbolique.*

3.3.2 Accélération et calcul d'un arbre symbolique d'accessibilité

L'algorithme `ArbreSymbolique` défini ci-après calcule un arbre symbolique d'accessibilité, en utilisant une fonction d'accélération. Contrairement à l'algorithme `ArbreSymbolique`, de la page 34, défini dans le cadre général des systèmes de transitions bien abstraits, ce nouvel algorithme fournit toute la branche en cours à la fonction d'accélération, afin de lui permettre de "mieux accélérer", et d'asurer ainsi la terminaison de l'algorithme. La fonction d'accélération utilisée dépendra du SAVE étendu considéré, et nous allons en fait définir dans la suite des fonctions d'accélération spécifiques à certaines classes de SAVE étendus.

Définition 3.3.5 *Une* accélération ∇ *pour un SAVE étendu* $\mathcal{V}$ *est une fonction associant à tout chemin symbolique* $r \overset{*}{\Rightarrow} t$ *de* $\mathcal{V}$ *un couple* $\nabla(r \overset{*}{\Rightarrow} t) = (P_\nabla, \Gamma_\nabla)$, *où* P_∇ *est un sous-ensemble fini de* $\mathbb{N}^C$ *et* Γ_∇ *est une étiquette d'accélération.*

[1] Une *branche* d'un arbre enraciné est un chemin dans l'arbre partant de la racine de l'arbre.

Définition 3.3.6 *Une accélération* ∇ *est* stable *si pour tout chemin symbolique* $t_0 : l_0 \overset{*}{\Rightarrow} t : (q, \mathsf{x}, P)$*, on a :*

$$\begin{array}{lll} [\![q, \mathsf{x}, P \cup P_\nabla]\!] & \subseteq & \bigcup_{\sigma \in \Gamma_\nabla^*} \mathrm{post}([\![(q, \mathsf{x}, P)]\!], \sigma) \qquad \textit{si } \Gamma_\nabla \neq \bot \\ [\![q, \mathsf{x}, P \cup P_\nabla]\!] & \subseteq & \mathrm{post}^*([\![l_0]\!]) \qquad \textit{si } \Gamma_\nabla = \bot \end{array}$$

où $\nabla(t_0 \overset{*}{\Rightarrow} t) = (P_\nabla, \Gamma_\nabla)$.

Algorithme 4 `ArbreSymbolique`$(\mathcal{V}, r \overset{*}{\Rightarrow} t : (q, \mathsf{x}, P), \nabla)$

Entrée: un SAVE étendu $\mathcal{V}$, un chemin symbolique $r \overset{*}{\Rightarrow} t : (q, \mathsf{x}, P)$ de $\mathcal{V}$ et une accélération stable ∇ pour $\mathcal{V}$

si $r \overset{*}{\Rightarrow} t$ est auto-couvrant **alors**
  retourner
sinon
  $(P_\nabla, \Gamma_\nabla) \leftarrow \nabla(r \overset{*}{\Rightarrow} t)$
  $W \leftarrow P \cup P_\nabla$
  {On calcule maintenant l'ensemble $\mathrm{post}([\![q, \mathsf{x}, W]\!])$}
  pour chaque transition $q \xrightarrow{\mathtt{add(v)}} q'$ de V **faire**
    pour chaque $\mathsf{m} \in \mathrm{Min}(((\mathsf{x} + W^*) + \mathsf{v}) \cap \mathbb{N}^n)$ **faire**
      construire un fils $t' : (q', \mathsf{m}, W)$ de t et étiqueter l'arc $t \xRightarrow{\Gamma_\nabla, \mathtt{add(v)}} t'$
  pour chaque transition $q \xrightarrow{\mathtt{test}(c)} q'$ de V **faire**
    si $\mathsf{x}[c] = 0$ **alors**
      $P' \leftarrow \{\mathsf{p} \in P \ / \ \mathsf{p}[c] = 0\}$
      construire un fils $t' : (q', \mathsf{x}, P')$ de t et étiqueter l'arc $t \xRightarrow{\Gamma_\nabla, \mathtt{test}(i)} t'$
  pour chaque transition $q \xrightarrow{\mathtt{weaktest}(c)} q'$ de V **faire**
    si $\mathsf{x}[c] = 0$ **alors**
      $P' \leftarrow \{\mathsf{p} \in P \ / \ \mathsf{p}[c] = 0\} \cup \{\mathsf{0}[c := 1]\}$
      construire un fils $t' : (q', \mathsf{x}, P')$ de t et étiqueter l'arc $t \xRightarrow{\Gamma_\nabla, \mathtt{weaktest}(i)} t'$
  pour chaque transition $q \xrightarrow{\mathtt{reset}(c)} q'$ de V **faire**
    $\mathsf{x}' \leftarrow \mathsf{x}[c := 0]$
    $P' \leftarrow \{\mathsf{p}[c := 0] \ / \ \mathsf{p} \in P\}$
    construire un fils $t' : (q', \mathsf{x}', P')$ de t et étiqueter l'arc $t \xRightarrow{\Gamma_\nabla, \mathtt{reset}(i)} t'$
  pour chaque transition $q \xrightarrow{\mathtt{transfer}(c \to c')} q'$ de V **faire**
    $\mathsf{x}' \leftarrow \mathsf{x}[c := 0][c' := (\mathsf{x}[c] + \mathsf{x}[c'])]$
    $P' \leftarrow \{\mathsf{p}[c := 0][c' := (\mathsf{p}[c] + \mathsf{p}[c'])] \ / \ \mathsf{p} \in P\}$
    construire un fils $t' : (q', \mathsf{x}', P')$ de t et étiqueter l'arc $t \xRightarrow{\Gamma_\nabla, \mathtt{transfer}(i \to j)} t'$
  {Appel récursif pour tous les fils}
  pour chaque fils t' de t **faire**
    {Puisque ∇ est stable, $r \overset{*}{\Rightarrow} t \Rightarrow t'$ est un chemin symbolique de $\mathcal{V}$}
    `ArbreSymbolique`$(\mathcal{V}, r \overset{*}{\Rightarrow} t', \nabla)$
  retourner

Remarque. L'algorithme **`ArbreSymbolique`** appliqué à $(\mathcal{V}, r \overset{*}{\Rightarrow} t, \nabla)$ construit un arbre dirigé de racine t. Observons qu'une branche $\mathtt{t} : l \overset{*}{\Rightarrow} \mathtt{t}' : l'$ dans cet arbre n'est pas nécessairement un chemin symbolique, puisque on peut avoir

$[\![l']\!] \not\subseteq \mathrm{post}^*([\![l]\!])$ alors que $[\![l']\!] \subseteq \mathrm{post}^*([\![l_r]\!])$, où $r : l_r$. Cependant, on se rend aisément compte que $r \stackrel{*}{\Rightarrow} t \stackrel{*}{\Rightarrow} t'$ est un chemin symbolique.

Théorème 3.3.7 *Soit $\mathcal{V}$ un SAVE étendu et $t_0 : l_0$ un noeud étiqueté par un ensemble linéaire $[\![l_0]\!]$ d'états de $\mathcal{V}$. Etant donné une accélération stable ∇ pour $\mathcal{V}$, l'algorithme* `ArbreSymbolique`*, appliqué à $(\mathcal{V}, t_0, \nabla)$, construit un arbre symbolique d'accessibilité pour $\mathcal{V}$ avec états initiaux $[\![l_0]\!]$.*

Preuve. Notons T l'arbre construit par `ArbreSymbolique`$(\mathcal{V}, t_0, \nabla)$. D'après la définition de l'algorithme `ArbreSymbolique`, et comme indiqué en ligne 28, on déduit facilement que toute branche de T est un chemin symbolique. Il reste donc à prouver que $\mathrm{post}^*([\![l_0]\!]) = \bigcup_{\mathtt{t}:l \in T} [\![l]\!]$.

Puisque toute branche de T est un chemin symbolique, on obtient que $\mathrm{post}^*([\![l_0]\!]) \supseteq \bigcup_{\mathtt{t}:l \in T} [\![l]\!]$. Pour montrer l'inclusion réciproque, on prouve par récurrence sur la longueur de σ que $\mathrm{post}([\![l_0]\!], \sigma) \subseteq \bigcup_{\mathtt{t}:l \in T} [\![l]\!]$, pour toute séquence d'opérations σ. La base est triviale car la racine t_0 de T est étiquetée par l_0. Pour montrer l'étape d'induction, supposons que l'hypothèse de récurrence est satisfaite pour toute séquence d'opérations σ de longueur k, pour un certain $k \in \mathbb{N}$. Soit α une opération et soit $(q, \mathsf{x}) \in \mathrm{post}([\![l_0]\!], \sigma\alpha)$. Il existe $(q', \mathsf{x}') \in \mathrm{post}([\![l_0]\!], \sigma)$ tel que $(q', \mathsf{x}') \xrightarrow{\alpha} (q, \mathsf{x})$. D'après l'hypothèse de récurrence, on sait qu'il existe un noeud $\mathtt{t}:l$ dans T tel que $(q', \mathsf{x}') \in [\![l]\!]$. De plus, on peut supposer que $t_0 \stackrel{*}{\Rightarrow} t$ n'est pas auto-couvrant, car sinon $t_0 \stackrel{*}{\Rightarrow} t$ s'écrirait sous la forme $t_0 \stackrel{*}{\Rightarrow} t' \stackrel{+}{\Rightarrow} t$, où t' est un noeud intermédiaire couvrant t, et on pourrait alors remplacer t par t'. Maintenant, d'après les lignes 5–25 de l'algorithme `ArbreSymbolique`, on obtient que t a un fils $t_s : l_s$ tel que $(q, \mathsf{x}) \in [\![l_s]\!]$. ∎

Etant donné un sous-ensemble `Op` d'opérations, on définit une relation d'équivalence sur les accélérations comme suit.

Définition 3.3.8 *Soit ∇_1, ∇_2 deux accélérations pour un SAVE étendu $\mathcal{V}$. On dit que ∇_1* est équivalente à *∇_2* modulo `Op`, *noté $\nabla_1 \equiv \nabla_2\,[\mathtt{Op}]$, si pour tout chemin symbolique $r \stackrel{*}{\Rightarrow} t \xRightarrow{\Gamma,\alpha} t'$ se terminant par une opération $\alpha \in \mathtt{Op}$, on a $\nabla_1(r \stackrel{*}{\Rightarrow} t \xRightarrow{\Gamma,\alpha} t') = \nabla_2(r \stackrel{*}{\Rightarrow} t \xRightarrow{\Gamma,\alpha} t')$.*

Définition 3.3.9 *On dit qu'une accélération ∇ est* `Op`-couvrante *si pour tout chemin symbolique $r \stackrel{*}{\Rightarrow} t$, toute branche infinie dans l'arbre construit par* `Arbre Symbolique`$(\mathcal{V}, r \stackrel{*}{\Rightarrow} t, \nabla)$ *contient une infinité d'opérations α qui n'appartiennent pas à* `Op`.

Proposition 3.3.10 *Une accélération ∇ est* `Op`*-couvrante si et seulement si pour tout chemin symbolique $r \stackrel{*}{\Rightarrow} t$ et pour toute branche π dans l'arbre construit par* `ArbreSymbolique`$(\mathcal{V}, r \stackrel{*}{\Rightarrow} t, \nabla)$*, si π contient uniquement des opérations α dans* `Op` *alors π est finie.*

Preuve. Le sens $\Rightarrow$ est trivial. Montrons la réciproque, et supposons que pour tout chemin symbolique $r \stackrel{*}{\Rightarrow} t$ et pour toute branche π dans l'arbre construit par `ArbreSymbolique`$(\mathcal{V}, r \overset{*}{\Rightarrow} t, \nabla)$, si π contient uniquement des opérations α dans `Op` alors π est finie. Soit $r \overset{*}{\Rightarrow} t$ un chemin symbolique et soit $\pi = t_0 \xrightarrow{\Gamma_0,\alpha_0} t_1 \xrightarrow{\Gamma_1,\alpha_1} t_2 \cdots t_k \xrightarrow{\Gamma_k,\alpha_k} t_{k+1} \cdots$ une branche infinie dans l'arbre construit par `ArbreSymbolique`$(\mathcal{V}, r \overset{*}{\Rightarrow} t_0, \nabla)$.

Par l'absurde, supposons que π ne contienne qu'un nombre fini d'opérations α qui n'appartiennent pas à `Op`. Alors il existe $h \in \mathbb{N}$ tel que $\alpha_i \in$ `Op` pour tout $i \geq h$. Par conséquent, toutes les opérations apparaissant dans le suffixe infini $\pi_h = t_h \xrightarrow{\Gamma_h,\alpha_h} t_{h+1} \cdots$ de π commençnt en t_h appartiennent à `Op`. Observons que π_h est une branche dans `ArbreSymbolique`$(\mathcal{V}, r \overset{*}{\Rightarrow} t_0 \overset{*}{\Rightarrow} t_h, \nabla)$. On en déduit, d'après notre hypothèse initiale, que π_h est finie, ce qui contredit l'hypothèse que π est infinie. On en conclut finalement que π contient une infinité d'opérations α qui n'appartiennent pas à `Op`. ■

Proposition 3.3.11 *Soit ∇ une accélération pour un SAVE étendu $\mathcal{V}$ d'ensemble* `Op` *d'opérations. Si ∇ est* `Op`*-couvrante alors pour tout chemin symbolique $r \overset{*}{\Rightarrow} t$, l'algorithme* `ArbreSymbolique` *appliqué à $(\mathcal{V}, r \overset{*}{\Rightarrow} t, \nabla)$ termine.*

Preuve. Supposons que ∇ est `Op`-couvrante et soit $r \overset{*}{\Rightarrow} t$ un chemin symbolique. Soit π une branche dans l'arbre construit par `ArbreSymbolique`$(\mathcal{V}, r \overset{*}{\Rightarrow} t, \nabla)$. Observons que toute opération α le long de la branche π appartient à `Op`. Comme ∇ est `Op`-couvrante, on en déduite que π est finie. Par conséquent, toute branche dans l'arbre construit par `ArbreSymbolique`$(\mathcal{V}, r \overset{*}{\Rightarrow} t, \nabla)$ est finie. Comme le degré de cet arbre est fini (voir les lignes 7–25), on obtient par application du lemme de König que l'arbre construit par `ArbreSymbolique`$(\mathcal{V}, r \overset{*}{\Rightarrow} t, \nabla)$ est fini. On en conclut donc que l'algorithme `ArbreSymbolique` appliqué à $(\mathcal{V}, r \overset{*}{\Rightarrow} t, \nabla)$ termine. ■

Proposition 3.3.12 *Soit* `Op` *un sous-ensemble d'opérations et ∇_1, ∇_2 deux accélérations pour un SAVE étendu $\mathcal{V}$ telles que $\nabla_1 \equiv \nabla_2\,[\texttt{Op}]$. Alors ∇_1 est* `Op`*-couvrante si et seulement si ∇_2 est* `Op`*-couvrante.*

Preuve. Supposons que ∇_1 est `Op`-couvrante et montrons que ∇_2 est `Op`-couvrante. Soit π une branche dans l'arbre construit par `ArbreSymbolique`$(\mathcal{V}, r \overset{*}{\Rightarrow} t, \nabla_2)$. Supposons que π contienne uniquement des opérations α dans `Op`. Puisque $\nabla_1 \equiv \nabla_2\,[\texttt{Op}]$, on obtient que π est aussi une branche dans l'arbre construit par `ArbreSymbolique`$(\mathcal{V}, r \overset{*}{\Rightarrow} t, \nabla_1)$. Or π contient uniquement des opérations α dans `Op`. Comme ∇_1 est `Op`-couvrante, on en déduit que π est finie. ■

Lorsque l'algorithme `ArbreSymbolique` termine sur un SAVE étendu $\mathcal{V}$ muni d'un ensemble linéaire $[\![l_0]\!]$ d'états initiaux et d'une accélération stable ∇, $\mathrm{post}^*([\![l_0]\!])$ est nécessairement semilinéaire. Comme les SAVE (non étendus) de dimension 3 peuvent avoir un ensemble d'états accessibles non semilinéaire [HP79], nous nous intéressons plus particulièrement aux SAVE étendus de dimension 2. Dans la suite de ce chapitre, notre but est de trouver des sous-classes $\mathfrak{C}$ de SAVE étendus de dimension 2, pour lesquelles une accélération dédiée assure la terminaison de l'algorithme `ArbreSymbolique`.

3.4 Classes de SAVE étendus de dimension 2

Pour toute la suite de ce chapitre, les SAVE étendus que nous considérons sont de dimension 2, et nous fixons l'ensemble C des compteurs à $C = \{1,2\}$, afin de simplifier les notations.

Nous définissons les sous-classes suivantes de SAVE étendus de dimension 2. Pour tout $I \subseteq \{1,2\}$, $J \subseteq \{1,2\}$ et $K \subseteq \{(1,2),(2,1)\}$, on désigne par $\mathfrak{T}_I\mathfrak{R}_J\mathfrak{Tr}_K$ la classe des SAVE étendus de dimension 2, dont l'ensemble d'opérations $\mathtt{Op}$ vérifie :

$$\mathtt{Op} \subseteq \mathtt{A} \cup \{\mathtt{test}(i), \mathtt{reset}(j), \mathtt{transfer}(k \to k') \ / \ i \in I, j \in J \text{ and } (k,k') \in K\}$$

Observons que $\mathfrak{T}_\emptyset$ (resp. $\mathfrak{R}_\emptyset$, $\mathfrak{Tr}_\emptyset$) signifie qu'il n'y a aucun test-à-zéro (resp. aucune remise-à-zéro, aucun transfert), et nous omettrons alors de préciser $\mathfrak{T}_\emptyset$ (resp. $\mathfrak{R}_\emptyset$, $\mathfrak{Tr}_\emptyset$). Par exemple, $\mathfrak{T}_1\mathfrak{R}_\emptyset\mathfrak{Tr}_{12,21}$ (écrit plus simplement $\mathfrak{T}_1\mathfrak{Tr}_{12,21}$) est la classe des SAVE étendus de dimension 2 où les opérations étendues "autorisées" sont $\mathtt{test}(1)$, $\mathtt{transfer}(1 \to 2)$ et $\mathtt{transfer}(2 \to 1)$.

Simulations

Soit $\mathfrak{C}$ et $\mathfrak{D}$ deux classes de SAVE étendus. On dit que $\mathfrak{C}$ est *effectivement simulable* par $\mathfrak{D}$ si pour tout SAVE étendu $\mathcal{V}$ dans $\mathfrak{C}$, d'ensemble d'états de contrôle Q_{V}, il existe un SAVE étendu $\mathcal{W}$ dans $\mathfrak{D}$, d'ensemble d'états de contrôle Q_{W}, tel que :

1. W est calculable à partir de V, et,
2. on a $Q_{\mathsf{W}} \supseteq \overline{Q_{\mathsf{V}}}$, où $\overline{Q_{\mathsf{V}}}$ est une "copie" de Q_{V}, et,
3. pour tout état (q,x) et (q',x') dans $\mathcal{V}$, on a $(q,\mathsf{x}) \xrightarrow{*}_{\mathcal{V}} (q',\mathsf{x}')$ si et seulement si $(\overline{q},\mathsf{x}) \xrightarrow{*}_{\mathcal{W}} (\overline{q'},\mathsf{x}')$.

Nous utiliserons cette notion de simulation effective, qui est différente de la notion classique de simulation entre systèmes de transitions étiquetés, afin de réduire le nombre de sous-classes à analyser. En effet, supposons que $\mathfrak{C}$ est effectivement simulable par $\mathfrak{D}$. Si post* (resp. pre*) est effectivement semilinéaire pour la classe $\mathfrak{D}$, alors on en déduit que post* (resp. pre*) est effectivement semilinéaire pour la classe $\mathfrak{C}$.

Théorème 3.4.1 *Nous avons les simulations suivantes entre classes de SAVE étendus de dimension 2 :*

1. *la classe $\mathfrak{T}_1\mathfrak{R}_2$ est effectivement simulable par la classe $\mathfrak{T}_{1,2}$,*
2. *la classe $\mathfrak{T}_1\mathfrak{R}_{1,2}\mathfrak{Tr}_{12}$ est effectivement simulable par la classe $\mathfrak{T}_1\mathfrak{R}_2$,*
3. *la classe $\mathfrak{T}_1\mathfrak{R}_1\mathfrak{Tr}_{12}$ est effectivement simulable par la classe $\mathfrak{T}_1$.*
4. *la classe $\mathfrak{T}_{1,2}$ est effectivement simulable par la classe $\mathfrak{T}_1\mathfrak{Tr}_{21}$,*
5. *la classe $\mathfrak{R}_{1,2}\mathfrak{Tr}_{12,21}$ est effectivement simulable par la classe $\mathfrak{R}_{1,2}\mathfrak{Tr}_{12}$,*

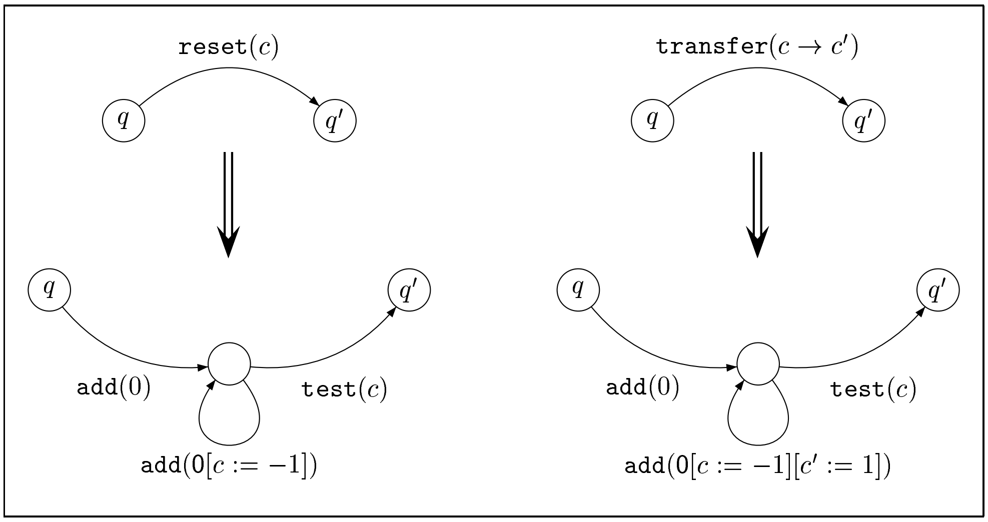


FIG. 3.3 – Simulation de $\texttt{reset}(c)$ et de $\texttt{transfer}(c \to c')$ par des $\texttt{test}(c)$ et des additions

Preuve. Comme nous l'avons indiqué sur la figure 3.3, les opérations $\texttt{reset}(c)$ et $\texttt{transfer}(c \to c')$ peuvent être facilement simulées par des $\texttt{test}(c)$ (et des opérations d'addition). Ceci prouve les assertions 1–3 du théorème. Nous montrons maintenant la quatrième assertion.

Soit $\mathcal{V}$ un SAVE étendu appartenant à la classe $\mathfrak{T}_1\mathfrak{T}_2$, et notons $\mathsf{V} = (Q_\mathsf{V}, \mathtt{Op}_\mathsf{V}, \to_\mathsf{V})$ son contrôle. Nous allons construire un SAVE étendu $\mathcal{W}$ appartenant à la classe $\mathfrak{T}_1\mathfrak{Tr}_{21}$ "simulant" $\mathcal{V}$. Intuitivement, $\mathcal{W}$ a la capacité de tester à zéro le premier compteur uniquement. Pour tester à zéro le second compteur, les rôles des deux compteurs vont être échangés. Nous ajoutons donc une information finie dans les état de contrôle, donnée par un élément de $\{12, 21\}$. Dans un état de contrôle $(q, 12)$ de $\mathcal{W}$, les compteurs se comportent normalement. Mais dans un état de contrôle $(q, 21)$, les rôles des deux compteurs sont échangés, ce qui permet, avec une opération $\texttt{test}(1)$, de tester à zéro le second compteur. Le bon moment pour échanger (de façon non déterministe) les rôles des compteurs se situe juste après une transition de test-à-zéro $\texttt{test}(1)$, en transférant le contenu du deuxième compteur dans le premier. Cette manipulation requière cependant un état de contrôle intermédiaire $(q, 12 \to 21)$ ou $(q, 21 \to 12)$. Plus formellement, le contrôle $\mathsf{W} = (Q_\mathsf{W}, \mathtt{Op}_\mathsf{W}, \to_\mathsf{W})$ est construit à partir de V comme suit :

- les états de contrôle sont $Q_\mathsf{W} = Q_\mathsf{V} \times \{12, 21, 12 \to 21, 21 \to 12\}$, et,
- les opérations sont $\mathtt{Op}_\mathsf{W} = (\mathtt{Op}_\mathsf{V} \setminus \{\texttt{test}(2)\}) \cup \{\texttt{transfer}(2 \to 1)\}$, et,
- l'ensemble des transitions de contrôle $\to_\mathsf{W} \subseteq Q_\mathsf{W} \times \mathtt{Op}_\mathsf{W} \times Q_\mathsf{W}$ est le plus petit sous-ensemble de $Q_\mathsf{W} \times \mathtt{Op}_\mathsf{W} \times Q_\mathsf{W}$ satisfaisant les quatres conditions suivantes :

1. si $q \xrightarrow{\texttt{add(v)}}_{\mathsf{v}} q'$ alors on a $\begin{cases} (q,12) \xrightarrow{\texttt{add(v)}}_{\mathsf{W}} (q',12), \text{ et} \\ (q,21) \xrightarrow{\texttt{add((v[2],v[1]))}}_{\mathsf{W}} (q',21) \end{cases}$

2. si $q \xrightarrow{\texttt{test}(1)}_{\mathsf{V}} q'$ alors on a $\begin{cases} (q,12) \xrightarrow{\texttt{test}(1)}_{\mathsf{W}} (q',12), \text{ et} \\ (q,12) \xrightarrow{\texttt{test}(1)}_{\mathsf{W}} (q',12 \to 21) \end{cases}$

3. si $q \xrightarrow{\texttt{test}(2)}_{\mathsf{V}} q'$ alors on a $\begin{cases} (q,21) \xrightarrow{\texttt{test}(1)}_{\mathsf{W}} (q',21), \text{ et} \\ (q,21) \xrightarrow{\texttt{test}(1)}_{\mathsf{W}} (q',21 \to 12) \end{cases}$

4. pour tout $q \in Q_{\mathsf{V}}$, on a $\begin{cases} (q,12 \to 21) \xrightarrow{\texttt{transfer}(2\to1)}_{\mathsf{W}} (q,21), \text{ et} \\ (q,21 \to 12) \xrightarrow{\texttt{transfer}(2\to1)}_{\mathsf{W}} (q,12) \end{cases}$

On vérifie que pour tout état (q,x) et (q',x') de $\mathcal{V}$, on a :

$$(q,\mathsf{x}) \xrightarrow{*}_{\mathcal{V}} (q',\mathsf{x}') \quad \text{ssi} \quad \begin{cases} ((q,12),\mathsf{x}) & \xrightarrow{*}_{\mathcal{W}} & ((q',12),\mathsf{x}') \text{ ou} \\ ((q,21),(\mathsf{x}[2],\mathsf{x}[1])) & \xrightarrow{*}_{\mathcal{W}} & ((q',12),\mathsf{x}') \end{cases}$$

La preuve de la dernière assertion est très similaire à la preuve ci-dessus de la quatrième assertion. Intuitivement, pour simuler un $\texttt{transfer}(2 \to 1)$, on commence par franchir un $\texttt{transfer}(1 \to 2)$ et on échange ensuite les rôles des deux compteurs. ∎

Nous avons représenté sur la figure 3.4 les classes "maximales" de SAVE étendus de dimension 2, ainsi que les simulations entre ces classes.

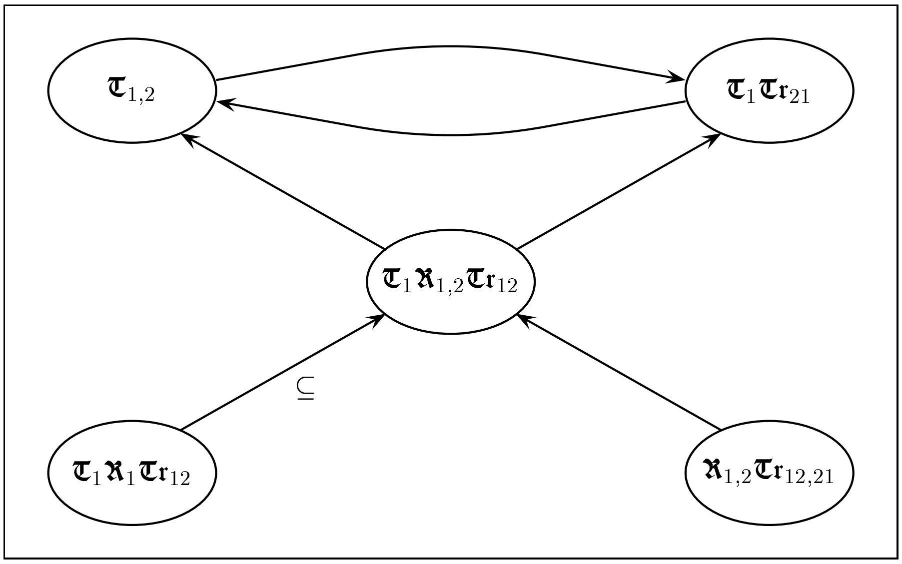


Fig. 3.4 – Hiérarchie des classes de SAVE étendus de dimension 2

3.5 Effective semilinéarité de post^* pour la classe $\mathfrak{T}_1\mathfrak{R}_1\mathfrak{Tr}_{12}$

Nous pouvons maintenant généraliser le résultat d'Hopcroft et Pansiot présenté dans [HP79], en montrant que post^* est encore semilinéaire et constructible pour la classe $\mathfrak{T}_1\mathfrak{R}_1\mathfrak{Tr}_{12}$ des SAVE étendus de dimension 2 avec test-à-zéro sur le premier compteur, remise-à-zéro sur le premier compteur et transfert du premier compteur vers le second. Nous concevons tout d'abord une accélération capturant la stratégie d'Hopcroft et Pansiot, ce qui nous permettra, en particulier, de retrouver leur résultat.

Conception d'une accélération stable et $\mathtt{A}$-couvrante

Un circuit de contrôle $\chi = q \xrightarrow{\alpha_1} q_1 \xrightarrow{\alpha_2} q_2 \cdots q_k \xrightarrow{\alpha_k} q$ d'un SAVE étendu $\mathcal{V}$ est *valide* depuis un état (q, x) s'il existe un chemin $(q, \mathsf{x}) \xrightarrow{\alpha_1} (q_1, \mathsf{x}_1) \xrightarrow{\alpha_2} (q_2, \mathsf{x}_2) \cdots (q_k, \mathsf{x}_k) \xrightarrow{\alpha_k} (q, \mathsf{x}')$ dans $\mathcal{V}$. On dit que χ est un circuit de contrôle *simple* si χ ne contient que des opérations d'addition. Le *déplacement* $\mathsf{d}(\chi)$ d'un circuit de contrôle simple $\chi = q \xrightarrow{\mathtt{add}(\mathsf{v}_1)} q_1 \xrightarrow{\mathtt{add}(\mathsf{v}_2)} q_2 \cdots q_k \xrightarrow{\mathtt{add}(\mathsf{v}_k)} q$ est défini par $\mathsf{d}(\chi) = \mathsf{v}_1 + \mathsf{v}_2 + \cdots + \mathsf{v}_k$. On note $D_{\mathcal{V}}(q, \mathsf{x})$ l'ensemble des déplacements de circuits de contrôle élémentaires simples valides en (q, x) :

$$D_{\mathcal{V}}(q, \mathsf{x}) = \{\mathsf{d}(\chi) \ / \ \chi \text{ est un circuit de contrôle élémentaire simple valide en } (q, \mathsf{x})\}$$

Algorithme 5 $\nabla^{\mathcal{V}}_{HP}(r \overset{*}{\Rightarrow} t : (q, \mathsf{x}, P))$

Entrée: un chemin symbolique $r \overset{*}{\Rightarrow} t$ pour un SAVE étendu $\mathcal{V}$ de dimension 2

1: $P_\nabla \leftarrow D_{\mathcal{V}}(q, \mathsf{x}) \cap \mathbb{N}^2$
2: **si** $(P_\nabla \cup P) \subseteq \{0\}$ **alors**
3:     **si** $r \overset{*}{\Rightarrow} t$ peut s'écrire sous la forme $r \overset{*}{\Rightarrow} t' \overset{\mathtt{A}^*}{\Longrightarrow} t$ où $t' : (q, \mathsf{x}', P')$ est un noeud intermédiaire tel que $P' \subseteq P$ et $\mathsf{x} - \mathsf{x}' > 0$ **alors**
4:         $P_\nabla \leftarrow \{\mathsf{x} - \mathsf{x}'\}$
5: **pour chaque** $\mathsf{e} \in (\{(0,1), (1,0)\} \setminus \widehat{P_\nabla \cup P})$ **faire**
6:     **si** il existe $\mathsf{u} \in (P_\nabla \cup P), \mathsf{v} \in D_{\mathcal{V}}(q, \mathsf{x})$ et $c, d \in \mathbb{N}$ tels que $(c\mathsf{u} + d\mathsf{v}) \in \mathsf{e}\,\mathbb{N}^*$ **alors**
7:         $P_\nabla \leftarrow P_\nabla \cup \{(c\mathsf{u} + d\mathsf{v})\}$
8:     **sinon si** $r \overset{*}{\Rightarrow} t$ peut s'écrire sous la forme $r \overset{*}{\Rightarrow} t' \overset{\mathtt{A}^*}{\Longrightarrow} t$ où $t' : (q, \mathsf{x}', P')$ est un noeud intermédiaire tel que $P' \subseteq P$ et $(\mathsf{x} - \mathsf{x}') \in \mathsf{e}\,\mathbb{N}^*$ **alors**
9:         $P_\nabla \leftarrow P_\nabla \cup \{\mathsf{x} - \mathsf{x}'\}$
10: **retourner** $(P_\nabla, \mathtt{Op} \cap \mathtt{A})$ où $\mathtt{Op}$ est l'ensemble d'opérations de $\mathcal{V}$

Proposition 3.5.1 *Pour tout SAVE étendu $\mathcal{V}$ de dimension 2, $\nabla^{\mathcal{V}}_{HP}$ est une accélération stable et $\mathtt{A}$-couvrante pour $\mathcal{V}$.*

Preuve. Soit $r \overset{*}{\Rightarrow} t : (q, \mathsf{x}, P)$ un chemin symbolique. Pour montrer que $\nabla^{\mathcal{V}}_{HP}$ est stable, nous prouvons que $[\![q, \mathsf{x}, P \cup P_\nabla]\!] \subseteq \bigcup_{\sigma \in \mathtt{A}^*} \mathrm{post}([\![(q, \mathsf{x}, P)]\!], \sigma)$. D'après la définition de $\nabla^{\mathcal{V}}_{HP}$, cette assertion est trivialement vraie pour la ligne 1 et pour les lignes 6–7. Examinons à présent les lignes 3–4 et 8–9. Rappelons que $t' : (q, \mathsf{x}', P') \overset{\mathtt{A}^*}{\Longrightarrow} t : (q, \mathsf{x}, P)$ implique que $[\![q, \mathsf{x}, P]\!] \subseteq \bigcup_{\sigma \in \mathtt{A}^*} \mathrm{post}([\![q, \mathsf{x}', P']\!], \sigma)$. Par conséquent, il existe $\mathsf{y} \in P'^*$ tel que $(q, \mathsf{x}' + \mathsf{y}) \xrightarrow{\sigma} (q, \mathsf{x})$ pour un certain

$\sigma \in \mathtt{A}^*$. Supposons que $P' \subseteq P$ et que $\mathsf{x} - \mathsf{x}' > 0$. Puisque σ est composée uniquement d'additions, on obtient que $(q, \mathsf{x} + \mathsf{y}) \xrightarrow{\sigma} (q, \mathsf{x} + (\mathsf{x} - \mathsf{x}'))$. D'une manière plus générale, on vérifie par récurrence sur k que pour tout $k \in \mathbb{N}$, on a $(q, \mathsf{x} + k\mathsf{y}) \xrightarrow{\sigma^k} (q, \mathsf{x} + k(\mathsf{x} - \mathsf{x}'))$. Comme $\mathsf{y} \in P'^* \subseteq P^*$, on conclut finalement que $[\![q, \mathsf{x}, P \cup \{\mathsf{x} - \mathsf{x}'\}]\!] \subseteq \bigcup_{\sigma \in \mathtt{A}^*} \mathrm{post}([\![(q, \mathsf{x}, P)]\!], \sigma)$.

Montrons à présent que $\nabla_{HP}^{\mathcal{V}}$ est $\mathtt{A}$-couvrante. Soit $\pi = t_0 \xRightarrow{\Gamma_0, \alpha_0} t_1 \xRightarrow{\Gamma_1, \alpha_1} t_2 \cdots t_k \xRightarrow{\Gamma_k, \alpha_k} t_{k+1} \cdots$ une branche dans $\mathtt{ArbreSymbolique}(\mathcal{V}, r \overset{*}{\Rightarrow} t, \nabla_{HP}^{\mathcal{V}})$, d'étiquettes $t_i : (q_i, \mathsf{x}_i, P_i)$, et supposons que π ne contienne que des opérations α dans $\mathtt{A}$. Observons que pour tout $i \in \mathbb{N}$, $\Gamma_i = \mathtt{Op} \cap \mathtt{A}$, où $\mathtt{Op}$ est l'ensemble d'opérations de $\mathcal{V}$. Prouvons par l'absurde que π est fini, et supposons donc que π est infini. Puisque π est infini, on déduit de la définition de l'algorithme $\mathtt{ArbreSymbolique}$ qu'aucun préfixe fini de π n'est auto-couvrant. Notre objectif est d'appliquer la proposition 3.3.3 afin de montrer que π admet un préfixe fini $t_0 \overset{*}{\Rightarrow} t_k$ auto-couvrant, ce qui terminera la preuve.

Commençons par montrer qu'il existe $h \in \mathbb{N}$ tel que pour tout $i \geq h$, $P_i = P_h$. Remarquons que pour tout $i \in \mathbb{N}$, $P_i \subseteq P_{i+1}$. Or le long du chemin π, les lignes 4, 7 et 9 de $\nabla_{HP}^{\mathcal{V}}$ peuvent être chacune exécutées au plus deux fois. Par conséquent, il existe $k \in \mathbb{N}$ tel que pour tout $i \geq k$, $P_i \subseteq P_{i+1} = P_i \cup (D_i \cap \mathbb{N}^2)$ où D_i est un ensemble de déplacements de circuits de contrôle élémentaires simples. Comme il n'y a qu'un nombre fini de circuits de contrôle élémentaires simples, on en déduit qu'il existe $h \in \mathbb{N}$ tel que pour tout $i \geq h$, $P_i = P_h$. Afin d'appliquer la proposition 3.3.3, il reste à prouver qu'il existe une sous-suite infinie $t_{i_0} \overset{*}{\Rightarrow} t_{i_1} \overset{*}{\Rightarrow} t_{i_2} \cdots t_{i_j} \overset{*}{\Rightarrow} t_{i_{j+1}} \cdots$, avec $i_0 \geq h$, telle que pour tout $j \in \mathbb{N}$, $q_{i_j} = q_{i_0}$ et $\mathsf{x}_{i_j} \in (\mathsf{x}_{i_0} + \widehat{P_h})$.

Choisissons un vecteur $\mathsf{z} \in \mathbb{N}^2$ tel que pour tout état de contrôle q et pour tout circuit de contrôle élémentaire simple χ sur q, χ est valide en (q, z). Deux cas se présentent :

1. un nombre infini de x_i du chemin π se trouvent sur la même ligne horizontale (resp. verticale). Comme il n'y a qu'un nombre fini d'états de contrôle, il existe $c \in \{(0,1), (1,0)\}$ et une sous-suite infinie $t_{i_0} \overset{*}{\Rightarrow} t_{i_1} \overset{*}{\Rightarrow} t_{i_2} \cdots t_{i_j} \overset{*}{\Rightarrow} t_{i_{j+1}} \cdots$ telle que $i_0 \geq h$ et telle que pour tout $j \in \mathbb{N}$, $q_{i_j} = q_{i_0}$ et $\mathsf{x}_{i_j} - \mathsf{x}_{i_0} \in c\,\mathbb{N}$. Puisque $P_{i_0} = P_{i_1} = P_h$ et comme $t_0 \overset{*}{\Rightarrow} t_{i_1}$ n'est pas auto-couvrant, on a $\mathsf{x}_{i_1} \neq \mathsf{x}_{i_0}$, et il vient donc $\mathsf{x}_{i_1} - \mathsf{x}_{i_0} \in c\,\mathbb{N}^*$. On déduit des lignes 8–9 de $\nabla_{HP}^{\mathcal{V}}$ qu'il existe $\mathsf{p} \in P_h$ tel que $\mathsf{p} \in c\,\mathbb{N}^*$. Par conséquent, on a $\mathsf{x}_{i_j} \in (\mathsf{x}_{i_0} + \widehat{P_h})$ pour tout $j \in \mathbb{N}$.

2. il existe $h' \geq h$ tel que pour tout $i \geq h'$, $\mathsf{x}_i \geq \mathsf{z}$. Comme il n'y a qu'un nombre fini d'états de contrôle, et puisque $\leq$ est un beau pré-ordre sur $\mathbb{N}^2$, il existe une sous-suite infinie $t_{i_0} \overset{*}{\Rightarrow} t_{i_1} \overset{*}{\Rightarrow} t_{i_2} \cdots t_{i_j} \overset{*}{\Rightarrow} t_{i_{j+1}} \cdots$ telle que $i_0 \geq h'$ et telle que pour tout $j \in \mathbb{N}$, $q_{i_j} = q_{i_0}$, et $\mathsf{x}_{i_j} \geq \mathsf{x}_{i_0}$. Puisque $P_{i_0} = P_{i_1} = P_h$ et comme $t_0 \overset{*}{\Rightarrow} t_{i_1}$ n'est pas auto-couvrant, on a $\mathsf{x}_{i_1} \neq \mathsf{x}_{i_0}$. Par conséquent, $\mathsf{x}_{i_1} > \mathsf{x}_{i_0}$ et on en déduit que $P_h \not\subseteq \{\mathsf{0}\}$ à cause des lignes 2–4 de $\nabla_{HP}^{\mathcal{V}}$. Ainsi, on obtient qu'il existe $\mathsf{u} \in P_h$ tel que $\mathsf{u} > \mathsf{0}$.

Nous montrons maintenant que $\mathsf{x}_{i_j} \in (\mathsf{x}_{i_0} + \widehat{P_h})$ pour tout $j \in \mathbb{N}$. Soit $j \in \mathbb{N}$. Observons que $[\![q_{i_j}, \mathsf{x}_{i_j}, P_h]\!] \subseteq \bigcup_{\sigma \in \mathtt{A}^*} \mathrm{post}([\![q_{i_0}, \mathsf{x}_{i_0}, P_h]\!], \sigma)$ car $t_{i_0} \xRightarrow{\mathtt{A}^*} t_{i_j}$. Par conséquent, il existe $\mathsf{y} \in {P_h}^*$ tel que $(q_{i_0}, \mathsf{x}_{i_0} + \mathsf{y}) \xrightarrow{\sigma} (q_{i_j}, \mathsf{x}_{i_j})$ pour un certain $\sigma \in \mathtt{A}^*$. Comme $q_{i_j} = q_{i_0}$, on en déduit que $\mathsf{x}_{i_j} - \mathsf{x}_{i_0} = \Sigma_{k=1}^{m} \mathsf{d}(\chi_k) + \mathsf{y}$ où chaque χ_k est un circuit de contrôle élémentaire simple sur un état de contrôle $q_{\varphi(k)}$ avec $i_0 \leq \varphi(k) \leq i_k$. Puisque $\mathsf{x}_{i_j} - \mathsf{x}_{i_0} = \Sigma_{k=1}^{m} \mathsf{d}(\chi_k) + \mathsf{y}$, on obtient qu'il existe $k \in [1\,..\,m]$ tel que $\mathsf{x}_{i_j} - \mathsf{x}_{i_0} \in \widehat{\{\mathsf{u}, \mathsf{d}(\chi_k)\}}$. Rappelons que χ_k est valide en $(q_{\varphi(k)}, \mathsf{x}_{\varphi(k)})$. Si $\mathsf{d}(\chi_k) \geq 0$ alors on déduit de la ligne 1 de $\nabla_{HP}^{\mathcal{V}}$ que $\mathsf{d}(\chi_k) \in P_h$, et par conséquent $\mathsf{x}_{i_j} - \mathsf{x}_{i_0} \in \widehat{P_h}$. Sinon, on a $\mathsf{d}(\chi_k) \not\geq \mathsf{0}$ et on en déduit qu'il existe $\mathsf{e} \in \underline{\{(0,1),(1,0)\}}$ tel que $\mathsf{e} \in \widehat{\{\mathsf{u}, \mathsf{d}(\chi_k)\}}$ et $\mathsf{x}_{i_j} - \mathsf{x}_{i_0} \in \widehat{\{\mathsf{u}, \mathsf{e}\}}$. Comme $\mathsf{e} \in \widehat{\{\mathsf{u}, \mathsf{d}(\chi_k)\}}$, on obtient qu'il existe $c, d \in \mathbb{N}$ tels que $(c\mathsf{u} + d\mathsf{d}(\chi_k)) \in \mathsf{e}\mathbb{N}^*$. D'après les lignes 6–7 de $\nabla_{HP}^{\mathcal{V}}$, on aboutit à $\mathsf{e} \in \widehat{P_h}$. Or on a $\mathsf{u} \in P_h$, et nous obtenons finalement que $\mathsf{x}_{i_j} - \mathsf{x}_{i_0} \in \widehat{P_h}$.

D'après la proposition 3.3.3, on obtient qu'un préfixe fini de π est auto-couvrant, ce qui conclut la preuve. ■

Théorème 3.5.2 ([HP79]) *Pour tout SAVE de dimension* 2*, muni d'un ensemble linéaire* S_0 *d'états initiaux,* $\mathrm{post}^*(S_0)$ *est semilinéaire et constructible.*

Conception d'une accélération stable et $(\mathtt{A} \cup \{\mathtt{test}(1)\})$-couvrante

Algorithme 6 $\nabla_{\mathfrak{T}_1}^{\mathcal{V}}(r \overset{*}{\Rightarrow} t : (q, \mathsf{x}, P))$

Entrée: un chemin symbolique $r \overset{*}{\Rightarrow} t$ pour un SAVE étendu $\mathcal{V}$ de dimension 2

1: **si** $r \overset{*}{\Rightarrow} t$ se termine par une opération d'addition $\alpha \in \mathtt{A}$ **alors**
2: **retourner** $\nabla_{HP}^{\mathcal{V}}(r \overset{*}{\Rightarrow} t)$
3: **sinon si** $r \overset{*}{\Rightarrow} t$ peut s'écrire sous la forme $r \overset{*}{\Rightarrow} t' \xRightarrow{(\mathtt{A} \cup \{\mathtt{test}(1)\})^*} t$ où $t' : (q, \mathsf{x}', P')$ est un noeud intermédiaire tel que $P' \subseteq P \subseteq \{\mathsf{0}\}$ et $(\mathsf{x} - \mathsf{x}') \in \{0\} \times \mathbb{N}^*$ **alors**
4: **retourner** $(\{\mathsf{x} - \mathsf{x}'\}, (\mathtt{Op} \cap \mathtt{A}) \cup \{\mathtt{test}(1)\})$ où $\mathtt{Op}$ est l'ensemble d'opérations de $\mathcal{V}$
5: **sinon**
6: **retourner** $(\emptyset, \emptyset)$

Proposition 3.5.3 *Pour tout SAVE étendu* $\mathcal{V}$ *de dimension* 2*,* $\nabla_{\mathfrak{T}_1}^{\mathcal{V}}$ *est une accélération stable et* $(\mathtt{A} \cup \{\mathtt{test}(1)\})$*-couvrante pour* $\mathcal{V}$*.*

Preuve. Soit $r \overset{*}{\Rightarrow} t : (q, \mathsf{x}, P)$ un chemin symbolique. Pour montrer que $\nabla_{\mathfrak{T}_1}^{\mathcal{V}}$ est stable, il suffit de prouver que lorsque les lignes 3–4 sont exécutées, on a $[\![q, \mathsf{x}, P \cup \{\mathsf{x} - \mathsf{x}'\}]\!] \subseteq \bigcup_{\sigma \in (\mathtt{A} \cup \{\mathtt{test}(1)\})^*} \mathrm{post}([\![(q, \mathsf{x}, P)]\!], \sigma)$. Rappelons que $t' \xRightarrow{(\mathtt{A} \cup \{\mathtt{test}(1)\})^*} t$ implique que $[\![q, \mathsf{x}, P]\!] \subseteq \bigcup_{\sigma \in (\mathtt{A} \cup \{\mathtt{test}(1)\})^*} \mathrm{post}([\![q, \mathsf{x}', P']\!], \sigma)$. Maintenant supposons que $P' \subseteq P \subseteq \{\mathsf{0}\}$ et $(\mathsf{x} - \mathsf{x}') \in \{0\} \times \mathbb{N}^*$. On obtient que $(q, \mathsf{x}') \xrightarrow{\sigma} (q, \mathsf{x})$ pour un certain $\sigma \in (\mathtt{A} \cup \{\mathtt{test}(1)\})^*$. Observons qu'on a $(\mathsf{x} - \mathsf{x}')[1] = 0$. Puisque σ est composées d'additions et de $\mathtt{test}(1)$, il vient

$(q, \mathsf{x}) \xrightarrow{\sigma} (q, \mathsf{x} + (\mathsf{x} - \mathsf{x}'))$. D'une manière plus générale, on vérifie par récurrence sur k que pour tout $k \in \mathbb{N}$, on a $(q, \mathsf{x}) \xrightarrow{\sigma^k} (q, \mathsf{x}+k(\mathsf{x}-\mathsf{x}'))$. On conclut finalement que $[\![q, \mathsf{x}, P \cup \{\mathsf{x} - \mathsf{x}'\}]\!] \subseteq \bigcup_{\sigma \in (\mathtt{A} \cup \{\mathtt{test}(1)\})^*} \mathrm{post}([\![(q, \mathsf{x}, P)]\!], \sigma)$.

Montrons à présent que $\nabla^{\mathcal{V}}_{\mathfrak{T}_1}$ est $(\mathtt{A} \cup \{\mathtt{test}(1)\})$-couvrante. Soit $\pi = t_0 \xRightarrow{\Gamma_0, \alpha_0} t_1 \xRightarrow{\Gamma_1, \alpha_1} t_2 \cdots t_k \xRightarrow{\Gamma_k, \alpha_k} t_{k+1} \cdots$ une branche dans $\mathtt{ArbreSymbolique}(\mathcal{V}, r \overset{*}{\Rightarrow} t, \nabla^{\mathcal{V}}_{\mathfrak{T}_1})$, d'étiquettes $t_i : (q_i, \mathsf{x}_i, P_i)$, et supposons que π ne contienne que des opérations α dans $(\mathtt{A} \cup \{\mathtt{test}(1)\})$. Observons que pour tout $i \in \mathbb{N}$, $\Gamma_i \subseteq (\mathtt{Op} \cap \mathtt{A}) \cup \{\mathtt{test}(1)\})$, où $\mathtt{Op}$ est l'ensemble d'opérations de $\mathcal{V}$. Prouvons par l'absurde que π est fini, et supposons donc que π est infini. Puisque π est infini, on déduit de la définition de l'algorithme $\mathtt{ArbreSymbolique}$ qu'aucun préfixe fini de π n'est auto-couvrant. Notre objectif est d'appliquer la proposition 3.3.3 afin de montrer que π admet un préfixe fini $t_0 \overset{*}{\Rightarrow} t_k$ auto-couvrant, ce qui terminera la preuve.

Puisque $\nabla^{\mathcal{V}}_{\mathfrak{T}_1} \equiv \nabla^{\mathcal{V}}_{HP}[\mathtt{A}]$, on déduit de la proposition 3.3.12 que $\nabla^{\mathcal{V}}_{\mathfrak{T}_1}$ est $\mathtt{A}$-couvrante. Par conséquent, il y a une infinité de $\mathtt{test}(1)$ le long de π. Comme il n'y a qu'un nombre fini d'états de contrôle, et puisque $\leq$ est un beau pré-ordre sur $\mathbb{N}^2$, on obtient qu'il existe une sous-suite infinie $t_{i_0} \overset{*}{\Rightarrow} t_{i_1} \overset{*}{\Rightarrow} t_{i_2} \cdots t_{i_j} \overset{*}{\Rightarrow} t_{i_{j+1}} \cdots$ telle que les deux conditions suivantes soient satisfaites :

1. pour tout $j \in \mathbb{N}$, on a $q_{i_j} = q_{i_0}$ et $\alpha_{i_j - 1} = \mathtt{test}(1)$,
2. la sous-suite infinie $(\mathsf{x}_{i_j})_{j \in \mathbb{N}}$ est croissante.

Observons que pour tout $j \in \mathbb{N}$, $t_{i_j - 1} \xRightarrow{\Gamma_{i_j - 1}, \mathtt{test}(1)} t_{i_j}$ est un arc symbolique pour $\mathcal{V}$, et par conséquent on obtient que $\mathsf{x}_{i_j}[1] = 0$ et $P_{i_j} \subseteq \{0\} \times \mathbb{N}$. Puisque π ne contient que des addition et des $\mathtt{test}(1)$, on déduit de la définition de l'algorithme $\mathtt{ArbreSymbolique}$ (lignes 10–13) que si un vecteur $\mathsf{p} \in \{0\} \times \mathbb{N}$ appartient à P_k pour un certain $k \in \mathbb{N}$, alors $\mathsf{p} \in P_i$ pour tout $i \geq k$. On en déduit que la sous-suite infinie $(P_{i_j})_{j \in \mathbb{N}}$ est croissante (pour l'inclusion). Deux cas se présentent :

– si $P_{i_1} \subseteq \{\mathsf{0}\}$ alors on obtient que $P_{i_0} \subseteq P_{i_1} \subseteq \{\mathsf{0}\}$. Par conséquent, il vient $\mathsf{x}_{i_1} \neq \mathsf{x}_{i_0}$ parce que $t_0 \overset{*}{\Rightarrow} t_{i_1}$ n'est pas auto-couvrant. Puisque $\mathsf{x}_{i_1} \geq \mathsf{x}_{i_0}$, on aboutit à $\mathsf{x}_{i_1} - \mathsf{x}_{i_0} \in \{0\} \times \mathbb{N}^*$, et on déduit des lignes 3–5 de $\nabla_{\mathfrak{T}_1}$ qu'il existe $\mathsf{p} \in \nabla^{\mathcal{V}}_{\mathfrak{T}_1}(t_0 \overset{*}{\Rightarrow} t_{i_1})$ tel que $\mathsf{p} \in \{0\} \times \mathbb{N}^*$.

– sinon, on a $P_{i_1} \not\subseteq \{\mathsf{0}\}$. Comme $P_{i_1} \subseteq \{0\} \times \mathbb{N}$, on obtient qu'il existe $\mathsf{p} \in P_{i_1}$ tel que $p \in \{0\} \times \mathbb{N}^*$.

Dans les deux cas, on obtient qu'il existe $\mathsf{p} \in P_{i_1} \cup \nabla^{\mathcal{V}}_{\mathfrak{T}_1}(t_0 \overset{*}{\Rightarrow} t_{i_1})$ tel que $\mathsf{p} \in \{0\} \times \mathbb{N}^*$. Par conséquent, d'après la définition de l'algorithme $\mathtt{ArbreSymbolique}$, il vient $\mathsf{p} \in P_{i_1+1}$ et donc $\mathsf{p} \in P_i$ pour tout $i \geq i_1+1$. On en déduit que pour tout $j \geq 2$, on a $\{\mathsf{p}\} \subseteq P_{i_j} \subseteq \widehat{\{\mathsf{p}\}}$ et $\mathsf{x}_{i_j} \in (\mathsf{x}_{i_2} + \widehat{\{\mathsf{p}\}})$. D'après la proposition 3.3.3, on obtient qu'un préfixe fini de π est auto-couvrant, ce qui conclut la preuve. ■

Puisque la classe $\mathfrak{T}_1\mathfrak{R}_1\mathfrak{Tr}_{12}$ est effectivement simulable par la classe $\mathfrak{T}_1$ (théorème 3.4.1), on déduit de la proposition précedente que :

Théorème 3.5.4 *Pour tout SAVE étendu dans la classe* $\mathfrak{T}_1\mathfrak{R}_1\mathfrak{Tr}_{12}$*, muni d'un ensemble linéaire* S_0 *d'états initiaux,* $\mathrm{post}^*(S_0)$ *est semilinéaire et constructible.*

3.6 Effective semilinéarité de pre^* pour la classe $\mathfrak{T}_1\mathfrak{R}_{1,2}\mathfrak{Tr}_{12}$

Nous présentons dans cette section un algorithme calculant pre^* pour la plus grande classe analysable (i.e. la plus grande classe non équivalente à $\mathfrak{T}_{1,2}$) de SAVE étendus de dimension 2, c'est-à-dire la classe $\mathfrak{T}_1\mathfrak{R}_{1,2}\mathfrak{Tr}_{12}$. Nous prouvons dans un premier temps que post^* est semilinéaire et constructible pour la classe $\mathfrak{T}_1\mathfrak{W}_2$ des SAVE étendus de dimension 2 avec test-à-zéro sur le premier compteur, et test-à-zéro faible sur le second compteur.

Conception d'une accélération stable et $(\mathtt{A}\cup\{\mathtt{weaktest}(2)\})$-couvrante

---

Algorithme 7 $\nabla^{\mathcal{V}}_{\mathfrak{W}_2}(r \overset{*}{\Rightarrow} t : (q, \mathsf{x}, P))$

Entrée: un chemin symbolique $r \overset{*}{\Rightarrow} t$ pour un SAVE étendu $\mathcal{V}$ de dimension 2

1: **si** $r \overset{*}{\Rightarrow} t$ se termine par une opération d'addition $\alpha \in \mathtt{A}$ **alors**

2: **retourner** $\nabla^{\mathcal{V}}_{HP}(r \overset{*}{\Rightarrow} t)$

3: **sinon si** $r \overset{*}{\Rightarrow} t$ peut s'écrire sous la forme $r \overset{*}{\Rightarrow} t' \xRightarrow{(\mathtt{A}\cup\{\mathtt{weaktest}(2)\})^*} t$ où $t' : (q, \mathsf{x}', P')$ est un noeud intermédiaire tel que $P' \subseteq P \subseteq \{\mathsf{0}, (0,1)\}$ et $(\mathsf{x}-\mathsf{x}') \in \mathbb{N}^* \times \{0\}$ **alors**

4: **retourner** $(\{\mathsf{x}-\mathsf{x}'\}, (\mathtt{Op}\cap\mathtt{A})\cup\{\mathtt{weaktest}(2)\})$ où $\mathtt{Op}$ est l'ensemble d'opérations de $\mathcal{V}$

5: **sinon**

6: **retourner** $(\emptyset, \emptyset)$

Proposition 3.6.1 *Pour tout SAVE étendu* $\mathcal{V}$ *de dimension 2,* $\nabla^{\mathcal{V}}_{\mathfrak{W}_2}$ *est une accélération stable et* $(\mathtt{A}\cup\{\mathtt{weaktest}(2)\})$*-couvrante pour* $\mathcal{V}$.

Preuve. On remarque que la définition de la fonction $\nabla^{\mathcal{V}}_{\mathfrak{W}_2}$ est très similaire à de celle de la fonction $\nabla^{\mathcal{V}}_{\mathfrak{T}_1}$. De la même façon, la preuve de la proposition est très similaire à celle de la proposition 3.5.3. Nous nous contentons donc de détailler la preuve de stabilité de $\nabla^{\mathcal{V}}_{\mathfrak{W}_2}$.

Soit $r \overset{*}{\Rightarrow} t : (q, \mathsf{x}, P)$ un chemin symbolique. Pour montrer que $\nabla^{\mathcal{V}}_{\mathfrak{W}_2}$ est stable, il suffit de prouver que $[\![q, \mathsf{x}, P\cup\{\mathsf{x}-\mathsf{x}'\}]\!] \subseteq \bigcup_{\sigma\in(\mathtt{A}\cup\{\mathtt{weaktest}(2)\})^*} \mathrm{post}([\![(q, \mathsf{x}, P)]\!], \sigma)$ lorsque les lignes 3–4 sont exécutées. Rappelons que $t' \xRightarrow{(\mathtt{A}\cup\{\mathtt{weaktest}(2)\})^*} t$ implique que $[\![q, \mathsf{x}, P]\!] \subseteq \bigcup_{\sigma\in(\mathtt{A}\cup\{\mathtt{weaktest}(2)\})^*} \mathrm{post}([\![q, \mathsf{x}', P']\!], \sigma)$. Maintenant supposons que $P' \subseteq P \subseteq \{\mathsf{0}, (0,1)\}$ et $(\mathsf{x}-\mathsf{x}') \in \mathbb{N}^* \times \{0\}$. On obtient qu'il existe $(0, y) \in P'^*$ tel que $(q, \mathsf{x}' + (0, y)) \xrightarrow{\sigma} (q, \mathsf{x})$ pour une certaine séquence d'opérations $\sigma \in (\mathtt{A}\cup\{\mathtt{weaktest}(2)\})^*$. Observons que $(\mathsf{x}-\mathsf{x}')[2] = 0$. Puisque σ est composée d'additions et de $\mathtt{weaktest}(2)$, il vient $(q, \mathsf{x}+(0,y)) \xrightarrow{\sigma} (q, \mathsf{x}+(\mathsf{x}-\mathsf{x}'))$. Deux cas se présentent :

– si σ ne contient aucun $\mathtt{weaktest}(2)$ alors on vérifie par récurrence sur k que pour tout $k \in \mathbb{N}$, on a $(q, \mathsf{x} + k(0,y)) \xrightarrow{\sigma^k} (q, \mathsf{x} + k(\mathsf{x} - \mathsf{x}'))$. Comme on a $(0,y) \in P'^* \subseteq P^*$, on en déduit que $[\![q, \mathsf{x}, P \cup \{\mathsf{x} - \mathsf{x}'\}]\!] \subseteq \bigcup_{\sigma \in (\mathtt{A} \cup \{\mathtt{weaktest}(2)\})^*} \mathrm{post}([\![(q, \mathsf{x}, P)]\!], \sigma)$.

– sinon, σ contient au moins un $\mathtt{weaktest}(2)$. D'après la définition des transitions de test-à-zéro faibles, on a $(q, \mathsf{x} + (0,y)) \xrightarrow{\sigma} (q, \mathsf{x} + (\mathsf{x} - \mathsf{x}') + (0,z))$ pour tout $z \in \mathbb{N}$. D'une manière plus générale, on vérifie par récurrence sur k que pour tout $k \in \mathbb{N}$, on a $(q, \mathsf{x} + (0,y)) \xrightarrow{\sigma^k} (q, \mathsf{x} + k(\mathsf{x} - \mathsf{x}') + (0,z))$. Comme on a $(0,y) \in P'^* \subseteq P^* \subseteq \{(0,1)\}^*$, on en déduit finalement que $[\![q, \mathsf{x}, P \cup \{\mathsf{x} - \mathsf{x}'\}]\!] \subseteq \bigcup_{\sigma \in (\mathtt{A} \cup \{\mathtt{weaktest}(2)\})^*} \mathrm{post}([\![(q, \mathsf{x}, P)]\!], \sigma)$. ■

Conception d'une accélération stable et $(\mathtt{A} \cup \{\mathtt{test}(1), \mathtt{weaktest}(2)\})$-couvrante

Algorithme 8 $\nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{W}_2}(r \overset{*}{\Rightarrow} t : (q, \mathsf{x}, P))$

Entrée: un chemin symbolique $r \overset{*}{\Rightarrow} t$ pour un SAVE étendu $\mathcal{V}$ de dimension 2
1: **si** $r \overset{*}{\Rightarrow} t$ se termine par une opération $\alpha \in \mathtt{A} \cup \{\mathtt{test}(1)\}$ **alors**
2: **retourner** $\nabla^{\mathcal{V}}_{\mathfrak{T}_1}(r \overset{*}{\Rightarrow} t)$
3: **sinon**
4: **retourner** $\nabla^{\mathcal{V}}_{\mathfrak{W}_2}(r \overset{*}{\Rightarrow} t)$

Proposition 3.6.2 *Pour tout SAVE étendu $\mathcal{V}$ de dimension 2, $\nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{W}_2}$ est une accélération stable et $(\mathtt{A} \cup \{\mathtt{test}(1), \mathtt{weaktest}(2)\})$-couvrante pour $\mathcal{V}$.*

Preuve. Soit $r \overset{*}{\Rightarrow} t : (q, \mathsf{x}, P)$ un chemin symbolique. La stabilité de $\nabla^{\mathcal{V}}_{\mathfrak{W}_2}$ découle directement de la stabilité de $\nabla^{\mathcal{V}}_{\mathfrak{T}_1}$ et de $\nabla^{\mathcal{V}}_{\mathfrak{W}_2}$.

Montrons à présent que $\nabla^{\mathcal{V}}_{\mathfrak{W}_2}$ est $(\mathtt{A} \cup \{\mathtt{test}(1), \mathtt{weaktest}(2)\})$-couvrante. Soit $\pi = t_0 \xRightarrow{\Gamma_0, \alpha_0} t_1 \xRightarrow{\Gamma_1, \alpha_1} t_2 \cdots t_k \xRightarrow{\Gamma_k, \alpha_k} t_{k+1} \cdots$ une branche dans $\mathtt{Arbre}$ $\mathtt{Symbolique}(\mathcal{V}, r \overset{*}{\Rightarrow} t, \nabla^{\mathcal{V}}_{\mathfrak{W}_2})$, d'étiquettes $t_i : (q_i, \mathsf{x}_i, P_i)$, et supposons que π ne contienne que des opérations α dans $(\mathtt{A} \cup \{\mathtt{test}(1), \mathtt{weaktest}(2)\})$. Prouvons par l'absurde que π est fini, et supposons donc que π est infini. Puisque π est infini, on déduit de la définition de l'algorithme $\mathtt{ArbreSymbolique}$ qu'aucun préfixe fini de π n'est auto-couvrant. Notre objectif est d'appliquer la proposition 3.3.3 afin de montrer que π admet un préfixe fini $t_0 \overset{*}{\Rightarrow} t_k$ auto-couvrant, ce qui terminera la preuve.

Puisque on a d'une part $\nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{W}_2} \equiv \nabla^{\mathcal{V}}_{\mathfrak{T}_1}[\mathtt{A} \cup \{\mathtt{test}(1)\}]$ et d'autre part $\nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{W}_2} \equiv \nabla^{\mathcal{V}}_{\mathfrak{W}_2}[\mathtt{A} \cup \{\mathtt{weaktest}(2)\}]$, on déduit de la proposition 3.3.12 que $\nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{W}_2}$ est à la fois $(\mathtt{A} \cup \{\mathtt{test}(1)\})$-couvrante et $(\mathtt{A} \cup \{\mathtt{weaktest}(2)\})$-couvrante. Par conséquent il y a une infinité de $\mathtt{test}(1)$ et aussi une infinité de $\mathtt{weaktest}(2)$

le long de π. Comme il n'y a qu'un nombre fini d'états de contrôle, et puisque $\leq$ est un beau pré-ordre sur $\mathbb{N}^2$, on obtient qu'il existe une sous-suite finie :

$$t_{i_0-1} \xrightarrow{\Gamma_{i_0-1},\texttt{weaktest}(2)} t_{i_0} \overset{*}{\Rightarrow} t_{i_1-1} \xrightarrow{\Gamma_{i_1-1},\texttt{test}(1)} t_{i_1} \overset{*}{\Rightarrow} t_{i_2-1} \xrightarrow{\Gamma_{i_2-1},\texttt{test}(1)} t_{i_2}$$

telle que $q_{i_1} = q_{i_2}$ et $\mathsf{x}_{i_1} \leq \mathsf{x}_{i_2}$. Observons qu'on a aussi $\mathsf{x}_{i_j}[1] = 0$ et $P_{i_j} \subseteq \{0\} \times \mathbb{N}$ pour tout $j \in [1..2]$. D'après la définition de l'algorithme `ArbreSymbolique`, on en déduit que $(0,1) \in P_{i_0}$; et puisque π ne contient que des additions, des $\texttt{test}(1)$ et des $\texttt{weaktest}(2)$, on aboutit à $(0,1) \in P_{i_1}$. Par conséquent, il vient $(\mathsf{x}_{i_2} + P_{i_2}{}^*) \subseteq (\mathsf{x}_{i_1} + P_{i_1}{}^*)$ et donc $t_0 \overset{*}{\Rightarrow} t_{i_2}$ est auto-couvrant. ■

Théorème 3.6.3 *Pour tout SAVE étendu dans la classe* $\mathfrak{T}_1\mathfrak{R}_{1,2}\mathfrak{Tr}_{12}$*, muni d'un ensemble linéaire* S_0 *d'états initiaux,* $\mathrm{pre}^*(S_0)$ *est semilinéaire et constructible.*

Preuve. Rappelons que l'opération $\texttt{weaktest}(2)$ est "le dual" de l'opération $\texttt{reset}(2)$ dans la mesure où on a $(q,\mathsf{x}) \xrightarrow{\texttt{weaktest}(2)} (q',\mathsf{x}')$ si et seulement si on a $(q',\mathsf{x}') \xrightarrow{\texttt{reset}(2)} (q,\mathsf{x})$. Par conséquent, on déduit de la proposition 3.6.2 que pour tout SAVE étendu $\mathcal{V}$ dans la classe $\mathfrak{T}_1\mathfrak{R}_2$, muni d'un ensemble linéaire S_0 d'états initiaux, $\mathrm{pre}^*(S_0)$ est semilinéaire et constructible. Puisque la classe $\mathfrak{T}_1\mathfrak{R}_{1,2}\mathfrak{Tr}_{12}$ est effectivement simulable par la classe $\mathfrak{T}_1\mathfrak{R}_2$ (théorème 3.4.1), on obtient que $\mathrm{pre}^*(S_0)$ est semilinéaire et constructible, pour tout SAVE étendu dans la classe $\mathfrak{T}_1\mathfrak{R}_{1,2}\mathfrak{Tr}_{12}$, muni d'un ensemble linéaire S_0 d'états initiaux. ■

Puisque la classe $\mathfrak{R}_{1,2}\mathfrak{Tr}_{12,21}$ est effectivement simulable par la classe $\mathfrak{T}_1\mathfrak{R}_{1,2}\mathfrak{Tr}_{12}$ (théorème 3.4.1), on déduit du théorème précédent que :

Corollaire 3.6.4 *Pour tout SAVE étendu dans la classe* $\mathfrak{R}_{1,2}\mathfrak{Tr}_{12,21}$*, muni d'un ensemble linéaire* S_0 *d'états initiaux,* $\mathrm{pre}^*(S_0)$ *est semilinéaire et constructible.*

3.7 Semilinéarité de post^* pour la classe $\mathfrak{T}_1\mathfrak{R}_{1,2}\mathfrak{Tr}_{12}$

Nous avons prouvé en section 3.5 que post^* est semilinéaire et constructible pour la classe $\mathfrak{T}_1\mathfrak{R}_1\mathfrak{Tr}_{12}$. Nous étendons maintenant ce résultat de semilinéarité à la classe $\mathfrak{T}_1\mathfrak{R}_{1,2}\mathfrak{Tr}_{12}$.

Conception d'une accélération stable et $(\mathtt{A}\cup\{\texttt{reset}(2)\})$-couvrante

La preuve de la proposition suivante est très similaire à la preuve de la proposition 3.5.3, affirmant que $\nabla^{\mathcal{V}}_{\mathfrak{T}_1}$ est une accélération stable et $(\mathtt{A}\cup\{\texttt{test}(1)\})$-couvrante.

Proposition 3.7.1 *Pour tout SAVE étendu* $\mathcal{V}$ *de dimension* 2, $\nabla^{\mathcal{V}}_{\mathfrak{R}_2}$ *est une accélération stable et* $(\mathtt{A}\cup\{\texttt{reset}(2)\})$*-couvrante pour* $\mathcal{V}$*.*

Algorithme 9 $\nabla^{\mathcal{V}}_{\mathfrak{R}_2}(r \overset{*}{\Rightarrow} t : (q, \mathsf{x}, P))$

Entrée: un chemin symbolique $r \overset{*}{\Rightarrow} t$ pour un SAVE étendu $\mathcal{V}$ de dimension 2

1: **si** $r \overset{*}{\Rightarrow} t$ se termine par une opération d'addition $\alpha \in \mathtt{A}$ **alors**
2:     **retourner** $\nabla^{\mathcal{V}}_{HP}(r \overset{*}{\Rightarrow} t)$
3: **sinon si** $r \overset{*}{\Rightarrow} t$ peut s'écrire sous la forme $r \overset{*}{\Rightarrow} t' \xrightarrow{(\mathtt{A}\cup\{\mathtt{reset}(2)\})^*} t$ où $t' : (q, \mathsf{x}', P')$ est un noeud intermédiaire tel que $P' \subseteq P \subseteq \{\mathsf{0}\}$ et $(\mathsf{x} - \mathsf{x}') \in \mathbb{N}^* \times \{0\}$ **alors**
4:     **retourner** $(\{\mathsf{x} - \mathsf{x}'\}, (\mathtt{Op} \cap \mathtt{A}) \cup \{\mathtt{reset}(2)\})$ où $\mathtt{Op}$ est l'ensemble d'opérations de $\mathcal{V}$
5: **sinon**
6:     **retourner** $(\emptyset, \emptyset)$

Conception d'une accélération stable et $(\mathtt{A} \cup \{\mathtt{test}(1), \mathtt{reset}(2)\})$-couvrante

Soit $\mathcal{V}$ un SAVE étendu de dimension 2, d'ensemble Q d'états de contrôle, et soit S_0 un ensemble d'états initiaux. On définit le sous-ensemble $\Omega(\mathcal{V}, S_0)$ d'états de contrôle comme suit : $\Omega(\mathcal{V}, S_0)$ est l'ensemble des états de contrôle q tels que l'ensemble $\{x \in \mathbb{N} \ / \ (q, (0, x)) \in \mathrm{post}(\mathrm{post}^*(S_0), \mathtt{test}(1))\}$ est infini.

L'accélération $\nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{R}_2}$ est construite de sorte que l'algorithme $\mathtt{SymbolicTree}$ termine pour la classe $\mathfrak{T}_1\mathfrak{R}_2$. Le problème majeur pour assurer la terminaison est qu'on peut avoir une infinité d'alternances de $\mathtt{reset}(2)$ et de $\mathtt{test}(1)$ le long d'une branche. Cette accélération utilise l'ensemble $\Omega(\mathcal{V}, S_0)$ afin d'itérer certains circuits. Intuitivement, si on vient juste de franchir une transiton de remise-à-zéro du second compteur, alors la valeur du second compteur avant le franchissement de ce $\mathtt{reset}(2)$ n'a pas d'importance. Par conséquent, si on sait que cette valeur peut être arbitrairement grande, alors, juste avant le franchissement de ce $\mathtt{reset}(2)$, on peut itérer un circuit de contrôle simple qui augmente la valeur du premier compteur, mais *diminue* la valeur du second compteur. A cet effet, nous étendons la notion de *validité* d'un circuit de contrôle de la façon suivante : un circuit de contrôle χ est valide en $(q, (x, \omega))$ s'il existe $y \in \mathbb{N}$ tel que χ est valide en $(q, (x, y))$.

Observons que cet ensemble $\Omega(\mathcal{V}, S_0)$ est fini, puisque c'est un sous-ensemble de Q. Même si on ne sait pas si $\Omega(\mathcal{V}, S_0)$ peut être calculé pour la classe $\mathfrak{T}_1\mathfrak{R}_2$, on peut néanmoins l'utiliser comme un "oracle" dans notre fonction $\nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{R}_2}$.

Lemme 3.7.2 *Pour tout SAVE étendu $\mathcal{V}$ de dimension 2, $\nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{R}_2}$ est une accélération stable pour $\mathcal{V}$.*

Preuve. Soit $r \overset{*}{\Rightarrow} t : (q, \mathsf{x}, P)$ un chemin symbolique. Rappelons que $\nabla^{\mathcal{V}}_{\mathfrak{T}_1}$ et $\nabla^{\mathcal{V}}_{\mathfrak{R}_2}$ sont stables. Par conséquent, pour montrer que $\nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{R}_2}$ est stable, nous devons prouver que $[\![q, \mathsf{x}, P \cup \{\mathsf{d}[2 := 0]\}]\!] \subseteq \mathrm{post}^*([\![l_r]\!])$, où $r : l_r$, lorsque les lignes 3–6 sont exécutées. Supposons donc que les lignes 3–6 sont exécutées pendant le calcul de $\nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{R}_2}(r \overset{*}{\Rightarrow} t)$, et soit $\mathsf{y} \in (\mathsf{x} + P^*)$. Nous montrons que $[\![q, \mathsf{y}, \{\mathsf{d}[2 := 0]\}]\!] \subseteq \mathrm{post}^*([\![l_r]\!])$, ce qui terminera la preuve.

Algorithme 10 $\nabla^{\mathcal{V}}_{\mathfrak{T}_1 \mathfrak{X}_2}(r \overset{*}{\Rightarrow} t : (q, \mathsf{x}, P))$

Entrée: un chemin symbolique $r \overset{*}{\Rightarrow} t$ pour un SAVE étendu $\mathcal{V}$ de dimension 2

1: **si** $r \overset{*}{\Rightarrow} t$ se termine par une opération $\alpha \in \mathtt{A} \cup \{\mathtt{test}(1)\}$ **alors**

2: **retourner** $\nabla^{\mathcal{V}}_{\mathfrak{T}_1}(r \overset{*}{\Rightarrow} t)$

3: **sinon si** $r \overset{*}{\Rightarrow} t$ s'écrit $r \overset{*}{\Rightarrow} t_1 \xRightarrow{\Gamma_1,\mathtt{test}(1)} t_2 \xRightarrow{\Gamma_2,\mathtt{add}(\mathsf{v})} t_3 \xRightarrow{\mathtt{A}^*} t_4 \xRightarrow{\Gamma_4,\mathtt{reset}(2)} t$ avec $\Gamma_2 \subseteq (\mathtt{A} \cup \{\mathtt{test}(1)\})$ et $\Gamma_4 \subseteq \mathtt{A}$ **alors**

4: **si** l'étiquette (q_2, x_2, P_2) de t_2 satisfait $q_2 \in \Omega(\mathcal{V}, [\![l_r]\!])$, où $r : l_r$ **alors**

5: **si** il existe $\mathsf{d} \in \mathbb{N}^* \times \mathbb{Z}$ et un noeud intermédiaire $u : (q_u, \mathsf{x}_u, P_u)$ entre t_3 et t_4 tel que $\mathsf{d} \in D_{\mathcal{V}}(q_u, (\mathsf{x}_u[1], \omega))$ **alors**

6: **retourner** $(\{\mathsf{d}[2 := 0]\}, \perp)$

7: {Si cette ligne est exécutée alors l'une des condition ci-dessus n'était pas vérifiée. Essayons d'accélérer avec $\nabla^{\mathcal{V}}_{\mathfrak{X}_2}$.}

8: **retourner** $\nabla^{\mathcal{V}}_{\mathfrak{X}_2}(r \overset{*}{\Rightarrow} t)$

9: **sinon**

10: **retourner** $\nabla^{\mathcal{V}}_{\mathfrak{X}_2}(r \overset{*}{\Rightarrow} t)$

Puisque les lignes 3–6 sont exécutées, le chemin symbolique $r \overset{*}{\Rightarrow} t$ s'écrit en fait $r \overset{*}{\Rightarrow} t_1 \xRightarrow{\Gamma_1,\mathtt{test}(1)} t_2 \xRightarrow{\Gamma_2,\mathtt{add}(\mathsf{v})} t_3 \xRightarrow{\mathtt{A}^*} t_4 \xRightarrow{\Gamma_4,\mathtt{reset}(2)} t$ avec $\Gamma_2 \subseteq (\mathtt{A} \cup \{\mathtt{test}(1)\})$ et $\Gamma_4 \subseteq \mathtt{A}$. Notons (q_i, x_i, P_i) l'étiquette de t_i pour tout $i \in [1..4]$. En analysant vers l'arrière le chemin symbolique $r \overset{*}{\Rightarrow} t$, on réalise que :

- puisque $\Gamma_4 \subseteq \mathtt{A}$ et $t_4 \xRightarrow{\Gamma_4,\mathtt{reset}(2)} t$, on obtient qu'il existe $\mathsf{y}_4 \in {P_4}^*$ tel que $(q_4, \mathsf{x}_4 + \mathsf{y}_4) \xrightarrow{\sigma_4\,\mathtt{reset}(2)} (q, \mathsf{y})$ pour un certain $\sigma_4 \in \mathtt{A}^*$.

- puisque $t_3 \xRightarrow{\mathtt{A}^*} u \xRightarrow{\mathtt{A}^*} t_1$, on obtient qu'il existe $\mathsf{y}_3 \in {P_3}^*$ et $\mathsf{y}_u \in {P_u}^*$ tel que $(q_3, \mathsf{x}_3 + \mathsf{y}_3) \xrightarrow{\sigma_3} (q_u, \mathsf{x}_u + \mathsf{y}_u) \xrightarrow{\sigma_u} (q_4, \mathsf{x}_4 + \mathsf{y}_4)$ pour des séquences d'opérations $\sigma_3, \sigma_u \in \mathtt{A}^*$.

- puisque $\Gamma_2 \subseteq (\mathtt{A} \cup \{\mathtt{test}(1)\})$ et $t_2 \xRightarrow{\Gamma_2,\mathtt{add}(\mathsf{v})} t_3$, on obtient qu'il existe $\mathsf{y}_2 \in {P_2}^*$ tel que $(q_2, \mathsf{x}_2 + \mathsf{y}_2) \xrightarrow{\sigma_2} (q_3, \mathsf{x}_3 + \mathsf{y}_3)$ pour un certain $\sigma_2 \in (\mathtt{A} \cup \{\mathtt{test}(1)\})^*$.

- puisque $t_1 \xRightarrow{\Gamma_1,\mathtt{test}(1)} t_2$, on obtient que $\mathsf{x}_2[1] = 0$ et $P_2 \subseteq \{0\} \times \mathbb{N}$. Par conséquent, $\mathsf{y}_2[1] = 0$ et on a $\mathsf{x}_2 + \mathsf{y}_2 = (0, z)$ où $z = \mathsf{x}_2[2] + \mathsf{y}_2[2]$.

On aboutit ainsi à :

$$(q_2, (0, z)) \xrightarrow{\sigma_2} (q_3, \mathsf{x}_3 + \mathsf{y}_3) \xrightarrow{\sigma_3} (q_u, \mathsf{x}_u + \mathsf{y}_u) \xrightarrow{\sigma_u} (q_4, \mathsf{x}_4 + \mathsf{y}_4) \xrightarrow{\sigma_4\,\mathtt{reset}(2)} (q, \mathsf{y})$$

Notons $\mathsf{x}'_3 = \mathsf{x}_3 + \mathsf{y}_3$, $\mathsf{x}'_u = \mathsf{x}_u + \mathsf{y}_u$ et $\mathsf{x}'_4 = \mathsf{x}_4 + \mathsf{y}_4$. D'après la ligne 5, on sait qu'il existe un circuit de contrôle simple χ valide en $(q_u, (\mathsf{x}_u[1], \omega))$, et de déplacement $\mathsf{d}(\chi) \in \mathbb{N}^* \times \mathbb{Z}$. Soit σ_χ la séquence d'opérations étiquetant χ, et notons $\mathsf{d}(\chi) = (d_1, d_2)$ avec $d_1 \in \mathbb{N}^*$ et $d_2 \in \mathbb{Z}$. Puisque χ est valide en $(q_u, (\mathsf{x}_u[1], \omega))$, on obtient que χ est valide en $(q_u, (\mathsf{x}'_u[1], \omega))$. Par conséquent, pour tout $k \in \mathbb{N}$, il existe $z'_k \in \mathbb{N}$ tel que χ est valide en $(q_u, (\mathsf{x}'_u[1] + i\, d_1, z'_k + i\, d_2))$

pour tout $i \in [0 \mathinner{..} k-1]$. Ainsi, on aboutit à :

$$\begin{array}{ccc} (q_2,(0,z+z'_k)) \xrightarrow{\sigma_2\,\sigma_3} (q_u,\mathsf{x}'_u+(0,z'_k)) & \xrightarrow{(\sigma_\chi)^k} & (q_u,\mathsf{x}'_u+(k\,d_1,z'_k+k\,d_2)) \\ & & \big\downarrow \sigma_u \\ (q,\mathsf{y}+(k\,d_1,0)) & \xleftarrow{\sigma_4\,\mathtt{reset}(2)} & (q_4,\mathsf{x}'_4+(k\,d_1,z'_k+k\,d_2)) \end{array}$$

Puisque $q_2 \in \Omega(\mathcal{V},[\![l_r]\!])$, on obtient que $(q,(0,z'')) \in \mathrm{post}^*([\![l_r]\!])$ pour un certain $z'' \geq z+z'_k$. Il vient $(q_2,(0,z'')) \xrightarrow{\sigma_2\,\sigma_3\,(\sigma_\chi)^k\,\sigma_u\,\sigma_4\,\mathtt{reset}(2)} (q,\mathsf{y}+(k\,d_1,0))$. On en conclut que $[\![q,\mathsf{y},\{\mathsf{d}[2:=0]\}]\!] \subseteq \mathrm{post}^*([\![l_r]\!])$. ∎

Lemme 3.7.3 *Pour tout SAVE étendu $\mathcal{V}$ de dimension 2, $\nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{R}_2}$ est à la fois $(\mathtt{A}\cup\{\mathtt{test}(1)\})$-couvrante et $(\mathtt{A}\cup\{\mathtt{reset}(2)\})$-couvrante.*

Preuve. Puisqu'on a $\nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{R}_2} \equiv \nabla^{\mathcal{V}}_{\mathfrak{T}_1}[\mathtt{A}\cup\{\mathtt{test}(1)\}]$, on déduit de la proposition 3.3.12 que $\nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{R}_2}$ est $(\mathtt{A}\cup\{\mathtt{test}(1)\})$-couvrante. Nous montrons maintenant que $\nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{R}_2}$ est $(\mathtt{A}\cup\{\mathtt{reset}(2)\})$-couvrante. Observons que malheureusement, $\nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{R}_2} \not\equiv \nabla^{\mathcal{V}}_{\mathfrak{R}_2}[\mathtt{A}\cup\{\mathtt{reset}(2)\}]$, à cause des lignes 3–6 de $\nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{R}_2}$. On ne peut donc pas utiliser la proposition 3.3.12 comme ci-dessus.

Soit $\pi = t_0 \xRightarrow{\Gamma_0,\alpha_0} t_1 \xRightarrow{\Gamma_1,\alpha_1} t_2 \cdots t_k \xRightarrow{\Gamma_k,\alpha_k} t_{k+1} \cdots$ une branche dans $\mathtt{SymbolicTree}(\mathcal{V}, r \overset{*}{\Rightarrow} t, \nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{R}_2})$, et supposons que π ne contienne que des opérations α dans $(\mathtt{A}\cup\{\mathtt{reset}(2)\})$. Puisque $\nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{R}_2}$ est $\mathtt{A}$-couvrante, on obtient que si π ne contient aucun $\mathtt{reset}(2)$ alors π est fini. Supposons maintenant que π contienne au moins un $\mathtt{reset}(2)$. Il existe $h \in \mathbb{N}$ tel que $\alpha_h = \mathtt{reset}(2)$. Notons $\pi_{h+2} = t_{h+2} \xRightarrow{\Gamma_{h+2},\alpha_{h+2}} t_{h+3} \cdots$ le suffixe de π commençant en t_{h+2}. Puisque π ne contient aucun $\mathtt{test}(1)$, on obtient que pour tout $i \geq h+2$, $r \overset{*}{\Rightarrow} t_i$ ne peut pas être écrit sous la forme donnée dans la ligne 3 de la définition de $\nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{R}_2}$. On en déduit que $\nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{R}_2}(r \overset{*}{\Rightarrow} t_i) = \nabla^{\mathcal{V}}_{\mathfrak{R}_2}(r \overset{*}{\Rightarrow} t_i)$ pour tout $i \geq h+2$. Par conséquent, π_{h+2} est une branche dans $\mathtt{SymbolicTree}(\mathcal{V}, r \overset{*}{\Rightarrow} t_{h+2}, \nabla^{\mathcal{V}}_{\mathfrak{R}_2})$, et on conclut que π_{h+2} (et donc aussi π) est finie. ∎

Notation. Pour tout contrôle de SAVE étendu $\mathsf{V} = (Q,\mathtt{Op},\rightarrow_{\mathsf{V}})$ de dimension 2, on définit $\theta(\mathsf{V})$ par $\theta(\mathsf{V}) = |Q| \cdot \mathrm{Max}\{|\mathsf{v}[1]| \;/\; \exists\, q,q' \in Q, q \xrightarrow{\mathtt{add}(\mathsf{v})}_{\mathsf{V}} q'\}$.

Remarque. Tout circuit de contrôle élémentaire simple χ dans un SAVE étendu $\mathcal{V}$ contient au plus Q transitions de contrôle, où Q est l'ensemble des états de contrôle de $\mathcal{V}$. Par conséquent, son déplacement $\mathsf{d}(\chi)$ vérifie $\mathsf{d}(\chi)[1] \leq \theta(\mathsf{V})$.

Lemme 3.7.4 *Soit* $\pi = (q_0,(x_0,y_0)) \xrightarrow{\mathtt{add}(\mathsf{v}_0)} (q_1,(x_1,y_1)) \xrightarrow{\mathtt{add}(\mathsf{v}_1)} (q_2,(x_2,y_2)) \cdots (q_{k-1},(x_{k-1},y_{k-1})) \xrightarrow{\mathtt{add}(\mathsf{v}_{k-1})} (q_k,(x_k,y_k))$ *un chemin dans $\mathcal{V}$. Supposons que :*

- *on a $q_k = q_0$, et,*
- *pour tout $i \in [1 \mathinner{..} k]$, on a $D_{\mathcal{V}}(q_i,(x_i,\omega)) \cap \mathbb{N}^* \times \mathbb{Z} = \emptyset$.*

Alors on a $x_k \leq x_0$.

Preuve. Par récurrence sur la longueur k de π. La base ($k = 0$) est triviale, aussi ne montrons-nous que l'étape d'induction. Soit $k \in \mathbb{N}^*$ et supposons que le lemme est vrai pour tout chemin de longueur strictement plus petite que k. Prouvons que le lemme est encore vrai pour tout chemin π de longueur k. Deux cas se présentent :

- si aucun état de contrôle ne se répète dans π, à part que $q_k = q_0$, alors on obtient que $\chi = q_0 \xrightarrow{\mathsf{add}(\mathsf{v}_0)} q_1 \xrightarrow{\mathsf{add}(\mathsf{v}_1)} q_2 \cdots q_{k-1} \xrightarrow{\mathsf{add}(\mathsf{v}_{k-1})} q_k$ est un circuit de contrôle élémentaire simple sur q_0. De plus, χ est valide en $(q_0, (x_0, y_0))$ puisque π est un chemin dans $\mathcal{V}$. Si on avait $x_k > x_0$ alors on obtiendrait que χ est aussi valide en $(q_k, (x_k, y_0))$. Comme $D_{\mathcal{V}}(q_k, (x_k, \omega)) \cap \mathbb{N}^* \times \mathbb{Z} = \emptyset$, on aboutirait à $x_k - x_0 = \mathsf{d}(\chi)[1] \leq 0$, une contradiction. Par conséquent, on a $x_k \leq x_0$.
- sinon, il existe $0 < i < j \leq k$ tel que :
  - on a $q_i = q_j$, et,
  - aucun état de contrôle ne se répète dans la séquence d'états de contrôle suivante : $q_0, q_1, \ldots, q_i, q_j, q_{j+1}, \ldots q_{k-1}$.

  Observons que ceci implique que π s'écrit $\pi = (q_0, (x_0, y_0)) \xrightarrow{+} (q_i, (x_i, y_i)) \xrightarrow{+} (q_j, (x_j, y_j)) \xrightarrow{*} (q_k, (x_k, y_k))$. Rappelons que pour tout $i \in [i + 1 \,..\, j]$, on a $D_{\mathcal{V}}(q_i, (x_i, \omega)) \cap \mathbb{N}^* \times \mathbb{Z} = \emptyset$. Ainsi, en appliquant l'hypothèse d'induction à $(q_i, (x_i, y_i)) \xrightarrow{+} (q_j, (x_j, y_j))$, dont la longueur est strictement plus petite que k, on obtient que $x_j \leq x_i$.

  Notons $d_x = x_i - x_j \geq 0$. En ajoutant $(d_x, 0)$ tout au long du chemin $(q_j, (x_j, y_j)) \xrightarrow{*} (q_k, (x_k, y_k))$, on obtient que $(q_j, (x_j + d_x, y_j)) \xrightarrow{*} (q_k, (x_k + d_x, y_k))$ est un chemin dans $\mathcal{V}$. Comme on a $q_i = q_j$ et $x_i = x_j + d_x$, on en déduit qu'il existe $y'_0, y'_1, \ldots, y'_i, y'_{j+1} \ldots, y'_k \in \mathbb{N}$, tels que π' défini ci-après est un chemin dans $\mathcal{V}$:

  $$\pi' = (q_0, (x_0, y'_0)) \xrightarrow{+} (q_i, (x_i, y'_i)) \xrightarrow{\mathsf{add}(\mathsf{v}_j)} (q_{j+1}, (x_{j+1} + d_x, y'_{j+1})) \xrightarrow{*} (q_k, (x_k + d_x, y'_k))$$

  Par conséquent, le circuit de contrôle élémentaire simple $\chi' = q_0 \xrightarrow{\mathsf{add}(\mathsf{v}_0)} q_1 \cdots q_i \xrightarrow{\mathsf{add}(\mathsf{v}_j)} q_{j+1} \cdots q_{k-1} \xrightarrow{\mathsf{add}(\mathsf{v}_{k-1})} q_k$ est valide en $(q_0, (x_0, y'_0))$. De plus, puisque π' défini ci-dessus est un chemin dans $\mathcal{V}$, il vient $\mathsf{d}(\chi')[1] = x_k + d_x - x_0$. Si on avait $x_k > x_0$ alors on aurait $\mathsf{d}(\chi')[1] > 0$ et on obtiendrait que χ est valide en $(q_k, (x_k, y'_0))$. Puisque $D_{\mathcal{V}}(q_k, (x_k, \omega)) \cap \mathbb{N}^* \times \mathbb{Z} = \emptyset$, on en déduirait que $\mathsf{d}(\chi')[1] \leq 0$, une contradiction. Par conséquent, on a $x_k \leq x_0$.

Dans les deux cas, on obtient que $x_k \leq x_0$. ∎

Lemme 3.7.5 *Soit $r \overset{*}{\Rightarrow} t_0 \xRightarrow{\Gamma_0, \mathsf{add}(\mathsf{v}_0)} t_1 \xRightarrow{\Gamma_1, \mathsf{add}(\mathsf{v}_1)} t_2 \cdots t_{k-1} \xRightarrow{\Gamma_{k-1}, \mathsf{add}(\mathsf{v}_{k-1})} t_k$ une branche dans* $\texttt{SymbolicTree}(\mathcal{V}, r \overset{*}{\Rightarrow} t, \nabla^{\mathcal{V}}_{\mathfrak{T}_1 \mathfrak{K}_2})$*, d'étiquettes $t_i : (q_i, \mathsf{x}_i, P_i)$. Supposons que :*

– *on a* $D_{\mathcal{V}}(q_i, (\mathsf{x}_i[1], \omega)) \cap \mathbb{N}^* \times \mathbb{Z} = \emptyset$ *pour tout* $i \in [1 .. k]$*, et,*

– *on a* $P_i \subseteq \{0\} \times \mathbb{N}$ *pour tout* $i \in [0 .. k]$.

Alors on a $\mathsf{x}_k[1] \leq \mathsf{x}_0[1]$ *si* $q_k = q_0$*, et* $\mathsf{x}_k[1] \leq \mathsf{x}_0[1] + \theta(\mathsf{V})$ *sinon.*

Preuve. D'après la définition de l'algorithme `SymbolicTree` (lignes 4–9), pour tout $i \in [0 .. k-1]$, on a $(q_i, \mathsf{x}_i + (0, y_i)) \xrightarrow{\mathtt{add}(\mathsf{v}_i)} (q_{i+1}, \mathsf{x}_{i+1})$ pour un certain $(0, y_i) \in {P_i}^*$. On en déduit qu'il existe $z_0, z_1, \ldots, z_k \in \mathbb{N}$ tels que $\pi = (q_0, (\mathsf{x}_0[1], z_0)) \xrightarrow{\mathtt{add}(\mathsf{v}_0)} (q_1, (\mathsf{x}_1[1], z_1)) \cdots (q_{k-1}, (\mathsf{x}_{k-1}[1], z_{k-1})) \xrightarrow{\mathtt{add}(\mathsf{v}_{k-1})} (q_k, (\mathsf{x}_k[1], z_k))$ est un chemin dans $\mathcal{V}$.

Maintenant si $q_k = q_0$ alors on déduit du lemme 3.7.4 que $\mathsf{x}_k[1] \leq \mathsf{x}_0[1]$. Sinon, il existe $0 < i \leq j \leq k$ tels que :

– on a $q_i = q_j$, et,

– $\chi = q_0 \xrightarrow{\mathtt{add}(\mathsf{v}_0)} q_1 \cdots q_i \xrightarrow{\mathtt{add}(\mathsf{v}_j)} q_{j+1} \cdots q_{k-1} \xrightarrow{\mathtt{add}(\mathsf{v}_{k-1})} q_k$ est un circuit de contrôle élémentaire simple.

Puisque $q_i = q_j$, on déduit du lemme 3.7.4 que $\mathsf{x}_j[1] \leq \mathsf{x}_i[1]$. Comme χ est un circuit de contrôle élémentaire simple, on obtient que $\mathsf{d}(\chi)[1] \leq \theta(\mathsf{V})$. On aboutit finalement à $\mathsf{x}_k[1] - \mathsf{x}_0[1] = \mathsf{x}_j[1] - \mathsf{x}_i[1] + \mathsf{d}(\chi)[1] \leq \theta(\mathsf{V})$. ∎

Lemme 3.7.6 *Soit* $r \overset{*}{\Rightarrow} t_1 \xRightarrow{\Gamma_1, \mathtt{test}(1)} t_2 \xRightarrow{\Gamma_2, \mathtt{add}(\mathsf{v}_2)} t_3 \xRightarrow{\mathtt{A}^*} t_k$ *une branche dans* `SymbolicTree`$(\mathcal{V}, r \overset{*}{\Rightarrow} t_1, \nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{K}_2})$*, d'étiquettes* $t_i : (q_i, \mathsf{x}_i, P_i)$*. Si on a* $D_{\mathcal{V}}(q_i, ((\mathsf{x}_i[1], \omega)) \cap \mathbb{N}^* \times \mathbb{Z} = \emptyset$ *pour tout* $i \in [3 .. k]$*, alors :*

– *on a* $\mathsf{x}_k[1] \leq \theta(\mathsf{V})$*, et,*

– *on a* $(P_k \cup \nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{K}_2}(r \overset{*}{\Rightarrow} t_k)) \subseteq \{0\} \times \mathbb{N}$.

Preuve. Observons tout d'abord que $\mathsf{x}_2[1] = 0$ et $P_2 \subseteq \{0\} \times \mathbb{N}$ parce que $t_1 \xRightarrow{\Gamma_1, \mathtt{test}(1)} t_2$ est un arc symbolique pour $\mathcal{V}$. Nous montrons par récurrence sur i que pour tout $i \in [2 .. k]$, on a $(P_i \cup \nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{K}_2}(r \overset{*}{\Rightarrow} t_i)) \subseteq \{0\} \times \mathbb{N}$. En effet, puisque $\mathsf{x}_2[1] = 0$ et en utilisant le lemme 3.7.5, ceci impliquera en particulier que $x_k[1] \leq \theta(\mathsf{V})$.

Pour montrer la base, supposons que $i = 2$. D'après la définition de $\nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{K}_2}$, on a $\nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{K}_2}(r \overset{*}{\Rightarrow} t_2) = \nabla^{\mathcal{V}}_{\mathfrak{T}_1}(r \overset{*}{\Rightarrow} t_2) \subseteq \{0\} \times \mathbb{N}$. On en déduit que $(P_2 \cup \nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{K}_2}(r \overset{*}{\Rightarrow} t_2)) \subseteq \{0\} \times \mathbb{N}$.

Montrons à présent l'étape d'induction et supposons que pour tout $2 \leq i < k$, on a $(P_i \cup \nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{K}_2}(r \overset{*}{\Rightarrow} t_i)) \subseteq \{0\} \times \mathbb{N}$. On déduit de la définition de l'algorithme `SymbolicTree` (lignes 4–9) que $P_k = (P_{k-1} \cup \nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{K}_2}(r \overset{*}{\Rightarrow} t_{k-1}))$ et donc $P_k \subseteq \{0\} \times \mathbb{N}$. Prouvons maintenant que $\nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{K}_2}(r \overset{*}{\Rightarrow} t_k)) \subseteq \{0\} \times \mathbb{N}$. D'après la définition de $\nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{K}_2}$, on a $\nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{K}_2}(r \overset{*}{\Rightarrow} t_k) = \nabla^{\mathcal{V}}_{HP}(r \overset{*}{\Rightarrow} t_k)$. Examinons en détails l'exécution de $\nabla^{\mathcal{V}}_{HP}(r \overset{*}{\Rightarrow} t_k)$:

– ligne 1 : puisque $D_\mathcal{V}(q_k, (\mathsf{x}_k[1], \omega)) \cap \mathbb{N}^* \times \mathbb{Z} = \emptyset$, on obtient immédiatement que $(D_\mathcal{V}(q_k, \mathsf{x}_k) \cap \mathbb{N}^2) \subseteq \{0\} \times \mathbb{N}$.

– lignes 3–4 et 8–9 : d'après le lemme 3.7.5, on obtient que pour tout $3 \leq i < k-1$ tel que $q_i = q_k$, on a $x_k[1] \leq x_i[1]$. Par conséquent, si la ligne 4 (resp. ligne 9) est exécutée alors on a $(\mathsf{x} - \mathsf{x}') \in \{0\} \times \mathbb{N}$.

– lignes 6–7 : puisque $P_k \subseteq \{0\} \times \mathbb{N}$ et d'après les cas ci-dessus, on en déduit que si la ligne 7 est exécutée alors on a $\mathsf{u} \in \{0\} \times \mathbb{N}$. De plus, on a $\mathsf{v} \in \{0\} \times \mathbb{N}$ car $D_\mathcal{V}(q_k, \mathsf{x}) \cap \mathbb{N}^* \times \mathbb{Z} = \emptyset$. Par conséquent, on obtient que $(c\mathsf{u} + d\mathsf{v}) \in \{0\} \times \mathbb{N}$.

On obtient finalement que $\nabla^\mathcal{V}_{HP}(r \stackrel{*}{\Rightarrow} t_k) \subseteq \{0\} \times \mathbb{N}$, et donc $P_k \subseteq \{0\} \times \mathbb{N}$, ce qui termine la preuve. ■

Proposition 3.7.7 *Pour tout SAVE étendu $\mathcal{V}$ de dimension 2, $\nabla^\mathcal{V}_{\mathfrak{T}_1\aleph_2}$ est une accélération stable et $(\mathtt{A} \cup \{\mathtt{test}(1), \mathtt{reset}(2)\})$-couvrante pour $\mathcal{V}$.*

Preuve. Nous savons déjà, d'après le lemme 3.7.2 que $\nabla^\mathcal{V}_{\mathfrak{T}_1\aleph_2}$ est stable. Il reste donc à prouver que $\nabla^\mathcal{V}_{\mathfrak{T}_1}$ est $(\mathtt{A} \cup \{\mathtt{test}(1)\})$-couvrante. Soit $\pi = t_0 \xRightarrow{\Gamma_0,\alpha_0} t_1 \xRightarrow{\Gamma_1,\alpha_1} t_2 \cdots t_k \xRightarrow{\Gamma_k,\alpha_k} t_{k+1} \cdots$ une branche dans $\mathtt{SymbolicTree}(\mathcal{V}, r \stackrel{*}{\Rightarrow} t_0, \nabla^\mathcal{V}_{\mathfrak{T}_1\aleph_2})$, d'étiquettes $t_i : (q_i, \mathsf{x}_i, P_i)$, et supposons que π ne contienne que des opérations α dans $(\mathtt{A} \cup \{\mathtt{test}(1), \mathtt{reset}(2)\})$. Prouvons par l'absurde que π est fini, et supposons donc que π est infini. Puisque π est infini, on déduit de la définition de l'algorithme $\mathtt{ArbreSymbolique}$ qu'aucun préfixe fini de π n'est auto-couvrant. Notre objectif est d'appliquer la proposition 3.3.3 afin de montrer que π admet un préfixe fini $t_0 \stackrel{*}{\Rightarrow} t_k$ auto-couvrant, ce qui terminera la preuve.

Rappelons que $\nabla^\mathcal{V}_{\mathfrak{T}_1\aleph_2}$ est à la fois $(\mathtt{A}\cup\{\mathtt{test}(1)\})$-couvrante et $(\mathtt{A}\cup\{\mathtt{reset}(2)\})$-couvrante (lemme 3.7.3). Par conséquent, il y a une infinité de $\mathtt{test}(1)$ et une infinité de $\mathtt{reset}(2)$ dans π. Comme il n'y a qu'un nombre fini d'états de contrôle, et puisque $\leq$ est un beau pré-ordre sur $\mathbb{N}^2$, on obtient qu'il existe une sous-suite infinie $t_{i_0} \stackrel{*}{\Rightarrow} t_{i'_0} \stackrel{*}{\Rightarrow} t_{i_1} \stackrel{*}{\Rightarrow} t_{i'_1} \cdots t_{i_j} \stackrel{*}{\Rightarrow} t_{i'_j} \cdots$ telle que les trois conditions suivantes soient satisfaites :

1. pour tout $j \in \mathbb{N}$, on a $\alpha_{i_j-1} = \mathtt{test}(1)$, $\alpha_{i'_j-1} = \mathtt{reset}(2)$ et $\alpha_i \in \mathtt{A}$ pour tout $i \in [i_j \,..\, i'_j - 2]$, et,
2. pour tout $j \in \mathbb{N}$, on a $q_{i_j} = q_{i_0}$ et $q_{i'_j} = q_{i'_0}$,
3. les deux sous-suites infinies $(\mathsf{x}_{i_j})_{j\in\mathbb{N}}$ et $(\mathsf{x}_{i'_j})_{j\in\mathbb{N}}$ sont croissantes.

D'après la définition de $\nabla^\mathcal{V}_{\mathfrak{T}_1\aleph_2}$, on obtient que pour tout $j \in \mathbb{N}$, le chemin symbolique $t_{i_j-1} \stackrel{*}{\Rightarrow} t_{i'_j}$ s'écrit comme suit :

$$t_{i_j-1} \xRightarrow{\Gamma_{i_j-1},\mathtt{test}(1)} t_{i_j} \xRightarrow{\Gamma_{i_j},\mathtt{add}(\mathsf{v})} t_{i_j+1} \xRightarrow{\mathtt{A}^*} t_{i'_j-1} \xRightarrow{\Gamma_{i'_j-1},\mathtt{reset}(2)} t_{i'_j}$$

avec $\Gamma_{i_j} \subseteq (\mathtt{A} \cup \{\mathtt{test}(1)\})$ et $\Gamma_{i'_j-1} \subseteq \mathtt{A}$. Observons que pour tout $j \in \mathbb{N}$, on a :

- $\mathsf{x}_{i_j}[1] = 0$ et $P_{i_j} \subseteq \{0\} \times \mathbb{N}$ parce que $t_{i_j-1} \xRightarrow{\Gamma_{i_j-1},\mathtt{test}(1)} t_{i_j}$ est un arc symbolique pour $\mathcal{V}$, et,
- $\mathsf{x}_{i'_j}[2] = 0$ et $P_{i'_j} \subseteq \mathbb{N} \times \{0\}$ parce que $t_{i'_j-1} \xRightarrow{\Gamma_{i'_j-1},\mathtt{reset}(2)} t_{i'_j}$ est un arc symbolique pour $\mathcal{V}$.

Commençons par montrer que $q_{i_0} \in \Omega(\mathcal{V}, S_0)$. Puisque $r \overset{*}{\Rightarrow} t_{i_j-1} \xRightarrow{\Gamma_{i_j-1},\mathtt{test}(1)} t_{i_j}$ est un chemin symbolique, on obtient que $[\![q_{i_j}, \mathsf{x}_{i_j}, P_{i_j}]\!] \subseteq \mathrm{post}(\mathrm{post}^*([\![l_r]\!]), \mathtt{test}(1))$. Deux cas se présentent :

- si pour tout $j \in \mathbb{N}$ on a $P_{i_j} \subseteq \{0\}$ alors on obtient que la sous-suite $(\mathsf{x}_{i_j})_{j\in\mathbb{N}}$ est *strictement* croissqnte, car pour tout $j \in \mathbb{N}$, le chemin $t_{i_0} \overset{*}{\Rightarrow} t_{i_j}$ n'est pas auto-couvrant. Puisque pour tout $j \in \mathbb{N}$, on a $\mathsf{x}_{i_j}[1] = 0$ et $(q_{i_j}, \mathsf{x}_{i_j}) \in \mathrm{post}(\mathrm{post}^*([\![l_r]\!]), \mathtt{test}(1))$, il vient $q_{i_j} = q_{i_0} \in \Omega(\mathcal{V}, S_0)$.
- sinon, il existe j tel que $P_{i_j} \not\subseteq \{0\}$. Comme $P_{i_j} \subseteq \{0\} \times \mathbb{N}$, on obtient qu'il existe $y \in \mathbb{N}^*$ tel que $(0, y) \in P_{i_j}$. Par conséquent, il vient $[\![q_{i_j}, \mathsf{x}_{i_j}, (0, y)]\!] \subseteq \mathrm{post}(\mathrm{post}^*([\![l_r]\!]), \mathtt{test}(1))$. On aboutit ainsi à $q_{i_j} = q_{i_0} \in \Omega(\mathcal{V}, S_0)$.

Nous montrons maintenant qu'il existe $\mathsf{p} \in \mathbb{N}^* \times \{0\}$ tel que pour une infinité de j, on a $\mathsf{p} \in \nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{R}_2}(r \overset{*}{\Rightarrow} t_{i'_j})$. Soit $j \in \mathbb{N}$. Deux cas se présentent à nouveau :

1. si pour tout $i \in [i_j + 1 \,..\, i'_j - 1]$ on a $D_{\mathcal{V}}(q_i, ((\mathsf{x}_i[1], \omega)) \cap \mathbb{N}^* \times \mathbb{Z} = \emptyset$, alors d'après le lemme 3.7.6, on obtient que :
   - on a $\mathsf{x}_{i'_j-1}[1] \leq \theta(\mathsf{V})$, et,
   - on a $(P_{i'_j-1} \cup \nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{R}_2}(r \overset{*}{\Rightarrow} t_{i'_j-1})) \subseteq \{0\} \times \mathbb{N}$.

   Par conséquent, on déduit de la définition de l'algorithme `SymbolicTree` que $\mathsf{x}_{i'_j} \in [0 \,..\, \theta(\mathsf{V})] \times \{0\}$ et $P_{i'_j} \subseteq \{0\}$.
2. sinon, il existe $i \in [i_j + 1 \,..\, i'_j - 1]$ tel que $D_{\mathcal{V}}(q_i, ((\mathsf{x}_i[1], \omega)) \cap \mathbb{N}^* \times \mathbb{Z} \neq \emptyset$. On déduit des lignes 3–6 de $\nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{R}_2}$ qu'il existe $\mathsf{p} \in [1 \,..\, \theta(\mathsf{V})] \times \{0\}$ tel que $\mathsf{p} \in \nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{R}_2}(r \overset{*}{\Rightarrow} t_{i'_j})$.

Observons que $[0 \,..\, \theta(\mathsf{V})] \times \{0\}$ est un ensemble fini. Puisque $t_{i_0} \overset{*}{\Rightarrow} t_{i'_j}$ n'est pas auto-couvrant, pour tout $j \in \mathbb{N}$, on obtient qu'on ne se trouve dans le cas (1) ci-dessus qu'au plus un nombre fini de fois. Par conséquent, on se trouve dans le cas (2) ci-dessus pour une infinité de j. Or il n'y a qu'un nombre fini de valeurs possibles pour p dans $[1 \,..\, \theta(\mathsf{V})] \times \{0\}$. On en déduit qu'il existe $\mathsf{p} \in \mathbb{N}^* \times \{0\}$ tel que $\mathsf{p} \in \nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{R}_2}(r \overset{*}{\Rightarrow} t_{i'_j})$ pour une infinité de j.

Comme il n'y a qu'un nombre fini de transitions de contrôle dans V, on obtient qu'il existe une transition de contrôle $q_{i'_0} \xrightarrow{\alpha} q$ et une extraction $\varphi : \mathbb{N} \to \mathbb{N}$ telles que pour tout $j \in \mathbb{N}$,

- on a $\mathsf{p} \in \nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{R}_2}(r \overset{*}{\Rightarrow} t_{i'_{\varphi(j)}})$, et,

– on a $\alpha_{i'_{\varphi(j)}} = \alpha$ et $q_{i'_{\varphi(j)}+1} = q$.

Afin de simplifier les notations pour la fin de la preuve, nous écrivons simplement j au lieu de $\varphi(j)$. En d'autres termes, nous supposons, sans perte de généralité, que la sous-suite initiale $t_{i_0} \overset{*}{\Rightarrow} t_{i'_0} \overset{*}{\Rightarrow} t_{i_1} \overset{*}{\Rightarrow} t_{i'_1} \cdots t_{i_j} \overset{*}{\Rightarrow} t_{i'_j} \cdots$ satisfaisait déjà les deux conditions ci-dessus.

Observons que pour tout $j \in \mathbb{N}$, on a $t_{i'_j} \xRightarrow{\Gamma_{i'_j},\alpha} t_{i'_j+1}$. Notons $W_{i'_j}$ l'ensemble calculé à la ligne 5 de $\texttt{SymbolicTree}(\mathcal{V}, r \overset{*}{\Rightarrow} t_{i'_j}, \nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{R}_2})$ et donné par $W_{i'_j} = P_{i'_j} \cup \nabla^{\mathcal{V}}_{\mathfrak{T}_1\mathfrak{R}_2}(r \overset{*}{\Rightarrow} t_{i'_j})$. On réalise que $\mathsf{p} \in W_{i'_j}$ et $W_{i'_j} \subseteq \mathbb{N} \times \{0\}$. Il reste trois cas à examiner pour terminer la preuve :

– si $\alpha = \texttt{test}(1)$ alors on obtient que $\mathsf{x}_{i'_j+1} = \mathsf{0}$ et $P_{i'_j+1} \subseteq \mathsf{0}$ pour tout $j \in \mathbb{N}$. Par conséquent, π admet un préfixe fini auto-couvrant.

– si $\alpha = \texttt{reset}(2)$ alors on obtient que $\mathsf{x}_{i'_j+1} = \mathsf{x}_{i'_j}$ et $P_{i'_j+1} = W_{i'_j}$ pour tout $j \in \mathbb{N}$. Rappelons que $\mathsf{p} \in W_{i'_j}$ et $W_{i'_j} \subseteq \mathbb{N} \times \{0\}$ pour tout $j \in \mathbb{N}$. On en déduit que $\{\mathsf{p}\} \subseteq P_{i'_j+1} \subseteq \widehat{\{\mathsf{p}\}}$ et $\mathsf{x}_{i'_j+1} \in (\mathsf{x}_{i'_0+1} + \widehat{\{\mathsf{p}\}})$. D'après la proposition 3.3.3, on obtient que π admet un préfixe fini auto-couvrant.

– si $\alpha = \texttt{add}(\mathsf{v})$ alors on obtient que pour tout $j \in \mathbb{N}$, on a $P_{i'_j+1} = W_{i'_j}$ et $\mathsf{x}_{i'_j+1} \in \mathrm{Min}(((\mathsf{x}_{i'_j} + W_{i'_j}{}^*) + \mathsf{v}) \cap \mathbb{N}^2)$. Comme $(\mathsf{x}_{i'_j} + W_{i'_j}{}^*) \subseteq \mathbb{N} \times \{0\}$, on obtient que $\mathsf{v}[2] \geq 0$ et donc on aboutit à $\mathsf{x}_{i'_j+1}[2] = \mathsf{v}[2]$ pour tout $j \in \mathbb{N}$. De plus, puisque $\leq$ est un ordre total sur $\mathbb{N}$, il existe $h \in \mathbb{N}$ tel que $\mathsf{x}_{i'_j+1}[2] \geq \mathsf{x}_{i'_h+1}[2]$ pour tout $j \geq h$. Rappelons que $\mathsf{p} \in W_{i'_j}$ et $W_{i'_j} \subseteq \mathbb{N} \times \{0\}$ pour tout $j \in \mathbb{N}$. Par conséquent, on a $\{\mathsf{p}\} \subseteq P_{i'_j+1} \subseteq \widehat{\{\mathsf{p}\}}$ et $\mathsf{x}_{i'_j+1} \in (\mathsf{x}_{i'_h+1} + \widehat{\{\mathsf{p}\}})$ pour tout $j \geq h$. D'après la proposition 3.3.3, on obtient que π admet un préfixe fini auto-couvrant.

Dans les trois cas ci-dessus, on obtient que π admet un préfixe fini auto-couvrant, ce qui achève la preuve. ■

Puisque la classe $\mathfrak{T}_1\mathfrak{R}_{1,2}\mathfrak{Tr}_{12}$ est effectivement simulable par la classe $\mathfrak{T}_1\mathfrak{R}_2$ (théorème 3.4.1), on déduit de la proposition précedente que :

Théorème 3.7.8 *Pour tout SAVE étendu dans la classe $\mathfrak{T}_1\mathfrak{R}_{1,2}\mathfrak{Tr}_{12}$, muni d'un ensemble linéaire S_0 d'états initiaux,* $\mathrm{post}^*(S_0)$ *est semilinéaire. Si de plus $\Omega(\mathcal{V}, S_0)$ est effectivement calculable, alors* $\mathrm{post}^*(S_0)$ *est constructible.*

Comme la classe $\mathfrak{R}_{1,2}\mathfrak{Tr}_{12,21}$ est effectivement simulable par la classe $\mathfrak{T}_1\mathfrak{R}_{1,2}\mathfrak{Tr}_{12}$ (théorème 3.4.1), on déduit du théorème précédent que :

Corollaire 3.7.9 *Pour tout SAVE étendu dans la classe $\mathfrak{R}_{1,2}\mathfrak{Tr}_{12,21}$ muni d'un ensemble linéaire S_0 d'états initiaux,* $\mathrm{post}^*(S_0)$ *est semilinéaire.*

3.8 Effective semilinéarité de post* pour la classe $\mathfrak{R}_{1,2}\mathfrak{Tr}_{12,21}$

Nous avons montré dans les sections précédentes que pour la classe $\mathfrak{R}_{1,2}\mathfrak{Tr}_{12,21}$, d'une part pre* est effectivement semilinéaire et d'autre part post* est semilinéaire. Remarquons que ces deux résultats ne montrent pas que post* est *effectivement* semilinéaire pour cette classe. En effet, pour la classe des SAVE avec perte étendus par des tests-à-zéro, introduite et analysée dans [BM99], pre* est effectivement semilinéaire, post* est semilinéaire, mais post* n'est pas constructible (même en dimension 3) car le problème du caractère borné est indécidable pour cette classe.

Nous prouvons dans cette section que post* est *effectivement* semilinéaire pour la classe $\mathfrak{R}_{1,2}\mathfrak{Tr}_{12,21}$. Comme conséquence immédiate de ce résultat, on obtient que post* est constructible pour la classe des SAVE avec perte étendus par des tests-à-zéro, de dimension 2.

3.8.1 SAVE avec reset/transfert

Un *SAVE avec reset/transfert* (resp. un *SAVE avec reset*) est un SAVE étendu, dont l'ensemble d'opérations `Op` satisfait $\mathtt{Op} \subseteq \mathtt{A}(C) \cup \mathtt{R}(C) \cup \mathtt{Tr}(C)$ (resp. $\mathtt{Op} \subseteq \mathtt{A}(C) \cup \mathtt{R}(C)$). Ainsi, $\mathfrak{R}_{1,2}\mathfrak{Tr}_{12,21}$ est la classe des SAVE avec reset/transfert de dimension 2 ; et $\mathfrak{R}_{1,2}$ est la classe des SAVE avec reset de dimension 2.

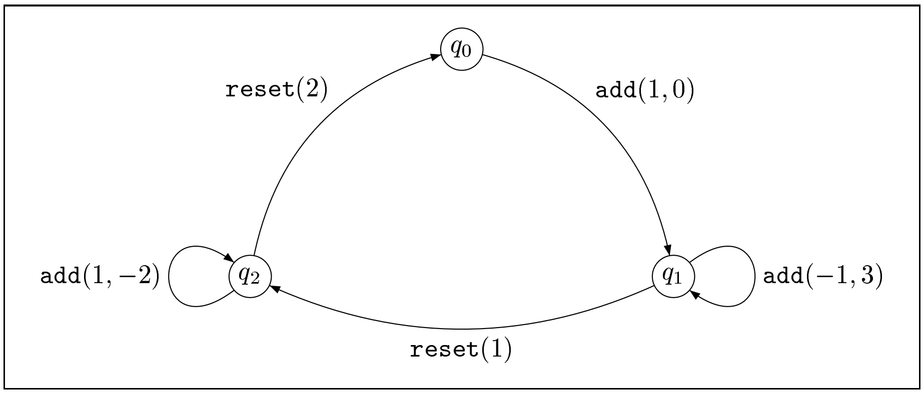


Fig. 3.5 – Un SAVE avec reset de dimension 2

Dans la suite de cette section, pour tout compteur $c \in [1..2]$, nous désignerons par $\overline{c}$ l'autre compteur. Plus précisément, $\overline{c}$ est l'unique élément de $[1..2] \setminus \{c\}$.

Rappelons que $\mathbb{N}_\omega$ (resp. $\mathbb{Z}_\omega$) désigne la fermeture classique $\mathbb{N} \cup \{\omega\}$ (resp. $\mathbb{Z} \cup \{\omega\}$) de l'ensemble $\mathbb{N}$ (resp. $\mathbb{Z}$), ou $v < \omega$ pour tout $v \in \mathbb{Z}$. Rappelons aussi que l'addition est étendue sur $\mathbb{Z}_\omega$ par $v + \omega = \omega + v = \omega$ pour $v \in \mathbb{Z}_\omega$. On étend également le produit sur $\mathbb{N}_\omega$ par $0 \cdot \omega = \omega \cdot 0 = 0$, et par $u \cdot \omega = \omega \cdot u = \omega$ pour tout $u \in \mathbb{N}_\omega \setminus \{0\}$. Ces opérations se prolongent naturellement sur les vecteurs, en traitant chaque composante indépendamment. L'ordre partiel $\leq$ sur $\mathbb{Z}^2$ (resp. $\preceq$ sur $Q \times \mathbb{N}^2$) se prolonge naturellement sur $\mathbb{Z}_\omega^2$ (resp. sur $Q \times \mathbb{N}_\omega^2$), tout en restant un ordre partiel.

3.8.2 Calcul de l'ω-base minimale de l'ensemble de couverture

Nous montrons dans cette section que l'ω-base minimale de l'ensemble de couverture d'un SAVE avec reset/transfert de dimension 2 est calculable. Les ω-bases minimales pour les ensemble clos inférieurement d'états de SAVE étendus ont été introduite en sous-section 3.2.1. Nous commençons par montrer ce résultat pour les SAVE avec reset de dimension 2, puis nous l'étendrons au cas des SAVE avec reset/transfert de dimension 2.

Etats paramétrés et schéma de chemins

Nous présentons dans ce paragraphe les notions d'*états paramétrés* et de *schéma de chemins*, qui nous permettront de calculer l'ensemble de couverture des SAVE avec reset de dimension 2. Cette présentation est fortement inspirée de l'analyse des réseaux de Petri avec reset de [DJS99] et de [DFJS00]. Nous avons adapté légèrement quelques définitions à notre contexte.

Pour tout $n \in \mathbb{N}$ et pour tout état $(q, \mathsf{x}), (q', \mathsf{x}')$ d'un SAVE avec reset $\mathcal{V}$ de dimension 2, on écrit $(q, \mathsf{x}) \xrightarrow{*}_n (q', \mathsf{x}')$ lorsqu'il existe un chemin allant de (q, x) à (q', x') et contenant au plus n transitions de remise-à-zéro. On écrit aussi $(q, \mathsf{x}) \xrightarrow{*\,\mathtt{reset}(c)}_n (q', \mathsf{x}')$ lorsqu'il existe un chemin se terminant par un $\mathtt{reset}(c)$ allant de (q, x) à (q', x') et contenant au plus n transitions de remise-à-zéro (observons qu'on a alors nécessairement $n \geq 1$).

Un *état paramétré* $s = \langle q, c \rangle$ est un état de contrôle q muni d'un compteur $c \in [1\,..\,2]$. Pour tout $x \in \mathbb{N}$, on désigne par $s(x)$ l'état $(q, \mathsf{0}[c := x])$ de $\mathcal{V}$.

Notation. Tout chemin $\pi = (q_0, \mathsf{x}_0) \xrightarrow{\alpha_0} (q_1, \mathsf{x}_1) \xrightarrow{\alpha_1} (q_2, \mathsf{x}_2) \cdots (q_k, \mathsf{x}_k) \xrightarrow{\alpha_k} (q_{k+1}, \mathsf{x}_{k+1})$ dans $\mathcal{V}$ s'écrit aussi plus simplement $\pi = (q_0, \mathsf{x}_0) \xrightarrow{\sigma} (q_{k+1}, \mathsf{x}_{k+1})$ où $\sigma = q_0 \xrightarrow{\alpha_0} q_1 \xrightarrow{\alpha_1} q_2 \cdots q_k \xrightarrow{\alpha_k} q_{k+1}$ (observons que σ est un chemin dans V).

Définition 3.8.1 *Un* schéma de chemins d'ordre n *(plus simplement un* schéma de chemins*) associé à un état paramétré $s = \langle q, c \rangle$ est un triplet (σ, f, x_0), où $\sigma : \mathbb{N} \rightarrow (\rightarrow_{\mathsf{V}})^+$ et $f : \mathbb{N} \rightarrow \mathbb{N}$ sont des fonctions partielles, où $x_0 \in \mathbb{N}$, et satisfaisant pour tout $x \geq x_0$ les trois conditions suivantes :*

1. *on a $s(x) \xrightarrow{\sigma(x)} s(f(x))$, et,*
2. *$\sigma(x)$ contient au plus n transitions de contrôle de reset, et,*
3. *$\sigma(x)$ se termine par une transition de contrôle de reset $p \xrightarrow{\mathtt{reset}(\overline{c})}_{\mathsf{V}} q$.*

Un schéma de chemins (σ, f, x_0) est maximal *si pour tout $x \geq x_0$ il n'existe pas de $y > f(x)$ tel que $s(x) \xrightarrow{*\,\mathtt{reset}(\overline{c})}_n s(y)$.*

Observons que les domaines de définition de f et de σ, respectivement $\mathrm{Dom}(f)$ et $\mathrm{Dom}(\sigma)$, contiennent nécessairement l'ensemble $\{x \in \mathbb{N} \ / \ x \geq x_0\}$. Remarquons également que pour tout schéma de chemins (σ, f, x_0) maximal, f est croissante.

Définition 3.8.2 *Un état paramétré $s = \langle q, c\rangle$ est n-sensé s'il existe $x_0 \in \mathbb{N}$ tel que l'ensemble $V(s,x) = \{y \in \mathbb{N} \ / \ s(x) \xrightarrow{*\,\texttt{reset}(\overline{c})}_n s(y)\}$ est fini, pour tout $x \geq x_0$.*

On se rend compte qu'un état paramétré s est n-sensé si et seulement s'il existe un schéma de chemins d'ordre n maximal associé à s. Nous utiliserons dans la suite la notion de schéma de chemins *régulier*, introduite dans [DJS99].

Définition 3.8.3 ([DJS99]) *Une fonction $f : \mathbb{N} \to \mathbb{N}$ est d-régulière, pour $d \in \mathbb{N}$, s'il existe $\rho \in \mathbb{Q}^+$ et une fonction $h : [0 \mathinner{..} d-1] \to \mathbb{Q}$ telles que $f(x) = \rho x + h(x \mod d)$ pour tout $x \in \mathrm{Dom}(f)$.*

Définition 3.8.4 ([DJS99]) *Un schéma de chemins (σ, f, x_0) est régulier si f est d-régulière pour un certain d, et si pour chaque $i \in [0 \mathinner{..} d-1]$, il existe $m \in \mathbb{N}$, $u_1, v_1, u_2, v_2, \ldots, u_m, v_m, u_{m+1} \in (\rightarrow_{\mathsf{V}})^*$ et m fonctions $g_1, g_2, \ldots, g_m$ d-régulières tels que pour tout $x \geq x_0$, si $x \mod d = i$ alors on a $\sigma(x) = u_1 v_1^{g_1(x)} u_2 v_2^{g_2(x)} \cdots u_m v_m^{g_m(x)} u_{m+1}$.*

D'après [DJS99], l'ensemble $RPS(s)$ des schéma de chemins réguliers associés à un état paramétré s est encodable. En d'autres termes, pour tout état paramétré s, on peut énumérer $RPS(s)$. On déduit aisément de [DJS99] que :

Lemme 3.8.5 *Soit s un état paramétré et $n \in \mathbb{N}$. Il existe un schéma de chemins régulier et maximal d'ordre n associé à s si et seulement si s est n-sensé.*

Arbre de couverture pour les SAVE avec reset de dimension 2

Définition 3.8.6 *Un* état paramétré non borné *est un couple (s, x_0) où s est un état paramétré, $x_0 \in \mathbb{N}$ et tel qu'il existe un schéma de chemins régulier (σ, f, x_0) associé à s satisfaisant $\forall\, x \geq x_0,\ f(x) > x$.*

D'après le lemme suivant, pour toute fonction d-régulière f, donnée par $f(x) = \rho x + h(x \mod d)$, la propriété $\forall\, x \geq x_0,\ f(x) > x$ est trivialement décidable.

Lemme 3.8.7 *Soit $x_0 \in \mathbb{N}$ et soit f une fonction d-régulière, donnée par $f(x) = \rho x + h(x \mod d)$. Supposons que $\mathrm{Dom}(f)$ contient l'ensemble $\{x \in \mathbb{N} \ / \ x \geq x_0\}$. Les deux assertions suivantes sont équivalentes :*

- *i) pour tout $x \geq x_0$, on a $f(x) > x$*
- *ii) $\rho \geq 1$ et pour tout $x \in [x_0 \mathinner{..} x_0 + d - 1]$, on a $f(x) > x$*

Preuve. Le sens $i) \Rightarrow ii)$ est immédiat, donc prouvons le sens $ii) \Rightarrow i)$. Supposons que $ii)$ est vraie et soit $x \geq x_0$. Il existe $k \in \mathbb{N}$ et $y \in [x_0 \mathinner{..} x_0 + d - 1]$ tels que $x = y + kd$. On a $f(x) = \rho x + h(x \mod d) = \rho y + \rho kd + h(y \mod d) = f(y) + \rho kd > y + kd = x$. ∎

Afin de calculer une ω-base pour l'ensemble de couverture d'un SAVE avec reset de dimension 2, nous exécutons *plusieurs* algorithmes *en parallèle*, comme décrit par l'algorithme `OmegaBaseEC`. D'un côté, avec plusieurs instances de l'algorithme `SchemaCheminsReguliers`, nous énumérons les schéma de chemins réguliers afin de trouver des états paramétrés non bornés. D'un autre côté, nous développons en parallèle un arbre de couverture semblable à celui de Karp et Miller, mais utilisant la liste (courante) des états paramétrés non bornés (voir l'algorithme `ArbreCouverture`).

Algorithme 11 `SchemaCheminsReguliers`$(\mathcal{V}, s, \texttt{var}\, L)$

Entrée: un SAVE avec reset $\mathcal{V}$ de dimension 2, un état paramétré s et une liste L d'états paramétrés non bornés

pour chaque schéma de chemins régulier (σ, f, x_0) associé à s **faire**
  si $\forall\, x \geq x_0,\ f(x) > x$ **alors**
    ajouter (s, x_0) à L
    retourner

Notation. Nous utilisons la notation suivante pour l'arbre construit par l'algorithme `ArbreCouverture` : $t \overset{\alpha}{\Longrightarrow} t'$ (ou plus simplement $t \Rightarrow t'$) désigne un arc, $t \overset{*}{\Rightarrow} t'$ désigne un chemin et $t \overset{*}{\Rightarrow}_n t'$ désigne un chemin avec au plus n arcs de reset.

L'algorithme `ArbreCouverture` construit un arbre en largeur d'abord (voir la ligne 3 de l'algorithme). Les lignes 11–13 de l'algorithme proviennent de la constatation suivante : si (s, x_0) est un état paramétré non borné alors il existe une fonction $f : \mathbb{N} \to \mathbb{N}$ telle que pour tout $x \geq x_0$,

– on a $\lim_{i\to+\infty} f^i(x) = +\infty$, et,

– on a $s(f^i(x)) \in \mathrm{post}^*(s(x))$ pour tout $i \in \mathbb{N}$.

Nous allons avoir besoin de quelques lemme techniques pour prouver que l'algorithme `ArbreCouverture` termine et calcule une ω-base de l'ensemble de couverture.

Lemme 3.8.8 *Soit f une fonction croissante d-régulière et soit $x_0 \in \mathbb{N}$. Supposons que* $\mathrm{Dom}(f)$ *contient l'ensemble* $\{x \in \mathbb{N} \ /\ x \geq x_0\}$. *Les deux assertions suivantes sont équivalentes :*

i) pour tout $x \geq x_0$, on a $f(x) > x$

ii) pour une infinité de x, on a $f(x) > x$, et il existe une suite finie strictement croissante $x_0 < x_1 < x_2 \cdots x_{d-1} < x_d$ dans $\mathbb{N}$ telle que $f(x_i) \geq x_{i+1}$ pour tout $i \in [0 \mathinner{..} d-1]$

Preuve. Le sens $i) \Rightarrow ii)$ est immédiat, donc prouvons le sens $ii) \Rightarrow i)$. Supposons que $ii)$ est vraie et que f est donnée par $f(x) = \rho x + h(x \mod d)$. D'après le lemme 3.8.7, il suffit de montrer que $\rho \geq 1$ et que pour tout $x \in [x_0 \mathinner{..} x_0{+}d{-}1]$, on a $f(x) > x$. Puisqu'on $f(x) > x$ pour une infinité de x, on obtient que $\rho \geq 1$. Soit $x \in [x_0 \mathinner{..} x_0 + d - 1]$. Il existe $k \in [0 \mathinner{..} d-1]$ tel que $x_k \leq x < x_{k+1}$. Puisque f est croissante, il vient $f(x) \geq f(x_k) \geq x_{k+1} > x$. ∎

Algorithme 12 $\texttt{ArbreCouverture}(\mathcal{V},(q_0,\mathsf{x}_0),\texttt{var}\ L)$

Entrée: un SAVE avec reset $\mathcal{V}$ de dimension 2, un état initial (q_0,x_0) et une liste L d'état paramétrés non bornés

1: créer la racine étiquetée par (q_0,x_0)
2: **tant que** il reste des noeuds non marqués **faire**
3: choisir un noeud non marqué $\mathtt{t}:(q,\mathsf{u})$ parmi ceux qui sont le moins profonds
4: **si** il existe un noeud marqué $\mathtt{t}':(q,\mathsf{u}')$ tel que $\mathsf{u}'\geq\mathsf{u}$ **alors**
5: passer {on saute à la ligne 19}
6: **sinon**
7: **si** il existe un ancêtre $\mathtt{t}':(q,\mathsf{u}')$ de t tel que $\mathsf{u}>\mathsf{u}'$ et $t'\overset{*}{\Rightarrow}_0 t$ **alors**
8: {Observons qu'il n'y a aucun reset entre t' et t}
9: $\mathsf{u}\leftarrow\mathsf{u}+\omega\cdot(\mathsf{u}-\mathsf{u}')$
10: **sinon**
11: **pour chaque** $c\in[1\,..\,2]$ **faire**
12: **si** le chemin de la racine à t se termine par un $\mathtt{reset}(\overline{c})$ et s'il existe $x_0\leq\mathsf{u}[c]$ tel que $(\langle q,c\rangle,x_0)\in L$ **alors**
13: $\mathsf{u}\leftarrow 0[c:=\omega]$
14: {Nous créons maintenant des fils pour les successeurs immédiats de (q,u)}
15: **pour chaque** transition $q\xrightarrow{\mathtt{add(v)}}q'$ dans V tel que $\mathsf{u}+\mathsf{v}\geq\mathsf{0}$ **faire**
16: construire un fils $\mathtt{t}':(q',\mathsf{u}+\mathsf{v})$ de t et étiqueter l'arc $t\xRightarrow{\mathtt{add(v)}}t'$
17: **pour chaque** transition $q\xrightarrow{\mathtt{reset}(c)}q'$ dans V **faire**
18: construire un fils $\mathtt{t}':(q',\mathsf{u}[c:=0])$ de t et étiqueter l'arc $t\xRightarrow{\mathtt{reset}(c)}t'$
19: marquer t

Pour tout alphabet Σ et pour tout $n\in\mathbb{N}$, on désigne par $\Sigma^{\leq n}$ l'ensemble $\{w\in\Sigma^*\ /\ |w|\leq n\}$.

Lemme 3.8.9 *Soit Σ un alphabet et soit $(w_i)_{i\in\mathbb{N}}$ une suite infinie de mots dans Σ^* telle que $\lim_{i\to\infty}|w_i|=+\infty$. Il existe $a\in\Sigma$ et $n\in\mathbb{N}$ tels que pour tout $m\in\mathbb{N}$, on a $w_{k_m}\in\Sigma^*(a\Sigma^{\leq n})^m a\Sigma^*$ pour un certain $k_m\in\mathbb{N}$.*

Preuve. Par récurrence sur $|\Sigma|$. La base ($|\Sigma|=1$) est triviale, aussi ne montrons-nous que l'étape d'induction. Soit Σ un alphabet tel que $|\Sigma|\geq 2$ et supposons que le lemme est vrai pour tout alphabet Σ' tel que $|\Sigma'|=|\Sigma|-1$. Choisissons un symbole $a\in\Sigma$. Si ce choix de a prouve le lemme, alors la preuve est terminée ! Sinon, pour tout $i\in\mathbb{N}$, il existe m_i tel que $w_j\notin\Sigma^*(a\Sigma^{\leq i})^{m_i}a\Sigma^*$ pour tout $j\in\mathbb{N}$. Rappelons que $\lim_{j\to\infty}|w_j|=\infty$. Par Conséquent, pour tout $i\in\mathbb{N}$, il existe $k_i\in\mathbb{N}$, tel que w_{k_i} contient un facteur[2] z_i tel que $|z_i|>i$ et $z_i\in(\Sigma\backslash\{a\})^*$. Ainsi, on construit une suite infinie $(z_i)_{i\in\mathbb{N}}$ dans $(\Sigma\backslash\{a\})^*$ telle que $\lim_{i\to\infty}|z_i|=+\infty$. Nous pouvons maintenant appliquer l'hypothèse de récurrence à (z_i), ce qui conclut la preuve puisque pour tout $i\in\mathbb{N}$, z_i est un facteur de w_{k_i}. ■

Lemme 3.8.10 *Soit $\mathtt{t}:(q,\mathsf{u})$ et $\mathtt{t}':(q,\mathsf{u}')$ deux noeuds distincts dans l'arbre construit par l'algorithme* `ArbreCouverture` *tels que $t'\overset{*}{\Rightarrow}t$. Si t a un fils alors on a $\mathsf{u}\not\leq\mathsf{u}'$.*

[2]Un *facteur* d'un mot $w\in\Sigma^*$ est un mot $w'\in\Sigma^*$ tel que $w=w_1w'w_2$ pour des mots $w_1,w_2\in\Sigma^*$

Preuve. Par l'absurde, supposons que t a un fils et que $\mathsf{u} \leq \mathsf{u}'$. Examinons le traitement de t (puisque t a un fils, t a nécessairement été traité). Observons que l'étiquette de t croît durant le traitement de t. Notons u_4 l'étiquette de t à la ligne 4 du traitement de t. On a $\mathsf{u}_4 \leq \mathsf{u} \leq \mathsf{u}'$. De plus, puisque $t' \overset{*}{\Rightarrow} t$, on obtient que t' a été traité avant t. Par conséquent, les lignes 7–18 n'ont pas pu être exécutées durant le traitement de t, ce qui conduit à une contradiction puisque t a un fils. ∎

Lemme 3.8.11 *Tout chemin infini dans l'arbre construit par l'algorithme* `ArbreCouverture` *contient une infinité de resets.*

Preuve. Soit $\pi = t_0 \Rightarrow t_1 \Rightarrow t_2 \cdots t_i \Rightarrow t_{i+1} \cdots$ un chemin infini dans l'arbre construit par l'algorithme `ArbreCouverture`, d'étiquettes $t_i : (q_i, \mathsf{u}_i)$. S'il n'y a qu'un nombre fini de resets le long de π, alors il existe $h \in \mathbb{N}$ tel que le suffixe π_h de π commençant en t_h ne contienne aucun reset. Puisqu'il n'y a aucun reset dans π_h, on obtient que pour tout $j > i \geq h$ et $c \in [1 \mathinner{..} 2]$, si $\mathsf{u}_i[c] = \omega$ alors $\mathsf{u}_j[c] = \omega$. Par conséquent, il existe $h' \geq h$ tel que pour tout $i > h'$ et $c \in [1 \mathinner{..} 2]$, on a $\mathsf{u}_i[c] = \omega$ si et seulement si $\mathsf{u}_{h'}[c] = \omega$. Observons que la ligne 9 de l'algorithme `ArbreCouverture` ajoute nécessairement un ω à l'étiquette du noeud en train d'être traité. Puisque π_h ne contient que des opérations d'addition, on en déduit que la ligne 9 n'est jamais exécutée pendant le traitement des noeuds t_i pour $i > h'$.

D'après le lemme de Dickson, il existe $k > j > h'$ tels que $q_k = q_j$ et $\mathsf{u}_k \geq \mathsf{u}_j$. Comme t_k a un fils, on déduit du lemme 3.8.10 que $\mathsf{u}_k > \mathsf{u}_j$. Puisque la ligne 9 n'est pas exécutée pendant le traitement de t_k, on obtient que l'étiquette de t_k n'est pas modifiée durant sont traitement. Par conséquent, on déduit des lignes 7–9 que pendant le traitement de t_k, la ligne 9 aurait du être exécutée, et on aboutit ainsi à une contradiction. ∎

Lemme 3.8.12 *Soit $\mathcal{V}$ un SAVE avec reset de dimension* 2, *muni d'un état initial (q_0, x_0), et notons T l'arbre construit par* `ArbreCouverture`$(\mathcal{V}, (q_0, \mathsf{x}_0), \texttt{var}\, L)$, *pour une certaine liste (variable) L. Pour tout état (q, x) tel que $(q_0, \mathsf{x}_0) \xrightarrow{*}_n (q, \mathsf{x})$, il existe un noeud $t : (q, \mathsf{u})$ dans T tel que $\mathsf{x} \leq \mathsf{u}$ et $r \overset{*}{\Rightarrow}_n t$, où r est la racine de T.*

Preuve. Par récurrence sur la longueur du chemin $(q_0, \mathsf{x}_0) \xrightarrow{*}_n (q, \mathsf{x})$. ∎

Rappelons que la fonction $[\![\cdot]\!] : Q \times \mathbb{N}_\omega^C \to \mathcal{P}(Q \times \mathbb{N}^C)$ est donnée dans la définition 3.2.1 de la page 78.

Lemme 3.8.13 *Soit $\mathcal{V}$ un SAVE avec reset de dimension* 2, *muni d'un état initial (q_0, x_0), et notons T l'arbre construit par* `ArbreCouverture`$(\mathcal{V}, (q_0, \mathsf{x}_0), \texttt{var}\, L)$, *pour une certaine liste (variable) L d'états paramétrés non bornés. Pour tout noeud $t : (q, \mathsf{u})$ dans T, on a $[\![(q, \mathsf{u})]\!] \subseteq \mathrm{post}^*(q_0, \mathsf{x}_0)$.*

Preuve. Par récurrence sur la profondeur d'un noeud $t : (q, \mathsf{u})$. ∎

Algorithme 13 `OmegaBaseEC`$(\mathcal{V}, (q_0, \mathsf{x}_0))$

Entrée: un SAVE avec reset $\mathcal{V}$ de dimension 2 et un état initial (q_0, x_0)

1: $L \leftarrow \emptyset$
2: **pour chaque** état paramétré s **faire**
3: {On lance un nouveau processus d'énumération des schéma de chemins réguliers associés à s}
4: **démarrer** `SchemaCheminsReguliers`$(\mathcal{V}, s, \texttt{var}\ L)$
5: {On exécute en parallèle l'algorithme `ArbreCouverture`}
6: **démarrer** `ArbreCouverture`$(\mathcal{V}, (q_0, \mathsf{x}_0), \texttt{var}\ L)$
7: **attendre que** `ArbreCouverture` termine
8: **arrêter** tous les processus `SchemaCheminsReguliers` démarrés à la ligne 4
9: $B \leftarrow \{(q, \mathsf{u}) \ / \ t : (q, \mathsf{u})$ est un noeud dans l'arbre construit par `ArbreCouverture`$\}$
10: **retourner** $\mathrm{Max}(B)$

Intuitivement, pour prouver que l'algorithme `OmegaBaseEC` termine, on montre que s'il y avait une branche infinie dans l'arbre de couverture développé, alors un état paramétré non borné aurait du être découvert (à un moment donné) et, par conséquent, l'algorithme `ArbreCouverture` aurait du utiliser cet état paramétré non borné pour obtenir un ω et ainsi terminer la branche. Plus formellement,

Proposition 3.8.14 *Pour tout SAVE avec reset* $\mathcal{V}$ *de dimension* 2*, muni d'un état initial* (q_0, x_0)*, l'algorithme* `OmegaBaseEC` *appliqué à* $(\mathcal{V}, (q_0, \mathsf{x}_0))$ *termine et retourne l'*ω*-base minimale de* $\downarrow \mathrm{post}^*(q_0, \mathsf{x}_0)$.

Preuve. Commençons par remarquer que si l'algorithme `OmegaBaseEC` termine, alors d'après les lemmes 3.8.12 et 3.8.13, on obtient que l'ensemble B calculé à la ligne 9 est une ω-base de $\downarrow \mathrm{post}^*(q_0, \mathsf{x}_0)$, et par conséquent $\mathrm{Max}(B)$ est l'unique ω-base minimale de $\downarrow \mathrm{post}^*(q_0, \mathsf{x}_0)$. Par l'absurde, supposons que l'algorithme `OmegaBaseEC` ne termine pas. D'après la définition de l'algorithme `OmegaBaseEC`, on en déduit que l'algorithme `ArbreCouverture` ne termine pas. Chaque instruction à l'intérieur de la boucle **tant que** (lignes 2–19) ne prend qu'un temps fini, et à chaque itération de cette boucle un nouveau noeud est visité, donc un arbre infini T est construit. Comme le degré de l'arbre est fini, on obtient par application du lemme de König qu'il existe un chemin infini $\pi = t_0 \xRightarrow{\alpha_0} t_1 \xRightarrow{\alpha_1} t_2 \cdots t_i \xRightarrow{\alpha_i} t_{i+1} \cdots$ dans T, d'étiquettes $t_i : (q_i, \mathsf{u}_i)$. On déduit du lemme 3.8.11 qu'il y a un nombre infini de resets le long de π. Or d'après le lemme 3.8.10, deux noeuds distincts de π ont nécessairement des étiquettes différentes. On peut donc supposer que pour tout $i \in \mathbb{N}$, on a $\mathsf{u}_i \notin \{(\omega, 0), (0, \omega)\}$.

Soit $R \subseteq \rightarrow_{\mathsf{V}}$ l'ensemble des *transitions de contrôle de reset* et notons w la projection de π sur R. Formellement, w s'obtient à partir de π en deux temps : nous remplaçons tout d'abord dans π chaque t_i par q_i, afin d'obtenir un mot infini μ sur l'ensemble fini $\rightarrow_{\mathsf{V}}$ des transitions de contrôle. Puis, nous projetons μ sur R pour obtenir w. Observons que w est un mot infini sur R. On associe à w la suite infinie $(w_i)_{i \in \mathbb{N}}$ où w_i est le facteur gauche de longueur i de w. D'après le lemme 3.8.9, il existe $n \in \mathbb{N}$ et une transition de contrôle de reset $r \in R$ tels que :

$$\text{pour tout } m \in \mathbb{N}, \text{ il existe } k_m \in \mathbb{N} \text{ tel que } w_{k_m} \in R^*(rR^{\leq n})^m rR^* \qquad (3.1)$$

Sans perte de généralité, on peut supposer que r est une transition de contrôle étiquetée par **reset**(2) : $r = p \xrightarrow{\texttt{reset}(2)}_{\mathcal{V}} q$ pour des états de contrôle p et q. Observons que r apparaît infiniment souvent dans π et notons t_{i_j} le noeud suivant la $j^{ème}$ *occurrence de* r *dans* π. Pour tout $j \in \mathbb{N}$, on a $q_{i_j} = q$ et $\mathsf{u}_{i_j} = (x_{i_j}, 0)$. De plus, on déduit du lemme 3.8.10 que la suite infinie $(x_{i_j})_{j\in\mathbb{N}}$ est strictement croissante. Par conséquent, pour tout $j \in \mathbb{N}$, on a $x_{i_j} \neq \omega$. La propriété (1) ci-dessus implique que : pour tout $m \in \mathbb{N}$, il existe $k_m \in \mathbb{N}$ tel que pour tout $j \in [k_m \,..\, k_m + m - 1]$ il y a au plus n resets entre t_{i_j} et $t_{i_{j+1}-1}$. De plus, puisque pour tout $i \in \mathbb{N}$ on a $\mathsf{u}_i \notin \{(\omega, 0), (0, \omega)\}$, on obtient que pour tout $m \in \mathbb{N}$,

$$(q, (x_{i_{k_m}}, 0)) \xrightarrow{*}_{n+1} (q, (x_{i_{k_m+1}}, 0)) \cdots (q, (x_{i_{k_m+m-1}}, 0)) \xrightarrow{*}_{n+1} (q, (x_{i_{k_m+m}}, 0))$$

Montrons maintenant que $s = \langle q, 1\rangle$ est un état paramétré $(n+1)$-sensé. D'après la définition 3.8.2, nous devons montrer que pour tout $x \in \mathbb{N}$, l'ensemble $V(s, x) = \{y \in \mathbb{N} \;/\; s(x) \xrightarrow{*\,\texttt{reset}(2)}_{n+1} s(y)\}$ est fini. Par l'absurde, supposons qu'il existe $x \in \mathbb{N}$ tel que $V(s, x)$ est infini. Alors, il existe $h \in \mathbb{N}$ tel que $x_{i_h} \geq x$, et on en déduit que $V(s, x_{i_h})$ est infini. Observons que $(q, (x_{i_h}, 0)) \in \mathrm{post}^*(q_0, \mathsf{x}_0)$, et par conséquent $(q_0, \mathsf{x}_0) \xrightarrow{*}_{n'} (q, (x_{i_h}, 0))$ pour un certain $n' \in \mathbb{N}$. Comme $V(s, x_{i_h})$ est infini, on obtient que $\{y \in \mathbb{N} \;/\; (q_0, \mathsf{x}_0) \xrightarrow{*}_{n+n'+1} (q, (y, 0))\}$ est infini. Considérons le plus grand sous-arbre T' de T tel que tout noeud t' de T' satisfait $r \overset{*}{\Rightarrow}_{(n+n'+1)} t'$. On déduit du lemme 3.8.11 que T' est fini et on obtient donc, avec le lemme 3.8.12, que T' contient un noeud t_ω étiqueté par $(q, (\omega, u_\omega))$ pour un certain $u_\omega \in \mathbb{N}_\omega$. Rappelons que l'algorithme **ArbreCouverture** construit un arbre en largeur d'abord (voir la ligne 3 de l'algorithme). On obtient donc que pour un certain $k \in \mathbb{N}$, le noeud t_{i_k} est traité après t_ω, et par conséquent t_{i_k} ne devrait pas avoir de fils ce qui conduit à une contradiction.

Puisque $\langle q, 1\rangle$ est un état paramétré $(n+1)$-sensé, on déduit du lemme 3.8.5 qu'il existe un schéma de chemins régulier et maximal (σ, f, y_0) d'ordre $(n+1)$ associé à s. Remarquons que f est d-régulière pour un certain $d \in \mathbb{N}$. Rappelons que pour une infinité de j on a $(q, (x_{i_j}, 0)) \xrightarrow{*}_{n+1} (q, (x_{i_{j+1}}, 0))$ avec $x_{i_{j+1}} > x_{i_j}$. Par conséquent, pour une infinité de j, on a $f(x_{i_j}) > x_{i_j}$. De plus, il existe $h \in \mathbb{N}$ tel que $x_{i_h} \geq y_0$ et tel que

$$(q, (x_{i_h}, 0)) \xrightarrow{*}_{n+1} (q, (x_{i_{h+1}}, 0)) \cdots (q, (x_{i_{h+d-1}}, 0)) \xrightarrow{*}_{n+1} (q, (x_{i_{h+d}}, 0))$$

On en déduit que pour tout $j \in [h \,..\, h + d - 1]$, on a $f(x_{i_j}) \geq x_{i_{j+1}}$. D'après le lemme 3.8.8, on obtient que pour tout $x \geq x_{i_h}$, on a $f(x) > x$, et par conséquent $(\langle q, 1\rangle, x_{i_h})$ est un état paramétré non borné. Donc il existe $y \in \mathbb{N}$ tel que, à un moment donné, $(\langle q, 1\rangle, y)$ est ajouté à la liste L. Cependant, il existe $h' \in \mathbb{N}$ tel que $x_{i_{h'}} \geq y$ et tel que le noeud $t_{i_{h'}}$ est traité après l'addition de $(\langle q, 1\rangle, y)$ à la liste L, ce qui conduit à une contradiction puisque $t_{i_{h'}}$ devrait alors être étiqueté par $(q, (\omega, 0))$. ■

Maintenant, étant donné un SAVE avec reset/transfert $\mathcal{V}$ de dimension 2, on construit un SAVE avec reset $\mathcal{W}$ de dimension 2, obtenu à partir de $\mathcal{V}$ en

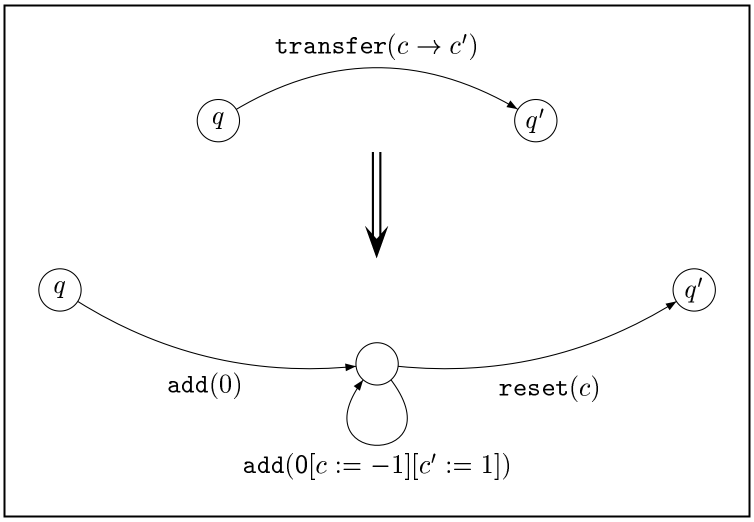


FIG. 3.6 – Simulation faible de $\texttt{transfer}(c \to c')$ par des $\texttt{reset}(c)$ et des additions

utilisant la “simulation faible” des transitions de transfert par des transitions de reset décrite en figure 3.6. On observe que cette simulation préserve l'ensemble de couverture (mais pas post*). Par conséquent, on aboutit à :

Théorème 3.8.15 *Pour tout SAVE avec reset/transfert $\mathcal{V}$ de dimension 2 et pour tout état initial (q_0, x_0), l'ω-base minimale de $\downarrow\mathrm{post}^*(q_0, \mathsf{x}_0)$ est calculable.*

Exemple 3.8.16 L'ensemble de couverture du SAVE avec reset de dimension 2, dont le contrôle est représenté en figure 3.5 et avec état initial $(q_0, (0, 0))$, a pour ω-base minimale $\{(q_0, (\omega, 0)), (q_1, (\omega, \omega)), (q_2, (\omega, \omega))\}$.

3.8.3 Construction de post* pour la classe $\mathfrak{R}_{1,2}\mathfrak{Tr}_{12,21}$

Nous montrons maintenant comment construire post* pour les SAVE avec reset/transfert de dimension 2.

Théorème 3.8.17 *Pour tout SAVE étendu dans la classe $\mathfrak{R}_{1,2}\mathfrak{Tr}_{12,21}$, muni d'un ensemble linéaire S_0 d'états initiaux,* $\mathrm{post}^*(S_0)$ *est semilinéaire et constructible.*

Preuve. Il suffit de prouver le théorème pour un unique état initial (q_0, x_0), puisque on peut facilement simuler S_0 par un SAVE (non étendu) $\mathcal{I}$, muni d'un unique état initial $(q_0, \mathsf{0})$ et tel que $\mathrm{post}^*_{\mathcal{I}}(q_0, \mathsf{0}) = S_0$. D'après le théorème 3.4.1, on sait qu'on peut simuler $\mathcal{V}$ par un SAVE dans la classe $\mathfrak{R}_{1,2}\mathfrak{Tr}_{12}$. On peut donc supposer sans perte de généralité que $\mathcal{V}$ ne contient aucune transition étiquetée par $\texttt{transfer}(2 \to 1)$. De plus, on peut aussi supposer que pour toute transition de contrôle $q \xrightarrow{\alpha} q'$ avec $\alpha \in \{\texttt{reset}(1), \texttt{transfer}(1 \to 2)\}$, il n'existe pas d'autre transition de contrôle se terminant en q'. Maintenant, notons $\mathcal{W}$ le

SAVE étendu dans la classe $\mathfrak{T}_1\mathfrak{R}_2$, obtenu à partir de $\mathcal{V}$ en utilisant la simulation des $\texttt{reset}(1)$ et des $\texttt{transfer}(1 \to 2)$ par des $\texttt{test}(1)$ (et des additions), donnée dans la figure 3.3 de la page 92. On observe que cette simulation préserve $\text{post}^*(q_0, \mathsf{x}_0)$, et donc aussi l'ensemble de couverture $\downarrow \text{post}^*(q_0, \mathsf{x}_0)$. De plus, pour toute transition de contrôle $q \xrightarrow{\texttt{test}(1)} q'$ dans $\mathcal{W}$, il n'y a pas d'autre transition de contrôle dans $\mathcal{W}$ se terminant en q'. Par conséquent, on obtient que $\Omega(\mathcal{W}, (q_0, \mathsf{x}_0))$ est calculable car ici, $\Omega(\mathcal{W}, (q_0, \mathsf{x}_0))$ est l'ensemble des états de contrôle q' tels que :

- d'une part il existe q tel que $q \xrightarrow{\texttt{test}(1)} q'$.
- d'autre part $(q', (0, \omega))$ appartient à l'ω-base minimale de $\downarrow \text{post}^*(q_0, \mathsf{x}_0)$, qui est calculable d'après le théorème 3.8.15, et

On déduit donc du théorème 3.7.8 que $\text{post}^*(q_0, \mathsf{x}_0)$ est semilinéaire et constructible. ∎

Exemple 3.8.18 Pour le SAVE avec reset de dimension 2, dont le contrôle est représenté en figure 3.5 et avec état initial $(q_0, (0, 0))$, l'ensemble des états accessibles depuis $(q_0, \mathsf{0})$ est donné par

$$\begin{array}{rcl} \text{post}^*(q_0, \mathsf{0}) = & & q_0 \times ((0,0) + \{(1,0)\}^*) \\ & \bigcup & q_1 \times (\{(1,0),(0,3)\} + \{(1,0),(0,3)\}^*) \\ & \bigcup & q_2 \times (\{(0,0),(1,1),(2,2)\} + \{(3,0),(0,3)\}^*) \end{array}$$

3.9 Application à la vérification des SAVE étendus

Nous résumons dans cette section les résultats principaux obtenus dans ce chapitre, et nous donnons ensuite quelques applications de ces résultats.

Nous avons analysé exhaustivement les classes de SAVE étendus de dimension 2. Nous avons obtenu une hiérarchie complète vis-à-vis de la décidabilité de l'accessibilité, donnée en figure 3.7, et presque complète vis-à-vis de l'effective semilinéarité de post* et de pre*. Il reste en effet un problème ouvert : nous ne savons pas, pour la classe “maximale” $\mathfrak{T}_1\mathfrak{R}_{1,2}\mathfrak{Tr}_{12}$, si l'ensemble semilinéaire post* est constructible.

Rappelons que lorsque pre* est effectivement semilinéaire, le model-checking du fragment existentiel de CTL est décidable. On en déduit donc que le model-checking du fragment existentiel de CTL est décidable pour la classe “maximale” $\mathfrak{T}_1\mathfrak{R}_{1,2}\mathfrak{Tr}_{12}$. Observons que cette classe est aussi stable par intersection avec un langage rationnel (en étiquetant les transitions de contrôle par un nouvel alphabet). On obtient ainsi que le model-checking de LTL sur les chemins finis est lui aussi décidable pour la classe $\mathfrak{T}_1\mathfrak{R}_{1,2}\mathfrak{Tr}_{12}$. Enfin, l'effective semilinéarité de pre* pour cette classe implique que les problèmes suivants sont décidables : l'accessibilité, l'accessibilité d'un état de contrôle et la couverture.

Nous avons montré que post* est lui aussi effectivement semilinéaire pour les classes $\mathfrak{T}_1\mathfrak{R}_1\mathfrak{Tr}_{12}$ et $\mathfrak{R}_{1,2}\mathfrak{Tr}_{12,21}$. Par conséquent, les problèmes suivants sont

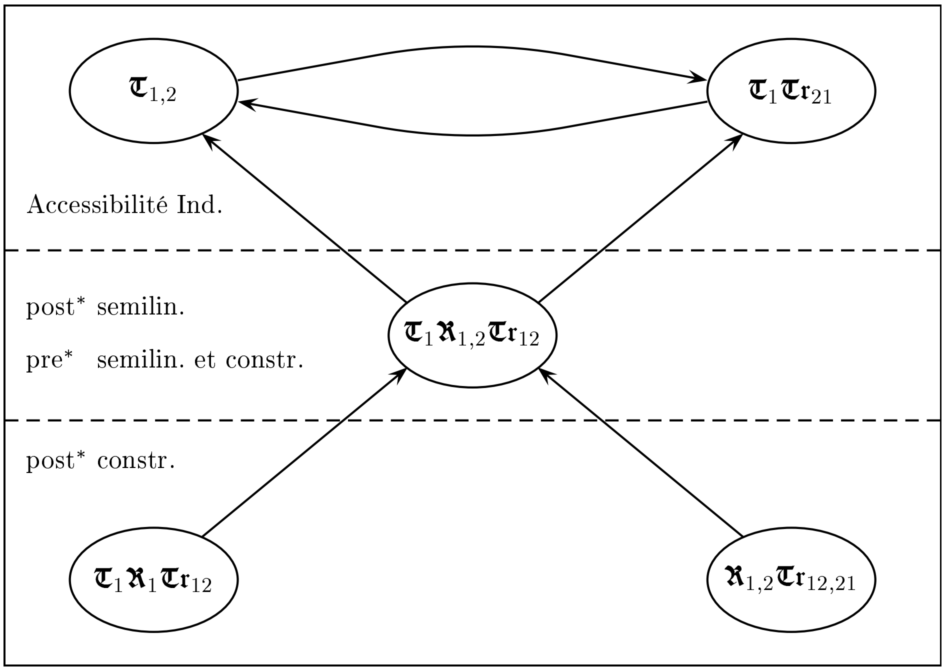


FIG. 3.7 – Hiérarchie des classes de SAVE étendus de dimension 2

aussi décidables pour ces classes : le caractère borné, le caractère borné d'un compteur donné, la terminaison, l'inclusion et l'équivalence d'ensembles d'états accessibles.

Cependant, deux compteurs ne sont en général pas suffisants pour modéliser des systèmes complexes. Les accélérations, que nous avons définies dans le cas de deux compteurs, peuvent être étendues au cas de n compteurs, mais sans garantie de terminaison de l'algorithme `ArbreSymbolique`. On obtient alors un semi-algorithme pour le cas général, qui produit une approximation inférieure de post* lorsqu'il est arrêté prématurément.

Notre analyse des SAVE étendus à deux compteurs nous permet également d'élaborer une stratégie pour le calcul d'une approximation supérieure de post* ou de pre* dans le cas de n compteurs. L'idée consiste à ne considérer que des paires de compteurs (en ignorant les autres), et à se ramener à une classe décidable de SAVE étendus de dimension 2 (par exemple, en simulant faiblement des test-à-zéro sur un compteur par des resets). On calcule alors l'ensemble post* (ou pre*) associé à cette paire de compteurs, et on calcule enfin l'intersection de ces ensembles pour toutes les paires possibles de compteurs. Cette intersection produit une approximation supérieure de post* (ou de pre*).

Chapitre 4

Analyse des automates à files Fifo

Nous faisons dans ce chapitre le point sur différentes représentations symboliques pour les automates à files Fifo — une structure de contrôle finie munie files Fifo non bornées. A notre connaissance, Pachl est le premier à utiliser des expression régulières pour représenter des ensembles infinis de contenus de files [Pac87]. Dans [FM96], les expression régulières linéaires ont été introduites et utilisées. Les QDD (*Queue-content Decision Diagrams*), une représentation basée sur des automates finis déterministes, furent examinés [BGWW97]. Ensuite, des formules de Presburger furent ajoutées pour former les CQDD (*Constrained QDDs*) [BH99]. Les SRE (*Simple regular expressions*) furent définies et utilisées dans le cadre des automates à files Fifo avec perte [AAB99].

Nous définissons dans ce chapitre une nouvelle représentation symbolique de contenus de files Fifo basée sur les SLRE. Nous montrons que :

- les SLRE sont faciles à manipuler. En particulier, nous prouvons que l'inclusion entre SLRE est coNP–complète.
- les SLRE sont au coeur des QDD et des CQDD. Plus précisément, nous montrons qu'un ensemble de contenus de files Fifo est représentable par un SLRE, si et seulement si il est à la fois représentable par un QDD et représentable par un CQDD.
- les SLRE sont souvent suffisants, car les SLRE sont "capables d'itérer" les mêmes circuits que les QDD. De plus, plusieurs exemples ont un ensemble d'états accessibles représentables avec les SLRE : le protocole du bit alterné [Pac87], le protocole de retransmission bornée de Philips [AAB99], l'exemple de producteur/consommateur décrit dans [BGWW97] et le protocole de connexion/déconnexion [JJ93].

Nous commencerons par introduire le modèle des automates à files Fifo en section 4.1, puis nous présenterons le cadre d'abstraction et d'accélération de ce modèle en section 4.2. Nous ferons un bref état de l'art des représentations symboliques FIFO dans la section 4.3. Nous présenterons la représentation symbolique FIFO basée sur les SLRE en section 4.4, puis nous comparerons les les QDD, les CQDD et les SLRE en section 4.5. Nous conclurons ce chapitre sur les automates à files FIFO non fiables dans la section 4.6.

4.1 Automates à files FIFO

Commençons par présenter le protocole bien connu du *bit alterné*. On le modélise généralement par un système de deux automates étendus, qui communiquent au travers de deux *canaux* uni-directionnels FIFO, comme indiqué sur la figure 4.1. On peut clairement calculer le produit cartésien de ces deux automates étendus, ce qui conduit à un unique automate, étendu avec deux canaux bi-directionnels (ou *files*) FIFO, qu'on appelle *automate à files* FIFO.

Les automates à files FIFO forment une sous-classe de la classe des *automates étendus*, définie et analysée dans la section 2.4 du premier chapitre. Un automate à files FIFO $\mathcal{F}$ est donc défini comme une sémantique opérationnelle (i.e. un système de transitions étiqueté) associée à une structure de contrôle F finie, cette dernière décrivant les opérations à effectuer sur les canaux lors du passage d'un état de contrôle à l'autre. Par exemple, la figure 4.1 dépeint le contrôle ABP de l'automate à files FIFO $\mathcal{ABP}$ modélisant le protocole du *bit alterné*.

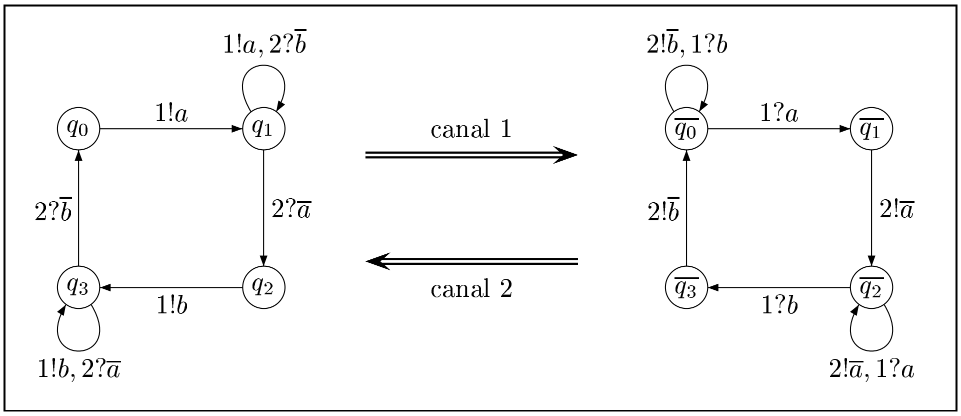


FIG. 4.1 – Un automate à files FIFO modélisant le protocole du bit alterné

La classe des automates à files FIFO, introduite dans [Boc78], est un modèle mathématique pour les langages de description de protocoles ESTELLE et SDL. Malheureusement, cette classe a la puissance des machines de Turing [BZ83, FM97], même lorsqu'on la restreint aux automates à une seule file. Par conséquent, l'ensemble des états accessibles post* n'est pas récursif pour cette classe. Nous allons néanmoins chercher à semi-calculer post*, en appliquant le cadre général d'*abstraction* et d'*accélération* des automates étendus décrit en section 2.4.

La structure de contrôle d'un automate à files FIFO est définie comme suit :

Définition 4.1.1 *Un* contrôle d'automate à files FIFO *est un quadruplet* $\mathsf{F} = (Q, C, M, \to)$*, où :*

- Q *est un ensemble fini d'*états de contrôle*, et*
- C *est un ensemble fini de* canaux *(ou* files*), et*
- M *est un alphabet de* messages*, et*
- *l'ensemble* $\to \subseteq Q \times (C \times \{?, !\} \times M) \times Q$ *est un ensemble fini de* transitions de contrôle*.*

Observons qu'un contrôle d'automate à files FIFO $\mathsf{F} = (Q, C, M, \to)$ définit implicitement le contrôle d'automate étendu $(Q, C \times \{?, !\} \times M, \to)$. Les opérations $(c, ?, m)$ et $(c, !, m)$ s'écrivent plus simplement $c?m$ et $c!m$ respectivement. Intuitivement, une transition de contrôle de la forme $(q, c?m, q')$ représente une réception du message m dans le canal c, et une transition de contrôle de la forme $(q, c!m, q')$ représente une émission du message m dans le canal c.

Exemple 4.1.2 Le contrôle d'automate à files FIFO ABP décrit en figure 4.1 est formellement défini par $\mathsf{ABP} = (Q, C, M, \to_{\mathsf{ABP}})$, où :

- $Q = \{q_0, q_1, q_2, q_3\} \times \{\overline{q_0}, \overline{q_1}, \overline{q_2}, \overline{q_3}\}$ est l'ensemble des états de contrôle, et
- $C = \{1, 2\}$ est l'ensemble des canaux, et
- $M = \{a, b, \overline{a}, \overline{b}\}$ est l'alphabet des messages, et
- l'ensemble $\to_{\mathsf{ABP}} \subseteq Q \times (C \times \{?, !\} \times M) \times Q$ des transitions de contrôle contient par exemple les éléments $(q_0, \overline{q_0}) \xrightarrow{1!a} (q_1, \overline{q_0})$, $(q_1, \overline{q_2}) \xrightarrow{1?b} (q_1, \overline{q_3})$ et $(q_0, \overline{q_2}) \xrightarrow{2!\overline{a}} (q_0, \overline{q_2})$.

Remarque. Dans toute la suite de ce chapitre, nous désignerons par M un alphabet de messages et par C un ensemble fini de canaux. L'alphabet d'opérations sera donc $C \times \{?, !\} \times M$, et les séquences d'opérations que nous considérerons seront dans $(C \times \{?, !\} \times M)^*$.

Nous supposons que les canaux sont parfaits dans le sens où les messages sont reçus dans le même ordre que celui de leur émission, d'où le nom FIFO (pour *First In First Out*). La signification des opérations d'émission et de réception est formellement définie par :

Définition 4.1.3 *La* sémantique FIFO*, notée* Δ_{FIFO}*, de l'alphabet d'opérations* $\mathtt{Op} = C \times \{?, !\} \times M$ *est le triplet* $\Delta_{\text{FIFO}} = (C, M^*, \delta_{\text{FIFO}})$*, où la fonction* δ_{FIFO} *de* $\mathtt{Op}$ *dans* $\mathcal{P}((M^*)^C \times (M^*)^C)$ *est définie par :*

$$\begin{array}{lcl} (\mathsf{w}, \mathsf{w}') \in \delta_{\text{FIFO}}(c?m) & si & \mathsf{w} = \mathsf{w}'[c := m\mathsf{w}'[c]] \\ (\mathsf{w}, \mathsf{w}') \in \delta_{\text{FIFO}}(c!m) & si & \mathsf{w}' = \mathsf{w}[c := \mathsf{w}[c]m] \end{array}$$

Ainsi, le contenu d'un canal est donné par un mot sur l'alphabet de messages. La définition de la fonction δ_{FIFO} correspond à l'idée intuitive suivante :

– lors d'une réception d'un message m depuis un canal c, on retire m du début du canal c,

– lors d'une émission d'un message m dans un canal c, on ajoute m à la fin du canal c.

Remarque. Dans toute la suite de ce chapitre, la sémantique utilisée sera la sémantique FIFO Δ_{FIFO}.

Un contrôle d'automate à files FIFO F, vu comme un contrôle d'automate étendu, définit l'automate étendu $\mathcal{F}$ conformément à la sémantique Δ_{FIFO} (voir la définition 2.4.5 page 54). L'automate étendu $\mathcal{F}$, qui est aussi appelé *automate à files* FIFO, décrit ainsi la sémantique opérationnelle du contrôle d'automate à files FIFO F. Redonnons la définition de l'automate étendu $\mathcal{F}$ dans le contexte des automates à files FIFO :

Définition 4.1.4 *Un contrôle d'automate à files* FIFO $\mathsf{F} = (Q, C, M, \rightarrow_{\mathsf{F}})$ *définit le système de transitions étiqueté* $\mathcal{F} = (S, \Sigma, \rightarrow_{\mathcal{F}})$*, appelé* automate à files FIFO*, comme suit :*

– $S = Q \times (M^*)^C$ *est l'ensemble des états, et*

– $\Sigma = C \times \{?, !\} \times M$ *est l'alphabet d'opérations, et*

– *la relation de transitions étiquetée* $\rightarrow_{\mathcal{F}}$ *est le plus petit sous-ensemble* $\rightarrow_{\mathcal{F}}$ *de* $S \times \Sigma \times S$ *satisfaisant pour tout* $\mathsf{w} \in (M^*)^C$ *les deux conditions suivantes :*

$$\begin{array}{llll} si & q \xrightarrow{c?m}_{\mathsf{F}} q' \;\; et \;\; m \leq \mathsf{w}[c] & alors & (q, \mathsf{w}) \xrightarrow{c?m}_{\mathcal{F}} (q', \mathsf{w}[c := m^{-1}\mathsf{w}[c]]) \\ si & q \xrightarrow{c!m}_{\mathsf{F}} q' & alors & (q, \mathsf{w}) \xrightarrow{c!m}_{\mathcal{F}} (q', \mathsf{w}[c := \mathsf{w}[c]m]) \end{array}$$

Ainsi, un état (q, w) d'un automate à files FIFO est composé d'un *état de contrôle* q et d'un vecteur de mots w représentant le contenu des canaux. Les deux conditions de franchissement d'une transition suivent la sémantique FIFO.

Exemple 4.1.5 Dans l'automate à files FIFO $\mathcal{ABP}$, on a par exemple le chemin suivant :

$$\begin{array}{c} ((q_0, \overline{q_0}), (\varepsilon, \varepsilon)) \xrightarrow{1!a} ((q_1, \overline{q_0}), (a, \varepsilon)) \xrightarrow{1!a} ((q_1, \overline{q_0}), (aa, \varepsilon)) \xrightarrow{1?a} ((q_1, \overline{q_1}), (a, \varepsilon)) \\ \Big\downarrow 2!\overline{a} \\ ((q_2, \overline{q_2}), (a, \overline{a})) \xleftarrow{2?\overline{a}} ((q_1, \overline{q_2}), (a, \overline{a}\overline{a})) \xleftarrow{2!\overline{a}} ((q_1, \overline{q_2}), (a, \overline{a})) \end{array}$$

Nous aurons parfois besoin de projeter une séquence d'opérations sur les émissions dans (resp. les réceptions depuis) un canal c donné. Nous utiliserons alors les notations suivantes :

Notation. Les fonctions *in* et *out*, de $C \times (C \times \{?,!\} \times M)^*$ dans M^*, sont définies récursivement par :

$$in(c,\sigma) \quad = \quad \begin{cases} \varepsilon & \text{si } \sigma = \varepsilon \\ m \cdot in(c,\sigma') & \text{si } \sigma = (c?m) \cdot \sigma' \\ in(\sigma') & \text{si } \sigma = \alpha \cdot \sigma' \text{ avec } \alpha \neq (c?m) \end{cases}$$

$$out(c,\sigma) \quad = \quad \begin{cases} \varepsilon & \text{si } \sigma = \varepsilon \\ m \cdot out(c,\sigma') & \text{si } \sigma = (c!m) \cdot \sigma' \\ out(\sigma') & \text{si } \sigma = \alpha \cdot \sigma' \text{ avec } \alpha \neq (c!m) \end{cases}$$

Rappelons que les automates à files FIFO ont la puissance des machines de Turing [BZ83, FM97]. Ce résutat se retrouve aussi comme conséquence du théorème suivant, qui montre les connexions entre ce modèle et celui des SAVE étendus.

Théorème 4.1.6 *Nous avons les simulations suivantes :*

- *Pour tout automate à files* FIFO $\mathcal{F}$ *dont l'alphabet de messages est réduit à un singleton, on peut construire un SAVE (non étendu) simulant* $\mathcal{F}$*.*
- *Pour tout SAVE étendu* $\mathcal{V}$*, on peut construire un automate à une seule file* FIFO $\mathcal{F}$*, d'alphabet de messages* $\{a, \#\}$*, simulant* $\mathcal{V}$*.*

4.2 Abstraction et accélération des automates à files FIFO

Rappelons que post* peut être infini dans un automate à files FIFO. Le cadre général d'abstraction et d'accélération développé dans le premier chapitre suggère que nous cherchions à représenter finiment des sous-ensembles infinis d'états, afin de semi-calculer post*. L'ensemble des états de contrôle étant fini, une manière naturelle de procéder consiste à représenter finiment des ensembles infinis de contenus de files, c'est-à-dire des ensembles infinis de vecteurs de mots.

Exemple 4.2.1 Pour l'automate à files FIFO $\mathcal{ABP}$, en partant de l'état initial $((q_0, \overline{q_0}), (\varepsilon, \varepsilon))$, et avec des notations simplifiées, on a :

$$\begin{array}{rcllcl} \text{post}^*((q_0,\overline{q_0}),(\varepsilon,\varepsilon)) & = & & (q_0,\overline{q_0}) \times (b^*, \overline{b}^*) & \cup & (q_1,\overline{q_0}) \times (b^* a\, a^*, \overline{b}^*) \\ & & \cup & (q_1,\overline{q_1}) \times (a^*, \overline{b}^*) & \cup & (q_1,\overline{q_2}) \times (a^*, \overline{b}^* \overline{a}\, \overline{a}^*) \\ & & \cup & (q_2,\overline{q_2}) \times (a^*, \overline{a}^*) & \cup & (q_3,\overline{q_2}) \times (a^* b\, b^*, \overline{a}^*) \\ & & \cup & (q_3,\overline{q_3}) \times (b^*, \overline{a}^*) & \cup & (q_3,\overline{q_0}) \times (b^*, \overline{a}^* \overline{b}\, \overline{b}^*) \end{array}$$

Nous suivrons l'idée originale de PACHL [Pac87] consistant à représenter des ensembles infinis de contenus de files par des expressions régulières. Rappelons que $REG(M)$ désigne l'ensemble des expressions régulières sur M, et que $[\![\rho]\!]$ dénote le langage (rationnel) associé à une expression régulière ρ. Plus particulièrement, nous développerons en section 4.4 une représentation symbolique par expressions régulières semilinéaires.

Exemple 4.2.2 On remarque que post* pour le protocole du bit alterné $\mathcal{ABP}$, depuis l'état initial $((q_0, \overline{q_0}), (\varepsilon, \varepsilon))$, est composé de 16 LRE. On peut encore citer d'autres exemples issus de la littérature où post* peut être décrit par des SLRE, plus précisément le protocole de retransmission bornée de PHILIPS [AAB99], l'exemple de producteur/consommateur décrit dans [BGWW97] et le protocole de connexion/déconnexion [JJ93].

Afin de représenter symboliquement les réceptions de messages, nous utiliserons la notion de *dérivatifs d'expression régulières* [Brz64]. Etant donné un message $m \in M$, l'opérateur de dérivation $m^{-1} : REG(M) \to REG(M)$ est défini par induction sur la structure des expression régulières :

$$m^{-1}(\rho) = \begin{cases} \varepsilon & \text{si } \rho = m \\ \emptyset & \text{si } \rho = \emptyset \text{ ou si } \rho = m' \text{ avec } m' \neq m \\ m^{-1}(\rho_1)\rho_2 & \text{si } \rho = \rho_1\rho_2 \text{ et si } \varepsilon \notin [\![\rho_1]\!] \\ m^{-1}(\rho_1)\rho_2 + m^{-1}(\rho_2) & \text{si } \rho = \rho_1\rho_2 \text{ et si } \varepsilon \in [\![\rho_1]\!] \\ m^{-1}(\rho_1) + m^{-1}(\rho_2) & \text{si } \rho = \rho_1 + \rho_2 \\ m^{-1}(\rho_1)\rho_1^* & \text{si } \rho = \rho_1^* \end{cases}$$

Proposition 4.2.3 *On a $[\![m^{-1}(\rho)]\!] = m^{-1}[\![\rho]\!]$, pour tout $\rho \in REG(M)$.*

Preuve. Par induction sur la structure d'une expression régulière ρ. ■

Nous allons manipuler et comparer dans la suite de ce chapitre différentes représentations symboliques pour la sémantique FIFO, basées respectivement sur les QDD [BGWW97], sur les CQDD [BH99] et sur les SLRE. La notion générale de *représentation symbolique*, donnée par la définition 2.4.7 de la page 56, nous amène naturellement à la notion de *représentation symbolique de files* FIFO, adaptée au contexte des automates à files FIFO. Les deux contraintes majeures sur les représentations symboliques de files FIFO sont la clôture par réception symbolique (notée ??), et la clôture par émission symbolique (notée !!).

Définition 4.2.4 *Une* représentation symbolique de files FIFO*, pour un ensemble fini C de canaux et pour un alphabet M de messages, est un sextuplet $\mathcal{R}_{C,M} = (R, !!, ??, [\![\cdot]\!], +, \bot)$ où :*

- *R est un ensemble de contenus (ou représentants) symboliques de files, et $+$ est une loi de composition interne sur R d'élément neutre $\bot$, et*

- *$?? : C \times M \times R \to R$, $!! : C \times M \times R \to R$ et $[\![\cdot]\!] : R \to 2^{(M^*)^C}$ sont trois fonctions satisfaisant pour tout $\chi, \chi' \in R$, $c \in C$ et $m \in M$:*

$$\begin{aligned} [\![\bot]\!] &= \emptyset && (4.1) \\ [\![??(c, m, \chi)]\!] &= \{\mathsf{w}[c := m^{-1}\mathsf{w}[c]] \;/\; \mathsf{w} \in [\![\chi]\!], m \leq \mathsf{w}[c]\} && (4.2) \\ [\![!!(c, m, \chi)]\!] &= \{\mathsf{w}[c := \mathsf{w}[c]m] \;/\; \mathsf{w} \in [\![\chi]\!]\} && (4.3) \\ [\![\chi + \chi']\!] &= [\![\chi]\!] \cup [\![\chi']\!] && (4.4) \end{aligned}$$

On note plus simplement $\mathcal{R}$ au lieu de $\mathcal{R}_{C,M}$ lorsque C et M se comprennent d'après le contexte. On dit que $\mathcal{R}$ est *effective* si d'une part R est un ensemble encodé, et d'autre part les fonctions $??, !!$ et $+$ sont récursives.

On peut fusionner les fonctions de réception symbolique $??$ et d'émission symbolique $!!$ en une unique fonction $\delta_{\mathcal{R}} : \mathtt{Op} \times R \to R$, où $\mathtt{Op} = C \times \{?, !\} \times M$, définie comme suit :

$$\begin{array}{rcl} \delta_{\mathcal{R}}(c?m, \chi) & = & ??(c, m, \chi) \\ \delta_{\mathcal{R}}(c!m, \chi) & = & !!(c, m, \chi) \end{array}$$

Une représentation symbolique de files FIFO $\mathcal{R}$ définit donc implicitement la représentation symbolique $(R, \delta_{\mathcal{R}}, [\![\cdot]\!], +, \bot)$ pour la sémantique Δ_{FIFO} (au sens de la définition 2.4.7 de la page 56).

Rappelons que pour toute séquence d'opérations $\sigma \in (C \times \{?, !\} \times M)^+$, les fonctions σ et σ^* de $\mathcal{P}((M^*)^C)$ dans $\mathcal{P}((M^*)^C)$ sont définies par :

$$\begin{array}{rcl} \sigma(X) & = & \{\mathsf{w}' \in (M^*)^C \ / \ \exists\ \mathsf{w} \in X, (\mathsf{w}, \mathsf{w}') \in \delta_{\text{FIFO}}(\sigma)\} \\ \sigma^*(X) & = & \displaystyle\bigcup_{i \in \mathbb{N}} \sigma^i(X) \end{array}$$

Redonnons également la définition 2.4.12 de la page 59 dans le contexte des représentation symbolique de files FIFO. Etant donné une représentation symbolique de files FIFO $\mathcal{R} = (R, !!, ??, [\![\cdot]\!], +, \bot)$ et une séquence d'opérations σ dans $(C \times \{?, !\} \times M)^+$, on dit que σ^* est $\mathcal{R}$–représentable lorsqu'il existe une fonction $\sigma^*_{\mathcal{R}} : R \to R$ telle que $[\![\sigma^*_{\mathcal{R}}(r)]\!] = \sigma^*([\![r]\!])$ pour tout $r \in R$.

Puisqu'une représentation symbolique de files FIFO $\mathcal{R}$ est aussi une représentation symbolique pour la sémantique Δ_{FIFO}, on peut suivre le cadre d'abstraction et d'accélération des automates étendus décrit dans la section 2.4, et on obtient que pour tout automate à files FIFO $\mathcal{F}$,

- $\mathcal{A}^{\mathcal{R}}_{\mathcal{F}}$ est une abstraction de $\mathcal{F}$, et
- la fonction $\nabla^{\mathcal{R}}_{\mathcal{F}}$, basée sur l'itération des séquences d'opérations σ dans $(C \times \{?, !\} \times M)^+$ telles que σ^* est $\mathcal{R}$–représentable, est une accélération correcte pour l'abstraction $\mathcal{A}^{\mathcal{R}}_{\mathcal{F}}$. De plus, si σ^* est $\mathcal{R}$–représentable pour tout circuit de contrôle $q \xrightarrow{\sigma} q$, alors l'accélération $\nabla^{\mathcal{R}}_{\mathcal{F}}$ est complète.

4.3 Etat de l'art des représentations symboliques FIFO

Nous montrons maintenant comment les notions de représentations symboliques de files FIFO et de séquences d'opérations σ telles que σ^* est symboliquement représentable permettent de présenter dans un cadre uniforme les travaux sur les QDD et les CQDD.

4.3.1 Les "Queue Decision Diagrams"

Les "Queue Decision Diagrams" (QDD) furent introduits par BOIGELOT et GODEFROID dans [BG96], afin de semi-calculer l'ensemble des états accessibles d'un automate à files FIFO. Un QDD D, pour un automate à files FIFO $\mathcal{F}$ d'alphabet de messages M et d'ensemble de canaux $C = \{c_1, c_2, \ldots, c_n\}$, est un automate fini déterministe sur l'alphabet $C \times M$ satisfaisant la condition suivante : pour tout mot w dans le langage $L(D)$ reconnu par D, la projection de w sur l'alphabet C est incluse dans $c_1{}^* c_2{}^* \cdots c_n{}^*$. Intuitivement, vecteur $\mathsf{w} \in (M^*)^C$ est dans l'ensemble des contenus de canaux représenté par D s'il existe un mot $(u_1 u_2 \cdots u_n) \in L(D)$ tel que pour tout $i \in [1 \mathinner{..} n]$, le mot u_i s'écrit $u_i = (c_i, m_1)(c_i, m_2) \cdots (c_i, m_k)$ avec $\mathsf{w}[c_i] = m_1 m_2 \cdots m_k$. Il est démontré dans [BGWW97] que les ensembles de contenus de canaux représentables par QDD sont précisément les ensembles reconnaissables de vecteurs de mots. Par conséquent, les QDD correspondent à la représentation symbolique de files FIFO basée sur les expressions régulières. Formellement,

Définition 4.3.1 *Soit* $\mathcal{QDD} = (\mathcal{P}_f(REG(M)^C), \emptyset, !!, ??, [\![\cdot]\!], \cup)$, *où les fonctions* $??$, $!!$ *et* $[\![\cdot]\!]$ *sont définies par :*

$$\begin{aligned} ??(c, m, \chi) &= \{\mathsf{r}[c := m^{-1}(\mathsf{r}[c])] \ / \ \mathsf{r} \in \chi\} \\ !!(c, m, \chi) &= \{\mathsf{r}[c := \mathsf{r}[c] \cdot m] \ / \ \mathsf{r} \in \chi\} \\ [\![\chi]\!] &= \bigcup_{\mathsf{r} \in \chi} \prod_{c \in C} [\![\mathsf{r}[c]]\!] \end{aligned}$$

Proposition 4.3.2 *$\mathcal{QDD}$ est une représentation symbolique effective de files* FIFO.

Par conséquent, pour tout automate à files FIFO $\mathcal{F}$, $\mathcal{A}_{\mathcal{F}}^{\mathcal{QDD}}$ est une abstraction effective de $\mathcal{F}$, et la fonction $\nabla_{\mathcal{F}}^{\mathcal{QDD}}$ est une accélération correcte pour $\mathcal{A}_{\mathcal{F}}^{\mathcal{QDD}}$.

Afin de caractériser les séquences d'opérations σ telles que σ^* est $\mathcal{QDD}$–représentable, nous utilisons une condition sur les séquences d'opérations introduite dans [BGWW97]. Observons que notre définition est légèrement différente de la définition originale, car les canaux ont des alphabets de messages distincts dans [BGWW97], alors que nous avons choisi d'utiliser le même alphabet de messages pour tous les canaux.

Définition 4.3.3 ([BGWW97]) *Une séquence d'opérations* $\sigma \in (C \times \{?, !\} \times M)^+$ compte *si l'une des conditions suivantes est satisfaite :*

i) $|M| > 1$ *et il existe* $c_1, c_2 \in C$ *avec* $c_1 \neq c_2$ *tels que* $out(c_1, \sigma) \neq \varepsilon$ *et* $out(c_2, \sigma) \neq \varepsilon$,

ii) $|M| = 1$ *et il existe* $c_1, c_2 \in C$ *avec* $c_1 \neq c_2$ *tels que* $out(c_1, \sigma) > in(c_1, \sigma)$ *et* $out(c_2, \sigma) > in(c_2, \sigma)$.

Expliquons cette définition. Lorsque $|M| > 1$, une séquence d'opérations σ compte si σ contient au moins deux opérations d'émission émettant dans des canaux distincts. Lorsque $|M| = 1$, une séquence d'opérations σ compte si au moins deux canaux (vus ici comme des compteurs) croissent avec σ. Remarquons que lorsque l'ensemble des canaux est réduit à un singleton, alors aucune séquence d'opérations ne compte.

Théorème 4.3.4 ([BGWW97]) *Pour toute séquence d'opérations σ, σ^* est $\mathcal{QDD}$–représentable si et seulement si σ ne compte pas.*

Comme conséquence immédiate du théorème, dont la preuve est constructive, on obtient que si $\mathcal{F}$ est un automate à files FIFO tel que σ ne compte pas pour tout circuit de contrôle $q \xrightarrow{\sigma} q$, alors la fonction $\nabla_{\mathcal{F}}^{\mathcal{QDD}}$ est une accélération complète récursive pour l'abstraction $\mathcal{A}_{\mathcal{F}}^{\mathcal{QDD}}$.

4.3.2 Les "Constrained Queue Decision Diagrams"

Une généralisation des QDD, les "Constrained QDDs" (CQDD), fut ensuite définie par BOUAJJANI et HABERMEHL dans [BH99]. L'idée principale des CQDD est d'introduire des formules de Presburger afin de pouvoir représenter l'effet de l'itération d'une séquence d'opération quelconque. La définition originale des CQDD était donnée en termes d'automates finis, déterministes, simples et restreints. Ces automates finis peuvent en fait se voir comme des LRE "déterministes", aussi traduisons-nous la définition des CQDD donnée dans [BH99], afin de la présenter plus simplement en termes de LRE.

Définition 4.3.5 *Une* composante acceptante *est un couple* (r, φ) *tel que :*

1. $\mathsf{r} \in LRE(M)^C$, *et*
2. *pour tout $c \in C$, le LRE $\mathsf{r}[c]$ ne contient aucun facteur de la forme $(au)^*a$ ou u^*v^* avec $u, v \in M^*$ et $a \in M$, et*
3. *φ est une formule de l'arithmétique de Presburger, d'ensemble de variables libres $V = \{x_c^i \ / \ c \in C, i \in [1 \mathinner{..} n_c]\}$ où n_c est le nombre d'occurrences de '$*$' dans $\mathsf{r}[c]$.*

Un CQDD *est une union finie de composantes acceptantes.*

On note $CQDD(C, M)$ l'ensemble des CQDD d'alphabet de messages M et d'ensemble de canaux C.

Afin de définir l'ensemble des contenus de canaux représenté par un CQDD, nous ordonnons les occurrences de '$*$' dans un LRE ρ en lisant ρ de gauche à droite. Nous associons à un LRE ρ comportant n occurences de '$*$' une fonction $w_\rho : \mathbb{N}^n \to M^*$, où $w_\rho(k_1, k_2, \ldots, k_n)$ est le mot obtenu à partir de ρ en remplaçant la $i^{\text{ème}}$ occurence de '$*$' par k_i, pour tout $i \in [1 \mathinner{..} n]$.

L'ensemble des contenus de canaux $[\![\chi]\!]$ représenté par un CQDD χ est donné par $[\![\chi]\!] = \bigcup_{(\mathsf{r},\varphi)\in\chi} [\![(\mathsf{r}, \varphi)]\!]$ où on désigne par $[\![(\mathsf{r}, \varphi)]\!]$ l'ensemble des contenus de canaux $\mathsf{w} \in (M^*)^C$ tels que, pour une certaine valuation $v : X \to \mathbb{N}$ *satisfaisant* φ, on a $\mathsf{w}[c] = f_{\mathsf{r}[c]}(v(x_c^1), v(x_c^2), \ldots, v(x_c^{n_c}))$ pour tout $c \in C$. Observons que si φ

est valide (i.e. satisfaite par toutes les valuations), alors $[\![(\mathsf{r}, \varphi)]\!] = \prod_{c \in C} [\![\mathsf{r}[c]]\!]$ représente un ensemble des contenus de canaux qui est aussi $\mathcal{QDD}$–représentable, et même $\mathcal{SLRE}$–représentable[1].

Les CQDD sont effectivement clos par émission symbolique et par réception symbolique :

Proposition 4.3.6 ([BH99]) *Etant donné un CQDD χ, un canal c et un message m, on peut construire deux CQDD χ' et χ'' tels que :*

$$\begin{array}{rcl} [\![\chi']\!] & = & \{\mathsf{w}[c := m^{-1}\mathsf{w}[c]] \ / \ \mathsf{w} \in [\![\chi]\!], m \leq \mathsf{w}[c]\} \\ [\![\chi'']\!] & = & \{\mathsf{w}[c := \mathsf{w}[c]m] \ / \ \mathsf{w} \in [\![\chi]\!]\} \end{array}$$

De plus, on note $\chi' = ??(c, m, \chi)$ et $\chi'' = !!(c, m, \chi)$.

Nous sommes maintenant prêts à définir la représentation symbolique de files Fifo basée sur les CQDD. On pose $\mathcal{CQDD} = (CQDD(C, M), \emptyset, !!, ??, [\![\cdot]\!], \cup)$, et on obtient trivialement que :

Proposition 4.3.7 *$\mathcal{CQDD}$ est une représentation symbolique effective de files Fifo.*

Par conséquent, pour tout automate à files Fifo $\mathcal{F}$, $\mathcal{A}_{\mathcal{F}}^{\mathcal{CQDD}}$ est une abstraction effective de $\mathcal{F}$, et la fonction $\nabla_{\mathcal{F}}^{\mathcal{CQDD}}$ est une accélération correcte pour $\mathcal{A}_{\mathcal{F}}^{\mathcal{CQDD}}$. Rappelons que l'effet de l'itération d'un circuit quelconque de $\mathcal{F}$ est symboliquement représentable avec les CQDD :

Théorème 4.3.8 ([BH99]) *Pour toute séquence d'opérations σ, σ^* est $\mathcal{CQDD}$–représentable.*

Comme conséquence immédiate du théorème, dont la preuve est constructive, on obtient que la fonction $\nabla_{\mathcal{F}}^{\mathcal{CQDD}}$ est une accélération complète récursive pour l'abstraction $\mathcal{A}_{\mathcal{F}}^{\mathcal{CQDD}}$.

4.4 Abstraction et accélération avec les SLRE

Nous considérons dans cette section les SLRE comme la base d'une nouvelle représentation symbolique de files Fifo. L'idée d'utiliser les SLRE pour représenter des ensembles infinis de contenus de files était déjà présente dans [FM96]. Cependant, une caractérisation des circuits dont l'effet est symboliquement itérable n'avait pas été établie. Nous allons montrer dans cette section que pour toute séquence d'opérations σ, σ^* est $\mathcal{SLRE}$–représentable si et seulement si σ^* est $\mathcal{QDD}$–représentable.

Commençons par définir la représentation symbolique de files Fifo basée sur les SLRE.

[1]La représentation symbolique de files Fifo $\mathcal{SLRE}$, basée sur les SLRE, sera définie en section 4.4.

Proposition 4.4.1 *Etant donné $m \in M$ et une expression régulière semilinéaire ρ, l'expression régulière $m^{-1}(\rho)$ est semilinéaire.*

Preuve. Comme la fonction $m^{-1} : REG(M) \to REG(M)$ se distribue par rapport à l'addition (i.e. $m^{-1}(\rho_1 + \rho_2) = m^{-1}(\rho_1) + m^{-1}(\rho_2)$), il suffit de montrer que pour tout LRE ρ, l'expression régulière $m^{-1}(\rho)$ est semilinéaire. La preuve de cette dernière assertion se fait par récurrence sur la taille d'un LRE ρ. ∎

Définition 4.4.2 *Soit $\mathcal{SLRE} = (\mathcal{P}_f(SLRE(M)^C), \emptyset, !!, ??, [\![\cdot]\!], \cup)$, où les fonctions ??, !! et $[\![\cdot]\!]$ sont définies par :*

$$\begin{array}{rcl} ??(c, m, \chi) & = & \{\mathsf{r}[c := m^{-1}(\mathsf{r}[c])] \ / \ \mathsf{r} \in \chi\} \\ !!(c, m, \chi) & = & \{\mathsf{r}[c := \mathsf{r}[c] \cdot m] \ / \ \mathsf{r} \in \chi\} \\ [\![\chi]\!] & = & \bigcup_{\mathsf{r} \in \chi} \prod_{c \in C} [\![\mathsf{r}[c]]\!] \end{array}$$

Proposition 4.4.3 *$\mathcal{SLRE}$ est une représentation symbolique effective de files* FIFO.

Par conséquent, pour tout automate à files FIFO $\mathcal{F}$, $\mathcal{A}_{\mathcal{F}}^{\mathcal{SLRE}}$ est une abstraction effective de $\mathcal{F}$, et la fonction $\nabla_{\mathcal{F}}^{\mathcal{SLRE}}$ est une accélération correcte pour $\mathcal{A}_{\mathcal{F}}^{\mathcal{SLRE}}$.

Nous allons maintenant caractériser les séquences d'opérations σ telles que σ^* est $\mathcal{SLRE}$–représentable. Nous aurons besoin de la notion de séquences infiniment itérables à partir d'ensembles de contenus de files :

Définition 4.4.4 *Une séquence d'opérations σ est* infiniment itérable *à partir d'un sous-ensemble $X \subseteq (M^*)^C$ si on a $\sigma^i(X) \neq \emptyset$ pour tout $i \in \mathbb{N}$.*

Nous commençons par analyser le cas plus simple où C est un singleton.

4.4.1 Accélération dans le cas d'un unique canal

Nous supposons dans toute cette sous-section que C est réduit à un singleton. Afin de simplifier les notations, et comme il n'y a pas d'ambiguité sur le canal, on omet de préciser le canal dans les fonctions *in* et *out*.

Considérons un chemin $(q, w) \xrightarrow{\sigma} (q, w')$ dans un automate à une seule file FIFO. Clairement, les messages émis et les messages réceptionnés le long de ce chemin proviennent tous de σ, et par conséquent on a $in(\sigma) \cdot w' = w \cdot out(\sigma)$. Supposons maintenant que σ est infiniment itérable à partir de w. Pour tout $i \in \mathbb{N}$, il existe un mot $w_i \in M^*$ tel que $(q, w) \xrightarrow{\sigma^i} (q, w_i)$, et on en déduit que $in(\sigma)^i \cdot w_i = w \cdot out(\sigma)^i$. Si de plus il y a au moins une réception dans σ (i.e. $in(\sigma) \neq \varepsilon$), alors $out(\sigma) \neq \varepsilon$ et quand i devient grand, ce qui est réceptionné du canal est précisément ce qui a été envoyé dedans. On obtient alors à la limite $in(\sigma)^\omega = w \cdot out(\sigma)^\omega$. Afin de décrire précisément $\sigma^*(\{w\})$, nous aurons besoin du lemme suivant :

Lemme 4.4.5 *Pour tout $w, x, y \in M^*$, les assertions suivantes sont équivalentes :*

1. $x^\omega = wy^\omega$
2. *il existe trois mots* $x', x'', z \in M^*$ *tels que* $x = x'x''$, $w \in x^*x'$, z *est primitif et* $(x''x'), y \in z^*$.

Preuve. Une application du *théorème du défaut* (voir par exemple [Per83]).

Nous pouvons maintenant décrire l'effet de l'itération d'une séquence d'opérations infiniment itérable à partir d'un mot donné :

Proposition 4.4.6 ([FM96]) *Lorsque l'ensemble des canaux est réduit à un singleton, alors pour toute séquence d'opérations* σ *infiniment itérable à partir d'un mot* $w \in M^*$, *il existe un mot* $z_{\sigma,w} \in M^*$ *tel que* $\sigma^*(\{w\}) = w \cdot (z_{\sigma,w})^*$.

Preuve. Supposons que σ est infiniment itérable à partir de w. Si $in(\sigma) = \varepsilon$, alors on a $\sigma^*(\{w\}) = w \cdot out(\sigma)^*$, ce qui termine la preuve. Supposons maintenant que $in(\sigma) \neq \varepsilon$. Comme σ est infiniment itérable à partir de w, on aboutit à $in(\sigma)^\omega = w \cdot out(\sigma)^\omega$. Posons $x = in(\sigma)$ et $y = out(\sigma)$. D'après le lemme 4.4.5, on obtient qu'il existe $x', x'', z \in M^*$ et $h, j, k \in \mathbb{N}$ tels que $x = x'x''$, $w = x^k x'$, z est primitif, $x''x' = z^h$ et $y = z^j$. Observons que $h \leq j$ puisque $|x| \leq |y|$. Par conséquent, on pose $z_{\sigma,w} = z^{j-h}$ et on obtient que $y = x''x'z_{\sigma,w}$. Maintenant, pour tout $i \in \mathbb{N}^*$, on a $\sigma^i(\{w\}) = \{(x^i)^{-1}(wy^i))\}$. On peut écrire $(x^i)^{-1}(wy^i)$ comme suit :

$$
\begin{array}{rcll}
(x^i)^{-1}(wy^i) & = & (x^i)^{-1}(x^k x' y^i) & (\text{car } w = x^k x')) \\
& = & (x'(x''x')^{i-1}x'')^{-1}(x'(x''x')^k y^i) & (\text{car } x = x'x'') \\
& = & ((x''x')^{i-1}x'')^{-1}((x''x')^{k+i} \cdot (z_{\sigma,w})^i) & (\text{car } y = x''x'z_\sigma) \\
& = & x'(x''x')^k \cdot (z_{\sigma,w})^i & \\
& = & x^k x' \cdot (z_{\sigma,w})^i & (\text{car } x = x'x'') \\
& = & w \cdot (z_{\sigma,w})^i & (\text{car } w = x^k x')
\end{array}
$$

On en conclut que $\sigma^*(\{w\}) = w \cdot (z_{\sigma,w})^*$. ∎

Soit σ une séquence d'opérations. En partant d'un unique contenu de file $w \in M^*$, on obtient d'après la proposition précédente que $\sigma^*(\{w\})$ est soit fini (si σ n'est pas infiniment itérable à partir de w), soit un ensemble infini $\mathcal{SLRE}$–représentable de contenus de file. Le théorème suivant, qui était utilisé sans être prouvé dans [FM96], exprime une propriété encore plus forte : en partant d'un ensemble $\mathcal{SLRE}$–représentable $[\![\chi]\!]$ de contenus de file, $\sigma^*([\![\chi]\!])$ est encore $\mathcal{SLRE}$–représentable.

Théorème 4.4.7 *Lorsque l'ensemble des canaux est réduit à un singleton, alors* σ^* *est* $\mathcal{SLRE}$*–représentable pour toute séquence d'opérations* σ.

Preuve. Nous devons montrer que pour toute séquence d'opérations σ et pour tout ensemble fini χ de SLRE, il existe un ensemble fini χ' de SLRE satisfaisant $\sigma^*([\![\chi]\!]) = [\![\chi']\!]$. Comme la fonction σ^* préserve la borne supérieure (i.e. se distribue par rapport à l'union), il suffit en fait de prouver que pour tout LRE ρ, il existe un SLRE ρ' tel que $\sigma^*([\![\rho]\!]) = [\![\rho']\!]$.

Si $in(\sigma) = \varepsilon$ alors pour tout LRE ρ, on a $\sigma^*(\llbracket\rho\rrbracket) = \llbracket\rho \cdot out(\sigma)^*\rrbracket$, ce qui termine la preuve pour ce cas particulier. Nous supposons maintenant pour tout le reste de la preuve que $in(\sigma) \neq \varepsilon$.

Rappelons que la fonction $\delta_{\mathcal{SLRE}} : \mathtt{Op}\times\mathcal{P}_f(SLRE(M)^C) \rightarrow \mathcal{P}_f(SLRE(M)^C)$, où $\mathtt{Op} = C \times \{?, !\} \times M$, est obtenue comme la fusion des fonctions de réception symbolique ?? et d'émission symbolique !! (voir la fin de la section 4.2). Elle vérifie $\llbracket\delta_{\mathcal{SLRE}}(\sigma, \rho)\rrbracket = \sigma(\llbracket\rho\rrbracket)$ pour toute séquence d'opérations σ et pour tout LRE ρ. Nous noterons plus simplement $\sigma(\rho)$ au lieu de $\delta_{\mathcal{SLRE}}(\sigma, \rho)$ au cours de cette preuve.

Pour tout LRE ρ, on désigne par n_ρ le nombre d'occurrences de '$*$' dans ρ. On introduit une *mesure* Δ sur les LRE comme suit :

$$\Delta(\rho) = \begin{cases} 2 * n_\rho - 1 & \text{si } \rho \text{ est de la forme } y^* \cdot \rho' \\ 2 * n_\rho & \text{sinon} \end{cases}$$

Nous montrons par récurrence sur $\Delta(\rho)$ que pour tout LRE ρ, pour toute séquence d'opérations σ telle que $in(\sigma) \neq \varepsilon$ et pour tout $\alpha \in \mathbb{N}^*$, on a :

$$\begin{array}{rcll} \sigma^*(\llbracket\rho\rrbracket) & = & \llbracket\rho'\rrbracket & \text{pour un certain SLRE } \rho' \qquad (\mathbf{H_1}) \\ \sigma^*(\llbracket\rho \cdot (out(\sigma)^\alpha)^*\rrbracket) & = & \llbracket\rho''\rrbracket & \text{pour un certain SLRE } \rho'' \qquad (\mathbf{H_2}) \end{array}$$

Observons que si $out(\sigma) = \varepsilon$, alors $(\mathbf{H_2})$ est équivalent à $(\mathbf{H_1})$. Aussi supposerons-nous toujours que $out(\sigma) \neq \varepsilon$ lorsque nous prouverons $(\mathbf{H_2})$ dans la preuve par récurrence suivante.

$\boxed{\text{base}}$ Soit ρ un LRE tel que $\Delta(\rho) = 0$. Il existe donc $x \in \Sigma^*$ tel que $\rho = x$. Afin de prouver $(\mathbf{H_1})$, remarquons que $\sigma^*(x)$ est :

- soit un ensemble fini (et par conséquent un langage SLRE-définissable) si σ n'est pas infiniment itérable à partir de x,
- soit le langage $x \cdot (z_{\sigma,x})^*$, d'après la proposition 4.4.6, si σ est infiniment itérable à partir de x.

Prouvons maintenant $(\mathbf{H_2})$ et supposons donc $out(\sigma) \neq \varepsilon$. Deux cas se présentent :

1. si $in(\sigma)^\omega \neq x \cdot out(\sigma)^\omega$ alors il existe $k \in \mathbb{N}$ tel que $\sigma^k(\llbracket x \cdot (out(\sigma)^\alpha)^*\rrbracket) = \emptyset$, et par conséquent $\sigma^*(\llbracket x \cdot (out(\sigma)^\alpha)^*\rrbracket) = \bigcup_{i=0}^{k-1} \sigma^i(\llbracket x \cdot (out(\sigma)^\alpha)^*\rrbracket) = \llbracket\rho'\rrbracket$ où $\rho' = \sum_{i=0}^{k-1} \sigma^i(x \cdot (out(\sigma)^\alpha)^*)$ est un SLRE.
2. si $in(\sigma)^\omega = x \cdot out(\sigma)^\omega$ alors on a $x \cdot out(\sigma)^{h\,|in(\sigma)|} = in(\sigma)^{h\,|out(\sigma)|} \cdot x$ pour tout $h \in \mathbb{N}$. Par conséquent, il existe $x' \in \Sigma^*$ et $k \in \mathbb{N}$ tels que $|x'| < |out(\sigma)|$ et $x = x' \cdot out(\sigma)^k$. On en déduit que :

   $$\sigma^{\alpha\,|out(\sigma)|}(\llbracket x \cdot (out(\sigma)^\alpha)^*\rrbracket) = x' \cdot out(\sigma)^{k+\alpha(\beta+|out(\sigma)|-|in(\sigma)|)} \cdot (out(\sigma)^\alpha)^*$$

   où $\beta \in \mathbb{N}$ est le nombre de fois que le mot $out(\sigma)^\alpha$ est utilisé pour le franchissement de la séquence d'opérations $\sigma^{\alpha\,|out(\sigma)|}$. En d'autres termes,

β est le plus petit entier tel que $k + \alpha(\beta + |out(\sigma)| - |in(\sigma)|) \geq 0$. Posons $y = x' \cdot out(\sigma)^{k+\alpha(\beta+|out(\sigma)|-|in(\sigma)|)}$. On obtient que :

$$\begin{aligned}\sigma^*([\![x \cdot (out(\sigma)^\alpha)^*]\!]) &= \sigma^*([\![y \cdot (out(\sigma)^\alpha)^*]\!]) \bigcup_{i=0}^{\alpha\,|out(\sigma)|-1} \sigma^i([\![x \cdot (out(\sigma)^\alpha)^*]\!]) \\ &= \sigma^*([\![y \cdot (out(\sigma)^\alpha)^*]\!]) \quad \bigcup \quad [\![\rho']\!]\end{aligned}$$

où $\rho' = \sum_{i=0}^{\alpha\,|out(\sigma)|-1} \sigma^i(x \cdot (out(\sigma)^\alpha)^*)$ est un SLRE. Deux se présentent à nouveau :

(a) Si $\beta + |out(\sigma)| - |in(\sigma)| \geq 0$, alors on aboutit à $y \in x \cdot (out(\sigma)^\alpha)^*$ et par conséquent $[\![y \cdot (out(\sigma)^\alpha)^*]\!] \subseteq [\![x \cdot (out(\sigma)^\alpha)^*]\!]$. Ainsi, on a $\sigma^{\alpha\,|out(\sigma)|}([\![x \cdot (out(\sigma)^\alpha)^*]\!]) \subseteq [\![x \cdot (out(\sigma)^\alpha)^*]\!]$ et on en déduit que $\sigma^*([\![x \cdot (out(\sigma)^\alpha)^*]\!]) = [\![\rho']\!]$.

(b) Sinon, on obtient que $y < x$. On peut remplacer x par y et répéter la démarche ci-dessus pour montrer qu'il existe un SLRE ρ'' tel que $\sigma^*([\![y \cdot (out(\sigma)^\alpha)^*]\!]) = [\![\rho'']\!]$. Ce processus finira par terminer car $y < x$.

$\boxed{\text{étape d'induction}}$ Soit $K \in \mathbb{N}$ et supposons que les assertions $(\mathbf{H_1})$ et $(\mathbf{H_2})$ sont vraies pour tout LRE ρ tel que $\Delta(\rho) \leq K$. Soit ρ un LRE tel que $\Delta(\rho) = K+1$. Nous avons deux cas distincts à examiner, suivant la parité de K.

Supposons que $K+1$ est pair. Il existe donc $x \in \Sigma^*$ et un LRE ρ' avec $\Delta(\rho') = K$ tels que $\rho = x \cdot \rho'$. Deux cas se présentent :

1. si $x \notin \mathrm{FG}(in(\sigma)^*)$ alors il existe $k \in \mathbb{N}$ tel que $x \not\leq in(\sigma)^k$ et $|x| \leq |in(\sigma)^k|$. Par conséquent, pour tout LRE π on a $\sigma^k([\![x \cdot \pi]\!]) = \emptyset$ et on obtient que :

$$\sigma^*([\![x \cdot \pi]\!]) = \bigcup_{i=0}^{k-1} \sigma^i([\![x \cdot \pi]\!]) = \left[\!\!\left[\sum_{i=0}^{k-1} \sigma^i(x \cdot \pi)\right]\!\!\right]$$

Cette égalité montre que $(\mathbf{H_1})$ et $(\mathbf{H_2})$ sont satisfaites pour ρ.

2. si $x \in \mathrm{FG}(in(\sigma)^*)$ alors il existe $k \in \mathbb{N}$ et $x' \leq in(\sigma)$ tels que $x = in(\sigma)^k \cdot x'$. Pour tout LRE π, on a :

$$\begin{aligned}\sigma^*([\![x \cdot \pi]\!]) &= \sigma^*(\sigma([\![x' \cdot \pi \cdot out(\sigma)^k]\!])) \bigcup_{i=0}^{k} in(\sigma)^{k-i} \cdot x' \cdot \pi \cdot out(\sigma)^i \\ &= \sigma^*(\sigma([\![x' \cdot \pi \cdot out(\sigma)^k]\!])) \bigcup \left[\!\!\left[\sum_{i=0}^{k} in(\sigma)^{k-i} \cdot x' \cdot \pi \cdot out(\sigma)^i\right]\!\!\right]\end{aligned}$$

Observons que $\sum_{i=0}^{k} in(\sigma)^{k-i} \cdot x' \cdot \pi \cdot out(\sigma)^i$ est un SLRE. La séquence d'opérations σ peut se décomposer sous la forme $\sigma = \sigma_1\sigma_2$ où $in(\sigma_1) = x'$. Puisque $(\sigma_2\sigma_1)$ est aussi une séquence d'opérations vérifiant $in(\sigma_2\sigma_1) \neq \varepsilon$, on en déduit que pour tout LRE π, on a :

$$\begin{aligned}\sigma^*(\sigma([\![x' \cdot \pi \cdot out(\sigma)^k]\!])) &= \sigma_2((\sigma_2\sigma_1)^*(\sigma_1([\![x' \cdot \pi \cdot out(\sigma)^k]\!]))) \\ &= \sigma_2((\sigma_2\sigma_1)^*([\![\pi \cdot out(\sigma)^k \cdot out(\sigma_1)]\!]))\end{aligned}$$

Observons que $\Delta(\rho' \cdot out(\sigma)^k \cdot out(\sigma_1)) = \Delta(\rho') = K$. Donc on obtient d'après l'hypothèse d'induction ($\mathbf{H_1}$) qu'il existe un SLRE ρ'' tel que $(\sigma_2\sigma_1)^*(\llbracket\rho' \cdot out(\sigma)^k \cdot out(\sigma_1)\rrbracket) = \llbracket\rho''\rrbracket$. Ainsi, $\sigma^*(\sigma(\llbracket x' \cdot \rho' \cdot out(\sigma)^k\rrbracket)) = \llbracket\rho''\rrbracket$ et il vient $\sigma^*(\llbracket\rho\rrbracket) = \sigma^*(\llbracket x \cdot \rho'\rrbracket) = \llbracket\rho'' + \sum_{i=0}^k in(\sigma)^{k-i} \cdot x' \cdot \rho' \cdot out(\sigma)^i\rrbracket$. Ceci montre ($\mathbf{H_1}$) for ρ. Pour prouver ($\mathbf{H_2}$) pour ρ, supposons que $out(\sigma) \neq \varepsilon$. On remarque que :

$$\begin{aligned}
&\sigma_2((\sigma_2\sigma_1)^*(\llbracket\rho' \cdot (out(\sigma)^\alpha)^* \cdot out(\sigma)^k \cdot out(\sigma_1)\rrbracket))) \\
&\qquad = \sigma_2((\sigma_2\sigma_1)^*(\llbracket\rho' \cdot out(\sigma)^k \cdot (out(\sigma)^\alpha)^* \cdot out(\sigma_1)\rrbracket))) \\
&\qquad = \sigma_2((\sigma_2\sigma_1)^*(\llbracket\rho' \cdot out(\sigma)^k \cdot out(\sigma_1) \cdot (out(\sigma_2\sigma_1)^\alpha)^*\rrbracket)))
\end{aligned}$$

Puisque $\Delta(\rho' \cdot out(\sigma)^k \cdot out(\sigma_1)) = \Delta(\rho') = K$, on déduit de l'hypothése d'induction ($\mathbf{H_2}$) que $(\sigma_2\sigma_1)^*(\llbracket\rho' \cdot out(\sigma)^k \cdot out(\sigma_1) \cdot (out(\sigma_2\sigma_1)^\alpha)^*\rrbracket) = \llbracket\rho''\rrbracket$ pour un certain ρ''. Par conséquent, il vient $\sigma^*(\sigma(\llbracket x' \cdot \rho' \cdot (out(\sigma)^\alpha)^* \cdot out(\sigma)^k\rrbracket)) = \llbracket\rho''\rrbracket$ et on aboutit à $\sigma^*(\llbracket\rho\rrbracket) = \sigma^*(\llbracket x \cdot \rho' \cdot (out(\sigma)^\alpha)^*\rrbracket) = \llbracket\rho'' + \sum_{i=0}^k in(\sigma)^{k-i} \cdot x' \cdot \rho' \cdot (out(\sigma)^\alpha)^* \cdot out(\sigma)^i\rrbracket$. Ceci montre ($\mathbf{H_2}$) for ρ.

Supposons que $K+1$ est impair. Il existe donc $y \in \Sigma^+$ et un LRE ρ' avec $\Delta(\rho') \leq K$ tels que $\rho = y^* \cdot \rho'$. Deux cas se présentent :

1. si $y^\omega \neq in(\sigma)^\omega$ alors il existe $k \in \mathbb{N}$ tel que $y^k \notin \mathrm{FG}(in(\sigma)^*)$. Par conséquent, il existe $l \in \mathbb{N}$ tel que $y^k \not\preceq in(\sigma)^l$ et $|y^k| \leq |in(\sigma)^l|$. Ainsi, pour tout LRE π on obtient que $\sigma^l(\llbracket y^k \cdot y^* \cdot \pi\rrbracket) = \emptyset$ et il vient donc :

$$\begin{aligned}
\sigma^*(\llbracket y^* \cdot \pi\rrbracket) &= \sigma^*(\llbracket y^k \cdot y^* \cdot \pi\rrbracket) \quad \bigcup_{j=0}^{k-1} \sigma^*(\llbracket y^j \cdot \pi\rrbracket) \\
&= \bigcup_{i=0}^{l-1} \sigma^i(\llbracket y^k \cdot y^* \cdot \pi\rrbracket) \quad \bigcup_{j=0}^{k-1} \sigma^*(\llbracket y^j \cdot \pi\rrbracket)
\end{aligned}$$

Maintenant, pour tout $j \in [0\,..\,k-1]$, on a $\Delta(y^j \cdot \rho') \leq K$. On déduit de l'hypothèse d'induction que :

($\mathbf{H_1}$) pour tout $0 \leq j \leq k-1$, il existe un SLRE ρ_j tel que $\sigma^*(\llbracket y^j \cdot \rho'\rrbracket) = \llbracket\rho_j\rrbracket$. On aboutit à :

$$\sigma^*(\llbracket\rho\rrbracket) = \left\llbracket \sum_{i=0}^{l-1} \sigma^i(y^k \cdot y^* \cdot \rho' \quad + \quad \sum_{j=0}^{k-1} \rho_j \right\rrbracket$$

ce qui prouve ($\mathbf{H_1}$) pour ρ.

($\mathbf{H_2}$) Supposons que $out(\sigma) \neq \varepsilon$. Pour tout $0 \leq j \leq k-1$, il existe un SLRE ρ_j tel que $\sigma^*(\llbracket y^j \cdot \rho' \cdot (out(\sigma)^\alpha)^*\rrbracket) = \llbracket\rho_j\rrbracket$. On aboutit à :

$$\sigma^*(\llbracket\rho \cdot (out(\sigma)^\alpha)^*\rrbracket) = \left\llbracket \sum_{i=0}^{l-1} \sigma^i(y^k \cdot y^* \cdot \rho' \cdot (out(\sigma)^\alpha)^*) \quad + \quad \sum_{j=0}^{k-1} \rho_j \right\rrbracket$$

ce qui prouve ($\mathbf{H_2}$) pour ρ.

2. si $y^\omega = in(\sigma)^\omega$ alors il vient $y^{|in(\sigma)|} = in(\sigma)^{|y|}$. On en déduit que pour tout $i, j, k, l \in \mathbb{N}$ et pour tout LRE π, on a :

$$\begin{array}{rcll} \sigma^{l|y|+i}([\![y^{k|in(\sigma)|+j} \cdot \pi]\!])) & = & \sigma^i([\![y^{(k-l)|in(\sigma)|+j} \cdot \pi \cdot out(\sigma)^{l|y|}]\!]) & \text{si } k \geq l \\ \sigma^{l|y|+i}([\![y^{k|in(\sigma)|+j} \cdot \pi]\!])) & = & \sigma^{(l-k)|y|+i}([\![y^j \cdot \pi \cdot out(\sigma)^{k|y|}]\!]) & \text{si } k \leq l \end{array}$$

Par conséquent, on obtient que :

$$\begin{array}{rcl} \sigma^*([\![y^* \cdot \pi]\!]) & = & \displaystyle\bigcup_{i,j \geq 0} \sigma^i([\![y^j \cdot \pi]\!]) \\ & = & \displaystyle\bigcup_{i=0}^{|y|-1} \bigcup_{j=0}^{|in(\sigma)|-1} \bigcup_{k,l \geq 0} \sigma^{l|y|+i}([\![y^{k|in(\sigma)|+j} \cdot \pi]\!])) \end{array}$$

Découpons l'union indicée par k et l ci-dessus en distinguant les cas où $k \geq l \geq 0$ et où $l \geq k \geq 0$. Pour tout $i, j \in \mathbb{N}$, on a :

$$\begin{array}{l} \displaystyle\bigcup_{k,l \geq 0} \sigma^{l|y|+i}([\![y^{k|in(\sigma)|+j} \cdot \pi]\!]) \\ \quad = \left\{ \begin{array}{l} \displaystyle\bigcup_{k \geq l \geq 0} \sigma^{l|y|+i}([\![y^{k|in(\sigma)|+j} \cdot \pi]\!]) \\ \displaystyle\bigcup_{l \geq k \geq 0} \sigma^{l|y|+i}([\![y^{k|in(\sigma)|+j} \cdot \pi]\!]) \end{array} \right. \\ \quad = \left\{ \begin{array}{l} \displaystyle\bigcup_{k \geq l \geq 0} \sigma^i([\![y^{(k-l)|in(\sigma)|+j} \cdot \pi \cdot out(\sigma)^{l|y|}]\!]) \\ \displaystyle\bigcup_{l \geq k \geq 0} \sigma^{(l-k)|y|+i}([\![y^j \cdot \pi \cdot out(\sigma)^{k|y|}]\!]) \end{array} \right. \\ \quad = \left\{ \begin{array}{l} \sigma^i([\![(y^{|in(\sigma)|})^* y^j \cdot \pi \cdot (out(\sigma)^{|y|})^*]\!]) \\ \bigcup \quad (\sigma^{|y|})^*(\sigma^i([\![y^j \cdot \pi \cdot (out(\sigma)^{|y|})^*]\!])) \end{array} \right. \end{array}$$

De plus, comme on a :

$$\begin{array}{rcl} \displaystyle\bigcup_{j=0}^{|in(\sigma)|-1} \sigma^i([\![(y^{|in(\sigma)|})^* y^j \cdot \pi \cdot (out(\sigma)^{|y|})^*]\!]) & = & \sigma^i([\![(y^* \cdot \pi \cdot (out(\sigma)^{|y|})^*]\!]) \\ \displaystyle\bigcup_{i=0}^{|y|-1} (\sigma^{|y|})^*(\sigma^i([\![y^j \cdot \pi \cdot (out(\sigma)^{|y|})^*]\!])) & = & \sigma^*([\![y^j \cdot \pi \cdot (out(\sigma)^{|y|})^*]\!]) \end{array}$$

on obtient finalement que :

$$\sigma^*([\![y^* \cdot \pi]\!]) = \bigcup_{i=0}^{|y|-1} \sigma^i([\![(y^* \cdot \pi \cdot (out(\sigma)^{|y|})^*]\!]) \bigcup_{j=0}^{|in(\sigma)|-1} \sigma^*([\![y^j \cdot \pi \cdot (out(\sigma)^{|y|})^*]\!])$$

Maintenant, pour tout $0 \leq j \leq |in(\sigma)| - 1$, on a $\Delta(y^j \cdot \rho') \leq K$. Par conséquent, on déduit de l'hypothèse d'induction ($\mathbf{H_2}$) qu'il existe un SLRE ρ_j

tel que $\sigma^*([\![y^j \cdot \rho' \cdot (out(\sigma)^{|y|})^*]\!]) = [\![\rho_j]\!]$. On aboutit à :

$$\sigma^*([\![\rho]\!]) = \left[\!\!\left[\sum_{i=0}^{|y|-1} \sigma^i(y^* \cdot \rho' \cdot (out(\sigma)^{|y|})^*) \quad + \sum_{j=0}^{|in(\sigma)|-1} \rho_j \right]\!\!\right]$$

Ceci montre $(\mathbf{H_1})$ pour ρ. Pour prouver $(\mathbf{H_2})$ pour ρ, supposons que $out(\sigma) \neq \varepsilon$. On remarque que pour tout $0 \leq j \leq |in(\sigma)| - 1$, on a :

$$[\![y^j \cdot \rho' \cdot (out(\sigma)^\alpha)^* \cdot (out(\sigma)^{|y|})^*]\!] = \bigcup_{h=0}^{\alpha-1} [\![y^j \cdot \rho' \cdot (out(\sigma)^{|y|})^h \cdot (out(\sigma)^\alpha)^*]\!]$$

Or pour tout $0 \leq j \leq |in(\sigma)| - 1$ et pour tout $0 \leq h \leq \alpha - 1$, on a $\Delta(y^j \cdot \rho' \cdot (out(\sigma)^{|y|})^h) \leq K$. Par conséquent, on déduit de l'hypothèse d'induction $(\mathbf{H_2})$ que $\sigma^*([\![y^j \cdot \rho' \cdot (out(\sigma)^{|y|})^h \cdot (out(\sigma)^\alpha)^*]\!]) = [\![\rho_j^h]\!]$ pour un certain SLRE ρ_j. On aboutit à :

$$\sigma^*([\![\rho]\!]) = \left[\!\!\left[\sum_{i=0}^{|y|-1} \sigma^i(y^* \cdot \rho' \cdot (out(\sigma)^\alpha)^* \cdot (out(\sigma)^{|y|})^*) \quad + \sum_{j=0}^{|in(\sigma)|-1} \sum_{h=0}^{\alpha-1} \rho_j^h \right]\!\!\right]$$

Ceci montre $(\mathbf{H_2})$ pour ρ.

Nous venons donc de montrer par récurrence que pour tout LRE ρ, pour toute séquence d'opérations σ telle que $in(\sigma) \neq \varepsilon$ et pour tout $\alpha \in \mathbb{N}^*$, les assertions $(\mathbf{H_1})$ et $(\mathbf{H_2})$ sont vraies. On conclut la preuve du théorème en notant que ce dernier est une conséquence de $(\mathbf{H_1})$. ■

Comme conséquence immédiate du théorème, on obtient que si $\mathcal{F}$ est un automate à une seule file FIFO, alors la fonction $\nabla_{\mathcal{F}}^{\mathcal{SLRE}}$ est une accélération complète récursive pour l'abstraction $\mathcal{A}_{\mathcal{F}}^{\mathcal{SLRE}}$.

4.4.2 Accélération dans le cas de plusieurs canaux

Nous allons généraliser dans cette section le théorème 4.4.7, au cas de plusieurs canaux. L'ensemble des canaux C est à nouveau quelconque. Contrairement aux CQDD, la représentation symbolique de files FIFO $\mathcal{SLRE}$ ne permet pas d'exprimer des relations fines entre les canaux, comme par exemple "les longueurs des canaux c_1 et c_2 sont égales". Nous allons donc traiter chaque canal de manière indépendante. Afin d'y parvenir, nous aurons besoin de la définition suivante :

Définition 4.4.8 *Etant donné un canal $c \in C$ et une séquence d'opérations σ, la* projection de σ sur c, *notée $\sigma_{|c}$, est le plus grand sous-mot de σ ne contenant que des opérations sur c.*

Observons que si σ et σ' sont deux séquences d'opérations telles que $\sigma_{|c} = \sigma'_{|c}$ pour tout canal c, alors l'effet de σ est exactement le même que celui de σ' (i.e. les fonctions σ et σ' de $(M^*)^C$ dans $(M^*)^C$ sont égales). Cette constatation permet de traiter chaque canal indépendamment, car pour toute séquence d'opérations σ et pour tout vecteur $\mathsf{X} \in \mathcal{P}(M^*)^C$ de sous-ensembles de M^*, on a :

$$\sigma\left(\prod_{c \in C} \mathsf{X}[c]\right) = \prod_{c \in C} \sigma_{|c}(\mathsf{X}[c])$$

On voit bien que cette égalité est encore vraie si on remplace σ par σ^k (et $\sigma_{|c}$ par $(\sigma_{|c})^k$). Malheureusement, elle ne tient plus lorsqu'on remplace σ par σ^* (et $\sigma_{|c}$ par $(\sigma_{|c})^*$). En effet, même en partant d'un ensemble "décorrelé" $\prod_{c \in C} \mathsf{X}[c]$ de contenus de files, l'ensemble de contenus de files obtenu par application de σ^* pourra être "correlé".

Remarque. Pour toute séquence d'opérations σ et pour tout canal c, on déduit du théorème 4.4.7 que $(\sigma_{|c})^*$ est $\mathcal{SLRE}$–représentable, puisque, la séquence d'opérations $\sigma_{|c}$ n'opèrant que sur un seul canal, le contenu des autres canaux reste inchangé par $\sigma_{|c}$.

Nous allons maintenant énoncer le théorème principal de cette section. Rappelons que la notion de séquences d'opérations "capables de compter" est décrite formellement dans la définition 4.3.3 de la page 128.

Théorème 4.4.9 *Pour toute séquence d'opérations σ, σ^* est $\mathcal{SLRE}$–représentable si et seulement si σ ne compte pas.*

Preuve. Soit σ une séquence d'opérations. Commençons par prouver le sens $\Rightarrow$ de l'équivalence, et supposons que σ compte. On peut supposer sans perte de généralité que $C = \{1, 2\}$. Montrons qu'il existe $\chi \in \mathcal{P}_f(SLRE(M)^2)$ tel que $\sigma^*([\![\chi]\!]) \neq [\![\chi']\!]$ pour tout $\chi' \in \mathcal{P}_f(SLRE(M)^2)$. Deux cas se présentent :

1. si $|M| > 1$ alors on a $out(1, \sigma) \neq \varepsilon$ et $out(2, \sigma) \neq \varepsilon$, puisque la condition *i)* de la définition 4.3.3 est satisfaite. Soit m_1 et m_2 deux messages tels que $m_1 \not\leq in(1, \sigma)$ et $m_2 \not\leq in(2, \sigma)$ respectivement. Observons que m_1 et m_2 existent obligatoirement car $|M| > 1$. Soit $\chi \in \mathcal{P}_f(SLRE(M)^2)$ le contenu symbolique de files FIFO défini par $\chi = \{(in(1, \sigma)^* \cdot m_1, in(2, \sigma)^* \cdot m_2)\}$. On obtient que :

$$\sigma^*([\![\chi]\!]) \;=\; \bigcup_{k \in \mathbb{N}} \left(in(1, \sigma)^* \cdot m_1 \cdot out(1, \sigma)^k\right) \times \left(in(2, \sigma)^* \cdot m_2 \cdot out(2, \sigma)^k\right)$$

   et donc $\sigma^*([\![\chi]\!])$ est un ensemble de contenus de files qui n'est pas $\mathcal{SLRE}$–représentable.

2. si $|M| = 1$ alors on a $out(1, \sigma) > in(1, \sigma)$ et $out(2, \sigma) > in(2, \sigma)$, puisque la condition *ii)* de la définition 4.3.3 est satisfaite. Soit $\chi \in \mathcal{P}_f(SLRE(M)^2)$ le contenu symbolique de files défini par $\chi = \{(in(1, \sigma), in(2, \sigma))\}$. On obtient que :

$$\sigma^*([\![\chi]\!]) = \{(in(1, \sigma) \cdot out(1, \sigma)^k,\ in(2, \sigma) \cdot out(2, \sigma)^k) \ /\ k \in \mathbb{N}\}$$

   et donc $\sigma^*([\![\chi]\!])$ est un ensemble de contenus de files qui n'est pas $\mathcal{SLRE}$–représentable.

Montrons à présent le sens $\Leftarrow$ de l'équivalence, et supposons que σ ne compte pas. Nous devons montrer que pour n'importe quel $\chi \in \mathcal{P}_f(SLRE(M)^C)$, il existe $\chi' \in \mathcal{P}_f(SLRE(M)^C)$ satisfaisant $\sigma^*(\llbracket\chi\rrbracket) = \llbracket\chi'\rrbracket$. Comme la fonction σ^* préserve la borne supérieure (i.e. se distribue par rapport à l'union), il suffit en fait de prouver que pour tout $\mathsf{r} \in SLRE(M)^C$, il existe $\mathsf{r}' \in SLRE(M)^C$ tel que $\sigma^*(\llbracket\mathsf{r}\rrbracket) = \llbracket\mathsf{r}'\rrbracket$. Soit donc $\mathsf{r} \in SLRE(M)^C$. Deux cas se présentent :

1. si $|M| > 1$ alors il existe au plus un canal c_0 tel que $out(c_0, \sigma) \neq \varepsilon$, puisque la condition $i)$ de la définition 4.3.3 n'est pas satisfaite. Ainsi, pour tout canal $c \neq c_0$, on a $out(c, \sigma) = \varepsilon$ et donc $(\sigma_{|c})^i(\llbracket\mathsf{r}[c]\rrbracket) = (in(c, \sigma)^i)^{-1}\llbracket\mathsf{r}[c]\rrbracket$. Or $\llbracket\mathsf{r}[c]\rrbracket$ est un langage rationnel, donc il existe $h_c, k_c \in \mathbb{N}$ avec $h_c > 0$ tels que $(in(c, \sigma)^{k_c+h_c})^{-1}\llbracket\mathsf{r}[c]\rrbracket = (in(c, \sigma)^{k_c})^{-1}\llbracket\mathsf{r}[c]\rrbracket$. En posant :
$$\begin{array}{rcl} k & = & \mathrm{Max}(\{k_c \;/\; c \in C, c \neq c_0\}) \\ h & = & \mathrm{ppcm}(\{h_c \;/\; c \in C, c \neq c_0\}) \end{array}$$
on obtient que $(in(c, \sigma)^{k+h})^{-1}\llbracket\mathsf{r}[c]\rrbracket = (in(c, \sigma)^{k})^{-1}\llbracket\mathsf{r}[c]\rrbracket$ pour tout canal $c \neq c_0$. Par conséquent, on aboutit à $(\sigma_{|c})^{k+h}(\llbracket\mathsf{r}[c]\rrbracket) = (\sigma_{|c})^{k}(\llbracket\mathsf{r}[c]\rrbracket)$ pour tout canal $c \neq c_0$.

2. si $|M| = 1$ alors il existe au plus un canal c_0 tel que $out(c_0, \sigma) > in(c_0, \sigma)$, puisque la condition $ii)$ de la définition 4.3.3 n'est pas satisfaite. Ainsi, pour tout canal $c \neq c_0$, on a $out(c, \sigma) \leq in(c, \sigma)$. Puisque $|M| = 1$, les canaux se comportent comme des compteurs, et on peut naturellement associer à $\sigma_{|c}$, avec $c \neq c_0$, une fonction d_c associant à un facteur gauche $x \leq \sigma_{|c}$ son "déplacement" dans $\mathbb{Z}$ défini par $d_c(x) = |out(c, x)| - |in(c, x)|$. Soit σ'_c un facteur gauche de $\sigma_{|c}$ pour lequel d_c est minimale, et soit $\sigma''_c = (\sigma'_c)^{-1}\sigma_{|c}$. Clairement, la fonction de déplacement associée à $(\sigma''_c\sigma'_c)$ est minimale en $(\sigma''_c\sigma'_c)$. On en déduit que tout ce qui est émis le long de la séquence d'opérations $(\sigma''_c\sigma'_c)$ finit par être réceptionné au cours de $(\sigma''_c\sigma'_c)$. Par conséquent, pour tout canal $c \neq c_0$, on a $(\sigma''_c\sigma'_c)(\llbracket\mathsf{r}[c]\rrbracket) = (w_c)^{-1}\llbracket\mathsf{r}[c]\rrbracket$, où $w_c = out(c, \sigma)^{-1}in(c, \sigma)$. Par un raisonnement analogue à celui du cas où $|M| > 1$ ci-dessus, on obtient qu'il existe $h, k \in \mathbb{N}$ avec $h > 0$ tels que $(\sigma''_c\sigma'_c)^{k+h}(\llbracket\mathsf{r}[c]\rrbracket) = (\sigma''_c\sigma'_c)^{k}(\llbracket\mathsf{r}[c]\rrbracket)$ pour tout canal $c \neq c_0$. Comme $(\sigma_{|c})^{i+1}(\mathsf{r}[c]) = \sigma'_c((\sigma''_c\sigma'_c)^i(\sigma''(\mathsf{r}[c])))$ pour tout $i \in \mathbb{N}$, on aboutit à $(\sigma_{|c})^{k+1+h}(\llbracket\mathsf{r}[c]\rrbracket) = (\sigma_{|c})^{k+1}(\llbracket\mathsf{r}[c]\rrbracket)$ pour tout canal $c \neq c_0$.

Dans les deux cas, nous avons montré qu'il existe $h, k \in \mathbb{N}$ avec $h > 0$ tels que $(\sigma_{|c})^{k+h}(\llbracket\mathsf{r}[c]\rrbracket) = (\sigma_{|c})^{k}(\llbracket\mathsf{r}[c]\rrbracket)$ pour tout canal $c \neq c_0$.

Décomposons maintenant $\sigma^*(\llbracket\mathsf{r}\rrbracket)$ de la façon suivante :
$$\begin{array}{rcl} \sigma^*(\llbracket\mathsf{r}\rrbracket) & = & \displaystyle\bigcup_{i\in\mathbb{N}} \sigma^i(\llbracket\mathsf{r}\rrbracket) \\ & = & \displaystyle\bigcup_{i=0}^{k-1} \sigma^i(\llbracket\mathsf{r}\rrbracket) \quad \bigcup \quad \bigcup_{i=k}^{k+h-1} \bigcup_{j\in\mathbb{N}} \sigma^{i+jh}(\llbracket\mathsf{r}\rrbracket) \end{array}$$

Observons que $\bigcup_{i=0}^{k-1} \sigma^i([\![\mathsf{r}]\!])$ est $\mathcal{SLRE}$–représentable, comme union finie d'ensembles $\mathcal{SLRE}$–représentable. Afin de conclure la preuve, montrons que l'union $\bigcup_{j\in\mathbb{N}} \sigma^{i+jh}([\![\mathsf{r}]\!])$ est $\mathcal{SLRE}$–représentable pour tout $i \in [k \mathinner{..} k+h-1]$. Soit donc $i \in [k \mathinner{..} k+h-1]$. On a :

$$\begin{aligned}
\bigcup_{j\in\mathbb{N}} \sigma^{i+jh}([\![\mathsf{r}]\!]) &= \bigcup_{j\in\mathbb{N}} \prod_{c\in C} (\sigma_{|c})^{i+jh}([\![\mathsf{r}[c]]\!]) \\
&= \bigcup_{j\in\mathbb{N}} \left(\left(\prod_{c\in C, c\neq c_0} (\sigma_{|c})^{i}([\![\mathsf{r}[c]]\!]) \right) \times (\sigma_{|c_0})^{i+jh}([\![\mathsf{r}[c_0]]\!]) \right) \\
&= \left(\prod_{c\in C, c\neq c_0} (\sigma_{|c})^{i}([\![\mathsf{r}[c]]\!]) \right) \times \left(\bigcup_{j\in\mathbb{N}} (\sigma_{|c_0})^{i+jh}([\![\mathsf{r}[c_0]]\!]) \right) \\
&= \left(\prod_{c\in C, c\neq c_0} (\sigma_{|c})^{i}([\![\mathsf{r}[c]]\!]) \right) \times \left(\left((\sigma_{|c_0})^h\right)^* \left((\sigma_{|c_0})^i([\![\mathsf{r}[c_0]]\!])\right) \right)
\end{aligned}$$

On déduit du théorème 4.4.7 que $\left((\sigma_{|c_0})^h\right)^*$ est $\mathcal{SLRE}$–représentable, car la séquence d'opérations $\sigma_{|c_0}$ n'opère que sur un seul canal, ce qui permet de conclure la preuve. ■

Rappelons que pour toute séquence d'opérations σ, σ^* est $\mathcal{QDD}$–représentable si et seulement si σ ne compte pas [BGWW97]. Il vient donc :

Corollaire 4.4.10 *Pour toute séquence d'opérations σ, σ^* est $\mathcal{SLRE}$–représentable si et seulement si σ^* est $\mathcal{QDD}$–représentable.*

Comme conséquence immédiate du théorème, on obtient que si $\mathcal{F}$ est un automate à files FIFO tel que σ ne compte pas pour tout circuit de contrôle $q \xrightarrow{\sigma} q$, alors la fonction $\nabla_{\mathcal{F}}^{\mathcal{SLRE}}$ est une accélération complète récursive pour l'abstraction $\mathcal{A}_{\mathcal{F}}^{\mathcal{SLRE}}$.

Remarquons que notre théorème est incomparable avec les théorèmes 4.3.4 et 4.3.8 rappelant les résultats de [BGWW97, BH99] sur l'itération des séquences d'opérations avec les QDD et les CQDD. En effet,

– on sait d'après le théorème 4.3.4 que σ^* est $\mathcal{QDD}$–représentable pour toute séquence d'opérations σ. Par conséquent $\sigma^*([\![\chi]\!])$ est $\mathcal{QDD}$–représentable pour tout $\chi \in \mathcal{P}_f(SLRE(M)^C)$, mais ceci n'implique pas que $\sigma^*([\![\chi]\!])$ soit $\mathcal{SLRE}$–représentable.

– le théorème 4.3.8 implique que σ^* est $\mathcal{CQDD}$–représentable pour toute séquence d'opérations σ. Cependant, la preuve donnée dans [BH99] se sert beaucoup des formules de Presburger, même dans le cas d'un seul canal, et il ne nous semble donc pas évident de déduire notre théorème de [BH99].

Nous verrons en section 4.5 qu'une preuve *indirecte* de notre théorème peut être obtenue en *combinant* les théorèmes 4.3.4 et 4.3.8.

Accélération complète approchée

Nous montrons maintenant comment définir une accélération complète (mais non correcte) pour l'abstraction $\mathcal{A}_{\mathcal{F}}^{\mathcal{SLRE}}$, dans le cas d'un ensemble de canaux C quelconque. Pour tout séquence d'opérations σ, nous définissons la fonction $\widetilde{\sigma^*}$ de $\mathcal{P}_f(SLRE(M))^C$ dans $\mathcal{P}_f(SLRE(M))^C$ comme suit :

$$\widetilde{\sigma^*}(\chi) = \begin{cases} \displaystyle\bigcup_{\mathsf{r}\in\chi} \prod_{c\in C} (\sigma_{|c})^*_{\mathcal{SLRE}}(\mathsf{r}[c]) & \text{si } \sigma \text{ est infiniment itérable à partir de } [\![\chi]\!] \\ \displaystyle\sum_{i=0}^{k} \sigma^i_{\mathcal{SLRE}}(\chi) & \text{si } \sigma^k([\![\chi]\!]) = \emptyset \end{cases}$$

On remarque que pour tout $\chi \in \mathcal{P}_f(SLRE(M))^C$, on a $\sigma^*([\![\chi]\!]) \subseteq [\![\widetilde{\sigma^*}(\chi)]\!]$. Ainsi, la fonction $\widetilde{\sigma^*}$ donne une approximation supérieure de l'effet de l'itération d'une séquence d'opérations.

L'accélération $\widetilde{\nabla}_{\mathcal{F}}^{\mathcal{SLRE}}$ basée sur la fonction $\widetilde{\sigma^*}$ est définie de la même façon que l'accélération $\nabla_{\mathcal{F}}^{\mathcal{SLRE}}$ (voir la définition 2.4.24 de la page 67), excepté que la fonction $\widetilde{\sigma^*}$ est utilisée à la place de la fonction $\sigma^*_{\mathcal{SLRE}}$. Formellement, nous définissons $\widetilde{\nabla}_{\mathcal{F}}^{\mathcal{SLRE}} : (\mathcal{P}_f(SLRE(M))^C)^Q \times (C \times \{?,!\} \times M)^+ \to (\mathcal{P}_f(SLRE(M))^C)^Q$ par :

$$\widetilde{\nabla}_{\mathcal{F}}^{\mathcal{SLRE}}(\mathsf{a}, \sigma) = \left\langle q \mapsto \sum_{q'\in Q} iter(q', q, \sigma, \mathsf{a}[q']) \right\rangle$$

où pour tout $q, q' \in Q$, $\sigma \in (C \times \{?,!\} \times M)^+$ et $\chi \in \mathcal{P}_f(SLRE(M))^C$, on définit $iter(q, q', \sigma, \chi)$ par :

$$iter(q, q', \sigma, \chi) = \sum_{x\in X} \delta_{\mathcal{SLRE}}(\sigma^x, \chi) \quad + \quad \sum_{i\in I} \widetilde{(\sigma^{p_i})^*} \left(\sum_{b\in B_i} \delta_{\mathcal{SLRE}}(\sigma^b, \chi) \right)$$

avec $(X, (B_i, p_i)_{i\in I})$ description canonique de $SL(\mathsf{F}, q, q', \sigma)$.

Proposition 4.4.11 *Etant donné un automate à files* FIFO $\mathcal{F}$*, la fonction* $\widetilde{\nabla}_{\mathcal{F}}^{\mathcal{SLRE}}$ *est une accélération complète récursive pour l'abstraction* $\mathcal{A}_{\mathcal{F}}^{\mathcal{SLRE}}$.

Rappelons que lorsque $|M| = 1$, un automate à files FIFO $\mathcal{F}$ peut être vu comme un réseau de Petri (ou de manière équivalente un SAVE). Dans ce cas particulier, l'accélération $\widetilde{\nabla}_{\mathcal{F}}^{\mathcal{SLRE}}$ est plus précise que la stratégie de KARP et MILLER pour le calcul d'un arbre de couverture [KM69], car elle produit pour chaque compteur un sous-ensemble semilinéaire de $\mathbb{N}$, et non pas un sous-ensemble clos inférieurement de $\mathbb{N}$.

4.5 Comparaison des QDD, CQDD et SLRE

Nous analysons dans cette section les différences entre les trois représentations symboliques de files FIFO que nous avons présentées. Commençons par comparer leur pouvoir d'expression :

Théorème 4.5.1 *Soit $X \subseteq (M^*)^C$ un sous-ensemble de contenus de files. Les deux assertions suivantes sont équivalentes :*

i) X est $\mathcal{SLRE}$–représentable,

ii) X est à la fois $\mathcal{QDD}$–représentable et $\mathcal{CQDD}$–représentable.

Preuve. Nous prouvons les deux directions de l'équivalence.

$\boxed{i) \Rightarrow ii)}$ Supposons que $X = [\![\chi]\!]$ pour un certain représentant symbolique de $\mathcal{SLRE}$ $\chi \in \mathcal{P}_f(SLRE(M)^C)$. Observons que χ est également un représentant symbolique de $\mathcal{QDD}$. Par conséquent, il reste à montrer que $X = [\![\chi']\!]$ pour un certain représentant symbolique de $\mathcal{CQDD}$ χ'.

Nous commençons par montrer que pour tout LRE ρ, il existe un SLRE $\rho' = \rho_0 + \rho_1 + \cdots + \rho_m$ tel que :

1. on a $[\![\rho']\!] = [\![\rho]\!]$, et
2. pour tout $i \in [0\,..\,m]$, le LRE ρ_i ne contient aucun facteur de la forme $(au)^*a$ ou u^*v^* avec $u, v \in M^*$ et $a \in M$.

Remarquons tout d'abord que si un LRE ρ contient un facteur de la forme $(au)^*a$, alors on peut remplacer $(au)^*a$ par $a(ua)^*$ dans ρ, puisque cette transformation préserve le langage $[\![\rho]\!]$ associé à ρ. Nous continuons de remplacer les facteur de la forme $(au)^*a$ par $a(ua)^*$ dans ρ, jusqu'à ce qu'il n'y ait plus aucun facteur de la forme $(au)^*a$ dans ρ. Le processus termine obligatoirement car ces remplacements déplacent les occurrences de '$*$' vers la droite dans ρ. Nous appliquons ensuite les deux transformations suivantes sur ρ, préservant elles aussi $[\![\rho]\!]$, afin de supprimer les facteurs de la forme u^*v^*.

– si $u^\omega = v^\omega$ alors on a $u^{|v|} = v^{|u|}$. Par conséquent $[\![u^*v^*]\!] = \left[\!\!\left[\sum_{i=0}^{|v|-1} u^i v^*\right]\!\!\right]$ et on peut donc éliminer le facteur u^*v^* de ρ en découpant ρ sous la forme d'une somme ρ'.

– si $u^\omega \neq v^\omega$ alors on a $u^{|v|} \neq v^{|u|}$. Par conséquent il existe $u', u'', v', v'' \in \Sigma^*$ et $k, l \in \mathbb{N}$ tels que $v^l v' = u^k u'$, $u = u'u''$, $v = v'v''$, $u'' \neq \varepsilon$, $v'' \neq \varepsilon$ et tels que u'' et v'' ne commencent pas par le même symbole. Pour tout $i \in [0\,..\,l]$, il existe des mots $u'_i, u''_i \in \Sigma^*$ tels que $v^i \in u^* u'_i$ et $u = u'_i u''_i$. On obtient que :

$$\begin{aligned} [\![u^*v^*]\!] &= \left[\!\!\left[\sum_{i=0}^{l} u^* v^i\right]\!\!\right] \quad \bigcup \quad \left[\!\!\left[u^* v^{l+1} v^*\right]\!\!\right] \\ &= \left[\!\!\left[\left(\sum_{i=0}^{l} v^i (u''_i u'_i)^*\right) + v^l v' (u'' u')^* v'' v^*\right]\!\!\right] \end{aligned}$$

et on peut donc éliminer u^*v^* de ρ en découpant ρ sous la forme d'une somme ρ' (remarquons que $(u''u')^* v'' v^*$ ne contient aucun facteur de la forme $(au)^*a$ ou u^*v^*).

Après avoir appliqué l'une des deux transformations ci-dessus, on ré-élimine les facteurs de la forme $(au)^*a$, qui auraient pu être ajoutés. Nous continuons de supprimer les facteurs de la forme u^*v^*, puis $(au)^*a$ dans ρ, jusqu'à ce qu'il n'y ait plus aucun facteur de la forme u^*v^* ou $(au)^*a$ dans ρ. Notons que dans les transformations d'élimination des facteurs de la forme u^*v^* ci dessus,

- soit le nombre d'occurrences de '$*$' diminue,
- soit aucun facteur de la forme $(au)^*a$ ou u^*v^* n'est ajouté par la transformation.

Par conséquent, le processus termine obligatoirement et on en déduit qu'il existe bien un SLRE ρ' satisfaisant les conditions $i)$ et $ii)$ ci-dessus.

Maintenant, d'après l'assertion que nous venons de prouver, on obtient que pour tout $\mathsf{r} \in \chi$ et $c \in C$, il existe une famille finie $(\rho_i)_{i\in I_{\mathsf{r},c}}$ de LRE tels que :

1. on a $[\![\mathsf{r}[c]]\!] = \bigcup_{i\in I_{\mathsf{r},c}} [\![\rho_i]\!]$, et
2. pour tout $i \in I_{\mathsf{r},c}$, le LRE ρ_i ne contient aucun facteur de la forme $(au)^*a$ ou u^*v^* avec $u, v \in M^*$ et $a \in M$.

Par conséquent, on aboutit à :

$$\begin{aligned} L = [\![\chi]\!] &= \bigcup_{\mathsf{r}\in\chi} \prod_{c\in C} [\![\mathsf{r}[c]]\!] \\ &= \bigcup_{\mathsf{r}\in\chi} \prod_{c\in C} \bigcup_{i\in I_{\mathsf{r},c}} [\![\rho_i]\!] \\ &= \bigcup_{\mathsf{r}\in\chi} \bigcup_{\mathsf{i}\in\prod_{c\in C} I_{\mathsf{r},c}} \prod_{c\in C} [\![\rho_{\mathsf{i}[c]}]\!] \\ &= [\![\chi']\!] \end{aligned}$$

où χ' est le représentant symbolique de $\mathcal{CQDD}$ défini par :

$$\chi' = \bigcup_{\mathsf{r}\in\chi} \bigcup_{\mathsf{i}\in\prod_{c\in C} I_{\mathsf{r},c}} (< c \mapsto \rho_{\mathsf{i}[c]} >, true)$$

$\boxed{ii) \Rightarrow i)}$ Supposons qu'il existe à la fois un représentant symbolique de $\mathcal{QDD}$ $\chi \in \mathcal{P}_f(REG(M)^C)$ et un représentant symbolique de $\mathcal{CQDD}$ $\chi' \in CQDD(C, M)$ tels que $[\![\chi]\!] = [\![\chi']\!] = X$. Observons que $[\![\chi']\!] \subseteq \bigcup_{(\mathsf{r}',\varphi)\in\chi'} \prod_{c\in C} [\![\mathsf{r}'[c]]\!]$. Par conséquent, il vient :

$$\begin{aligned} [\![\chi]\!] &= \left(\bigcup_{\mathsf{r}\in\chi} \prod_{c\in C} [\![\mathsf{r}[c]]\!]\right) \cap \left(\bigcup_{(\mathsf{r}',\varphi)\in\chi'} \prod_{c\in C} [\![\mathsf{r}'[c]]\!]\right) \\ &= \bigcup_{\mathsf{r}\in\chi,(\mathsf{r}',\varphi)\in\chi'} \left(\prod_{c\in C} [\![\mathsf{r}[c]]\!] \cap \prod_{c\in C} [\![\mathsf{r}'[c]]\!]\right) \\ &= \bigcup_{\mathsf{r}\in\chi,(\mathsf{r}',\varphi)\in\chi'} \prod_{c\in C} \left([\![\mathsf{r}[c]]\!] \cap [\![\mathsf{r}'[c]]\!]\right) \end{aligned}$$

Rappelons que r' est un vecteur de LRE pour tout $(\mathsf{r}', \varphi) \in \chi'$. Par conséquent, d'après la proposition 2.2.4, pour tout $\mathsf{r} \in \chi$, $(\mathsf{r}', \varphi) \in \chi'$ et $c \in C$, il existe un SLRE $\rho_{\mathsf{r},\mathsf{r}',c}$ tel que $\rho_{\mathsf{r},\mathsf{r}',c} = [\![\mathsf{r}[c]]\!] \cap [\![\mathsf{r}'[c]]\!]$. On obtient finalement que $X = [\![\chi]\!] = [\![\chi'']\!]$ où χ'' est le représentant symbolique de $\mathcal{SLRE}$ défini par :

$$\chi'' = \bigcup_{\mathsf{r} \in \chi, (\mathsf{r}, \varphi) \in \chi'} < c \mapsto \rho_{\mathsf{r},\mathsf{r}',c} >$$

■

Remarque. Soit σ une séquence d'opérations. Rappelons que σ^* est $\mathcal{SLRE}$–représentable si et seulement si pour tout sous-ensemble $X \subseteq (M^*)^C$ $\mathcal{SLRE}$–représentable, $\sigma^*(X)$ est $\mathcal{SLRE}$–représentable. Or d'après le théorème précédent, $\sigma^*(X)$ est $\mathcal{SLRE}$–représentable si et seulement si $\sigma^*(X)$ est à la fois $\mathcal{QDD}$–représentable et $\mathcal{CQDD}$–représentable. On en déduit que σ^* est $\mathcal{SLRE}$–représentable si et seulement si σ^* est à la fois $\mathcal{QDD}$–représentable et $\mathcal{CQDD}$–représentable. Or d'après les théorèmes 4.3.4 et 4.3.8, on sait que :

– σ^* est nécessairement $\mathcal{CQDD}$–représentable, et

– σ^* est $\mathcal{QDD}$–représentable si et seulement si σ ne compte pas.

On retrouve donc ainsi, en combinant les théorèmes 4.3.4 et 4.3.8, notre théorème 4.4.9 affirmant que σ^* est $\mathcal{SLRE}$–représentable si et seulement si σ ne compte pas. L'intérêt de la preuve directe de ce résultat, donnée dans la section 4.4, est qu'elle est constructive et n'utilise que les LRE. On peut donc en déduire un algorithme direct pour le calcul de la fonction $\sigma^*_{\mathcal{SLRE}}$, et cet algorithme est a priori plus efficace que d'utiliser les fonctions $\sigma^*_{\mathcal{QDD}}$ et $\sigma^*_{\mathcal{CQDD}}$, puis le théorème ci-dessus pour combiner leur résultat.

Il est clair que les opérations d'émission symbolique, de réception symbolique et de somme peuvent être calculées en temps *linéaire* pour la représentation symbolique $\mathcal{SLRE}$. De plus, nous avons les résultats de complexité suivants pour les problèmes d'inclusion et d'équivalence :

Théorème 4.5.2 *Les problèmes d'inclusion et d'équivalence entre deux SLRE sont* coNP*–complets.*

Preuve. Observons que si ρ et ρ' sont deux SLRE, alors on a :

1. $[\![\rho]\!] \not\subseteq [\![\rho']\!]$ si et seulement si $[\![\rho']\!] \neq [\![\rho + \rho']\!]$, et
2. $[\![\rho]\!] \neq [\![\rho']\!]$ si et seulement si $[\![\rho']\!] \not\subseteq [\![\rho]\!]$ ou $[\![\rho]\!] \not\subseteq [\![\rho']\!]$

Par conséquent, il suffit de prouver que le problème d'inéquivalence entre deux SLRE est NP–complet.

Pour montrer que le problème d'inéquivalence entre deux SLRE est dans NP, nous utilisons un résultat de [HRS76], affirmant que le problème d'inéquivalence entre deux expressions régulières dénotant des langages *bornés* est dans NP. Puisque les SLRE sont clairement des expressions régulières dénotant des langages bornés, on en déduit que le problème d'inéquivalence entre deux SLRE est aussi dans NP.

La preuve que ce problème est NP–difficile se trouve en fait dans [SM73]. Il y est démontré que le problème d'inéquivalence entre deux expressions régulières sur un alphabet à une seule lettre $\{a\}$ est NP–complet. Cependant, l'auteur prouve en réalité un résultat plus fort, puisqu'il réduit en espace logarithmique 3–SAT au problème $\{\rho \in SLRE(\{a\}) \ / \ [\![\rho]\!] \neq a^*\}$. Ceci montre que le problème d'inéquivalence entre deux SLRE est NP–difficile (et cela reste vrai si on ne considère que des SLRE sur un alphabet à une lettre). ■

Nous ne connaissons pas de résultat précis de complexité concernant les QDD et des CQDD, mais les problèmes d'inclusion et d'équivalence pour ces contenus symboliques de files FIFO requièrent au moins un espace polynomial.

	un unique canal	plusieurs canaux
$\mathcal{SLRE}$	$\nabla_{\mathcal{F}}^{\mathcal{SLRE}}$: correcte, complète	$\nabla_{\mathcal{F}}^{\mathcal{SLRE}}$: correcte, non complète $\widetilde{\nabla}_{\mathcal{F}}^{\mathcal{SLRE}}$: non correcte, complète
$\mathcal{QDD}$	$\nabla_{\mathcal{F}}^{\mathcal{QDD}}$: correcte, complète	$\nabla_{\mathcal{F}}^{\mathcal{QDD}}$: correcte, non complète
$\mathcal{CQDD}$	$\nabla_{\mathcal{F}}^{\mathcal{CQDD}}$: correcte, complète	$\nabla_{\mathcal{F}}^{\mathcal{CQDD}}$: correcte, complète

TAB. 4.1 – Comparaison des accélérations

Comparés aux QDD et aux CQDD, les SLRE semblent avoir plus d'intérêt pratique, car ils sont en général *suffisants* pour représenter symboliquement des ensembles de contenus de files FIFO et pour itérer des séquences d'opérations (voir la table 4.1), et de plus ils ont une *meilleure complexité*. En partant d'un unique état initial d'un automate à une seule file FIFO, l'algorithme générique **`ArbreSymbolique`** de la page 34 appliqué aux abstractions $\mathcal{A}_{\mathcal{F}}^{\mathcal{QDD}}$, $\mathcal{A}_{\mathcal{F}}^{\mathcal{CQDD}}$ et $\mathcal{A}_{\mathcal{F}}^{\mathcal{SLRE}}$, munies de leur accélération générique, donnera le même résultat. Dans le cas des automates à plusieurs files FIFO, les accélérations $\nabla_{\mathcal{F}}^{\mathcal{SLRE}}$ et $\nabla_{\mathcal{F}}^{\mathcal{QDD}}$, appliquées à un état abstrait de l'abstraction basée sur la représentation symbolique $\mathcal{SLRE}$, produisent le même résultat.

4.6 Automates à files FIFO non fiables

D'autres sémantiques opérationnelles peuvent être associées à un contrôle d'automate à files FIFO, afin de modéliser par exemple le cas où les canaux sont non fiables [AJ96b, CFP96, Céc98]. Les canaux peuvent alors *perdre*, *insérer* ou *dupliquer* des messages, ce qui donne lieu à des automates à files FIFO *avec perte*, *avec insertion* ou *avec duplication* respectivement. Rappelons que la duplication de messages n'altère pas la puissance du modèle, car les automates à files FIFO avec duplication ont toujours la puissance des machines de Turing [CFP96]. En revanche, le problème de l'accessibilité est décidable pour les automates à files FIFO avec perte, et pour les automates à files FIFO avec insertion [AJ96b, CFP96].

On remarque que l'automate à files FIFO avec perte (resp. avec insertion) associé à un contrôle d'automate à files FIFO $\mathsf{F} = (Q, C, M, \to_{\mathsf{F}})$ est en fait la clôture inférieure $\downarrow (\mathcal{F}, \preceq)$ (resp. clôture supérieure $\uparrow (\mathcal{F}, \preceq)$) de l'automate à files FIFO $\mathcal{F}$ pour l'ordre partiel $\preceq$ sur l'ensemble $S = Q \times (M^*)^C$ des états de $\mathcal{F}$ défini comme suit :

$$(q, \mathsf{w}) \preceq (q', \mathsf{w}') \quad \text{si} \quad q = q' \;\; \text{et} \;\; \mathsf{w}[c] \,|\, \mathsf{w}'[c] \text{ pour tout canal } c \in C$$

De plus, le système de transtions étiqueté structuré $(\mathcal{F}, \preceq)$ ainsi que son dual $(\overline{\mathcal{F}}, \preceq)$ admettent une base supérieure post finie effective. Par conséquent, on peut appliquer le schéma d'abstraction décrit en sous-section 2.3.1 à $\uparrow \mathcal{F}$ et à $\overline{\downarrow \mathcal{F}} = \uparrow \overline{\mathcal{F}}$. Puisque l'ordre partiel sous-mot est un beau pré-ordre sur M^*, on obtient que $\preceq$ est un beau pré-ordre sur S. Par conséquent, on obtient d'après le théorème 2.3.7 que pour tout sous-ensemble fini $X \subseteq S$ d'états initiaux,

– une base finie de $\mathrm{post}^*(\uparrow \mathcal{F}, X)$ est calculable, et

– une base finie de $\mathrm{pre}^*(\downarrow \mathcal{F}, X) = \mathrm{post}^*(\uparrow \overline{\mathcal{F}}, X)$ est calculable.

On retrouve ainsi les résultats décrits dans [CFP96] de reconnaissabilité effective de post* pour les automates à files FIFO avec insertion, et de pre* pour les automates à files FIFO avec perte.

Observons que l'ensemble des états accessibles post* dans un automate à files FIFO avec perte est nécessairement clos inférieurement, et par conséquent il est reconnaissable. Cependant, il n'est pas effectivement reconnaissable, car il n'existe pas d'algorithme prenant en entrée un contrôle d'automate à files FIFO F et un état de contrôle initial q_0, et calculant une représentation finie sous forme d'automates de $\mathrm{post}^*(\downarrow \mathcal{F}, (q_0, (\varepsilon, \varepsilon, \ldots, \varepsilon)))$ [CFP96]. Notons que la preuve décrite dans [CFP96] utilise une file intermédiaire, et ne prouve donc pas l'ennoncé ci-dessus lorsque F ne dispose que d'une seule file. De plus, le problème du caractère borné était conjecturé indécidable pour les automates à files FIFO avec perte dans [CFP96], mais ce problème était resté ouvert depuis. En utilisant le résultat (plus récent) d'indécidabilité du caractère borné dans les SAVE étendus avec perte [DFS98, BM99], nous prouvons maintenant que le problème du caractère borné est indécidable pour les automates à files FIFO avec perte, même restreints à une seule file. Une conséquence immédiate de ce résultat est que post* n'est pas *effectivement* reconnaissable pour les automates à une seule file FIFO avec perte.

Théorème 4.6.1 *Le problème du caractère borné est indécidable pour les automates à une seule file* FIFO *avec perte et d'alphabet de messages* $\{a, \#\}$.

Preuve. On réduit le problème du caractère borné dans les SAVE de dimension 3 étendus par des test-à-zéro avec perte au problème du caractère borné pour les automates à une seule file FIFO avec perte. L'idée de la construction consiste à encoder le contenu $(x, y, z) \in \mathbb{N}^3$ des compteurs par le mot $w = a^x \# a^y \# a^z \#$, le symbole # permettant de délimiter les parties de w encodant x, y et z respectivement. Afin d'incrémenter (resp. de décrémenter) le contenu d'un compteur, on "déroule" la file afin de placer le début de la zone encodant le compteur en tête de file, puis on émet (resp. réceptionne) juste une fois de trop le message correspondant au compteur. Le test-à-zéro sur un compteur se fera d'une manière similaire, en vérifiant que la partie encodant le compteur est vide. On note que le # peut également être perdu dans l'automate à une seule file FIFO avec perte, mais cela conduit alors immédiatement à un état de blocage, car l'opération de "déroulage" ne pourra plus se faire correctement. On vérifie que pout tout $(x, y, z) \in \mathbb{N}^3$, l'état $(q, (x, y, z))$ est accessible dans le SAVE avec perte si et seulement si $(q, a^x \# a^y \# a^z \#)$ est accessible dans l'automate à une seule file FIFO avec perte. D'un autre côté, pour toute exécution infinie de l'automate à une seule file FIFO avec perte, sur laquelle la file est non bornée, on ne perd jamais de #, et par conséquent, pour tous les états (q, w) le long de cette exécution, on obtient que $w \in a^* \# a^* \# a^* \#$. On en déduit que l'ensemble des états accessibles dans le SAVE avec perte est infini si et seulement si l'ensemble des états accessibles dans l'automate à une seule file FIFO avec perte est infini. ∎

Corollaire 4.6.2 *L'ensemble des états accessibles* post* *dans un automate à une seule file* FIFO *avec perte n'est pas effectivement reconnaissable.*

Une variante des automates à files FIFO avec perte, les *protocoles complètement spécifiés*, a été introduite dans [Fin94], où le problème de la terminaison est démontré décidable pour cette classe. Les protocoles complètement spécifiés sont des automates à files FIFO dont le contrôle contient une transition $q \xrightarrow{c?m} q$ pour chaque état de contrôle q, pour chaque canal c et pour chaque message m. Un tel automate à files FIFO peut se voir comme un automate à files FIFO avec perte *en tête de file* [Céc98]. Notons $\mathcal{F}_{pt}$ l'automate à files FIFO avec *perte en tête* associé à un contrôle d'automate à files FIFO quelconque F. Des liens étroits existent entre l'automate à files FIFO avec perte $\downarrow \mathcal{F}$ et l'automate à files FIFO avec perte en tête $\mathcal{F}_{pt}$. Plus précisément, il est démontré dans [Céc98, page 41] qu'étant donné un contrôle d'automate à files FIFO F et un ensemble X d'états initiaux, $\mathrm{post}^*(\downarrow \mathcal{F}, X)$ est fini si et seulement si $\mathrm{post}^*(\mathcal{F}_{pt}, X)$ est fini. On déduit donc du théorème 4.6.1 le corollaire suivant, qui ferme un problème ouvert depuis [Fin94].

Corollaire 4.6.3 *Le problème du caractère borné est indécidable pour les automates à une seule file* FIFO *avec perte en tête et d'alphabet de messages* $\{a, \#\}$.

Observons que les résultats d'indécidabilité du caractère borné que nous avons obtenus sont des "résultats limites", car une seule file et un alphabet de messages restreint à deux symboles suffisent à rendre ce problème indécidable. Si on diminue la puissance du modèle, en n'autorisant qu'un seul symbole dans l'alphabet de messages, alors les files se comportent comme des compteurs et le problème du caractère borné devient donc décidable, même avec plusieurs files.

Abstraction et accélération des automates à files FIFO avec perte

On peut chercher à semi-calculer l'ensemble des états accessibles dans un automate à files FIFO avec perte $\downarrow \mathcal{F}$, en utilisant l'abstraction $\mathcal{A}^b_{\downarrow \mathcal{F}}$ définie en sous-section 2.3.1. Cependant, on remarque que la clôture inférieure $\downarrow(X, \preceq)$ d'un sous-ensemble fini d'états X est ici nécessairement finie, car l'ensemble des sous-mots d'un mot donné $w \in M^*$ est fini. Ainsi, l'abstraction $\mathcal{A}^b_{\downarrow\mathcal{F}}$ ne permet de manipuler que des sous-ensembles finis d'états !

Pour pallier à ce problème, une représentation symbolique pour les sous-ensembles d'états clos inférieurement — les *expressions régulières simples* (*SRE*) — a été introduite dans [ABJ98]. Les SRE sont suffisamment expressives, puisque tout sous-ensemble d'états clos inférieurement (et en particulier post*) est SRE-définissable. De plus, pour tout circuit de contrôle de $\downarrow\mathcal{F}$, un SRE représentant l'ensemble des états accessibles par itération de ce circuit (avec perte) est calculable [ABJ98]. Ceci conduit naturellement à une abstraction de $\downarrow\mathcal{F}$ basée sur les SRE, et à une accélération correcte et complète pour cette abstraction.

Chapitre 5

Analyse des automates à file réactifs

Les programmes réactifs doivent réagir instantanément aux occurrences d'événements provenant de leur environnement. Cependant, un système réactif peut parfois se trouver dans un état où il lui est impossible de réagir "normalement" à une occurrence d'événement. Dans ce cas, le programme peut soit *oublier* une telle occurrence d'événement, soit la *mémoriser* afin de la traiter ultérieurement.

Nous analysons dans ce chapitre les *automates à file réactifs*, qui permettent de modéliser des systèmes réactifs avec mémorisation d'événements, comme par exemple ceux écrits dans le langage réactif ELECTRE [CR95, SFRC99]. Les occurrences d'événements ne pouvant pas être traitées dans l'état courant du système sont mémorisés dans une file afin d'être traités ultérieurement. Cependant, la file ne respecte pas la sémantique FIFO standard. En effet, afin de réagir au plus vite sur une occurrence d'événement mémorisée, la priorité est donnée aux "post-traitements" à partir de la file, et on parle alors de sémantique FIFFO (pour *First In First Fireable Out*).

Le modèle initial introduit dans [CR95, SFRC99] satisfaisait de fortes contraintes structurelles, correspondant au langage réactif ELECTRE dont il était issu, et qui nous ont permis d'obtenir de nombreux résultats de décidabilité pour la vérification [SFRC99]. Ces résultats de décidabilité sont néanmoins surprenants car ce modèle est proche de celui des automates à une seule file FIFO, qui a la puissance des machines de Turing. Nous isolons dans ce chapitre les contraintes structurelles d'une part, et les différentes sémantiques entre la sémantique FIFO et la sémantique FIFFO d'autre part. Nous analysons ainsi exhaustivement les différents modèles obtenus, du point de vue de la décidabilité de l'accessibilité, ainsi que de l'effective reconnaissabilité de post* et de pre*.

Nous motiverons dans un premier temps l'introduction du modèle des automates à files réactifs en section 5.1, puis nous définirons ce modèle en section 5.2. Nous analyserons le problème de l'accessibilité pour les différentes sous-classes d'automates à files réactifs dans la section 5.3. Nous étudierons ensuite la reconnaissabilité de post* pour ces sous-classes en section 5.4. Nous conclurons ce chapitre en rappelant les résultats les plus importants que nous avons obtenus sur les automates à files réactifs dans la section 5.5.

5.1 Systèmes réactifs avec mémorisation et protocoles de communication

Nous motivons dans cette section l'introduction du modèle des automates à file réactifs, à l'aide de deux exemples : les systèmes réactifs avec mémorisation écrits en ELECTRE et les protocoles de communication écrits en SDL.

5.1.1 Le langage ELECTRE

Un programme réactif est censé réagir instantanément aux occurrences d'événements provenant de son environnement. Bien souvent, de tels systèmes réactifs doivent gérer des cas d'exclusion mutuelle entre deux types d'événements, comme dans le cas bien classique des *lecteurs/écrivains* [CHP71]. Ainsi, le système peut parfois se trouver dans un état où il ne peut pas traiter une occurrence d'événement (comme par exemple une requête de lecture alors qu'un écrivain est en train de modifier le livre). Une solution qui a le mérite d'être simple à mettre en œuvre, consiste à *oublier* de telles occurrences d'événements, mais cette approche n'est pas très équitable ! On souhaiterait plutôt stocker de telles occurrences d'événements, pour les *traiter ultérieurement*, mais *le plus tôt possible.*

Afin de permettre la spécification et la programmation de tels systèmes, le langage réactif ELECTRE [RCCE92, CR95] propose deux types d'événements :

– les événements *fugaces*, dont les occurrences sont soit traitées immédiatement, soit perdues.

– les événements *mémorisables*, dont les occurrences ne pouvant pas être traitées immédiatement sont mémorisées dans une file (a priori non bornée), afin d'être prises en compte *ultérieurement*, mais *le plus tôt possible*, et *dans l'ordre* (temporel) de leurs occurrences.

Un programme ELECTRE se compile en une structure de contrôle finie [CR95], à laquelle on adjoint une file (non bornée), afin de stocker les occurrences d'événements mémorisables [SFRC99]. Intuitivement, lorsque le système change d'état de contrôle, la file est scrutée afin de traiter, si possible, une occurrence d'événement mémorisée, auquel cas on la retire de la file. Si deux occurrences d'événements mémorisées peuvent être traitées, alors la priorité est donnée à l'occurrence la plus ancienne, c'est-à-dire à celle se trouvant le plus près du début de la file.

Exemple 5.1.1 La figure 5.1 présente la structure de contrôle obtenue par compilation d'un programme ELECTRE résolvant le problème des lecteurs/écrivains, à deux lecteurs et à deux écrivains. Cet exemple est détaillé dans [Sut97, SFRC99], ainsi que le passage du programme ELECTRE à cette structure de contrôle. Nous nous contentons ici de remarquer que :

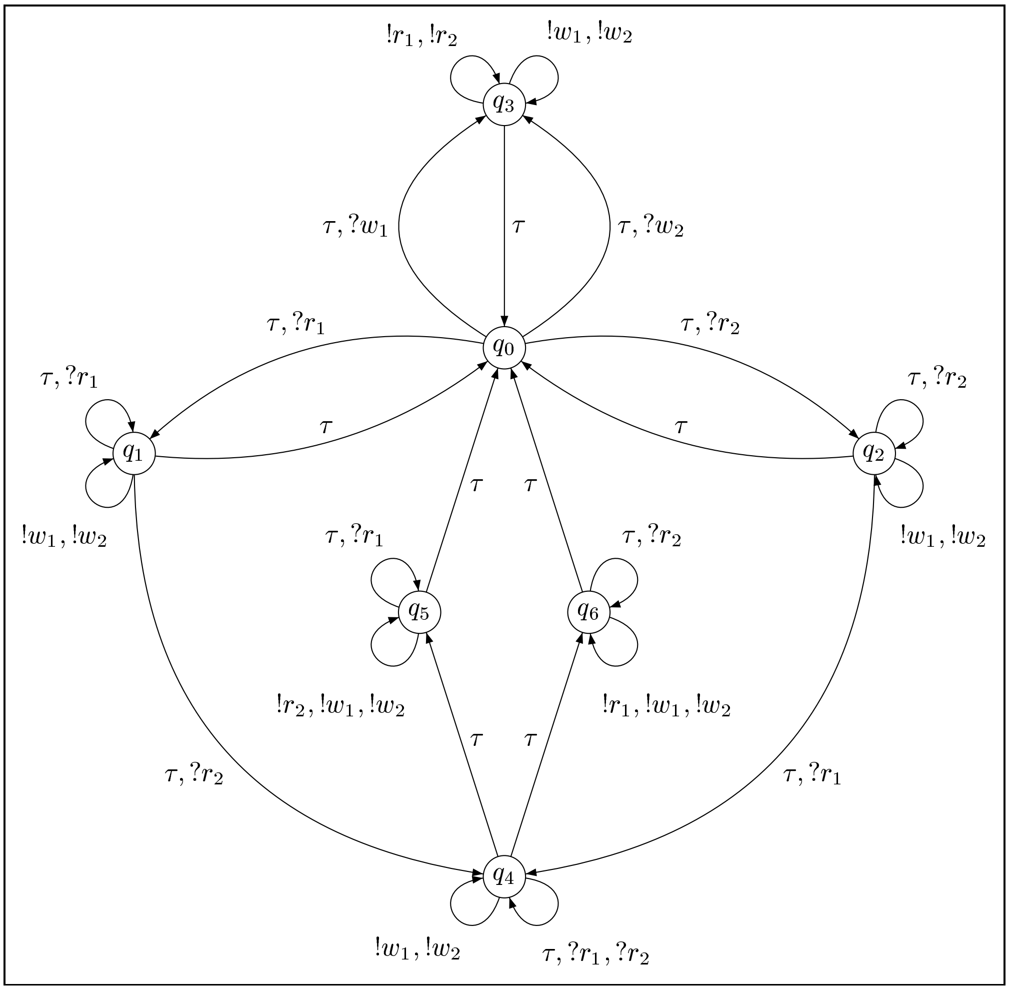


FIG. 5.1 – Un automate à file réactif modélisant les lecteurs/écrivains

– les requêtes de lecture (resp. d'écriture) sont r_1 et r_2 (resp. w_1 et w_2).

– les τ-transitions dénotent des réactions immédiates à des requêtes de lecture ou d'écriture, ou encore à des fins de requête. Ces transitions étaient originellement étiquetées par $r_1, r_2, w_1, w_2, end_{R_1}, end_{R_2}, end_{W_1}$ et end_{W_2}, mais leur étiquette n'a en fait pas d'importance du point de vue des propriétés d'accessiblité ; aussi avons-nous choisi de les étiqueter uniformément par τ.

– une transition étiquetée par $!e$ dénote une mémorisation de l'événement e. Par exemple dans l'état q_3, un écrivain est en train d'accéder au livre, et par conséquent les requêtes de lecture ne peuvent pas être traitées ; elles sont donc mémorisées.

– une transition étiquetée par $?e$ dénote un "post-traitement" de l'événement e. Par exemple lorsque l'écrivain termine sa requête d'écriture, ce qui est représenté par la transition $q_3 \xrightarrow{\tau} q_0$, alors une requête mémorisée d'un écrivain peut être traitée, ce qui est représenté par les transitions $q_0 \xrightarrow{?r_1} q_1$ et $q_0 \xrightarrow{?r_2} q_2$.

La sémantique formelle des programmes Electre est ainsi donnée sous la forme d'automates à file, proches des automates à file Fifo, mais :

– intégrant une notion de *priorité* : les "post-traitements" (i.e. les réceptions) à partir de la file sont prioritaires sur les autres opérations, et

– permettant de réceptionner des éléments de la file se situant arbitrairement loin dans la file, et

– comportant de fortes contraintes structurelles. Par exemple, les mémorisations (i.e. les émissions) se font en restant dans le même état de contrôle.

Comme nous le verrons dans la suite de ce chapitre, ces automates à file particuliers forment une sous-classe d'automates à file réactifs, pour laquelle de nombreux problèmes de vérification sont décidables.

5.1.2 Le standard SDL

Le langage SDL est un standard pour la description et la conception de protocoles de communication, issu de l'Union Internationale des Télécommunications (UIT). Ce langage est basé sur le modèle formel des automates à files Fifo, et un *diagramme* SDL se présente en fait comme un morceau d'automate à files Fifo, "au sucre syntaxique près". Cependant, SDL comporte quelques variantes sémantiques *implicites* par rapport au modèle classique des automates à files Fifo.

Par exemple, il est couramment admis que les messages ne doivent pas bloquer les canaux, c'est-à-dire que le système a toujours la possibilité de supprimer, en tête des canaux, les messages pour lesquels aucune réception n'est spécifiée. Cette hypothèse revient en fait à supposer, pour tout état de contrôle q et pour tout canal c, l'existence de transitions $q \xrightarrow{c?m} q$ pour tous les messages m tels que m ne peut pas être réceptionné depuis le canal c en q.

Cependant, il pourrait être plus intéressant de pouvoir scruter le canal à la recherche d'un message particulier (comme par exemple un acquittement). D'une manière plus générale, on pourrait chercher dans le canal le premier message pour lequel une réception est spécifiée. Cette idée rejoint le fonctionnement de la file des systèmes réactifs avec mémorisation écrits en Electre. On pourrait aussi choisir de privilégier la réception de messages, afin de vider les canaux le plus possible, ce qui, d'une certaine façon, forcerait le protocole à être "réactif".

Enfin, une dernière variante serait aussi envisageable. Le fonctionnement décrit ci-dessus consiste à scruter la file jusqu'à ce qu'un message puisse être réceptionné. Le sémantique standard Fifo, quant à elle, ne regarde que le premier élément de la file. On pourrait choisir un fonctionnement intermédiaire, plus facile à implanter en pratique, où on regarderait les l premiers éléments de la file, avec $l \in \mathbb{N}$ fixé. L'entier l servirait ainsi de "taille de fenêtre" des canaux.

5.2 Automates à files réactifs

Nous décrivons dans cette section le modèle des automates à file réactifs, ainsi que plusieurs sous-classes définies par des contraintes structurelles. Nos définitions généralisent à la fois la notion d'*automate réactif* FIFFO (*RFA*) que nous avions donnée dans [SFRC99], et la notion d'automate à file FIFO du chapitre 4. La structure de contrôle d'un automate à file réactif est définie comme suit :

Définition 5.2.1 *Un* contrôle d'automate à file réactif *est un triplet* $\mathsf{F} = (Q, M, \rightarrow)$*, où :*

- Q *est un ensemble fini d'*états de contrôle*, et*

- M *est un alphabet de* messages *(ou d'*événements*), et*

- *l'ensemble* $\rightarrow \subseteq Q \times ((\{?, !\} \times M) \cup \{\tau\}) \times Q$ *est un ensemble fini de* transitions de contrôle*.*

Ainsi, un contrôle d'automate à file réactif peut être vu comme contrôle d'automate à une seule file FIFO, comportant de plus des τ-transitions. Ces τ-transitions représentent des actions internes de l'automate à file réactif, comme par exemple des réactions immédiates à des occurrence d'événements, et correspondent donc à des ε-transitions[1]. En revanche, tout comme dans le modèle des automates à une seule file FIFO, une transition de contrôle de la forme $(q, ?m, q')$ représente une réception du message m (ou un post-traitement d'une occurrence d'événement m) depuis la file, et une transition de contrôle de la forme $(q, !m, q')$ représente une émission du message m (ou une mémorisation d'une occurrence d'événement m) dans la file.

Exemple 5.2.2 Le contrôle d'automate à file réactif LE décrit en figure 5.1 est formellement défini par $\mathsf{LE} = (Q, M, \rightarrow_{\mathsf{LE}})$, où :

- $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$ est l'ensemble des états de contrôle, et

- $M = \{r_1, r_2, w_1, w_2\}$ est l'alphabet des messages, et

- l'ensemble $\rightarrow_{\mathsf{LE}} \subseteq Q \times ((\{?, !\} \times M) \cup \{\tau\}) \times Q$ des transitions de contrôle contient par exemple les éléments $q_0 \xrightarrow{?r_1} q_1$, $q_1 \xrightarrow{!w_1} q_1$ et $q_5 \xrightarrow{\tau} q_0$.

Remarque. Dans toute la suite de ce chapitre, nous désignerons par M un alphabet de messages. L'alphabet d'opérations sera donc $(\{?, !\} \times M) \cup \{\tau\}$.

Afin d'associer des sémantiques opérationnelles "réactives" à un contrôle d'automate à file réactif, nous aurons besoin de la notation suivante :

[1]Nous avons choisi de noter les actions internes avec un τ au lieu d'un ε afin de mieux les distinguer du mot vide.

Notation. Etant donné un contrôle d'automate à file réactif $\mathsf{F} = (Q, M, \rightarrow)$, on associe à tout état de contrôle q les sous-ensembles de messages R_q et E_q définis comme suit :

$$\begin{array}{rcl} R_q & = & \{m \in M \;/\; \exists\; q' \in Q,\, q \xrightarrow{?m} q'\} \\ E_q & = & \{m \in M \;/\; \exists\; q' \in Q,\, q \xrightarrow{!m} q'\} \end{array}$$

Remarque. Dans le cas des systèmes réactifs avec mémorisation d'événements écrits en ELECTRE, R_q dénote l'ensemble des événements que le système peut traiter lorsqu'il se trouve dans l'état de contrôle q.

5.2.1 Sémantiques opérationnelles

Nous définissons maintenant les différentes sémantiques opérationnelles "réactives" que nous étudierons dans la suite de ce chapitre. Ces sémantiques opérationnelles formalisent les idées intuitives de la section 5.1 de ce chapitre. Contrairement aux systèmes d'addition de vecteurs avec états étendus et aux automates à files FIFO, nous n'associons pas ici de sémantique à l'alphabet d'opérations $\mathtt{Op} = (\{?, !\} \times M) \cup \{\tau\}$, car le franchissement d'une transition de contrôle d'un automate à file réactif dépend non seulement du contenu de la file, mais aussi des autres transitions de contrôle partant du même état de contrôle, données par les ensembles R_q et E_q.

Rappelons que $\mathbb{N}_\omega$ désigne la fermeture classique $\mathbb{N} \cup \{\omega\}$ de l'ensemble $\mathbb{N}$, où $n < \omega$ pour tout $n \in \mathbb{N}$.

Définition 5.2.3 *Un contrôle d'automate à file réactif* $\mathsf{F} = (Q, M, \rightarrow_{\mathsf{F}})$*, muni d'une* taille de fenêtre $l \in \mathbb{N}_\omega$*, définit le système de transitions étiqueté* $\mathcal{F} = (S, \Sigma, \rightarrow_{\mathcal{F}})$*, appelé* automate à file réactif FIFFO$[l]$*, comme suit :*

- $S = Q \times (M^*)$ *est l'ensemble des états, et*
- $\Sigma = (\{?, !\} \times M) \cup \{\tau\}$ *est l'alphabet d'opérations, et*
- *la relation de transitions étiquetée* $\rightarrow_{\mathcal{F}}$ *est le plus petit sous-ensemble* $\rightarrow_{\mathcal{F}}$ *de* $S \times \Sigma \times S$ *satisfaisant pour tout* $w, w_1, w_2 \in M^*$*, avec* $|w_1| \leq l$*, les deux conditions suivantes :*

$$\begin{array}{llll} \textit{si} & q \xrightarrow{?m}_{\mathsf{F}} q' \;\; \textit{et} \;\; w_1 \in (M \setminus R_q)^* & \textit{alors} & (q, w_1 m w_2) \xrightarrow{?m}_{\mathcal{F}} (q', w_1 w_2) \\ \textit{si} & q \xrightarrow{!m}_{\mathsf{F}} q' & \textit{alors} & (q, w) \xrightarrow{!m}_{\mathcal{F}} (q', wm) \\ \textit{si} & q \xrightarrow{\tau}_{\mathsf{F}} q' & \textit{alors} & (q, w) \xrightarrow{\tau}_{\mathcal{F}} (q', w) \end{array}$$

Ainsi, comme dans le modèle des automate à une seule file FIFO, un état (q, w) d'un automate à file réactif FIFFO$[l]$ est composé d'un *état de contrôle* q et d'un mot w représentant le contenu de la file. Les conditions de franchissement des transitions d'émission et des τ-transitions sont classiques. Cependant, le cas des transitions de réception est moins classique car il ne suit pas la sémantique FIFO standard. Les réceptions tiennent compte de la "taille de fenêtre" $l \in \mathbb{N}_\omega$, qui indique jusqu'où la file peut être scrutée pour trouver un message pouvant être réceptionné (i.e. appartenant à R_q). Plus précisément, un message m peut être réceptionné si :

– d'une part il se trouve parmi les $(l+1)$ premiers messages de la file (en partant du début de la file), et

– d'autre part il est le premier, parmi ces $(l+1)$ premiers messages de la file, à pouvoir être réceptionné (i.e. à appartenir à R_q).

Lors de la réception de m, on retire la première occurrence de m présente dans la file. On observe que lorsque $l = \omega$, le message m peut être réceptionné, même si la première occurrence de m se situe arbitrairement loin dans la file.

Exemple 5.2.4 Dans l'automate à file réactif FIFFO$[\omega]$ associé au contrôle d'automate à file réactif LE, on a par exemple le chemin suivant :

$$\begin{array}{c}(q_0,\varepsilon) \xrightarrow{\tau} (q_3,\varepsilon) \xrightarrow{!r_1} (q_3,r_1) \xrightarrow{!w_2} (q_3,r_1w_2) \xrightarrow{!r_2} (q_3,r_1w_2r_2) \xrightarrow{\tau} (q_0,r_1w_2r_2) \\ \Big\downarrow{\scriptstyle ?r_1} \\ \ldots(q_3,r_2w_1) \xleftarrow{?w_2} (q_0,w_2r_2w_1) \xleftarrow{\tau} (q_4,w_2r_2w_1) \xleftarrow{!w_1} (q_1,w_2r_2)\end{array}$$

Remarque. La sémantique opérationnelle FIFFO$[0]$ coïncide avec la sémantique opérationnelle FIFO standard, dans le sens où l'automate à file réactif FIFFO$[0]$ associé à un contrôle d'automate à file réactif F est égal à l'automate à file FIFO associé à F (vu comme contrôle d'automate à file FIFO).

Remarque. Si $R_q = M$ pour tout état de contrôle q, alors pour tout $l \in \mathbb{N}_\omega$, la sémantique opérationnelle FIFFO$[l]$ coïncide avec la sémantique opérationnelle FIFO standard.

Ajout de la priorité aux réceptions

Nous analyserons également la variante de la sémantique opérationnelle ci-dessus où la priorité est donnée aux réceptions. Intuitivement, cette variante est obtenue à partir de la définition ci-dessus en “désactivant” les transitions d'émission et les τ-transitions lorsqu'une transition de réception est franchissable. Formellement,

Définition 5.2.5 *Un contrôle d'automate à file réactif* $\mathsf{F} = (Q, M, \to_{\mathsf{F}})$, *muni d'une* taille de fenêtre $l \in \mathbb{N}_\omega$, *définit le système de transitions étiqueté* $\mathcal{F} = (S, \Sigma, \to_{\mathcal{F}})$, *appelé* automate à file réactif FIFFO$[l]$ avec priorité aux réceptions, *comme suit :*

– $S = Q \times (M^*)$ *est l'ensemble des états, et*

– $\Sigma = (\{?,!\} \times M) \cup \{\tau\}$ *est l'alphabet d'opérations, et*

– *la relation de transitions étiquetée* $\to_{\mathcal{F}}$ *est le plus petit sous-ensemble* $\to_{\mathcal{F}}$ *de* $S \times \Sigma \times S$ *satisfaisant pour tout* $w, w_1, w_2 \in M^*$, *avec* $|w_1| \leq l$, *les deux conditions suivantes :*

$$\begin{array}{llllr} si & q \xrightarrow{?m}_{\mathsf{F}} q' & et\ \ w_1 \in (M \setminus R_q)^* & alors & (q, w_1mw_2) \xrightarrow{?m}_{\mathcal{F}} (q', w_1w_2) \\ si & q \xrightarrow{!m}_{\mathsf{F}} q' & et\ \ w \notin \mathit{R\acute{e}cep}(q,l) & alors & (q, w) \xrightarrow{!m}_{\mathcal{F}} (q', wm) \\ si & q \xrightarrow{\tau}_{\mathsf{F}} q' & et\ \ w \notin \mathit{R\acute{e}cep}(q,l) & alors & (q, w) \xrightarrow{\tau}_{\mathcal{F}} (q', w) \end{array}$$

où Récep(q, l) *est l'ensemble des mots* $x \in M^*$ *s'écrivant sous la forme* $x = x_1 m x_2$ *avec* $x_1, x_2 \in M^*$, $|x_1| \leq l$ *et* $m \in R_q$.

Remarque. L'automate à file réactif FIFFO$[l]$ avec priorité aux réceptions associé à un contrôle d'automate à file réactif F contient moins de transitions (et par conséquent moins de chemins) que l'automate à file réactif FIFFO$[l]$ associé à F.

Remarque. La sémantique formelle d'un programme ELECTRE est donnée par un automate à file réactif FIFFO$[\omega]$ avec priorité aux réceptions, et satisfaisant, de plus, de fortes contraintes structurelles [CR95, SFRC99].

Exemple 5.2.6 Dans l'automate à file réactif FIFFO$[\omega]$ associé au contrôle d'automate à file réactif LE, on a par exemple le chemin suivant :

$$\begin{array}{ccc} (q_0, \varepsilon) \xrightarrow{\tau} (q_3, \varepsilon) \xrightarrow{!r_1} (q_3, r_1) \xrightarrow{!w_2} (q_3, r_1 w_2) \xrightarrow{!r_2} (q_3, r_1 w_2 r_2) \xrightarrow{\tau} & & (q_0, r_1 w_2 r_2) \\ & & \big\downarrow {?r_1} \\ \ldots (q_3, \varepsilon) \xleftarrow{?w_2} (q_0, w_2) \xleftarrow{\tau} (q_6, w_2) \xleftarrow{\tau} (q_4, w_2) \xleftarrow{?r_2} & & (q_1, w_2 r_2) \end{array}$$

En revanche, il ne contient pas le chemin de l'exemple 5.2.4, car la transition $(q_1, w_2 r_2) \xrightarrow{!w_1} (q_4, w_2 r_2 w_1)$ ne respecte pas la priorité aux réceptions. En effet, la seule transition possible depuis l'état $(q_1, w_2 r_2)$ est $(q_1, w_2 r_2) \xrightarrow{?r_2} (q_4, w_2)$.

Remarque. Si $R_q \in \{\emptyset, M\}$ pour tout état de contrôle q, alors pour tout $l \in \mathbb{N}_\omega$, la sémantique opérationnelle FIFFO$[l]$ avec priorité aux réceptions coïncide avec la sémantique opérationnelle FIFFO$[0]$ avec priorité aux réceptions.

Remarque. Il est assez naturel de donner la priorité aux transitions de réception, car ces dernières vident la file et permettent de réagir plus rapidement aux occurrences d'événement mémorisées. En revanche, il serait absurde de donner la priorité aux transitions d'émission ou aux τ-transitions car ces dernières sont toujours franchissables.

5.2.2 Classes d'automates à files réactifs

On se rend compte aisément que les automates à file réactifs ont la puissance des machines de Turing, quelle que soit la sémantique opérationnelle choisie. En effet, pour tout automate à une seule file FIFO $\mathcal{F}$, on peut construire un automate à file réactif simulant $\mathcal{F}$, comme indiqué par la figure 5.2, et ceci quelle que soit la sémantique opérationnelle réactive choisie. L'idée de la construction consiste à :

- introduire des états de contrôle intermédiaires, pour dissocier les transitions d'émission des transitions de réception, et s'affranchir ainsi de l'éventuelle priorité aux réceptions, et à
- ajouter des transitions de réception vers un état puits afin de se ramener à une sémantique FIFO standard.

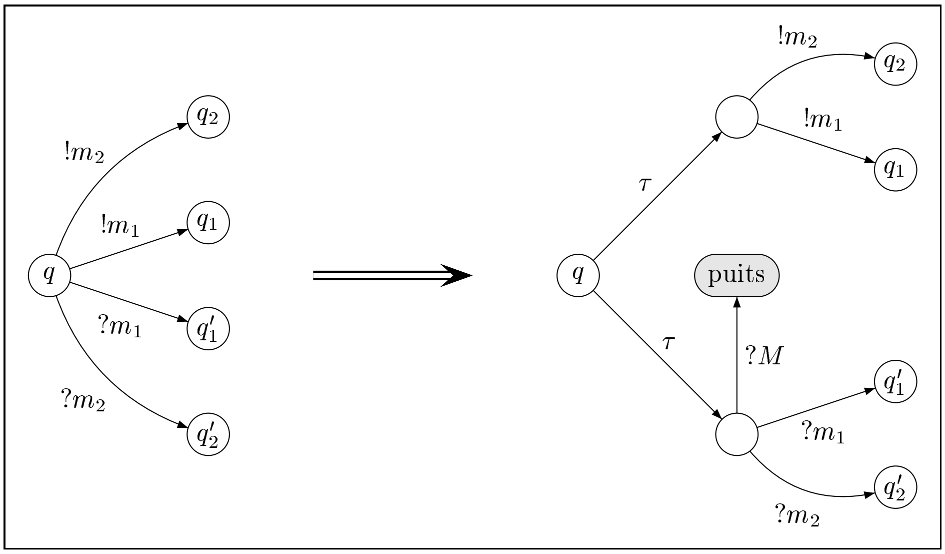
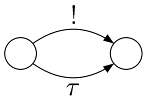
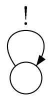
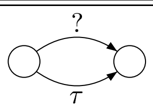
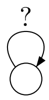


FIG. 5.2 – Simulation d'un automate à file FIFO par un automate à file réactif

Aussi allons-nous analyser des sous-classes d'automates à file réactifs, définies par des *contraintes structurelles* provenant de notre étude des systèmes réactifs avec mémorisation d'événements écrits en ELECTRE [Sut97, SFRC99].

Contraintes structurelles sur les transitions de contrôle

	pour tout $q, q' \in Q$ et $m \in M$, si $q \xrightarrow{!m} q'$ alors $q \xrightarrow{\tau} q'$
	pour tout $q, q' \in Q$ et $m \in M$, si $q \xrightarrow{!m} q'$ alors $q = q'$
	pour tout $q, q' \in Q$ et $m \in M$, si $q \xrightarrow{?m} q'$ alors $q \xrightarrow{\tau} q'$
	pour tout $q, q' \in Q$ et $m \in M$, si $q \xrightarrow{?m} q'$ alors $q = q'$

TAB. 5.1 – Contraintes structurelles sur les transitions de contrôle

Nous considérons quatre contraintes structurelles sur la forme des transitions de contrôle, données dans le tableau 5.1. Les lignes 2 et 3 du tableau sont des conditions naturelles pour des systèmes réactifs avec mémorisation d'événements, comme par exemple dans le cas d'ELECTRE. Elles ont la signification intuitive suivante :

– ligne 2 : lorsqu'une occurrence d'événement ne peut pas être traitée dans l'état de contrôle courant, alors on la stocke dans la file, mais *on ne change pas d'état de contrôle* car le système n'a pas réagi !

– ligne 3 : un post-traitement d'une occurrence d'événement doit avoir le *même effet qu'un traitement immédiat* d'une occurrence du même événement, c'est-à-dire qu'une τ-transition, puisque nous avons choisi de représenter uniformément par des τ-transitions les réactions immédiates.

Les lignes 1 et 4 du tableau ont été ajoutées par souci d'exhaustivité de l'analyse. Cependant, elles sont également intéressantes, car :

– la ligne 1 correspond en fait à la notion de *perte de message*, comme indiqué dans [ABIJ00]. En effet, lorsque cette condition est satisfaite, tous les messages qui ont été envoyés auraient aussi pu ne pas l'être, en choisissant de franchir la τ-transition au lieu de la transition d'envoi.

– la ligne 4 est proche de la notion de *perte en tête*, ou de *protocole complètement spécifié*, introduite dans [Fin94].

Contraintes structurelles de complétude

$E_q, R_q \subseteq M$	aucune contrainte (E_q et R_q quelconques)
$E_q \cup R_q = M$	pour tout $q \in Q$, $E_q \cup R_q = M$
$E_q \oplus R_q = M$	pour tout $q \in Q$, $E_q \cup R_q = M$ et $E_q \cap R_q = \emptyset$
$R_q = M$	pour tout $q \in Q$, $R_q = M$
$E_q = M$	pour tout $q \in Q$, $E_q = M$

TAB. 5.2 – Contraintes structurelles de complétude

Les systèmes réactifs doivent en général être capable de réagir à tout moment (i.e. dans tout état du système), et ceci pour toute occurrence d'événement. Dans le cas des systèmes réactifs avec mémorisation d'événements écrits en ELECTRE, cette contrainte de *complétude* est donnée par la ligne 3 du tableau 5.2, qui exprime que les occurrences d'événements qui sont mémorisées dans un état de contrôle q sont précisément celles que le système ne peut pas traiter en q. Nous avons également choisi d'analyser une condition plus faible définie par la ligne 2, où l'on relâche l'hypothèse $E_q \cap R_q = \emptyset$.

D'un autre côté, les protocoles de communication satisfont généralement la condition donnée par la ligne 4 du tableau 5.2, qui permet de s'assurer qu'aucun message ne peut bloquer indéfiniment le canal. La condition duale $E_q = M$ a été ajoutée par souci d'exhaustivité de l'analyse.

Remarque. Les trois conditions $E_q \oplus R_q = M$, $R_q = M$ et $E_q = M$ sont incomparables, mais elles sont toutes plus fortes que la condition $E_q \cup R_q = M$.

Remarque. On peut toujours compléter un automate à file réactif en rajoutant des boucles $q \xrightarrow{\tau} q$ sur chaque état de contrôle q, sans changer sa relation binaire d'accessibilité. De plus cette modification du contrôle respecte les contraintes structurelles de complétude, car ces dernières ne portent pas sur les τ-transitions. Ainsi, si un contrôle d'automate à file réactif satisfait la condition de la ligne 2 (resp. de la ligne 4) du tableau 5.1, alors il satisfait aussi la condition de la ligne 1 (resp. de la ligne 3).

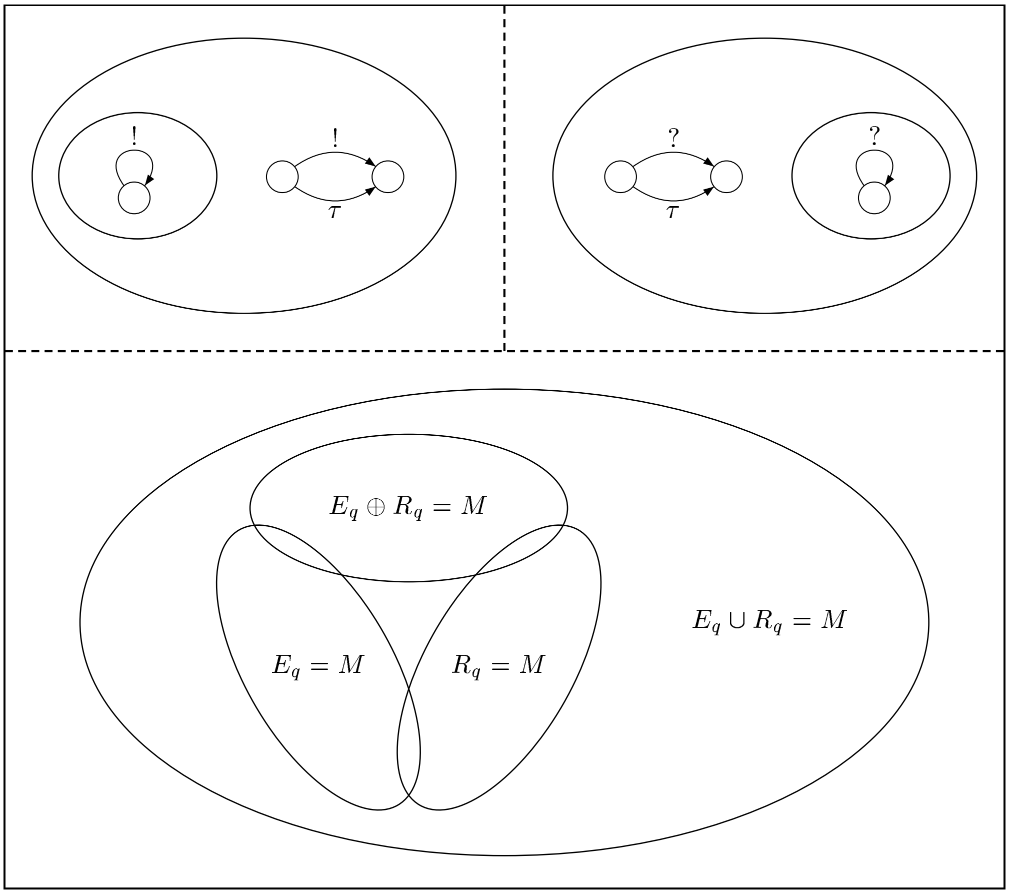


FIG. 5.3 – Hiérarchie des contraintes structurelles sur les automates à file réactifs

Les sous-classes d'automates à file réactifs que nous considérerons dans toute la suite de ce chapire sont données par le choix :

- d'une sémantique opérationnelle (par exemple FIFFO[5] avec priorité aux réceptions), et
- d'une contrainte structurelle sur les transitions de contrôle (par exemple pour tout $q, q' \in Q$ et $m \in M$, si $q \xrightarrow{?m} q'$ alors $q = q'$), et
- d'une contrainte structurelle de complétude (par exemple pour tout $q \in Q$, $E_q \cup R_q = M$ et $E_q \cap R_q = \emptyset$).

Nous étudierons en général plusieurs sous-classes en même temps, en nous donnant uniquement des contraintes structurelles, et en autorisant implicitement n'importe quelle sémantique opérationnelle.

Remarque. Nous n'examinons pas le cas des sous-classes d'automates à file réactifs où les transitions de contrôle sont quelconques (i.e. les sous-classes définies uniquement par une sémantique opérationnelle et par des contraintes structurelles de complétude). En effet, ces sous-classes se décomposent en deux catégories :

- les sous-classes de sémantique opérationnelle FIFFO[l] avec priorité aux réceptions et satisfaisant la condition $R_q = M$ sont trivialement analysables. Effectivement, le long d'une exécution d'un automate à file réactif appartenant à l'une de ces sous-classes, la file reste bornée à un message, sauf si elle est initialement non vide, auquel cas elle est tout d'abord vidée.
- toutes les autres sous-classes ont clairement la puissance des machines de Turing, car on peut toujours compléter le contrôle d'automate à file réactif, construit dans la figure 5.2, par des transitions d'émission dirigées vers un état puits.

5.3 Décidabilité de l'accessibilité

Notre analyse des automates à file réactifs porte sur le calcul des états accessibles, à partir d'un état initial fixé. Plus précisément, pour toute sous-classe $\mathfrak{C}$ d'automates à file réactifs, nous examinerons :

- le problème de l'accessibilité pour $\mathfrak{C}$, et
- la reconnaissabilité de $\text{post}^*(q_0, w_0)$ pour les automates à file réactifs de $\mathfrak{C}$ avec unique état initial (q_0, w_0), et
- la construction effective de $\text{post}^*(q_0, w_0)$ lorsqu'il est reconnaissable. Formellement, lorsque $\text{post}^*(q_0, w_0)$ est reconnaissable pour $\mathfrak{C}$, nous dirons qu'il est *constructible* pour $\mathfrak{C}$ lorsqu'on peut calculer, pour tout automate à file réactif de $\mathfrak{C}$ et pour tout état initial (q_0, w_0), un automate fini reconnaissant $\text{post}^*(q_0, w_0)$.

Nous nous intéressons dans cette section au problème de l'accessibilité, pour les sous-classes d'automates à file réactifs où les ensembles E_q et R_q sont quelconques (i.e. les sous-classes définies uniquement par une sémantique opérationnelle et par des contraintes structurelles sur les transitions de contrôle).

Nous commençons par étudier la sous-classe $\mathfrak{L}$ des automates à file réactifs dont le contrôle satisfait la ligne 1 du tableau 5.1. Rappelons que cette condition exprime que pour tout $q, q' \in Q$ et $m \in M$, si $q \xrightarrow{!m} q'$ alors $q \xrightarrow{\tau} q'$.

Proposition 5.3.1 *Le problème de l'accessibilité est décidable pour $\mathfrak{L}$. De plus, pour tout automate à file réactif $\mathcal{F} \in \mathfrak{L}$ et pour tout état initial (q_0, w_0) de $\mathcal{F}$,* $\text{post}^*(q_0, w_0)$ *est reconnaissable.*

Preuve. Soit $\mathcal{F}$ un automate à file réactif de $\mathfrak{L}$ et soit (q_0, w_0) un état initial de $\mathcal{F}$. Posons $k = |w_0|$ et définissons la relation binaire $|_k$ sur les mots de M^* par $w \,|_k\, w'$ si $w = w'$ ou s'il existe $z, x, x' \in M^*$ tels que $w = zx$, $w' = zx'$, $|z| = k$ et $x \,|\, x'$. Intuitivement on a $w \,|_k\, w'$ lorsque d'une part w est un sous-mot de w', et d'autre part le facteur gauche de longueur k de w coïncide avec le facteur gauche de longueur k de w'. On vérifie aisément que cette relation binaire est un ordre partiel sur M^*, et un beau pré-ordre sur M^*.

Maintenant, considérons un chemin $(q_0, w_0) \xrightarrow{*} (q, w)$ dans $\mathcal{F}$. Clairement, toutes les émissions le long de ce chemin peuvent être remplacées par des τ-transitions, et ceci quelle que soit la sémantique opérationnelle choisie. Par conséquent, si w s'écrit $w = zx$ avec $|z| = k$, alors les occurrences de messages composant x proviennent de transitions d'émission. On en déduit que $(q_0, w_0) \xrightarrow{*} (q, w')$ pour tout $w' \,|_k\, w$. On obtient donc que $\mathrm{post}^*(q_0, w_0)$ est clos inférieurement, pour l'ordre partiel $\preceq_k$, sur l'ensemble des états de $\mathcal{F}$, défini par $(q, w) \preceq_k (q', w')$ si $q = q'$ et $w \,|_k\, w'$.

Tout langage $L \subseteq M^*$ clos supérieurement pour $|_k$ est clairement régulier. Par conséquent, $\mathrm{post}^*(q_0, w_0)$ est reconnaissable, car son complémentaire est clos supérieurement pour $\preceq_k$.

Puisque $\mathrm{post}^*(q_0, w_0)$ est clos inférieurement pour $\preceq_k$, on déduit de la proposition 2.3.8 que $\mathrm{post}^*(\mathcal{F}, (q_0, w_0)) = \mathrm{post}^*(\downarrow(\mathcal{F}, \preceq_k), (q_0, w_0))$. Ainsi, on réduit le problème de l'accessibilité pour $\mathcal{F}$ au problème de l'accessibilité pour $\downarrow(\mathcal{F}, \preceq_k)$. Or ce dernier est décidable puisque, d'après le théorème 2.3.7, on peut effectivement calculer pre^* pour $\downarrow(\mathcal{F}, \preceq_k)$, car d'une part $\preceq_k$ est un beau pré-ordre décidable sur l'ensemble des états de $\mathcal{F}$, et d'autre part $\overline{\mathcal{F}}$ admet une base supérieure post finie et récursive. ∎

Nous étudions maintenant la sous-classe $\mathfrak{R}$ des automates à file réactifs de sémantique Fiffo$[l]$, et dont le contrôle satisfait la ligne 3 du tableau 5.1. Rappelons que cette condition exprime que pour tout $q, q' \in Q$ et $m \in M$, si $q \xrightarrow{?m} q'$ alors $q \xrightarrow{\tau} q'$.

Proposition 5.3.2 *Le problème de l'accessibilité est décidable pour* $\mathfrak{R}$.

Preuve. Soit $\mathcal{F}$ un automate à file réactif de $\mathfrak{R}$. Commençons par montrer que le problème de l'accessibilité avec file finale vide est décidable. Soit q_f un état de contrôle. Considérons un chemin $(q, w) \xrightarrow{*} (q_f, \varepsilon)$ dans $\mathcal{F}$. Pour tout $w' \,|\, w$, on a $(q, w') \xrightarrow{*} (q_f, \varepsilon)$, car on peut remplacer les transitions de réception des messages manquants par des τ-transitions. On en déduit que $\mathrm{pre}^*(q_f, \varepsilon)$ est clos inférieurement pour l'ordre partiel $\preceq$ défini par $(q, w) \preceq (q', w')$ si $q = q'$ et $w \,|\, w'$. Ainsi, on réduit le problème de l'accessibilité avec file finale vide pour $\mathcal{F}$ au problème de l'accessibilité avec file finale vide pour $\uparrow(\mathcal{F}, \preceq)$. Or ce dernier est décidable puisque, d'après le théorème 2.3.7, on peut effectivement calculer post^* pour $\uparrow(\mathcal{F}, \preceq)$, car d'une part $\preceq$ est un beau pré-ordre décidable sur l'ensemble des états de $\mathcal{F}$, et d'autre part $\mathcal{F}$ admet une base supérieure post finie et récursive.

Nous montrons maintenant que le problème général de l'accessibilité est décidable. Soit (q_0, w_0) et (q_f, w_f) deux états de $\mathcal{F}$. On peut supposer sans perte de généralité que la file initiale w_0 est vide. On observe que les messages composant w_f proviennent nécessairement d'émissions, puisque la file initiale est vide. L'idée de la preuve consiste à dupliquer le contrôle de l'automate, afin de se ramener au cas de l'accessibilité avec file finale vide. Le nouvel ensemble d'états de contrôle devient $Q \times \mathrm{FG}(w_f)$, et le facteur gauche de w_f stocké dans les nouveaux états de contrôle désigne la portion de w_f qui a déjà été envoyée et qui ne sera pas consommée. On passe d'un état de contrôle (q, z) à un état de contrôle (q', zm) lors d'une émission du message m, en choisissant de façon non déterministe que cette occurrence de m envoyée ne sera pas consommée, et sera donc dans la file finale w_f. On duplique également l'alphabet des messages afin de savoir, pour chaque occurrence de message présente dans la file, depuis quel exemplaire du contrôle de l'automate initial elle a été envoyée. En effet, supposons qu'on se trouve dans un état de contrôle (q, z), et qu'un message de z peut être réceptionné en q. Clairement, les occurrences de messages présentes dans la file, et qui ont été envoyées depuis un état de contrôle (q', z) du même exemplaire, ne peuvent pas être réceptionnées, quelle que soit la sémantique opérationnelle choisie.

On observe que le nouvel automate à file réactif $\mathcal{F}'$ obtenu de cette façon appartient lui aussi à la sous-classe $\aleph$, et il a la même sémantique opérationnelle que $\mathcal{F}$. On se ramène de cette façon au cas de l'accessibilité avec file finale vide, car (q_f, w_f) est accessible dans $\mathcal{F}$ si et seulement si l'état de contrôle (q_f, w_f) de $\mathcal{F}'$ est accessible avec file finale vide. ■

La dernière sous-classe analysable est la sous-classe $\aleph'$ des automates à file réactifs de sémantique FIFFO$[l]$ avec priorité aux réceptions, et dont le contrôle satisfait la ligne 4 du tableau 5.1. Rappelons que cette condition exprime que pour tout $q, q' \in Q$ et $m \in M$, si $q \xrightarrow{?m} q'$ alors $q = q'$.

Proposition 5.3.3 *Le problème de l'accessibilité est décidable pour $\aleph'$.*

Preuve. Soit $\mathcal{F}$ un automate à file réactif de $\aleph'$. Malgré la priorité aux réceptions, pour tout chemin $(q, w) \xrightarrow{*} (q_f, \varepsilon)$ de $\mathcal{F}$, et pour tout $w' \mid w$, on a encore $(q, w') \xrightarrow{*} (q_f, \varepsilon)$, car on peut tout simplement supprimer les transitions de réception des messages manquants. La proposition se prouve donc de manière très similaire à la proposition précédente. ■

Enfin, pour tout $l \in \mathbb{N}_\omega$, notons $\aleph''(l)$ la sous-classe des automates à file réactifs de sémantique FIFFO$[l]$ avec priorité aux réceptions, et dont le contrôle satisfait la ligne 3 du tableau 5.1. Rappelons que cette condition exprime que pour tout $q, q' \in Q$ et $m \in M$, si $q \xrightarrow{?m} q'$ alors $q \xrightarrow{\tau} q'$.

Proposition 5.3.4 *Pour tout $l \in \mathbb{N}_\omega$, le problème de l'accessibilité est indécidable pour la sous-classe $\aleph''(l)$.*

Preuve. On montre que pour tout automate à une file FIFO $\mathcal{F}$, on peut construire un automate à file réactif de sémantique FIFFO$[l]$ avec priorité aux réceptions simulant $\mathcal{F}$, et dont le contrôle satisfait la ligne 3 du tableau 5.1. La construction est similaire à celle donnée dans la figure 5.2. Cependant, la contrainte structurelle impose ici des transitions $q \xrightarrow{\tau} q'$ dès que $q \xrightarrow{?m} q'$. Par conséquent, on ne peut pas être certain qu'une transition de réception soit franchie.

Pour résoudre cette difficulté, on observe que dans notre construction, les τ-transitions imposées par la condition de la ligne 3 du tableau 5.1 ne sont franchissables que si la file est vide, car on a $R_q \in \{\emptyset, M\}$ pour tout état de contrôle q. L'idée consiste donc à utiliser un marqueur de fin de file #, de telle sorte que :

- la file ne soit jamais vide, ce qui rend infranchissables les τ-transitions non voulues, et que
- ce marqueur ne gêne pas les réceptions initialement spécifiées de l'automate à une file FIFO $\mathcal{F}$, puisqu'il se trouve en fin de file.

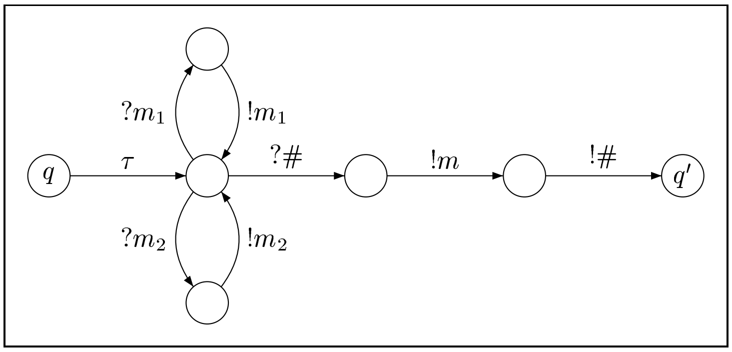


FIG. 5.4 – Simulation d'une émission $q \xrightarrow{!m} q'$

Au cours d'une émission, on fait "tourner la file" afin de remettre le marqueur en fin de file, comme indiqué sur la figure 5.4. ■

5.4 Reconnaissabilité et construction effective de post*

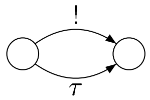
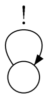
Cette section est dévolue à l'analyse exhaustive des différentes sous-classes d'automates à file réactifs, du point de vue de *l'effectivité de* post*, c'est-à-dire du point de vue :

- de la décidabilité du problème de l'accessibilité, et
- de la reconnaissabilité de post*, lorsque l'accessibilité est décidable, et
- de la construction effective de post* lorsqu'il est reconnaissable.

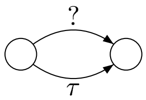
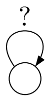
Nous présentons cette étude sous la forme de quatre tableaux. Chaque case d'un tableau indique, pour la classe considérée, si le problème de l'accessibilité est décidable, si post* est reconnaissable et, le cas échéant, si post* est constructible. Lorsqu'un résultat est enoncé pour plusieurs sémantiques opérationnelles (par exemple FIFFO$[l]$, avec $l \in \mathbb{N}_\omega$), alors il est implicitement valable pour *chaque* sémantique opérationnelle.

5.4.1 Analyse de la sémantique FIFFO

Nous étudions dans cette sous-section les différentes sous-classes d'automates à file réactifs, de sémantique opérationnelle FIFFO$[l]$.

		
$E_q, R_q \subseteq M$	Déc/Reco/¬Cons	Déc/Reco/¬Cons
$E_q \cup R_q = M$	Déc/Reco/¬Cons	Déc/Reco/¬Cons
$E_q \oplus R_q = M$	Déc/Reco/¬Cons	Déc/Reco/¬Cons
$R_q = M$	Déc/Reco/¬Cons	Déc/Reco/¬Cons
$E_q = M$	Déc/Reco/¬Cons	Déc/Reco/Cons

TAB. 5.3 – Sémantique FIFFO$[l]$, avec $l \in \mathbb{N}_\omega$

			
	$l \in \mathbb{N}$	$l = \omega$	$l \in \mathbb{N}_\omega$
$E_q, R_q \subseteq M$	Déc/Reco/Cons	Déc/¬Reco	Déc/Reco/Cons
$E_q \cup R_q = M$	Déc/Reco/Cons	Déc/¬Reco	Déc/Reco/Cons
$E_q \oplus R_q = M$	Déc/Reco/Cons	Déc/¬Reco	Déc/Reco/Cons
$R_q = M$	Déc/Reco/Cons	Déc/Reco/Cons	Déc/Reco/Cons
$E_q = M$	Déc/Reco/Cons	Déc/¬Reco	Déc/Reco/Cons

TAB. 5.4 – Sémantique FIFFO$[l]$, avec $l \in \mathbb{N}_\omega$

Théorème 5.4.1 *La situation de l'effectivité de* post*, *pour les sous-classes d'automates à file réactifs de sémantique opérationnelle* FIFFO$[l]$, *avec* $l \in \mathbb{N}_\omega$, *est décrite dans les tableaux 5.3 et 5.4.*

Preuve. Commençons par montrer les résultats du tableau 5.3. D'après la proposition 5.3.1, il reste en fait à prouver que post* n'est pas constructible pour toutes les classes sauf celle correspondant à la dernière case de la seconde colonne, pour laquelle post* est trivialement constructible. Rappelons que d'après le théorème 4.6.1 de la page 146, post* n'est pas constructible pour les automates à une seule file FIFO avec perte. En utilisant une construction similaire à celle donnée dans [CFP96], permettant de simuler un automate à files FIFO avec perte par un automate à files FIFO avec perte et duplication, on montre que post* n'est pas constructible pour la première case de la deuxième colonne. Les autres résultats de non constructibilité de post* s'obtiennent par réduction à cette case.

Montrons à présent les résultats du tableau 5.4. Il suffit en fait de prouver la première ligne du tableau (i.e. les résultats correspondant aux sous-classes où les ensembles E_q et R_q sont quelconques). La figure 5.5 décrit un automate à file réactif appartenant à la classe de la première case de la deuxième colonne, et d'ensemble d'états accessibles $\text{post}^*(q_0, \varepsilon)$ non reconnaissable, car l'ensemble des contenus de file accessibles en q_1, en partant de l'état initial (q_0, ε), est le langage non régulier $\{b^n(ab)^i c^m \ / \ i, n, m \in \mathbb{N} \text{ et } m \geq n-1\}$. Cependant, on sait d'après la proposition 5.3.2 que le problème de l'accessibilité est décidable pour cette colonne.

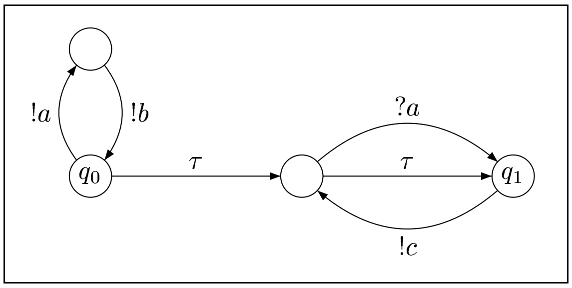


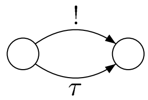
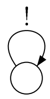
FIG. 5.5 – Automate à file réactif FIFFO[ω] d'ensemble d'états accessibles non reconnaissable

Pour construire post* dans le cas des automates à file réactifs de la première colonne, on développe un arbre symbolique d'accessibilité. Nous décrivons cet arbre symbolique dans le cas où $l = 1$, le cas où $l \in \mathbb{N}$ s'obtient en généralisant cette méthode. L'idée de l'algorithme consiste à calculer en une étape l'effet des chemins ne comportant aucune réception. Puis on franchit une transition de réception, et on recommence avec uniquement des τ et des émissions, etc. On note $L(q, q')$ le langage rationnel des mots émis le long des chemins allant de q à q' et ne comportant aucune réception. Les noeuds de l'arbre sont étiquetés par des couples (q, L) où q est un état de contrôle, et où L est un langage rationnel de la forme $((m_1 m_3 m_5 \cdots m_{2k-1})^{-1} L(q_0, q_1)) \cdot L(q_2, q_3) \cdot L(q_4, q_5) \cdots L(q_{2k}, q_{2k+1})$, avec $q_{2i+1} \xrightarrow{?m_i} q_{2i+2}$ pour tout $i \in [0 \mathinner{..} k-1]$. La contrainte structurelle sur les réceptions impose que $q_{2i+1} \xrightarrow{\tau} q_{2i+2}$ pour tout $i \in [0 \mathinner{..} k-1]$. Par conséquent, on a $L(q_2, q_3) \cdot L(q_4, q_5) \cdots L(q_{2k}, q_{2i+1}) \subseteq L(q_2, q_{2i+1})$ pour tout $i \in [0 \mathinner{..} k-1]$. Cette propriété permet d'assurer la terminaison de l'algorithme construisant l'arbre symbolique.

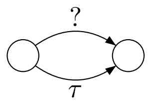
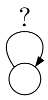
Montrons enfin comment construire post* dans le cas des automates à file réactifs de la dernière colonne, et avec $l = \omega$ (le cas $l \in \mathbb{N}$ s'obtenant avec la première colonne). Ici, les réceptions se font nécessairement en restant sur le même état de contrôle. Un chemin est donc composé d'alternances entre des portions où seules des émissions et des τ-transitions sont franchies, et des portions où on réceptionne en restant sur le même état de contrôle. On voit bien que lorsqu'un chemin retourne sur un état de contrôle q qu'il avait déjà visité, alors toutes les réceptions sur q qu'il avait faites auparavant lors de ses "séjours" précédents sur q peuvent se faire plutôt au cours de son dernier "séjour" sur q. De cette façon, on peut se contenter, pour calculer post*, de ne regarder que les chemins qui ont au plus $|Q|$ portions de réceptions, ce qui nous permet de conclure la preuve car d'une part l'effet d'une portion de réceptions sur un état de contrôle donné s'exprime par un transducteur rationnel, et d'autre part l'effet d'une portion sans réception ajoute un langage rationnel dans la file. ■

5.4.2 Analyse de la sémantique Fiffo avec priorité aux réceptions

Nous étudions dans cette sous-section les différentes sous-classes d'automates à file réactifs, de sémantique opérationnelle Fiffo$[l]$ avec priorité aux réceptions.

			
	$l \in \mathbb{N}_\omega$	$l \in \mathbb{N}$	$l = \omega$
$E_q, R_q \subseteq M$	Déc/Reco/¬Cons	Déc/Reco/¬Cons	Déc/Reco/¬Cons
$E_q \cup R_q = M$	Déc/Reco/¬Cons	Déc/Reco/¬Cons	Déc/Reco/Cons
$E_q \oplus R_q = M$	Déc/Reco/¬Cons	Déc/Reco/¬Cons	Déc/Reco/Cons
$R_q = M$	Déc/Reco/Cons	Déc/Reco/Cons	Déc/Reco/Cons
$E_q = M$	Déc/Reco/¬Cons	Déc/Reco/Cons	Déc/Reco/Cons

Tab. 5.5 – Sémantique Fiffo$[l]$ avec priorité aux réceptions, avec $l \in \mathbb{N}_\omega$

			
	$l \in \mathbb{N}_\omega$	$l \in \mathbb{N}$	$l = \omega$
$E_q, R_q \subseteq M$	Ind	Déc/**?**	Déc/Reco/Cons
$E_q \cup R_q = M$	Ind	Déc/**?**	Déc/Reco/Cons
$E_q \oplus R_q = M$	Ind	Déc/**?**	Déc/Reco/Cons
$R_q = M$	Déc/Reco/Cons	Déc/Reco/Cons	Déc/Reco/Cons
$E_q = M$	Ind	Déc/**?**	Déc/Reco/Cons

Tab. 5.6 – Sémantique Fiffo$[l]$ avec priorité aux réceptions, avec $l \in \mathbb{N}_\omega$

Théorème 5.4.2 *La situation de l'effectivité de* post*, *pour les sous-classes* *d'automates à file réactifs de sémantique opérationnelle* FIFFO$[l]$ *avec priorité aux réceptions, avec* $l \in \mathbb{N}_\omega$, *est décrite dans les tableaux 5.5 et 5.6.*

Preuve. Rappelons tout d'abord que post* est trivialement reconnaissable et constructible lorsque $R_q = M$, du fait de la priorité aux réceptions. Commençons par montrer les résultats du tableau 5.6. Les deux premières colonnes se déduisent des propositions 5.3.4 et 5.3.3 respectivement. Le calcul de post* pour la dernière colonne se fait de façon similaire au cas sans priorité aux réceptions (i.e. dernière colonne du tableau 5.4). Ici, les portions de réceptions sur un état de contrôle q enlèvent nécessairement de la file toutes les occurrences de messages appartenant à R_q. Ces portions de réceptions ont donc un effet de projection sur $M \setminus R_q$. On vérifie qu'on peut encore se contenter du dernier passage dans l'état de contrôle q pour prendre en compte l'effet des réceptions, ce qui permet de construire post*.

Prouvons maintenant les résultats du tableau 5.5. D'après la proposition 5.3.1, il reste en fait à prouver les résultats de constructibilité et de non constructibilité de post*. La première ligne s'obtient par la technique habituelle de séparation des transitions d'émission et des transitions de réception, afin de s'affranchir de la priorité aux réceptions (voir la figure 5.2). Les autres résultats de non constructibilité de post* s'obtiennent par des réductions à cette première ligne.

Montrons enfin comment construire post* dans le cas des automates à file réactifs de la dernière colonne, et avec $E_q \cup R_q = M$. On se ramène dans un premier temps au cas où la file initiale est vide, par une construction consistant à dupliquer le contrôle d'automate à file réactif en autant d'exemplaires que de sous-mots de la file initiale. Ensuite, on remarque que pour tout état de contrôle q, si $E_q \cap R_q \neq \emptyset$, alors lorsqu'on franchit depuis q une transition d'émission de $m \in E_q \cap R_q$, on doit tout de suite après réceptionner ce message, puisqu'on est resté dans le même état de contrôle q. Ainsi, quitte à rajouter des τ-transitions et à supprimer des transitions d'émission, on peut supposer que $E_q \oplus R_q = M$ pour tout état de contrôle q. Maintenant, on se rend compte facilement que pour tout état de contrôle q accessible depuis l'état de contrôle initial par des τ-transitions, on a $\{q\} \times E_q^* \subseteq \mathrm{post}^*(q_0, \varepsilon)$. On propage ensuite ces langages E_q^* le long des τ-transitions et des transitions de réception[2], pour finalement obtenir tout l'ensemble $\mathrm{post}^*(q_0, \varepsilon)$. ∎

5.5 Ce qu'il faut retenir !

Parmi les résultats exposés dans les tableaux 5.3, 5.4, 5.5 et 5.6, on remarque notamment que :

- post* est reconnaissable et constructible pour la classe des automates à file réactifs de sémantique FIFFO$[\omega]$ avec priorité aux réceptions, et satisfaisant les contraintes structurelles :
  - $E_q \cup R_q = M$ pour tout $q \in Q$, et

[2]La preuve détaillée de cette construction effective de post* se trouve dans [SFRC99].

  - pour tout $q, q' \in Q$ et $m \in M$, si $q \xrightarrow{!m} q'$ alors $q = q'$.

  Cette classe nous semble naturelle pour la spécification des systèmes réactifs avec mémorisation d'événements, et elle contient en fait la classe des RFA que nous avions étudiée dans [SFRC99].

- La contrainte structurelle, exprimant que pour tout $q, q' \in Q$ et $m \in M$, si $q \xrightarrow{?m} q'$ alors $q = q'$, permet pour de nombreuses sémantiques opérationnelles de calculer effectivement post*, car ce dernier est alors reconnaissable et constructible.

- pour la classe des automates à file réactifs de sémantique FIFFO[0] (i.e de sémantique FIFO), et satisfaisant la contrainte structurelle pour tout $q, q' \in Q$ et $m \in M$, si $q \xrightarrow{!m} q'$ alors $q \xrightarrow{\tau} q'$, post* est reconnaissable et pre* est effectivement reconnaissable (car le dual de la contrainte structurelle ci-dessus s'obtient en replaçant $q \xrightarrow{!m} q'$ par $q \xrightarrow{?m} q'$). Cette classe a donc les même propriétés de reconnaissabilité de post* et de pre* que la classe des automates à une seule FIFO avec perte.

On pourrait donc envisager de développer des stratégies pour le calcul d'une approximation supérieure de l'ensemble des états accessibles d'un automate à file réactif, en complétant par exemple sa structure de contrôle afin qu'elle satisfasse une contrainte structurelle permettant le calcul de post* (ou de pre*).

Conclusion et perspectives

Nous avons cherché, dans cette thèse, à analyser les systèmes de transitions étiquetés infinis, par des méthodes d'*abstraction* et d'*accélération*.

Nous avons commencé par montrer que le cadre général de l'interprétation abstraite n'était pas adapté à une analyse *exacte* et *accélérée* des systèmes infinis. Nous avons donc introduit un nouveau cadre d'*abstraction* et d'*accélération*, ainsi qu'un semi-algorithme générique pour le calcul de post*. Nous avons donné un algorithme calculant l'ensemble des états accessibles d'un système de transitions étiqueté plat, pourvu qu'on le munisse d'une abstraction et d'une accélération correcte et complète pour cette abstraction. Nous avons également montré que les travaux sur les *systèmes de transitions bien structurés* [Fin90, AČJ00, EN98, FS00a] peuvent s'exprimer en termes d'abstractions et d'accélérations. Nous avons décrit une méthode générale pour construire une abstraction et une "bonne" accélération dans le cadre des *automates étendus*.

Cette approche a ensuite été illustrée avec plusieurs modèles de systèmes infinis : les *systèmes d'addition de vecteurs avec états étendus*, les *automates à files* FIFO et les *automates à file réactifs*.

Nous avons montré comment les résultats sur le calcul d'un arbre de couverture d'un SAVE étendu [KM69, Fin93] peuvent de voir en termes d'abstraction et d'accélération. Nous avons défini une abstraction et une accélération basées sur les ensembles semilinéaires de vecteurs d'entiers. Nous nous sommes ensuite plus particulièrement intéressés aux SAVE étendus de dimension 2, pour lesquels nous avons obtenu une hiérarchie complète vis-à-vis de la décidabilité de l'accessibilité et presque complète vis-à-vis de l'effective semilinéarité de post* et de pre*.

Les travaux récents sur les représentations symboliques de contenus de files, à savoir les QDD [BGWW97] et les CQDD [BH99], s'expriment naturellement en termes d'abstraction et d'accélération. Nous avons développé une nouvelle représentation symbolique de contenus de files basée sur les SLRE. Nous avons ensuite comparé ces trois représentations symboliques ce qui nous a permis de voir que les SLRE semblent *suffisants* pour représenter symboliquement des contenus de files FIFO et pour itérer des séquences d'opérations, et de plus ils ont une *meilleure complexité*. En effet, l'algorithme générique de semi-calcul de post*, en partant d'un unique état initial, terminera avec les SLRE dès qu'il terminera avec les QDD (et réciproquement).

La dernière partie de notre étude portait sur l'analyse des automates à file réactifs. Nous avons analysés les différents modèles d'automates à file réactifs, du point de vue de la décidabilité de l'accessibilité, ainsi que de l'effective reconnaissabilité de post^* et de pre^*.

Problèmes ouverts

Nous nous sommes intéressés essentiellement au calcul de post^* et de pre^* pour certaines classes de systèmes infinis. De ce point de vue, notre analyse des SAVE étendus de dimension 2 est presque complète, mais nous avons laissé un problème ouvert : nous ne savons pas si post^* est *effectivement* semilinéaire pour la classe des SAVE de dimension 2 étendus par des tests-à-zéro sur le premier compteur et par des reset sur le second compteur. Nous avons cependant montré que ce problème se réduit au calcul d'une base minimale de l'ensemble de couverture.

Nous n'avons pas étudié le problème du model-checking de la logique temporelle LTL pour les SAVE étendus, et nous laissons donc ce problème ouvert. On peut néanmoins déduire immédiatement de [BM99] que le model-checking de LTL est indécidable pour les SAVE étendus par des resets.

Nous n'avons pas trouvé d'algorithme polynomial pour décider de l'inclusion entre deux SLRE.

Perspectives

Le cadre d'abstraction et d'accélération que nous avons introduit nous semble bien adapté à l'analyse des systèmes infinis, car il permet de capturer d'une part de nombreux travaux sur les systèmes de transitions bien structurés, et il permet d'autre part de présenter d'une manière uniforme les représentations symboliques et les accélérations associées.

Nous avons commencé la programmation d'un prototype d'outil mettant en oeuvre cette approche. Nous avons l'intention d'intégrer dans cet outil les représentations symboliques de contenus compteurs par ensembles semilinéaires, ainsi que les représentations symboliques de contenus de files avec les SLRE. Cependant, il reste encore des problèmes théoriques à examiner :

- l'algorithmique des représentations symboliques n'a pas été traitée dans cette thèse. Par exemple, les SLRE gagneraient sans doute (d'un point de vue pratique) à être représentés d'une façon plus compacte, en les fermant syntaxiquement par concaténation.

- la composition des abstractions reste un sujet très intéressant à étudier. On peut bien sûr choisir en première approche d'utiliser plusieurs abstractions d'un même système de manière indépendante, puis de composer leur résultats. Il serait cependant plus avantageux de pouvoir réaliser des analyses *relationnelles* en composant des abstractions non-relationnelles.

Bibliographie

[AAB99] P. Abdulla, A. Annichini, and A. Bouajjani. Symbolic verification of lossy channel systems : Application to the bounded retransmission protocol. In *Proc. 5th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99), Amsterdam, The Netherlands, March 1999*, volume 1579 of *Lecture Notes in Computer Science*, pages 208–222. Springer, 1999.

[ABIJ00] P. A. Abdulla, C. Baier, P. Iyer, and B. Jonsson. Reasoning about probabilistic lossy channel systems. In *Proc. 11th Int. Conf. Concurrency Theory (CONCUR'2000), University Park, PA, USA, Aug. 2000*, volume 1877 of *Lecture Notes in Computer Science*, pages 320–333. Springer, 2000.

[ABJ98] P. A. Abdulla, A. Bouajjani, and B. Jonsson. On-the-fly analysis of systems with unbounded, lossy FIFO channels. In *Proc. 10th Int. Conf. Computer Aided Verification (CAV'98), Vancouver, BC, Canada, June-July 1998*, volume 1427 of *Lecture Notes in Computer Science*, pages 305–318. Springer, 1998.

[AČ98] P. A. Abdulla and K. Čerāns. Simulation is decidable for one-counter nets. In *Proc. 9th Int. Conf. Concurrency Theory (CONCUR'98), Nice, France, Sep. 1998*, volume 1466 of *Lecture Notes in Computer Science*, pages 253–268. Springer, 1998.

[AČJ00] P. A. Abdulla, K. Čerāns, and B. Jonsson. Algorithmic analysis of programs with well quasi-ordered domains. *Information and Computation*, 160(1–2) :109–127, 2000.

[AD94] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2) :183–235, 1994.

[AHH96] R. Alur, T. A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 22(3) :181–201, 1996.

[AJ96a] P. A. Abdulla and B. Jonsson. Undecidable verification problems for programs with unreliable channels. *Information and Computation*, 130(1) :71–90, 1996.

[AJ96b] P. A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2) :91–101, 1996.

[Arn92] A. Arnold. *Systèmes de transitions finis et sémantique des processus communicants.* Collection Études et Recherches en Informatique. Masson, Paris, 1992.

[BBC+97] B. Barras, S. Boutin, C. Cornes, J. Courant, J. C. Filliâtre, E. Giménez, H. Herbelin, G. Huet, C. Muñoz, C. Murthy, C. Parent, C. Paulin-Mohring, A. Saïbi, and B. Werner. The Coq proof assistant reference manual – version v6.1. Technical Report 0203, INRIA, August 1997.

[BCM+92] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and J. Hwang. Symbolic model checking : 10^{20} states and beyond. *Information and Computation*, 98(2) :142–170, 1992.

[BEM97] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata : Application to model-checking. In *Proc. 8th Int. Conf. Concurrency Theory (CONCUR'97), Warsaw, Poland, Jul. 1997*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 1997.

[Ber79] J. Berstel. *Transductions and Context-Free Languages.* B. G. Teubner, Stuttgart, 1979.

[Ber80] L. Berman. The complexity of logical theories. *Theoretical Computer Science*, 11 :71–77, 1980.

[BG96] B. Boigelot and P. Godefroid. Symbolic verification of communication protocols with infinite state spaces using qdds. In *Proc. 8th Int. Conf. Computer Aided Verification (CAV'96), New Brunswick, NJ, USA, July-Aug. 1996*, volume 1102 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 1996.

[BGWW97] B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The power of QDDs. In *Proc. Static Analysis 4th Int. Symp. (SAS'97), Paris, France, Sep. 1997*, volume 1302 of *Lecture Notes in Computer Science*, pages 172–186. Springer, 1997.

[BH99] A. Bouajjani and P. Habermehl. Symbolic reachability analysis of FIFO-channel systems with nonregular sets of configurations. *Theoretical Computer Science*, 221(1–2) :211–250, 1999.

[BJNT00] A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In *Proc. 12th Int. Conf. Computer Aided Verification (CAV'2000), Chicago, IL, USA, July 2000*, volume 1855 of *Lecture Notes in Computer Science*, pages 403–418. Springer, 2000.

[BJP90] G. Berthelot, C. Johnen, and L. Petrucci. PAPETRI : environment for the analysis of Petri nets. In *Proc. 2nd Int. Workshop*

Computer-Aided Verification (CAV'90), New Brunswick, NJ, USA, June 1990, volume 531 of *Lecture Notes in Computer Science*, pages 13–22. Springer, 1990.

[BM99] A. Bouajjani and R. Mayr. Model checking lossy vector addition systems. In *Proc. 16th Ann. Symp. Theoretical Aspects of Computer Science (STACS'99), Trier, Germany, Mar. 1999*, volume 1563 of *Lecture Notes in Computer Science*, pages 323–333. Springer, 1999.

[Boc78] G. V. Bochmann. Finite state description of communication protocols. *Computer Networks (and ISDN Systems)*, 2 :361–372, 1978.

[Bér95] B. Bérard. Untiming timed languages. *Information Processing Letters*, 55(3) :129–135, 1995.

[Bry86] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8) :677–691, 1986.

[Brz64] J. A. Brzozowski. Derivatives of regular expressions. *Journal of the ACM*, 11(4) :481–494, 1964.

[BS86] M. Benois and J. Sakarovitch. On the complexity of some extended word problems defined by cancellation rules. *Information Processing Letters*, 23(6) :281–287, 1986.

[BW94] B. Boigelot and P. Wolper. Symbolic verification with periodic sets. In *Proc. 6th Int. Conf. Computer Aided Verification (CAV'94), Stanford, CA, USA, June 1994*, volume 818 of *Lecture Notes in Computer Science*, pages 55–67. Springer, 1994.

[BZ83] D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2) :323–342, 1983.

[Cau92] D. Caucal. On the regular structure of prefix rewriting. *Theoretical Computer Science*, 106(1) :61–86, 1992.

[CC77] P. Cousot and R. Cousot. Abstract interpretation : A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. 4th ACM Symp. Principles of Programming Languages, Los Angeles, CA, USA*, pages 238–252. ACM Press, 1977.

[CC79] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proc. 6th ACM Symp. Principles of Programming Languages, San Antonio, TX, USA*, pages 269–282. ACM Press, 1979.

[CC92a] P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *Journal Logic Programming*, 13(2–3) :103–179, 1992.

[CC92b] P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4) :511–547, 1992.

[Céc98] G. Cécé. Vérification, analyse et approximations symboliques des automates communicants. Thèse ENS de Cachan, January 1998.

[CC00] H. Comon and V. Cortier. Flatness is not a weakness. In *Proc. 14th Int. Workshop Computer Science Logic (CSL'2000), Fischbachau, Germany, Aug. 2000*, volume 1862 of *Lecture Notes in Computer Science*, pages 262–276. Springer, 2000.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2) :244–263, 1986.

[CF87] A. Choquet and A. Finkel. Simulation of linear fifo nets by Petri nets having a structured set of terminal markings. In *Proc. 8th European Workshop on Application and Theory of Petri Nets, Zaragoza, Spain, June 1987*, pages 95–112, 1987.

[CF97] G. Cécé and A. Finkel. Programs with quasi-stable channels are effectively recognizable. In *Proc. 9th Int. Conf. Computer Aided Verification (CAV'97), Haifa, Israel, June 1997*, volume 1254 of *Lecture Notes in Computer Science*, pages 304–315. Springer, 1997.

[CFP96] G. Cécé, A. Finkel, and S. Purushothaman Iyer. Unreliable channels are easier to verify than perfect channels. *Information and Computation*, 124(1) :20–31, 1996.

[CH78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proc. 5th ACM Symp. Principles of Programming Languages, Tucson, AZ, USA*, pages 84–96. ACM Press, 1978.

[CHP71] P. J. Courtois, F. Heymans, and D. L. Parnas. Concurrent control with 'readers' and 'writers'. *Communications of the ACM*, 14(10) :667–668, 1971.

[CJ98] H. Comon and Y. Jurski. Multiple counters automata, safety analysis and Presburger arithmetic. In *Proc. 10th Int. Conf. Computer Aided Verification (CAV'98), Vancouver, BC, Canada, June-July 1998*, volume 1427 of *Lecture Notes in Computer Science*, pages 268–279. Springer, 1998.

[CJ99] H. Comon and Y. Jurski. Timed automata and the theory of real numbers. In *Proc. 10th Int. Conf. Concurrency Theory (CONCUR'99), Eindhoven, The Netherlands, Aug. 1999*, volume 1664 of *Lecture Notes in Computer Science*, pages 242–257. Springer, 1999.

[CJK97] S. Christensen, J. B. Jørgensen, and L. M. Kristensen. Design/CPN — a computer tool for coloured Petri nets. In *Proc. 3rd Int. Workshop Tools and Algorithms for the Construction and Analysis of Systems (TACAS'97), Enschede, The Netherlands, April 1997*, volume 1217 of *Lecture Notes in Computer Science*, pages 209–223. Springer, 1997.

[Cou00] P. Cousot. Interprétation abstraite. *Technique et Science Informatiques*, 19(1–2–3) :155–164, 2000.

[CR95] F. Cassez and O. Roux. Compilation of the ELECTRE reactive language into finite transition systems. *Theoretical Computer Science*, 146(1–2) :109–143, 1995.

[Dam94] M. Dam. CTL* and ECTL* as fragments of the modal μ-calculus. *Theoretical Computer Science*, 126(1) :77–96, 1994.

[DFJS00] C. Dufourd, A. Finkel, P. Jančar, and Ph. Schnoebelen. A survey about decidability questions on reset/copy/transfer nets. manuscript note, 2000.

[DFS98] C. Dufourd, A. Finkel, and Ph. Schnoebelen. Reset nets between decidability and undecidability. In *Proc. 25th Int. Coll. Automata, Languages, and Programming (ICALP'98), Aalborg, Denmark, July 1998*, volume 1443 of *Lecture Notes in Computer Science*, pages 103–115. Springer, 1998.

[Dic13] L. E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with r distinct prime factors. *Amer. Journal Math.*, 35 :413–422, 1913.

[DJS99] C. Dufourd, P. Jančar, and Ph. Schnoebelen. Boundedness of Reset P/T nets. In *Proc. 26th Int. Coll. Automata, Languages, and Programming (ICALP'99), Prague, Czech Republic, July 1999*, volume 1644 of *Lecture Notes in Computer Science*, pages 301–310. Springer, 1999.

[Duf98] C. Dufourd. Réseaux de Petri avec reset/transfert : Décidabilité et indécidabilité. Thèse ENS de Cachan, October 1998.

[EFM99] J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *Proc. 14th IEEE Symp. Logic in Computer Science (LICS'99), Trento, Italy, July 1999*, pages 352–359, 1999.

[EHRS00] J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. In *Proc. 12th Int. Conf. Computer Aided Verification (CAV'2000), Chicago, IL, USA, July 2000*, volume 1855 of *Lecture Notes in Computer Science*, pages 232–247. Springer, 2000.

[Eme90] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 16, pages 995–1072. Elsevier Science, 1990.

[EN98] E. A. Emerson and K. S. Namjoshi. On model checking for non-deterministic infinite-state systems. In *Proc. 13th IEEE Symp. Logic in Computer Science (LICS'98), Indianapolis, IN, USA, June 1998*, pages 70–80, 1998.

[Esp97] J. Esparza. Decidability of model checking for infinite-sate concurrent systems. *Acta Informatica*, 34(2) :85–107, 1997.

[FC87] A. Finkel and A. Choquet. Fifo nets without order deadlock. *Acta Informatica*, 25(1) :15–36, 1987.

[Fin82] A. Finkel. About monogeneous fifo Petri nets. In *Proc. 3rd European Workshop Applications and Theory of Petri Nets, Varenna, Italy, Sep. 1982*, pages 175–192, 1982.

[Fin88] A. Finkel. A new class of analyzable CFSMs with unbounded FIFO channels. In *Proc. 8th IFIP WG 6.1 Int. Symp. Protocol Specification, Testing and Verification, Atlantic City, New Jersey, USA, June 1988*, pages 1–12, 1988.

[Fin90] A. Finkel. Reduction and covering of infinite reachability trees. *Information and Computation*, 89(2) :144–179, 1990.

[Fin93] A. Finkel. The minimal coverability graph algorithm. In *Advances in Petri Nets 1993*, volume 674 of *Lecture Notes in Computer Science*, pages 210–243. Springer, 1993.

[Fin94] A. Finkel. Decidability of the termination problem for completely specificied protocols. *Distributed Computing*, 7 :129–135, 1994.

[FIS00] A. Finkel, S. P. Iyer, and G. Sutre. Well-abstracted transition systems. In *Proc. 11th Int. Conf. Concurrency Theory (CONCUR'2000), University Park, PA, USA, Aug. 2000*, volume 1877 of *Lecture Notes in Computer Science*, pages 566–580. Springer, 2000.

[FM96] A. Finkel and O. Marcé. Verification of infinite regular communicating automata. Internal report 96-4, LIFAC, École Normale Supérieure de Cachan, April 1996.

[FM97] A. Finkel and P. McKenzie. Verifying identical communicating processes is undecidable. *Theoretical Computer Science*, 174(1–2) :217–230, 1997.

[FO97a] L. Fribourg and H. Olsén. A decompositional approach for computing least fixed-points of Datalog programs with Z-counters. *Constraints*, 2(3/4) :305–335, 1997.

[FO97b] L. Fribourg and H. Olsén. Proving safety properties of infinite state systems by compilation into Presburger arithmetic. In *Proc. 8th Int. Conf. Concurrency Theory (CONCUR'97), Warsaw, Poland, Jul. 1997*, volume 1243 of *Lecture Notes in Computer Science*, pages 213–227. Springer, 1997.

[FR88] A. Finkel and L. E. Rosier. A survey on the decidability questions for classes of FIFO nets. In *Advances in Petri Nets 1988*, volume 340 of *Lecture Notes in Computer Science*, pages 106–132. Springer, 1988.

[FS00a] A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere ! *Theoretical Computer Science*, 2000. To appear.

[FS00b] A. Finkel and G. Sutre. An algorithm constructing the semilinear *post** for 2-dim reset/transfer vass. In *Proc. 25th Int. Symp. Math. Found. Comp. Sci. (MFCS'2000), Bratislava, Slovakia, Aug. 2000*, volume 1893 of *Lecture Notes in Computer Science*, pages 353–362. Springer, 2000.

[FS00c] A. Finkel and G. Sutre. Decidability of reachability problems for classes of two counters automata. In *Proc. 17th Ann. Symp. Theoretical Aspects of Computer Science (STACS'2000), Lille, France, Feb. 2000*, volume 1770 of *Lecture Notes in Computer Science*, pages 346–357. Springer, 2000.

[FWW97] A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems. In *Proc. 2nd Int. Workshop on Verification of Infinite State Systems (INFINITY'97), Bologna, Italy, July 1997*, volume 9 of *Electronic Notes in Theor. Comp. Sci.*, pages 30–40. Elsevier Science, 1997.

[GGLR87] M. G. Gouda, E. M. Gurari, T. Lai, and L. E. Rosier. On deadlock detection in systems of communicating finite state machines. *Computers and Artificial Intelligence*, 6(3) :209–228, 1987.

[Gin66] S. Ginsburg. *The Mathematical Theory of Context-Free Languages.* McGraw-Hill, 1966.

[Gra97] B. Grahlmann. The reference component of PEP. In *Proc. 3rd Int. Workshop Tools and Algorithms for the Construction and Analysis of Systems (TACAS'97), Enschede, The Netherlands, April 1997*, volume 1217 of *Lecture Notes in Computer Science*, pages 65–80. Springer, 1997.

[Hal93] N. Halbwachs. Delay analysis in synchronous programs. In *Proc. 5th Int. Conf. Computer Aided Verification (CAV'93), Elounda, Greece, June 1993*, volume 697 of *Lecture Notes in Computer Science*, pages 333–346. Springer, 1993.

[HHWT97] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HYTECH : A model checker for hybrid systems. *Journal of Software Tools for Technology Transfer*, 1(1–2) :110–122, 1997.

[Hig52] G. Higman. Ordering by divisibility in abstract algebras. *Proc. London Math. Soc. (3)*, 2(7) :326–336, 1952.

[HM00a] P. Habermehl and R. Mayr. A note on SLRE, October 2000. Personal communication.

[HM00b] T. A. Henzinger and R. Majumdar. A classification of symbolic transition systems. In *Proc. 17th Ann. Symp. Theoretical Aspects of Computer Science (STACS'2000), Lille, France, Feb. 2000*, volume 1770 of *Lecture Notes in Computer Science*, pages 13–34. Springer, 2000.

[Hol91] G. J. Holzmann. *Design and Validation of Computer Protocols.* Prentice Hall Int., 1991.

[HP79] J. E. Hopcroft and J.-J. Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theoretical Computer Science*, 8(2) :135–159, 1979.

[HRHY86] R. R. Howell, L. E. Rosier, D. T. Huynh, and H-C. Yen. Some complexity bounds for problems concerning finite and 2-dimensional vector addition systems with states. *Theoretical Computer Science*, 46(3) :107–140, 1986.

[HRS76] H. B. Hunt III, D. J. Rosenkrantz, and T. G. Szymanski. On the equivalence, containment, and covering problems for the regular and context-free languages. *Journal of Computer and System Sciences*, 12(2) :222–268, 1976.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[Jan94] P. Jančar. Decidability questions for bisimilarity of Petri nets and some related problems. In *Proc. 11th Ann. Symp. Theoretical Aspects of Computer Science (STACS'94), Caen, France, Feb. 1994*, volume 775 of *Lecture Notes in Computer Science*, pages 581–592. Springer, 1994.

[Jan99] P. Jančar. A note on well quasi-orderings for powersets. *Information Processing Letters*, 72(5–6) :155–160, 1999.

[JJ93] T. Jéron and C. Jard. Testing for unboundedness of fifo channels. *Theoretical Computer Science*, 113(1) :93–117, 1993.

[JKM98] P. Jančar, A. Kučera, and R. Mayr. Deciding bisimulation-like equivalences with finite-state processes. In *Proc. 25th Int. Coll. Automata, Languages, and Programming (ICALP'98), Aalborg,* *Denmark, July 1998*, volume 1443 of *Lecture Notes in Computer Science*, pages 200–211. Springer, 1998.

[JN95] N. D. Jones and F. Nielson. Abstract interpretation : a semantics-based tool for program analysis. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 4, pages 527–636. Oxford Univ. Press, 1995.

[Jur99] Y. Jurski. Expression de la relation binaire d'accessiblité pour les automates à compteurs plats et les automates temporisés. Thèse ENS de Cachan, October 1999.

[KM69] R. M. Karp and R. E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3(2) :147–195, 1969.

[Kos82] S. R. Kosaraju. Decidability of reachability in vector addition systems. In *Proc. 14th ACM Symp. Theory of Computing (STOC'82), San Francisco, CA, May 1982*, pages 267–281, 1982.

[Koz83] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27 :333–354, 1983.

[LMR+98] P. Lacan, J. N. Monfort, L. V. Q. Ribal, A. Deutsch, and G. Gonthier. The software reliability verification process : the ARIANE 5 example. In *Proc. DAta Systems In Aerospace (DASIA'98), Athens, Greece, May 1998*, volume ESA SP-422. ESA Publications, 1998.

[LPY97] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *Journal of Software Tools for Technology Transfer*, 1(1–2) :134–152, 1997.

[LS00] D. Lugiez and Ph. Schnoebelen. The regular viewpoint on PA-processes. *Theoretical Computer Science*, 2000. To appear.

[May84] E. W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM J. Comput.*, 13(3) :441–460, 1984.

[McM93] K. L. McMillan. *Symbolic Model Checking.* Kluwer Academic, 1993.

[MF85] G. Memmi and A. Finkel. An introduction to FIFO nets–monogeneous nets : A subclass of FIFO nets. *Theoretical Computer Science*, 35(2–3) :191–214, 1985.

[ORS92] S. Owre, J. M. Rushby, and N. Shankar. PVS : A prototype verification system. In *Proc. 11th Int. Conf. Automated Deduction (CADE'92), Saratoga Springs, NY, USA, June, 1992*, volume 607 of *Lecture Notes in Computer Science*, pages 748–752. Springer, 1992.

[Pac87] J. Pachl. Protocol description and analysis based on a state transition model with channel expressions. In *Proc. 7th IFIP WG 6.1 Int. Symp. Protocol Specification, Testing and Verification*, 1987.

[Per83] D. Perrin. Words. In M. Lothaire, editor, *Combinatorics on Words*, volume 17 of *Encyclopedia of Mathematics and its Applications*, chapter 1, pages 1–17. Addison-Wesley, 1983.

[Pet62] C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Univ. Bonn, 1962. Schriften des Instituts für Instrumentelle Mathematik.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. 18th IEEE Symp. Foundations of Computer Science (FOCS'77), Providence, RI, USA, Oct.-Nov. 1977*, pages 46–57, 1977.

[PP91] W. Peng and S. Purushothaman. Data flow analysis of communicating finite state machines. *ACM Transactions on Programming Languages and Systems*, 13(3) :399–442, 1991.

[RCCE92] O. Roux, D. Creusot, F. Cassez, and J. P. Elloy. Le langage réactif asynchrone Electre. *Technique et Science Informatiques*, 11(5) :35–66, 1992.

[Rei95] K. Reinhardt. Reachability in Petri nets with inhibitor arcs. Unpublished manuscript available at `http://www-fs.informatik.uni-tuebingen.de/~reinhard`, November 1995.

[SFRC98] G. Sutre, A. Finkel, O. Roux, and F. Cassez. Effective recognizability and model checking of reactive fiffo automata. Research Report LSV-98-10, Lab. Specification and Verification, ENS de Cachan, Cachan, France, October 1998. 18 pages.

[SFRC99] G. Sutre, A. Finkel, O. Roux, and F. Cassez. Effective recognizability and model checking of reactive fiffo automata. In *Proc. 7th Int. Conf. Algebraic Methodology and Software Technology (AMAST'98), Amazonia, Brazil, Jan. 1999*, volume 1548 of *Lecture Notes in Computer Science*, pages 106–123. Springer, 1999.

[SM73] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time : Preliminary report. In *Proc. 5th ACM Symp. Theory of Computing (STOC'73), Austin, TX, USA, Apr.-May 1973*, pages 1–9, 1973.

[SS99] H. Saïdi and N. Shankar. Abstract and model check while you prove. In *Proc. 11th Int. Conf. Computer Aided Verification (CAV'99), Trento, Italy, July 1999*, volume 1633 of *Lecture Notes in Computer Science*, pages 443–454. Springer, 1999.

[Sut97] G. Sutre. Vérification de propriétés sur les automates à file réactifs produits par compilation de programmes Electre. Mémoire de DEA, Univ. Paris VII et Ecole Polytechnique, September 1997.

[VW86] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st IEEE Symp. Logic in Computer Science (LICS'86), Cambridge, MA, USA, June 1986*, pages 332–344, 1986.

[Yov97] S. Yovine. KRONOS : A verification tool for real-time systems. *Journal of Software Tools for Technology Transfer*, 1(1–2) :123–133, 1997.

[Yu97] S. Yu. Regular languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, chapter 2, pages 41–110. Springer, 1997.

Index