RECHERCHE

Automates Concurrents à Contraintes

Laurent Fribourg — Marcos Veloso Peixoto

L.I.E.N.S. - URA 1327 du CNRS
45, rue d'Ulm
75230 PARIS cedex 05

RÉSUMÉ. Nous considérons le problème de la spécification et de la vérification de systèmes concurrents manipulant un nombre non borné de données. Nous proposons de modéliser les systèmes concurrents par des automates étendus appelés "automates généralisés avec contraintes arithmétiques". Un automate généralisé est un automate dont les actions et les états sont paramétrés par des entiers relatifs. A chaque transition est associée une contrainte arithmétique qui doit être vérifiée par les paramètres d'action et d'état pour que la transition puisse se déclencher. Deux automates généralisés peuvent communiquer de façon synchrone. Chaque automate généralisé sera représenté sous la forme d'un programme logique avec contraintes arithmétiques. Nous donnerons quelques exemples d'application et comparerons notre approche avec des travaux similaires récents.

ABSTRACT. We address the problem of the specification and the verification of concurrent systems which manipulate an unbounded number of data. We propose to modelize concurrent systems by means of extended automata, called "generalized automata with arithmetical constraints". A generalized automaton is an automaton whose actions and states contain parameters that take their value in the set of integers. With each transition is associated an arithmetic constraint that must be satisfied by the the action and states parameters for enabling the transition. Two generalized automata can synchronously communicate. Each generalized automaton will be written as a logic program with arithmetic constraints. We shall give some applied examples and compare our approach with some recent similar works.

MOTS-CLÉS : systèmes concurrents, programmation logique avec contraintes, calcul de point-fixe, inclusion de langage, propriétés de sûreté.

KEY WORDS : concurrent systems, logic programming with constraints, fixpoint computation, language inclusion, safety properties.

1. Introduction

La notion d'automate à états finis est couramment utilisée pour modéliser et vérifier les systèmes concurrents. Une approche classique consiste à représenter par un automate chaque processus du système concurrent considéré, et à modéliser les interactions entre processus en synchronisant ces automates entre eux [ARN 82, ARN 93]. La preuve de propriétés du système se fait en analysant le graphe d'états de l'automate composé obtenu.

Le défaut majeur d'une telle approche est qu'elle produit un nouvel automate à chaque fois que la valeur d'un paramètre du système est changée. Or la taille du graphe des états de l'automate explose de façon combinatoire lorsque la valeur d'un paramètre augmente. Cela rend impraticable la vérification de propriétés pour les "grandes" valeurs. Il y a donc un intérêt évident à tenter de raisonner, de façon générique, sur des automates dont les états contiennent des paramètres *non instanciés* (et pouvant prendre leurs valeurs sur des domaines infinis). Pour de tels automates, il n'y a malheureusement plus de procédure de décision connue. Le problème de l'*inclusion de langage* en particulier (c'est-à-dire, savoir si, étant donné deux automates, toute suite d'actions reconnue par l'un est reconnue par l'autre, voir [EIL 74]) devient indécidable.

Nous proposons ici une approche de représentation et d'analyse des automates paramétrés, qui repose sur la Programmation Logique avec Contraintes [JAF 87]. Les paramètres des automates que nous considérons prennent leurs valeurs sur le domaine des entiers relatifs. A chaque transition est associée une contrainte d'arithmétique linéaire liant les paramètres d'action et d'états de la transition. L'automate paramétré sera représenté sous la forme d'un programme logique avec contraintes arithmétiques. L'utilisation de techniques d'évaluation ascendante pour ces programmes nous permettra de montrer que, pour une certaine sous-classe de contraintes, le problème de l'inclusion de langage est décidable.

Le plan du papier est le suivant : les parties 2 et 3 présentent des définitions de base et expliquent comment représenter des automates sous forme de programmes logiques ; dans les parties 4 et 5, nous définissons de façon formelle les automates généralisés et leur représentation sous forme de programmes logiques avec contraintes ; dans la partie 6, nous définissons les opérations d'intersection, mélange, composition parallèle et complétion pour les automates généralisés ; la partie 7 montre que tout automate généralisé est équivalent à un automate généralisé déterministe ; la partie 8 décrit une méthode de résolution du problème de l'inclusion de langage ; dans la partie 9, nous comparons notre travail avec des travaux récents similaires ; enfin, dans la partie 10, nous appliquons notre méthode sur un exemple tiré de la littérature.

2. Automates

Dans cette partie, nous rappelons certaines notions de base. Les lettres a et b (avec d'éventuels indices) représenteront des symboles d'actions et les lettres s et t (avec d'éventuels indices) représenteront des symboles d'états des automates.

Définition 1. Un *automate* est un quadruplet $\ll \Xi, s_0, \Sigma, \Delta \gg$, où :

1. Ξ est un ensemble fini d'états ;
2. $s_0 \in \Xi$ est l'état initial ;
3. Σ est un ensemble fini d'actions ;
4. Δ est un sous-ensemble de $\Xi \times \Sigma \times \Xi$.

Le fait qu'un triplet $< s, a, t >$ appartient à Δ signifie qu'une transition de l'état s à l'état t peut intervenir en exécutant l'action a. Cette transition est notée $s \xmapsto{a} t$.

L'automate n'a pas un ensemble explicite d'états finaux : tout état est considéré comme final.

Une *chaîne de transitions* d' un automate est une suite finie de transitions de la forme $s_0 \xmapsto{a_0} s_1 \xmapsto{a_1} \ \ldots \ \xmapsto{a_{n-1}} s_n$, qui part de l'état initial s_0.

Le *langage reconnu* par un automate AUT est l'ensemble $Rec(AUT)$ des suites d'actions $[a_0, a_1, \ldots, a_{n-1}]$ correspondant à une chaîne de transitions de cet automate.

L'expression $[a_0, a_1, \ldots, a_{n-1}]$, ci-dessus, dénote la liste formée des actions $a_0, a_1, \ldots, a_{n-1}$. Noter que nous ne considérons que les listes *finies* d'actions reconnues par l'automate. La liste vide, notée [ ], appartient trivialement à $Rec(AUT)$.

Exemple 2. Considérons l'automate AUT_0 représenté figure 1. Nous avons :

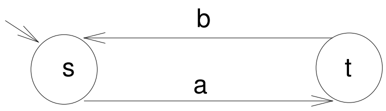

Figure 1. . *Automate* AUT_0

$$Rec(AUT_0) \ = \ \{ \ [\,] \ \} \ \cup \ \{ \ [a] \ \} \ \cup \ \{ \ [a,b] \ \} \ \cup \ \{ \ [a,b,a] \ \} \ \cup \\ \{ \ [a,b,a,b] \ \} \ \cup \ \{ \ [a,b,a,b,a] \ \} \ \cup \ \ldots \qquad \square$$

3. Programmes Logiques

Etant donné un automate, nous allons construire un programme logique [LLO 87] tel que toute suite d'actions reconnue par l'automate soit la réponse à un certain but soumis au programme, et vice-versa. Nous rappelons qu'un programme logique est composé de *clauses* de la forme $p(\overline{t}) \leftarrow p_1(\overline{t}_1), ..., p_m(\overline{t}_m)$ dans laquelle $\leftarrow$ dénote l'implication logique, $p, p_1, ..., p_m$ des prédicats, et $\overline{t}, \overline{t}_1, ..., \overline{t}_m$ des vecteurs de termes. Un *but* est une formule de la forme $\leftarrow p_1(\overline{t}_1), ..., p_m(\overline{t}_m)$. Un interprète de programmes logiques utilise des chaînes d'opérations appelées *SLD-dérivations* pour faire ses calculs. Une étape de SLD-dérivation consiste à unifier l'atome de gauche d'un but avec la tête d'une clause du programme, et à remplacer l'atome du but par le corps de la clause. Le calcul *réussit* lorsque la chaîne de la SLD-dérivation aboutit à un but vide.

La construction d'un programme logique simulant un automate à états finis donné, ne pose pas de difficultés (voir, par exemple, [STE 86]). Etant donné un automate $AUT = \ll \Xi, s_0, \Sigma, \Delta \gg$, on associe à AUT, de manière standard (std), le programme Π_{AUT}^{std} suivant :

$$aut^{\text{std}}\ ([\,], X)$$

$$\{\ aut^{\text{std}}\ ([a|L], s) \quad \leftarrow \quad aut^{\text{std}}\ (L, t)\ \}$$
$$\text{pour tout} \quad < s, a, t > \in \Delta$$

La lettre majuscule X représente une variable logique (d'état). La lettre majuscule L représente une variable logique (de liste). L'expression $[a|L]$ représente la liste ayant l'action a comme premier élément, et la liste L comme queue. Il est facile de voir qu'une étape de SLD-dérivation au niveau du programme correspond à l'exécution d'une transition au niveau de l'automate. Plus précisément : pour toute liste l d'actions données, le but $\leftarrow aut^{\text{std}}\ (l, s_0)$ réussit avec le programme Π_{AUT}^{std} ssi l est reconnue par l'automate.

D'un point de vue déclaratif, le programme dit que l'atome $aut^{\text{std}}\ (L, X)$ est vrai ssi L est une liste d'actions correspondant à une suite de transitions qui part de l'état X.

Afin d'exploiter des procédures d'évaluation ascendante, il est préférable que l'état initial s_0 apparaisse dans la clause de base et non pas dans le but, et qu'inversement, la variable d'état X apparaisse dans le but et non dans la clause de base. En utilisant des techniques comme celles dites d'"ensembles magiques" (voir [BAN 86]), on peut transformer le programme précédent en un programme Π_{AUT}^{dual} de la forme :

$$aut^{\text{dual}}\ ([\,], s_0)$$

$$\{\ aut^{\text{dual}}\ ([a|L], t) \quad \leftarrow \quad aut^{\text{dual}}\ (L, s)\ \}$$
$$\text{pour tout} \quad < s, a, t > \in \Delta$$

Le but $\leftarrow aut^{\text{dual}}\ (l, X)$ réussit dans Π_{AUT}^{dual} ssi le but $\leftarrow aut^{\text{std}}\ (l^{inv}, s_0)$ réussit dans Π_{AUT}^{std} (c'est-à-dire, ssi $l^{inv} \in Rec(AUT)$). L'expression l^{inv} ci-dessus dénote la liste inverse de l.

D'un point de vue déclaratif, le programme dit que l'autome $aut^{\text{dual}}\ (L, X)$ est vrai ssi L^{inv} est une liste d'actions correspondant à une *chaîne* (partant de

s_0) de transitions qui aboutit à l'état X.

Dans un souci de concision, on notera dorénavant le programme Π^{dual}_{AUT} par Π_{AUT} et le prédicat aut^{dual} par aut.

Exemple 3. Considérons l'automate AUT_0 de l'exemple 2. Il peut être caractérisé par le programme Π_{AUT_0} :

$$\begin{array}{lll} aut_0([\,], s) & & \\ aut_0([a|L], t) & \leftarrow & aut_0(L, s) \\ aut_0([b|L], s) & \leftarrow & aut_0(L, t) \end{array}$$

Le but $\leftarrow aut_0([b, a, b, a], X)$ réussit dans Π_{AUT_0}, ce qui signifie que $[b, a, b, a]^{inv}$ (c'est-à-dire : $[a, b, a, b]$) est une suite d'actions reconnue par AUT_0. □

4. Automates Généralisés avec Contraintes Arithmétiques

Dans cette partie, nous généralisons la notion d'automate : les actions comme les états contiendront des paramètres arithmétiques, et les transitions entre états seront déclenchées seulement si certaines contraintes arithmétiques sur ces paramètres sont satisfaites.

Dorénavant, les lettres a, b, c, d, e, f, g, h (avec d'éventuels indices et primes) représenteront les symboles d'actions et les lettres q, r, s, t, u, v, w (avec d'éventuels indices et primes) les symboles d'états des automates généralisés.

Les lettres $\overline{A}, \overline{B}, \overline{C}, \overline{D}, \overline{E}, \overline{F}, \overline{G}, \overline{H}$ représenteront des vecteurs de variables distinctes qui seront utilisées comme arguments d'actions.

Les lettres $\overline{Q}, \overline{R}, \overline{S}, \overline{T}, \overline{U}, \overline{V}, \overline{W}$ représenteront des vecteurs de variables distinctes qui seront utilisées comme arguments d'états.

On notera par $\mathcal{V}$ l'ensemble des variables utilisées comme arguments d'actions et d'états.

Les symboles $\overline{\alpha}, \overline{\sigma}, \overline{\theta}$ (avec d'éventuels indices) représenteront des vecteurs d'entiers relatifs.

Les contraintes arithmétiques que nous considérons dans ce travail sont des conjonctions et/ou disjonctions d' équations et inéquations sur le domaine des entiers relatifs. Ces contraintes appartiennent à l'arithmétique de Presburger (arithmétique sans opération de multiplication), pour laquelle on dispose de procédures de décision [PRE 29, COO 71].

Définition 4. Un *automate généralisé (avec contraintes arithmétiques)* est un quadruplet $\ll \Xi, s_0(\overline{S}_0), \Sigma, \Delta \gg$, où :

1. Ξ, l' ensemble d'états paramétrés, est un ensemble de termes de la forme $s(\overline{S})$;

2. $s_0(\overline{S}_0) \in \Xi$ est l'état initial ;
3. Σ, l'ensemble d'actions paramétrées, est un ensemble de termes de la forme $a(\overline{A})$;
4. Δ est un ensemble de transitions paramétrées, c'est à dire,un ensemble de quadruplets $< s(\overline{S}), a(\overline{A}), t(\overline{T}), \varphi >$, avec $s(\overline{S}),\ t(\overline{T}) \in \Xi,\ a(\overline{A}) \in \Sigma$. Les vecteurs de variables $\overline{S}, \overline{T}$ et $\overline{A}$ sont disjoints deux à deux, et φ est une contrainte d'arithmétique linéaire à variables dans $\overline{S} \cup \overline{T} \cup \overline{A}$. [1]

Une transition $< s(\overline{S}), a(\overline{A}), t(\overline{T}), \varphi >$ est définie à un renommage des variables de $\overline{S}, \overline{A}, \overline{T}$ près. Dans un souci de concision, on notera parfois la transition $< s(\overline{S}), a(\overline{A}), t(\overline{T}), \varphi >$ par $< s, a, t, \varphi >$. Noter que dans le cas où les symboles s et t coincident, la transition s'écrit $< s(\overline{S}), a(\overline{A}), s(\overline{T}), \varphi >$ (en abrégé : $< s, a, s, \varphi >$).

Pour définir le langage reconnu par un automate généralisé, nous introduisons d'abord les concepts de valuation, transition valide et chaîne de transitions valides.

On appelle *valuation* toute fonction $\rho : \mathcal{V} \rightarrow \mathbb{Z}$, où $\mathbb{Z}$ désigne l'ensemble des entiers relatifs. La fonction ρ peut être facilement étendue pour instancier un vecteur de variables, une action paramétrée, un état paramétré, une contrainte ou une transition. Par exemple :

$$\begin{array}{rcl} \rho(a(\overline{A})) & \stackrel{\text{def}}{=} & a(\rho(\overline{A})) \\ \rho(s(\overline{S})) & \stackrel{\text{def}}{=} & s(\rho(\overline{S})) \\ \rho(\varphi(\overline{S}, \overline{A}, \overline{T})) & \stackrel{\text{def}}{=} & \varphi(\rho(\overline{S}), \rho(\overline{A}), \rho(\overline{T})) \\ \rho(< s(\overline{S}), a(\overline{A}), t(\overline{T}), \varphi >) & \stackrel{\text{def}}{=} & < \rho(s(\overline{S})), \rho(a(\overline{A})), \rho(t(\overline{T})), \rho(\varphi) > \end{array}$$

On définit de manière naturelle (par récurrence) la valeur de vérité d'une contrainte φ par une valuation ρ. En commettant un léger abus de langage, cette valeur de vérité sera aussi notée $\rho(\varphi)$ (comme l'instance).

Etant donné un automate généralisé $\ll \Xi, s_0(\overline{S}_0), \Sigma, \Delta \gg$, une *transition valide* est une instance d'une transition paramétrée $< s(\overline{S}), a(\overline{A}), t(\overline{T}), \varphi > \in \Delta$ par une valuation ρ rendant vraie la contrainte arithmétique φ.

Si on pose $\overline{\alpha} \equiv \rho(\overline{A})$, $\overline{\sigma} \equiv \rho(\overline{S})$ et $\overline{\theta} \equiv \rho(\overline{T})$, on note une telle transition par $s(\overline{\sigma}) \xmapsto[\rho(\varphi)]{a(\overline{\alpha})} t(\overline{\theta})$.

Une *chaîne de transitions valides* pour un automate généralisé est une suite finie de transitions valides pour cet automate, de la forme :

$$s_0(\overline{\sigma}_0) \xmapsto[\rho_0(\varphi_0)]{a_0(\overline{\alpha}_0)} s_1(\overline{\sigma}_1) \xmapsto[\rho_1(\varphi_1)]{a_1(\overline{\alpha}_1)} \cdots\ s_{n-1}(\overline{\sigma}_{n-1}) \xmapsto[\rho_{n-1}(\varphi_{n-1})]{a_{n-1}(\overline{\alpha}_{n-1})} s_n(\overline{\sigma}_n)$$

1. Quand on veut expliciter les variables de cette contrainte, on l' écrira $\varphi(\overline{S}, \overline{A}, \overline{T})$.

qui part d'une instance $s_0(\overline{\sigma}_0)$ de l'état initial $s_0(\overline{S}_0)$.

Le *langage reconnu* par un automate généralisé AUT est l'ensemble $Rec(AUT)$ des suites d'actions instanciées $[a_0(\overline{\alpha}_0), a_1(\overline{\alpha}_1), \ldots, a_{n-1}(\overline{\alpha}_{n-1})]$ correspondant à une chaîne de transitions valides pour cet automate.

Un automate généralisé peut ainsi être vu comme un automate ordinaire mais infini, dont les états et les actions sont obtenus par instanciation des états et actions paramétrés, et dont les transitions sont exactement les transitions valides pour l'automate généralisé.

On représentera graphiquement de manière informelle [2], un automate par un diagramme représentant ses transitions : la transition $< s(\overline{S}), a(\overline{A}), t(\overline{T}), \varphi >$ sera représentée par le diagramme de la figure 2 et la transition $< s(\overline{S}), a(\overline{A}), s(\overline{T}), \varphi >$ sera représentée par le diagramme de la figure 3.

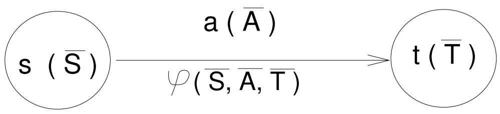

Figure 2. . *Diagramme de Transition* $< s, a, t, \varphi >$

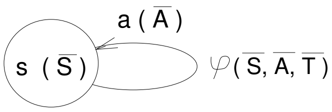

Figure 3. . *Diagramme de Transition* $< s, a, s, \varphi >$

Exemple 5. Soit AUT_1 l'automate généralisé $\ll \{s(S), t(T)\},\ s(S),\ \{a(A), b(B)\},\ \Delta \gg$, où $\Delta = \{ <\ s(S),\ a(A),\ t(T),\ A = S\ >,\ <\ t(T),\ b(B),\ s(S),\ B = S\ > \}$.

Cet automate est représenté par la figure 4.

Le langage reconnu est :

$$\begin{aligned} Rec(AUT_1) = {} & \{[\,]\} \cup \{\ [a(m)]\ /\ m \in \mathbb{Z}\} \cup \{\ [a(m), b(m')]\ /\ m, m' \in \mathbb{Z}\} \cup \\ & \{\ [a(m), b(m'), a(m')]\ /\ m, m' \in \mathbb{Z}\ \} \cup \\ & \{\ [a(m), b(m'), a(m'), b(m'')]\ /\ m, m', m'' \in \mathbb{Z}\ \} \cup \ldots \end{aligned}$$ □

2. La représentation graphique est informelle car le diagramme de transition d'un automate ne peut pas prendre en compte le renommage de variables qui devrait intervenir lorsqu'un état est atteint pour la seconde fois lors du parcours d'un chemin (cycle).

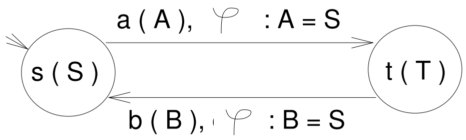

Figure 4. . *Automate* AUT_1

5. Programmes Logiques avec Contraintes

Nous allons à présent étendre les résultats de la section 3 sur la correspondance entre automates et programmes logiques, dans le cas des automates généralisés. De façon naturelle, ces derniers seront représentés par des *programmes logiques avec contraintes* [JAF 87].

En Programmation Logique avec Contraintes [JAF 87, COH 90], l'univers des termes de Herbrand de la programmation logique classique est augmenté avec des termes interprétés sur un domaine particulier. Le langage des clauses est lui-même enrichi par des "contraintes" qui portent sur les termes interprétés. La notion classique de SLD-dérivation s'étend en adjoignant au mécanisme d'unification un test de satisfiabilité pour les contraintes. Dans le cas qui nous concerne, le domaine d'interprétation est l'ensemble des entiers relatifs, et les contraintes sont des contraintes linéaires sur ce domaine. Le test de satisfiabilité pourra être réalisé par une procédure de décision de l'arithmétique linéaire quelconque (par exemple, la procédure de Presburger [PRE 29] ou celle de Cooper [COO 71]).

A l'automate généralisé AUT décrit dans la partie précédente, nous associons, de façon naturelle, le programme logique avec contraintes Π_{AUT} :

$$aut([\,], s_0(\overline{S}_0))$$
$$\{\ aut([a(\overline{A})|L], t(\overline{T})) \ \leftarrow\ \varphi(\overline{S}, \overline{A}, \overline{T}),\ aut(L, s(\overline{S}))\ \}$$
$$\text{pour tout } < s, a, t, \varphi > \in \Delta$$

L'automate de l'exemple 5 est ainsi représenté par le programme :

$$\begin{array}{lcl} aut_1([\,], s(S)) & & \\ aut_1([a(A)|L], t(T)) & \leftarrow & A = S,\ aut_1(L, s(S)) \\ aut_1([b(B)|L], s(S)) & \leftarrow & B = S,\ aut_1(L, t(T)) \end{array}$$

L'équivalence entre un automate généralisé AUT et son programme logique avec contraintes Π_{AUT} associé est formellement exprimée par la proposition suivante.

Proposition 6. *Etant donné une liste* l *d'actions instanciées avec des entiers relatifs (de la forme* $[a_1(\overline{\alpha}_1), \ldots, a_n(\overline{\alpha}_n)]$*), le but* $\leftarrow aut(l, X)$ *réussit pour*

Π_{AUT} *ssi la liste inverse* l^{inv} *(de la forme* $[a_n(\overline{\alpha}_n), \ldots, a_1(\overline{\alpha}_1)]$*) appartient à* $Rec(AUT)$*.*

En vertu de cette proposition, on pourra identifier un automate généralisé et le programme logique avec contraintes associé.

Dans la suite, on sera amené à faire un certain nombre de manipulations sur le programme logique Π_{AUT} (voir partie 8). Ainsi il nous arrivera de *tronquer l'argument de liste*. Pour alléger la notation, on réutilisera le même symbole de prédicat *aut* dans le programme tronqué Π_{AUT}^{tronc} :

$$aut(s_0(\overline{S}_0))$$

$$\{\ aut(t(\overline{T})) \ \leftarrow \ \varphi(\overline{S},\overline{A},\overline{T}),\ aut(s(\overline{S}))\ \} \quad \text{pour tout } < s,a,t,\varphi > \in \Delta$$

Dans un second temps, on pourra simplifier l'expression de ce dernier programme en *éliminant le vecteur* $\overline{A}$ *des variables d'actions*. Ces variables n'apparaissent plus en effet que dans les corps des clauses et peuvent être considérées comme étant existentiellement quantifiées. On peut toujours éliminer les variables $\overline{A}$ dans la contrainte $\exists \overline{A}\ \varphi(\overline{S},\overline{A},\overline{T})$ au moyen d'une procédure d'élimination des quantificateurs pour l'arithmétique linéaire [PRE 29, COO 71]. Avec un léger abus de notation, on notera par $\varphi(\overline{S},\overline{T})$ la contrainte simplifiée. Le programme simplifié est alors :

$$aut(s_0(\overline{S}_0))$$

$$\{\ aut(t(\overline{T})) \ \leftarrow \ \varphi(\overline{S},\overline{T}),\ aut(s(\overline{S}))\ \} \quad \text{pour tout } < s,a,t,\varphi > \in \Delta$$

Enfin, on tentera de calculer par *évaluation ascendante* le plus petit point-fixe de l'opérateur de conséquence immédiate $\mathcal{T}$ associé au programme simplifié. Dans notre cas, l'opérateur de conséquence immédiate (cf. [VAN 76, JAF 87]) est défini, pour un ensemble d'atomes clos (sans variables) $\mathcal{I}$, par :

$$\mathcal{T}(\mathcal{I}) \ = \ \bigcup_{<s,a,t,\varphi>\in\Delta} \{\ aut(t(\rho(\overline{T})))\ /\ \rho \text{ est une valuation telle que :} \\ \varphi(\rho(\overline{S}),\rho(\overline{T})) \text{ vrai et } aut(s(\rho(\overline{S}))) \ \in \ \mathcal{I}\}$$

Le plus petit point-fixe de cet opérateur est défini par $\bigcup_{j\geq 0} \mathcal{T}\!\uparrow^j$ (ce qu'on abrège en $\mathcal{T}\!\uparrow^\omega$), avec :

$$\begin{array}{lll} \mathcal{T}\!\uparrow^0 & = & \{aut(s_0(\rho(\overline{S}_0)))\ /\ \rho \text{ est une valuation }\} \\ \mathcal{T}\!\uparrow^{j+1} & = & \mathcal{T}(\mathcal{T}\!\uparrow^j), \qquad \text{pour } j \ \geq \ 0 \end{array}$$

L'intérêt de $\mathcal{T}\!\uparrow^\omega$ réside dans la propriété suivante : $aut(s(\overline{\sigma})) \in \mathcal{T}\!\uparrow^\omega$ ssi il existe, pour AUT, une chaîne de transitions valides (partant d'une instance de l'état initial) qui aboutit à l'état instancié $s(\overline{\sigma})$. $\mathcal{T}\!\uparrow^\omega$ caractérise donc l'ensemble des états instanciés *accessibles* de l'automate.

6. Opérations de Base sur les Automates

Dans cette partie, nous étendons aux automates généralisés les opérations classiques d'intersection et de mélange (voir, par exemple, [EIL 74]) et de composition parallèle [ARN 82]. Nous définissons aussi une opération de complétion d'automate.

Soit AUT et AUT' deux automates généralisés définis respectivement par $\ll \Xi, s_0(\overline{S}_0), \Sigma, \Delta \gg$ et $\ll \Xi', s'_0(\overline{S}'_0), \Sigma', \Delta' \gg$. On suppose que les ensembles d'états Ξ and Ξ' sont disjoints (pour cela, on renommera au besoin les états de l'un d'entre eux).

Les programmes Π_{AUT} et $\Pi_{AUT'}$ associés à AUT et AUT' sont :

$$aut([\,], s_0(\overline{S}_0))$$

$$\left\{ \; aut([a(\overline{A})|L], t(\overline{T})) \;\leftarrow\; \varphi(\overline{S}, \overline{A}, \overline{T}),\; aut(L, s(\overline{S})) \; \right\} \quad \text{pour tout} \quad < s, a, t, \varphi > \in \Delta$$

$$aut'([\,], s'_0(\overline{S}'_0))$$

$$\left\{ \; aut'([a'(\overline{A}')|L], t'(\overline{T}')) \;\leftarrow\; \varphi'(\overline{S}', \overline{A}', \overline{T}'),\; aut'(L, s'(\overline{S}')) \; \right\} \quad \text{pour tout} \quad < s', a', t', \varphi' > \in \Delta'$$

6.1. *Intersection*

Définition 7 . L' *automate intersection* de AUT et AUT' est l'automate généralisé AUT^{inter} caractérisé par le programme :

$$aut^{inter}([\,], s_0(\overline{S}_0), s'_0(\overline{S}'_0))$$

$$\left\{ \begin{array}{lcr} aut^{inter}([a(\overline{A})|L], t(\overline{T}), t'(\overline{T}')) & \leftarrow & \varphi(\overline{S}, \overline{A}, \overline{T}) \;\wedge\; \varphi'(\overline{S}', \overline{A}', \overline{T}') \;\wedge \\ & & A = A',\; aut^{inter}(L, s(\overline{S}), s'(\overline{S}')) \end{array} \right\}$$
$$\text{pour tout} \quad < s, a, t, \varphi > \in \Delta \quad \text{et} \quad < s', a', t', \varphi' > \in \Delta' \text{ tels que } a = a'$$

On peut facilement montrer :

Proposition 8. $Rec(AUT^{inter}) = Rec(AUT) \cap Rec(AUT')$

6.2. *Mélange*

Etant donné deux sous-ensembles A et A' de listes d'actions, le mélange $A \sqcup\!\sqcup A'$ représente l'ensemble de listes obtenu en "interfoliant" les éléments d'une liste d'actions l de A avec ceux d'une liste d'actions l' de A'.

Définition 9. L' *automate mélange* de AUT et AUT' est l'automate généralisé AUT^{mel} caractérisé par le programme :

$$aut^{mel}([\,], s_0(\overline{S}_0), s'_0(\overline{S}'_0))$$

$$\left\{ \begin{array}{lcr} aut^{mel}([a(\overline{A})|L], t(\overline{T}), s'(\overline{S}')) & \leftarrow & \varphi(\overline{S}, \overline{A}, \overline{T}), \\ & & aut^{mel}(L, s(\overline{S}), s'(\overline{S}')) \end{array} \right\}$$

pour tout $< s, a, t, \varphi > \in \Delta$ et $s' \in \Xi'$

$$\left\{ \begin{array}{lcr} aut^{mel}([a'(\overline{A}')|L], s(\overline{S}), t'(\overline{T}')) & \leftarrow & \varphi'(\overline{S}', \overline{A}', \overline{T}'), \\ & & aut^{mel}(L, s(\overline{S}), s'(\overline{S}')) \end{array} \right\}$$

pour tout $< s', a', t', \varphi' > \in \Delta'$ et $s \in \Xi$

On peut montrer facilement :

Proposition 10. *$Rec(AUT^{mel}) = Rec(AUT) \sqcup\!\sqcup\, Rec(AUT')$.*

6.3. *Composition Parallèle*

On peut, par analogie avec les automates à états finis [ARN 82], définir une opération de composition parallèle pour les automates généralisés en combinant les opérations d'intersection et de mélange. Informellement, l'opération de composition parallèle correspond à l'opération d'intersection pour les transitions que l'on désire synchroniser, et correspond à l'opération de mélange pour les autres actions. Notons en passant que, ici, contrairement à la programmation logique concurrente [SHP 89, SAR 90], la communication se fait de façon synchrone par "rendez-vous", et non pas de façon asynchrone au moyen d'opérations de mise-à-jour et de requête (voir [BOE 93]).

On suppose que, pour tout automate AUT, l'ensemble Σ contient une action spéciale τ. Cette action, appelée "action silencieuse", représente une transition spontanée entre états. Cela signifie que les clauses

$$aut([\tau|L], s(\overline{T})) \leftarrow \overline{S} = \overline{T},\ aut(L, s(\overline{S}))$$

pour tout $s \in \Xi$, appartiennent implicitement au programme associé.

Nous expliquons à présent sur un exemple caractéristique comment procéder à la composition parallèle d'automates généralisés. (Le procédé s'étend sans problème au cas général.)

Soit AUT et AUT' deux automates définis par les programmes suivants :

$$\begin{array}{lcl} aut([\,], s(S)) & & \\ aut([a(A)|L], t(T)) & \leftarrow & A = T,\ aut(L, s(S)) \\ aut([b(B)|L], s(S)) & \leftarrow & B = S,\ aut(L, t(T)) \end{array}$$

$$\begin{array}{lcl} aut'([\,], w(W)) & & \\ aut'([c(C)|L], v(V)) & \leftarrow & C = W,\ aut'(L, w(W)) \\ aut'([d(D)|L], w(W)) & \leftarrow & aut'(L, v(V)) \end{array}$$

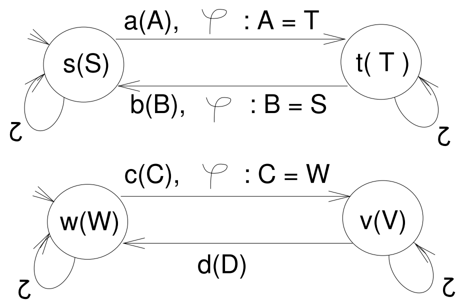

Figure 5. . *Automates AUT et AUT'*

Les automates AUT et AUT' sont représentés figure 5.

Supposons qu'on veuille *synchroniser* la transition étiquetée avec l'action $b(B)$ dans AUT avec la transition étiquetée avec l'action $c(C)$ dans AUT'. Nous caractérisons l'automate de composition parallèle AUT^{paral} par le programme:

$$\begin{array}{lcl}
aut^{paral}([\,], s(S), w(W)) & & \\
aut^{paral}([a(A)|L], t(T), w(W)) & \leftarrow & A = T,\ aut^{paral}(L, s(S), w(W)) \\
aut^{paral}([a(A)|L], t(T), v(V)) & \leftarrow & A = T,\ aut^{paral}(L, s(S), v(V)) \\
aut^{paral}([d(D)|L], s(S), w(W)) & \leftarrow & aut^{paral}(L, s(S), v(V)) \\
aut^{paral}([d(D)|L], t(T), w(W)) & \leftarrow & aut^{paral}(L, t(T), v(V)) \\
aut^{paral}([\tau|L], s(S), v(V)) & \leftarrow & B = C \wedge B = S \wedge C = W, \\
 & & \qquad aut^{paral}(L, t(T), w(W))
\end{array}$$

Les $2^{\text{ème}}$, $3^{\text{ème}}$, $4^{\text{ème}}$ et $5^{\text{ème}}$ clauses proviennent du programme associé au mélange de AUT et AUT'. La dernière clause provient du programme associé à l'intersection de AUT avec AUT', en identifiant les actions a et b; ces actions sont renommées en τ et la contrainte $B = C$ (égalisation des paramètres d'action) est ajoutée[3].

L'automate AUT^{paral} est représenté figure 6.

Un exemple de composition plus réaliste (composition de processus dans un protocole de bit alterné) est donné dans [FRI 93].

6.4. *Complétion*

Nous voulons ici construire un automate qui reconnaisse toute liste d'actions l instanciée, quelle qu'elle soit, et puisse décider si, oui ou non, cette liste l est reconnue par AUT.

3. Deux actions synchronisées doivent avoir même arité (pour permettre l'égalisation de leurs paramètres).

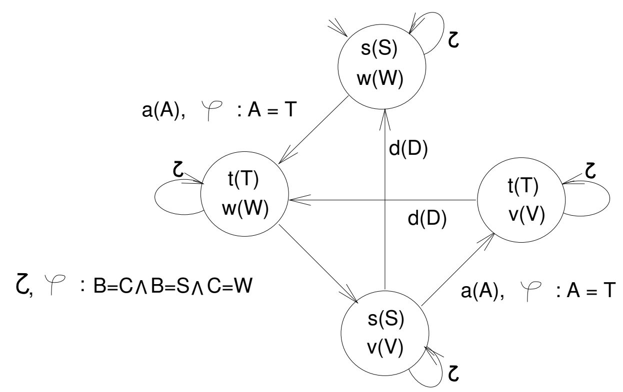

Figure 6. . *Automate* AUT^{paral}

Dans cette partie, on ne considèrera que des automates déterministes : un automate est déterministe ssi, pour toute action $a(\overline{A})$ et tout état $s(\overline{S})$, il y a au plus une transition étiquetée avec $a(\overline{A})$ et qui parte de $s(\overline{S})$. Cette hypothèse de déterminisme n'entraîne pas de perte de généralité, car tout automate est équivalent à un automate déterministe (voir partie 7).

Pour définir l'automate complété, on ajoutera un nouvel état, noté s_{erreur}, à l'ensemble des états de l'automate original. On a :

Définition 11. L'*automate complété* de AUT est l'automate généralisé AUT^{comp} caractérisé par le programme :

$$aut^{comp}([\,], s_0(\overline{S}_0))$$

$$\{\ aut^{comp}([a(\overline{A})|L], t(\overline{T}))\ \leftarrow\ \varphi(\overline{S}, \overline{A}, \overline{T}),\ aut^{comp}(L, s(\overline{S}))\ \}$$
$$\text{pour tout}\quad < s, a, t, \varphi > \in \Delta$$

$$\left\{ \begin{array}{lcl} aut^{comp}([a(\overline{A})|L], s_{erreur}) & \leftarrow & (\ \forall\ \overline{T}\ \neg\varphi(\overline{S}, \overline{A}, \overline{T})), \\ & & \quad aut^{comp}(L, s(\overline{S})) \end{array} \right\}$$
$$\text{pour tout } a \in \Sigma,\ s \in \Xi,\ \text{ et }\ \varphi\ \text{ tel que}$$
$$\exists\ t \in \Xi\ /\ \ < s, a, t, \varphi > \in \Delta\ ^4$$

$$\{\ aut^{comp}([a(\overline{A})|L], s_{erreur})\ \leftarrow\ aut^{comp}(L, s(\overline{S}))\ \}$$
$$\text{pour tout } a \in \Sigma,\ s \in \Xi\ \text{ et }\ \varphi\ \text{ tel que}$$
$$\forall\ t \in \Xi\ /\ \ < s, a, t, \varphi > \notin \Delta\ ^5$$

4. Cas où il existe une (unique) transition étiquetée avec $a(\overline{A})$ qui parte de $s(\overline{S})$.
5. Cas où il n'y a pas de transition étiquetée avec $a(\overline{A})$ qui parte de $s(\overline{S})$.

$$\{ \ aut^{comp}([a(\overline{A})|L], s_{erreur}) \leftarrow \ aut^{comp}(L, s_{erreur}) \ \}$$

pour tout $a \in \Sigma$

Noter que la contrainte ($\forall \ \overline{T} \ \neg\varphi(\overline{S}, \overline{A}, \overline{T})$) peut toujours être remplacée par une expression arithmétique équivalente sans quantificateur au moyen d'une procédure d'élimination des quantificateurs pour l'arithmétique linéaire [PRE 29, COO 71].

On peut facilement montrer :

Proposition 12. *Soit* $AUT = \ll \Xi, s_0(\overline{S}_0), \Sigma, \Delta \gg$ *un automate généralisé,* Π_{AUT} *le programme associé et* Π_{AUT}^{comp} *le programme complété de* Π_{AUT}*. Pour toute liste instanciée* l *et toute variable logique d'état* X*:*

- *Le but* $\leftarrow aut^{comp}(l, X)$ *réussit dans* Π_{AUT}^{comp} *ssi le but* $\leftarrow aut(l, X)$ *réussit dans* Π_{AUT}*.*
- *Le but* $\leftarrow aut^{comp}(l, s_{erreur})$ *réussit dans* Π_{AUT}^{comp} *ssi le but* $\leftarrow aut(l, X)$ *échoue dans* Π_{AUT}*.*

Il découle de la proposition 12 que, comme désiré, l'automate AUT^{comp} est capable de décider si, oui ou non, une liste l d'actions est reconnue par AUT.

6.5. *Un exemple d'automate complété*

Considérons l'automate AUT_2 caractérisé par le programme Π_{AUT_2}.

$$\begin{array}{lcl}
aut_2([\,], s(S)) & & \\
aut_2([a(A)|L], t(T)) & \leftarrow & A = T, \ aut_2(L, s(S)) \\
aut_2([b(B)|L], s(S)) & \leftarrow & B = S, \ aut_2(L, t(T)) \\
aut_2([c(C)|L], t(S)) & \leftarrow & C = T, \ aut_2(L, t(T))
\end{array}$$

Cet automate est représenté figure 7.

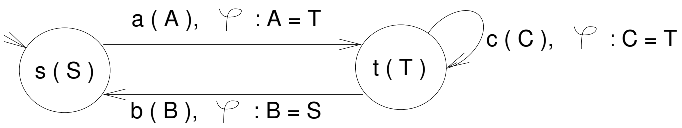

Figure 7. . *Automate* AUT_2

Le langage reconnu par AUT_2 est :

$$\begin{aligned} Rec(AUT_2) \;=\; & \{[\,]\} \;\cup\; \{\;[a(m)],\; m \in \mathbb{Z}\} \;\cup\; \{\;[a(m), c(m)],\; m \in \mathbb{Z}\} \;\cup \\ & \cup\; \{\;[a(m), b(m')],\; m, m' \in \mathbb{Z}\} \;\cup\; \{\;[a(m), c(m), b(m')], m, m' \in \mathbb{Z}\;\} \;\cup \\ & \{\;[a(m), b(m'), a(m'')],\; m, m', m'' \in \mathbb{Z}\;\;\} \;\cup\; \ldots \end{aligned}$$

L'automate AUT_2^{comp} est représenté figure 8.

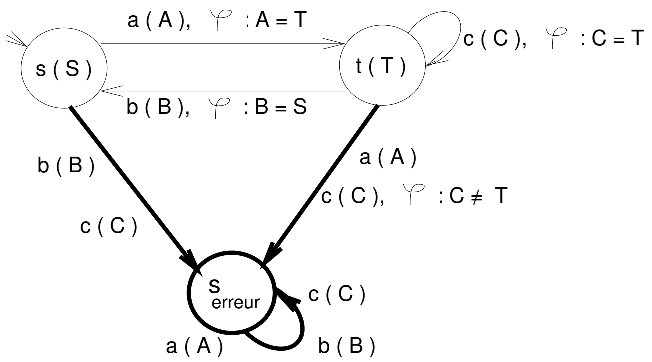

Figure 8. . *Automate aut_2^{comp}*

Le programme $\Pi_{AUT_2}^{comp}$ associé à AUT_2^{comp} est :

$$\begin{array}{lll}
aut_2^{comp}([\,], s(S)) & & \\
aut_2^{comp}([a(A)|L], t(T)) & \leftarrow & A = T,\; aut_2^{comp}(L, s(S)) \\
aut_2^{comp}([b(B)|L], s(S)) & \leftarrow & B = S,\; aut_2^{comp}(L, t(T)) \\
aut_2^{comp}([c(C)|L], t(S)) & \leftarrow & C = T,\; aut_2^{comp}(L, t(T)) \\
aut_2^{comp}([c(C)|L], s_{erreur}) & \leftarrow & aut_2^{comp}(L, s(S)) \\
aut_2^{comp}([b(B)|L], s_{erreur}) & \leftarrow & aut_2^{comp}(L, s(S)) \\
aut_2^{comp}([a(A)|L], s_{erreur}) & \leftarrow & aut_2^{comp}(L, t(T)) \\
aut_2^{comp}([c(C)|L], s_{erreur}) & \leftarrow & C \neq T,\; aut_2^{comp}(L, t(T)) \\
aut_2^{comp}([a(A)|L], s_{erreur}) & \leftarrow & (\forall T\; A \neq T),\; aut_2^{comp}(L, s(S)) \quad (*) \\
aut_2^{comp}([b(B)|L], s_{erreur}) & \leftarrow & (\forall S\; B \neq S),\; aut_2^{comp}(L, t(T)) \quad (**) \\
aut_2^{comp}([a(A)|L], s_{erreur}) & \leftarrow & aut_2^{comp}(L, s_{erreur}) \\
aut_2^{comp}([b(B)|L], s_{erreur}) & \leftarrow & aut_2^{comp}(L, s_{erreur}) \\
aut_2^{comp}([c(C)|L], s_{erreur}) & \leftarrow & aut_2^{comp}(L, s_{erreur})
\end{array}$$

Noter que les contraintes $\forall T\; A \neq T$ et $\forall S\; B \neq S$ dans les clauses $(*)$ et $(**)$ peuvent être remplacées par "faux". Ces deux clauses peuvent donc être éliminées.

Le but $\leftarrow aut_2^{comp}([b(7), c(5), a(5)], X)$ réussit dans $\Pi_{AUT_2}^{comp}$ car le but $\leftarrow aut_2([b(7), c(5), a(5)], X)$ réussit dans Π_{AUT_2}.

Le but $\leftarrow aut_2^{comp}([b(7), c(5), a(3)], s_{erreur})$ réussit dans $\Pi_{AUT_2}^{comp}$ car le but

$\leftarrow aut_2([b(7), c(5), a(3)], X)$ échoue dans Π_{AUT_2}.

7. Automates Déterministes

Etant donné un automate généralisé non-déterministe AUT, nous montrons maintenant comment construire un automate déterministe AUT^{det} tel que AUT et AUT^{det} reconnaissent les mêmes listes d'actions. Rappelons qu'un automate est déterministe ssi, pour toute action a et tout état s, il y a au plus une transition étiquetée avec a qui parte de s.

Sans perte de généralité, on supposera que, quels que soient les états $s(\overline{S})$ et $t(\overline{T})$ et l'action $a(\overline{A})$, il existe au plus une transition dans Δ de $s(\overline{S})$ vers $t(\overline{T})$, étiquetée par $a(\overline{A})$ [6].

Soit $\Xi(u,b)$ l'ensemble $\{\ t\ \in \Xi /\ \exists\varphi\ \ < u,b,t,\varphi > \ \in\ \ \Delta\ \}$. Intuitivement, l'ensemble $\Xi(u,b)$ contient tous les états t qui sont atteints depuis l'état u en exécutant l'action b. Pour un automate déterministe, l'ensemble $\Xi(u,b)$ contient au plus un élément, quels que soient $u \in \Xi$ et $b \in \Sigma$.

La construction de l'automate déterministe se fait par transformations successives, chaque transformation consistant à fusionner tous les états t (s'ils sont plusieurs) de $\Xi(u,b)$ en un seul nouvel état.

Plus précisément, chaque étape de transformation consiste à :

1. Sélectionner un état $u(\overline{U})$ et une action $b(\overline{B})$ tels que $\Xi(u,b)$ contienne plus de 1 élément (disons, n éléments).
2. Renommer (en les indexant) les états de $\Xi(u,b)$.
   ($\Xi(u,b)$ se réécrit $\{t_1(\overline{T}_1), t_2(\overline{T}_2), \ldots, t_n(\overline{T}_n)\}$).
3. Transformer l'automate courant AUT en AUT', défini par :

$$aut'([\,], s_0(S_0))$$

$$aut'([b(B)|L], q(I,\overline{Q})) \ \leftarrow \ \bigvee_{<u,b,t_j,\varphi>\in\Delta} (\varphi(\overline{U},\overline{B},\overline{Q}) \wedge I = j),\ aut'(L, u(\overline{U}))$$

$$\{\ aut'([a(A)|L], t(\overline{T}))\ \ \leftarrow\ \ \varphi(\overline{S},\overline{A},\overline{T}),\ aut'(L, s(S))\ \}$$
$$\text{pour tout } s,t \in \Xi \text{ et } a\ \in \Sigma \text{ tels que } < s,a,t,\varphi > \in \Delta,\ s \notin \Xi(u,b)\ \text{ et }\ t \notin \Xi(u,b)$$

$$\{\ aut'([a(A)|L], t(\overline{T}))\ \ \leftarrow\ \ \varphi(\overline{Q},\overline{A},\overline{T})\ \wedge\ I = i,\ \ aut'(L,\ q(I,\overline{Q})))\ \}$$
$$\text{pour tout } t, t_i \in \Xi \text{ et } a \in \Sigma \text{ tels que } < t_i,a,t,\varphi > \in \Delta,\ t_i \in \Xi(u,b)\ \text{ et }\ t \notin \Xi(u,b)$$

6. S'il en existe deux, par exemple, $< s,a,t,\varphi_1 >$ et $< s,a,t,\varphi_2 >$, il suffit de remplacer ces deux transitions par la transition $< s,a,t,\varphi_1 \vee \varphi_2 >$.

$$\left\{ \; aut'([a(A)|L], q(I,\overline{Q})) \quad \leftarrow \quad \varphi(\overline{S},\overline{A},\overline{Q}) \wedge \; I=j, \quad aut'(L, s(\overline{S})) \; \right\}$$

pour tout $s, t_j \in \Xi$ et $a \in \Sigma$ tels que $< s, a, t_j, \varphi > \in \Delta$, $s \notin \Xi(u,b)$ et $t \in \Xi(u,b)$

$$\left\{ \begin{array}{lll} aut'([a(A)|L], q(J,\overline{R})) & \leftarrow & \varphi(\overline{Q},\overline{A},\overline{R}) \; \wedge \; I = i \wedge \; J = j, \\ & & \qquad aut'(L, q(I,\overline{Q})) \end{array} \right\}$$

pour tout $a \in \Sigma$ et $t_i, t_j \in \Xi$ tels que $< t_i, a, t_j, \varphi > \in \Delta$, $t_i \in \Xi(u,b)$ et $t_j \in \Xi(u,b)$

où q représente un nouveau symbole d'état, I et J sont des variables représentant des entiers et $\overline{Q}$ et $\overline{R}$ sont des vecteurs de variables utilisées comme arguments d'états. Noter que tous les états de $\Xi(u,b)$ ont été fusionnés en q.

Puisque chaque étape fait décroître le nombre d'états (de $n-1$), et puisque AUT a un nombre fini d'états, la procédure de transformation termine nécessairement et engendre un automate déterministe AUT^{det}.

On montre facilement d'autre part, que chaque étape préserve le langage reconnu, et que, par conséquent :

Proposition 13. $Rec(AUT) = Rec(AUT^{det})$

Exemple 14. Considérons l'automate AUT_3 caractérisé par :

$$\begin{array}{lll} aut_3([\,], s(S)) & & \\ aut_3([a(A)|L], t(T)) & \leftarrow & A = T, \quad aut_3(L, s(S)) \\ aut_3([a(A)|L], u(U)) & \leftarrow & A = S, \quad aut_3(L, s(S)) \\ aut_3([c(C)|L], u(U)) & \leftarrow & C = T, \quad aut_3(L, t(T)) \\ aut_3([b(B)|L], v(V)) & \leftarrow & T = V, \quad aut_3(L, t(T)) \\ aut_3([b(B)|L], w(W)) & \leftarrow & B = W, \quad aut_3(L, u(U)) \\ aut_3([a(A)|L], v(V)) & \leftarrow & aut_3(L, w(W)) \end{array}$$

Cet automate est représenté figure 9.

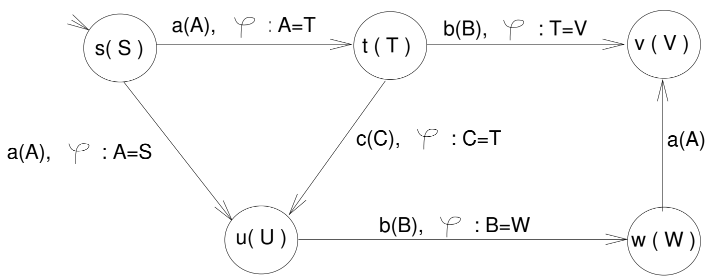

Figure 9. . *Automate AUT_3*

Dans la première étape de transformation, on fusionne les états de $\Xi(s,a)$.

On a :

$\Xi(s,a) = \{t(T), u(U)\}$

On renomme alors ces deux états en $\{t_1(T_1), t_2(T_2)\}$. Par fusion en un nouvel état q, on obtient ensuite l'automate AUT_3' (voir figure 10) :

$$
\begin{array}{lll}
aut'_3([\,], s(S)) & & \\
aut'_3([a(A)|L], q(I,Q)) & \leftarrow & (A = Q \wedge I = 1) \vee (A = S \wedge I = 2), \\
 & & \qquad aut'_3(L, s(S)) \\
aut'_3([c(C)|L], q(J,R)) & \leftarrow & C = Q \wedge I = 1 \wedge J = 2,\ aut'_3(L, q(I,Q)) \\
aut'_3([b(B)|L], v(V)) & \leftarrow & Q = V \wedge I = 1,\ \ aut'_3(L, q(I,Q)) \\
aut'_3([b(B)|L], w(W)) & \leftarrow & B = W \wedge I = 2,\ \ aut'_3(L, q(I,Q)) \\
aut'_3([a(A)|L], v(V)) & \leftarrow & aut'_3(L, w(W))
\end{array}
$$

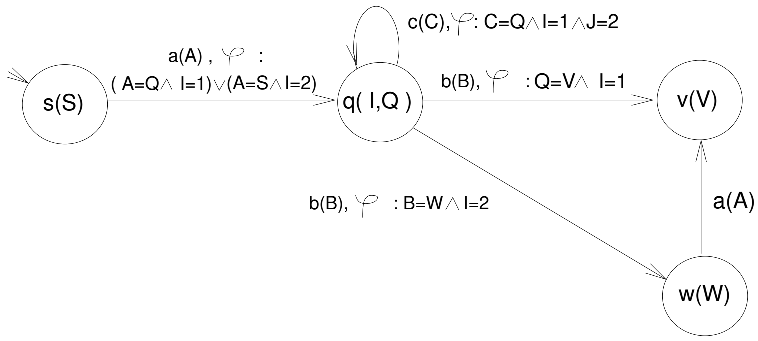

Figure 10. . *Automate* AUT_3'

A l'étape suivante, on fusionne les états de $\Xi(q,b)$. On a :

$\Xi(q,b) = \{v(V), w(W)\}$

On renomme alors ces deux états en $\{v_1(V_1), v_2(V_2)\}$. Par fusion en un nouvel état r, on obtient ensuite l'automate AUT_3'' (voir figure 11) :

$$
\begin{array}{lll}
aut''_3([\,], s(S)) & & \\
aut''_3([a(A)|L], q(I,Q)) & \leftarrow & (A = Q \wedge I = 1) \vee (A = S \wedge J = 2), \\
 & & aut''_3(L, s(S)) \\
aut''_3([c(C)|L], q(J,R)) & \leftarrow & C = Q \wedge I = 1 \wedge J = 2,\ aut''_3(L, q(I,Q)) \\
aut''_3([b(B)|L], r(J,R)) & \leftarrow & (Q = R \wedge I = 1 \wedge J = 1) \vee \\
 & & (B = R \wedge I = 2 \wedge J = 2),\ \ aut''_3(L, q(I,Q)) \\
aut''_3([a(A)|L], r(I,Q)) & \leftarrow & J = 2 \wedge I = 1,\ \ aut''_3(L, r(J,R))
\end{array}
$$

L'automate AUT_3'' est déterministe, et notre procédure termine ainsi en deux coups ($AUT_3^{det} = AUT_3''$). □

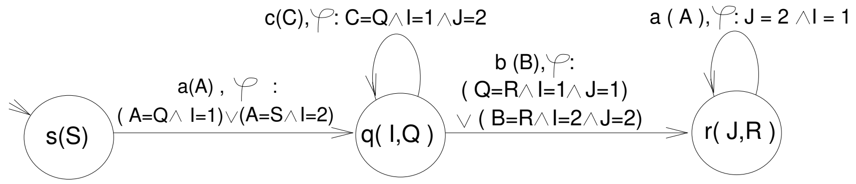

Figure 11. . *Automate* AUT_3''

8. Problème de l'Inclusion de Langage

Dans cette partie, nous étudions le problème de l'inclusion de langage pour les automates généralisés avec contraintes arithmétiques : étant donnés deux automates généralisés AUT_1 et AUT_2, décider si, oui ou non, $Rec(AUT_1) \subseteq Rec(AUT_2)$.

Le problème de l'inclusion de langage est décidable pour les automates à états finis [EIL 74]. Nous décrivons ici une méthode pour traiter ce problème au niveau des automates généralisés avec contraintes arithmétiques. De la proposition 12 et de la définition des prédicats aut_1 et aut_2, il découle :

Proposition 15. *Etant donnés deux automates généralisés* AUT_1 *et* AUT_2, $Rec(AUT_1) \subseteq Rec(AUT_2)$ *ssi* $\forall L\ \forall X_1\ \forall X_2\ (\ aut_1(L, X_1) \ \wedge\ aut_2^{comp}(L, X_2)\ \Rightarrow X_2 \neq s_{erreur})$.

Le problème de l'inclusion de langage se ramène donc à décider de la validité de la formule :

$$aut_1(L, X_1)\ \wedge\ aut_2^{comp}(L, X_2) \Rightarrow\ X_2 \neq s_{erreur} \qquad (\star)$$

Ceci se fait suivant les étapes ci-dessous (cf. [FRI 92a]) :

1. Construire le programme Π (définissant l'atome $aut^{inter}(L, X_1, X_2)$) associé à l'automate intersection de AUT_1 et AUT_2^{comp}.
2. Construire le programme Π^{tronc} obtenu en tronquant l' argument de liste L et en éliminant les paramètres d'action $\overline{A}$ des clauses de Π.
3. Calculer le résultat $\mathcal{T}\!\uparrow^{\omega}$ de la procédure d'évaluation ascendante pour le programme Π^{tronc}.

En général, la procédure d'évaluation ascendante de l'étape 3 ne termine pas. Lorqu'elle termine, le résultat $\mathcal{T}\!\uparrow^{\omega}$ est de la forme $\mathcal{T}\!\uparrow^{\omega}{}_1 \;\cup\; \mathcal{T}\!\uparrow^{\omega}{}_2$, où :

$$\mathcal{T}\!\uparrow^{\omega}{}_1 \quad = \quad \bigcup_{\substack{s_1 \in \Xi_1 \\ s_2 \in \Xi_2}} \{\ aut^{inter}(s_1(\overline{S_1}), s_2(\overline{S}_2)) \ /\ \eta_{s_1,s_2}(\overline{S}_1, \overline{S}_2) \ \text{ vrai } \}$$

$$\mathcal{T}\!\uparrow^{\omega}{}_2 \quad = \quad \bigcup_{s_1 \in \Xi_1} \{\ aut^{inter}(s_1(\overline{S}_1), s_{erreur}) \ /\ \zeta_{s_1}(\overline{S}_1) \ \text{ vrai } \}$$

et η_{s_1,s_2} et ζ_{s_1} sont des formules arithmétiques.

D'après [FRI 92a], l'hypothèse $\exists L\ aut_1(L, X_1) \ \wedge aut_2^{comp}(L, X_2)$ de la formule $(\star)$ est vraie ssi $aut^{inter}(X_1, X_2) \in \mathcal{T}\!\uparrow^{\omega}$. Il s'ensuit que la formule $(\star)$ est vraie ssi $\mathcal{T}\!\uparrow^{\omega}{}_2$ est vide.

Le problème d'inclusion de langage est donc décidable quand la procédure d'évaluation ascendante de l'étape 3 termine et la validité des formules engendrées $\zeta_s(\overline{S})$ est décidable.

Cela se produit pour certaines classes de programmes, comme par exemple lorsque les contraintes arithmétiques de AUT_1 et AUT_2 ne contiennent que les symboles $=$, $\neq$, $<$ (mais aucun $+$) : les contraintes du programme Π^{tronc} ne contiennent alors elles-mêmes aucun $+$, et le plus petit point-fixe du programme peut être calculé en temps fini selon la procédure de Revesz [REV 93] (qui engendre une formule d'arithmétique linéaire décidable). En conséquence, on a :

Proposition 16. *Le problème de l' inclusion de langage pour deux automates généralisés est décidable lorsque les contraintes arithmétiques des automates ne contiennent que les symboles* $=$, $\neq$, $<$.

L'automate modélisant le protocole du bit alterné, dans lequel les messages transmis sont représentés par des paramètres entiers, est un exemple d'automate généralisé dont les contraintes arithmétiques ne contiennent que les symboles $=, \neq, <$ (voir [FRI 93]). On peut donc utiliser la procédure de Revesz pour décider si le langage reconnu par un tel automate est inclus dans ceux reconnus par des automates de même nature. Cela peut servir, par exemple, à montrer que le message transmis par le processus émetteur du protocole coincide avec celui reçu par le processus récepteur (intégrité des données).

Noter que, même dans le cas où certaines contraintes arithmétiques dans AUT_1 (et AUT_2) contiennent le symbole $+$, on peut parfois résoudre le problème d'inclusion en utilisant d'autres procédures d'évaluation ascendante que celle de Revesz (voir, par exemple, [BAU 91, CHO 88, FRI 92b, KAN 90]), mais la terminaison n'est pas alors garantie dans le cas général.

Exemple 17. Considérons l'automate AUT_4 (adapté de [HIG 94]) représenté figure 12.

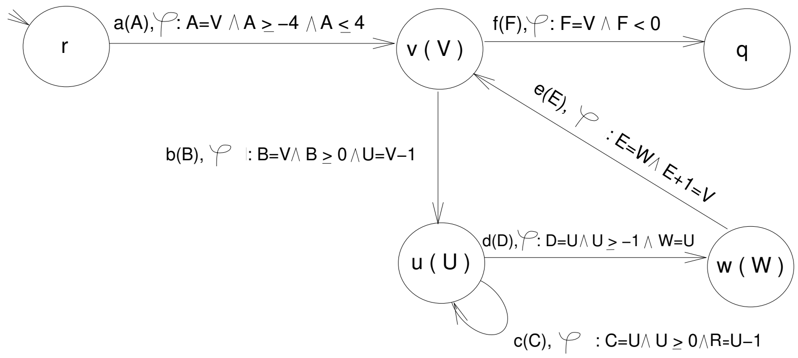

Figure 12. . *Automate AUT_4*

$$
\begin{array}{lll}
aut_4([\,], r) & & \\
aut_4([a(A)|L], v(V)) & \leftarrow & A = V \wedge A \geq -4 \ \wedge A \leq 4, \quad aut_4(L, r) \\
aut_4([b(B)|L], u(U)) & \leftarrow & B = V \wedge B \geq 0 \wedge U = V - 1, \quad aut_4(L, v(V)) \\
aut_4([c(C)|L], u(R)) & \leftarrow & C = U \wedge U \geq 0 \wedge R = U - 1, \quad aut_4(L, u(U)) \\
aut_4([d(D)|L], w(W)) & \leftarrow & D = U \wedge U \geq -1 \wedge W = U, \quad aut_4(L, u(U)) \\
aut_4([e(E)|L], v(V)) & \leftarrow & E = W \wedge E + 1 = V, \quad aut_4(L, w(W)) \\
aut_4([f(F)|L], q) & \leftarrow & F = V \wedge F < 0, \quad aut_4(L, v(V))
\end{array}
$$

Considérons à présent l'automate AUT_5 représenté figure 13. Cet automate reconnaît une liste d'action ssi elle ne contient pas plus de 4 actions c consécutives.

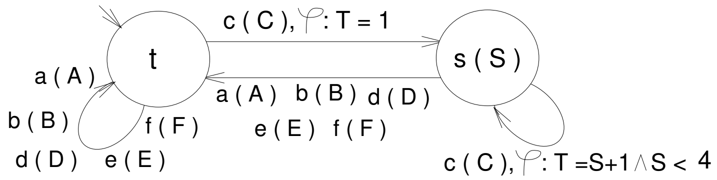

Figure 13. . *Automate AUT_5*

$$
\begin{array}{lll}
aut_5([\,], t) & & \\
aut_5([a(A)|L], t) & \leftarrow & aut_5(L, t) \\
aut_5([b(B)|L], t) & \leftarrow & aut_5(L, t) \\
aut_5([d(D)|L], t) & \leftarrow & aut_5(L, t) \\
aut_5([e(E)|L], t) & \leftarrow & aut_5(L, t) \\
aut_5([f(F)|L], t) & \leftarrow & aut_5(L, t) \\
aut_5([c(C)|L], s(S)) & \leftarrow & S = 1, \quad aut_5(L, t) \\
aut_5([c(C)|L], s(T)) & \leftarrow & T = S + 1 \wedge S < 4, \quad aut_5(L, s(S))
\end{array}
$$

$$\begin{array}{lcl}
aut_5([a(A)|L],t) & \leftarrow & aut_5(L,s(S)) \\
aut_5([b(B)|L],t) & \leftarrow & aut_5(L,s(S)) \\
aut_5([d(D)|L],t) & \leftarrow & aut_5(L,s(S)) \\
aut_5([e(E)|L],t) & \leftarrow & aut_5(L,s(S)) \\
aut_5([f(F)|L],t) & \leftarrow & aut_5(L,s(S))
\end{array}$$

Notre objectif est de démontrer que $Rec(AUT_4) \subseteq Rec(AUT_5)$. Ceci revient à montrer que l'automate AUT_4 n'accepte jamais plus de 4 actions c consécutives.

Le programme $\Pi^{comp}_{AUT_5}$ associé avec AUT_5^{comp} est (voir figure 14) :

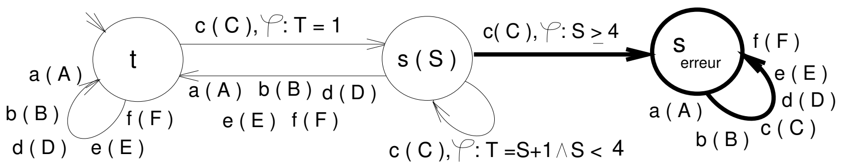

Figure 14. . *Automate* AUT_5^{comp}

$$\begin{array}{lcl}
aut_5^{comp}([\,],t) & & \\
aut_5^{comp}([a(A)|L],t) & \leftarrow & aut_5^{comp}(L,t) \\
aut_5^{comp}([b(B)|L],t) & \leftarrow & aut_5^{comp}(L,t) \\
aut_5^{comp}([d(D)|L],t) & \leftarrow & aut_5^{comp}(L,t) \\
aut_5^{comp}([e(E)|L],t) & \leftarrow & aut_5^{comp}(L,t) \\
aut_5^{comp}([f(F)|L],t) & \leftarrow & aut_5^{comp}(L,t) \\
aut_5^{comp}([c(C)|L],s(S)) & \leftarrow & S=1,\ aut_5^{comp}(L,t) \\
aut_5^{comp}([c(C)|L],s(T)) & \leftarrow & T=S+1 \wedge S<4,\ aut_5^{comp}(L,s(S)) \\
aut_5^{comp}([a(A)|L],t) & \leftarrow & aut_5^{comp}(L,s(S)) \\
aut_5^{comp}([b(B)|L],t) & \leftarrow & aut_5^{comp}(L,s(S)) \\
aut_5^{comp}([d(D)|L],t) & \leftarrow & aut_5^{comp}(L,s(S)) \\
aut_5^{comp}([e(E)|L],t) & \leftarrow & aut_5^{comp}(L,s(S)) \\
aut_5^{comp}([f(F)|L],t) & \leftarrow & aut_5^{comp}(L,s(S)) \\
aut_5^{comp}([c(C)|L],s_{erreur}) & \leftarrow & S \geq 4,\ aut_5^{comp}(L,s(S)) \\
aut_5^{comp}([\varepsilon(\mathcal{E}\ )|L],s_{erreur}) & \leftarrow & aut_5^{comp}(L,s_{erreur}) \quad (\varepsilon \in \{a,b,c,d,e,f\})
\end{array}$$

Notre problème se ramène à vérifier la formule :

$$aut_4(L,X_3)\ \wedge aut_5^{comp}(L,X_4) \Rightarrow\ X_4 \neq s_{erreur}$$

Ceci se fait en suivant les étapes 1, 2 et 3 décrites précédemment.

1. Construction du programme Π, résultant de l'intersection de AUT_4 et AUT_5^{comp}. (Noter que les états paramètres des actions dans AUT_5^{comp} sont renommés avec des primes pour les distinguer des paramètres de même nom dans AUT_4)

$$
\begin{array}{lcl}
aut^{inter}([\,], r, t) & & \\
aut^{inter}([a(A)|L], v(V), t) & \leftarrow & A = A' \wedge A = V \wedge A \geq -4 \ \wedge A \leq 4, \\
 & & \qquad aut^{inter}(L, r, t) \\
aut^{inter}([b(B)|L], u(U), t) & \leftarrow & B = B' \wedge B = V \wedge B \geq 0 \wedge U = V - 1, \\
 & & \qquad aut^{inter}(L, v(V), t) \\
aut^{inter}([d(D)|L], w(W), t) & \leftarrow & D = D' \wedge D = U \wedge U \geq -1 \wedge W = U, \\
 & & \qquad aut^{inter}(L, u(U), t) \\
aut^{inter}([e(E)|L], v(V), t) & \leftarrow & E = E' \wedge E = W \wedge E + 1 = V, \\
 & & \qquad aut^{inter}(L, w(W), t) \\
aut^{inter}([f(F)|L], q, t) & \leftarrow & F = F' \wedge F = V \wedge F < 0, \\
 & & \qquad aut^{inter}(L, v(V), t) \\
aut^{inter}([c(C)|L], u(R), s(S)) & \leftarrow & C = C' \wedge C = U \wedge U \geq 0 \wedge R = U - 1 \wedge \\
 & & \qquad S = 1, \ aut^{inter}(L, u(U), t) \\
aut^{inter}([c(C)|L], u(R), s(T)) & \leftarrow & C = C' \wedge C = U \wedge U \geq 0 \wedge R = U - 1 \wedge \\
 & & \quad T = S + 1 \wedge S < 4, \ aut^{inter}(L, u(U), s(S)) \\
aut^{inter}([a(A)|L], v(V), t) & \leftarrow & A = A' \wedge A = V \wedge A \geq -4 \ \wedge A \leq 4, \\
 & & \qquad aut^{inter}(L, r, s(S)) \\
aut^{inter}([b(B)|L], u(U), t) & \leftarrow & B = B' \wedge B = V \wedge B \geq 0 \wedge U = V - 1, \\
 & & \qquad aut^{inter}(L, v(V), s(S)) \\
aut^{inter}([d(D)|L], w(W), t) & \leftarrow & D = D' \wedge D = U \wedge U \geq -1 \wedge W = U, \\
 & & \qquad aut^{inter}(L, u(U), s(S)) \\
aut^{inter}([e(E)|L], v(V), t) & \leftarrow & E = E' \wedge E = W \wedge E + 1 = V, \\
 & & \qquad aut^{inter}(L, w(W), s(S)) \\
aut^{inter}([f(F)|L], q, t) & \leftarrow & F = F' \wedge F = V \wedge F < 0, \\
 & & \qquad aut^{inter}(L, v(V), s(S)) \\
aut^{inter}([c(C)|L], u(R), s_{erreur}) & \leftarrow & C = C' \wedge C = U \wedge U \geq 0 \wedge R = U - 1 \wedge \\
 & & \qquad S \geq 4, \ \ aut^{inter}(L, u(U), s(S)) \\
aut^{inter}([a(A)|L], v(V), s_{erreur}) & \leftarrow & A = V \wedge A \geq -4 \ \wedge A \leq 4, \\
 & & \qquad aut^{inter}(L, r, s_{erreur}) \\
aut^{inter}([b(B)|L], u(U), s_{erreur}) & \leftarrow & B = V \wedge B \geq 0 \wedge U = V - 1, \\
 & & \qquad aut^{inter}(L, v(V), s_{erreur}) \\
aut^{inter}([c(C)|L], u(R), s_{erreur}) & \leftarrow & C = U \wedge U \geq 0 \wedge R = U - 1, \\
 & & \qquad aut^{inter}(L, u(U), s_{erreur}) \\
aut^{inter}([d(D)|L], w(W), s_{erreur}) & \leftarrow & D = U \wedge U \geq -1 \wedge W = U, \\
 & & \qquad aut^{inter}(L, u(U), s_{erreur}) \\
aut^{inter}([e(E)|L], v(V), s_{erreur}) & \leftarrow & E = W \wedge E + 1 = V, \\
 & & \qquad aut^{inter}(L, w(W), s_{erreur}) \\
aut^{inter}([f(F)|L], q, s_{erreur}) & \leftarrow & F = V \wedge F < 0, \ aut^{inter}(L, v(V), s_{erreur})
\end{array}
$$

2. Troncation de l'argument de liste L et élimination des paramètres d'actions du programme Π, ce qui engendre le programme Π^{tronc} :

$$
\begin{array}{lll}
aut^{inter}(r,t) & & \\
aut^{inter}(v(V),t) & \leftarrow & V \geq -4 \ \wedge V \leq 4,\ aut^{inter}(r,t) \\
aut^{inter}(u(U),t) & \leftarrow & V \geq 0 \wedge U = V-1,\ \ aut^{inter}(v(V),t) \\
aut^{inter}(w(W),t) & \leftarrow & U \geq -1 \wedge W = U,\ \ aut^{inter}(u(U),t) \\
aut^{inter}(v(V),t) & \leftarrow & W+1 = V,\ \ aut^{inter}(w(W),t) \\
aut^{inter}(q,t) & \leftarrow & V < 0,\ aut^{inter}(v(V),t) \\
aut^{inter}(u(R),s(S)) & \leftarrow & U \geq 0 \wedge R = U-1 \ \wedge S = 1,\ \ aut^{inter}(u(U),t) \\
aut^{inter}(u(R),s(T)) & \leftarrow & U \geq 0 \wedge R = U-1 \ \wedge T = S+1 \wedge S < 4, \\
 & & \qquad\qquad\qquad aut^{inter}(u(U),s(S)) \\
aut^{inter}(v(V),t) & \leftarrow & V \geq -4 \ \wedge V \leq 4,\ \ aut^{inter}(r,s(S)) \\
aut^{inter}(u(U),t) & \leftarrow & V \geq 0 \wedge U = V-1,\ \ aut^{inter}(v(V),s(S)) \\
aut^{inter}(w(W),t) & \leftarrow & U \geq -1 \wedge W = U,\ \ aut^{inter}(u(U),s(S)) \\
aut^{inter}(v(V),t) & \leftarrow & W+1 = V,\ \ aut^{inter}(w(W),s(S)) \\
aut^{inter}(q,t) & \leftarrow & V < 0,\ \ aut^{inter}(v(V),s(S)) \\
aut^{inter}(u(R),s_{erreur}) & \leftarrow & U \geq 0 \wedge R = U-1 \ \ \wedge S \geq 4,\ \ aut^{inter}(u(U),s(S)) \\
aut^{inter}(v(V),s_{erreur}) & \leftarrow & V \geq -4 \ \wedge A \leq 4,\ \ aut^{inter}(r,s_{erreur}) \\
aut^{inter}(u(U),s_{erreur}) & \leftarrow & U = V-1,\ \ aut^{inter}(v(V),s_{erreur}) \\
aut^{inter}(u(R),s_{erreur}) & \leftarrow & U \geq 0 \wedge R = U-1,\ \ aut^{inter}(u(U),s_{erreur}) \\
aut^{inter}(w(W),s_{erreur}) & \leftarrow & U \geq -1 \wedge W = U,\ \ aut^{inter}(u(U),s_{erreur}) \\
aut^{inter}(v(V),s_{erreur}) & \leftarrow & W+1 = V,\ \ aut^{inter}(w(W),s_{erreur}) \\
aut^{inter}(q,s_{erreur}) & \leftarrow & V < 0,\ aut^{inter}(v(V),s_{erreur})
\end{array}
$$

3. Calcul du résultat de l'évaluation ascendante de Π^{tronc}. L'application de l'opérateur $\mathcal{T}$ engendre :

$$
\begin{array}{lll}
\mathcal{T}\uparrow^0 & = & \{\ aut^{inter}(r,t)\ \ \} \\[2ex]
\bigcup_{0\leq j\leq 1}\mathcal{T}\uparrow^j & = & \mathcal{T}\uparrow^0\ \cup \{aut^{inter}(v(m),t)\ /\ -4 \leq m \leq 4\ \} \\[2ex]
\bigcup_{0\leq j\leq 2}\mathcal{T}\uparrow^j & = & \bigcup_{0\leq j\leq 1}\mathcal{T}\uparrow^j\ \cup\ \{\ aut^{inter}(q,t)\ \}\ \cup \\
 & & \qquad\qquad\qquad \{\ aut^{inter}(u(m),t)\ /\ -1 \leq m \leq 3\ \} \\[2ex]
\bigcup_{0\leq j\leq 3}\mathcal{T}\uparrow^j & = & \bigcup_{0\leq j\leq 2}\mathcal{T}\uparrow^j\ \cup\ \{\ aut^{inter}(w(m),t)\ /\ -1 \leq m \leq 3\ \}\ \cup \\
 & & \qquad \{\ aut^{inter}(u(m),s(n))\ /\ -1 \leq m \leq 2\ \wedge\ n = 1\ \} \\[2ex]
\bigcup_{0\leq j\leq 4}\mathcal{T}\uparrow^j & = & \bigcup_{0\leq j\leq 3}\mathcal{T}\uparrow^j\ \cup\ \{\ aut^{inter}(u(m),s(n))\ /\ -1 \leq m \leq 1\ \wedge\ n = 2\ \} \\[2ex]
\bigcup_{0\leq j\leq 5}\mathcal{T}\uparrow^j & = & \bigcup_{0\leq j\leq 4}\mathcal{T}\uparrow^j\ \cup\ \{\ aut^{inter}(u(m),s(n))\ /\ -1 \leq m \leq 0\ \wedge\ n = 3\ \} \\[2ex]
\bigcup_{0\leq j\leq 6}\mathcal{T}\uparrow^j & = & \bigcup_{0\leq j\leq 5}\mathcal{T}\uparrow^j\ \cup\ \{\ aut^{inter}(u(m),s(n))\ /\ m = -1\ \wedge\ n = 4\ \} \\[2ex]
\bigcup_{0\leq j\leq 7}\mathcal{T}\uparrow^j & = & \bigcup_{0\leq j\leq 6}\mathcal{T}\uparrow^j
\end{array}
$$

Le plus petit point-fixe de Π^{tronc} est donc $\bigcup_{0\leq j\leq 6}\mathcal{T}\uparrow^j$. Comme l'état s_{erreur} n'est jamais atteint, on a $\mathcal{T}\uparrow^{\omega}{}_2\ = \emptyset$, et donc $Rec(AUT_4)\ \subseteq\ Rec(AUT_5)$. □

9. Comparaison avec des travaux récents

Récemment, plusieurs travaux se sont indépendamment attaqués au problème de la preuve de propriétés d'automates ayant des paramètres entiers. Il s'agit notamment du travail de Halbwachs [HAL 93] et de celui de Higashino-Bochmann [HIG 94]. Les automates considérés dans ces travaux sont, comme les nôtres, des automates dont les états et les actions contiennent des paramètres entiers. Les transitions d'un état à l'autre sont également conditionnées par des contraintes d'arithmétique linéaire. Ces travaux présentent cependant plusieurs différences avec notre approche, qui sont rapidement expliquées ici.

Une première différence tient au cadre conceptuel : les automates étudiés dans les travaux de Halbwachs et Higashino-Bochmann sont obtenus par compilation à partir de programmes écrits dans des langages synchrones tels que ESTEREL [HAL 93] et LOTOS [HIG 94], alors que nos automates sont écrits directement sous forme de programmes logiques avec contraintes.

Une seconde différence tient à la nature des propriétés prouvées dans les deux approches. Halbwachs et Higashino-Bochmann s'intéressent surtout à la preuve de *propriétés de sûreté* et font appel à la notion d'*automate observateur* [HAL 93], tandis que nous nous sommes intéressés au problème d'inclusion de langage. Il s'agit là, en fait, d'une différence essentiellement formelle, car notre codage du problème d'inclusion peut facilement être interprété comme une propriété de sûreté faisant intervenir un automate observateur. Nous avons en effet exprimé le problème d'inclusion sous la forme $aut^{inter}(L, X_1, X_2) \Rightarrow X_2 \neq s_{erreur}$, ce qui correspond à une propriété de sûreté (l'état s_{erreur} n'est jamais atteint). En outre, l'automate aut^{inter} a été obtenu par intersection d'un automate aut_1 avec un automate aut_2^{comp}. Ce dernier peut être considéré comme un automate observateur du premier. De façon générale, notre méthode peut être utilisée pour prouver des propriétés de sûreté d'un automate $aut(L, X)$: cela se fait en codant la propriété de sûreté sous la forme d'une formule arithmétique $\psi(X)$, et en démontrant la formule $aut(L, X) \Rightarrow \psi(X)$ selon la méthode de la section 8 (troncation de l'argument L, puis calcul du plus petit point-fixe associé au programme aut tronqué). Un exemple est donné dans la partie suivante.

Une troisième différence concerne l'analyse du programme manipulant les paramètres arithmétiques de l'automate. Halbwachs et Higashino-Bochmann ne cherchent pas, comme nous, à calculer le plus petit point-fixe de ce programme. Ils cherchent simplement à en calculer des invariants pertinents. Ces invariants sont soit obtenus manuellement [HIG 94], soit engendrés automatiquement [HAL 93] à l'aide de techniques d'interprétation abstraite [COU 78]. Ces invariants sont, en fait, des approximations du plus petit point-fixe : ils sont plus faciles à calculer que celui-ci, mais peuvent se révéler trop faibles pour prouver la propriété. En sens inverse, le plus petit point-fixe peut être vu comme un invariant "optimal" qui, lorsqu'il est connu, permet toujours de prouver la propriété. Son calcul par évaluation ascendante, malheureusement, ne termine pas toujours, ainsi que nous l'avons déjà souligné. Ceci suggère, à l'avenir, d'essayer de combiner les deux approches d'analyse pour engendrer des invariants plus puissants ou améliorer la convergence des algorithmes de calcul de plus petit point fixe.

Signalons enfin les travaux de Shankar et Lam [SHA 87, SHA 93]. Ces travaux portent également sur la vérification de systèmes concurrents ayant des paramètres entiers. Cependant les invariants manipulés dans ces travaux ne sont pas des formules purement arithmétiques, comme ici, mais font intervenir d'autres types de données que les entiers, telles que des files ou des tableaux. Ces invariants peuvent donc exprimer des propriétés plus fines, mais leur génération automatique est plus difficile. Noter que les heuristiques de génération d'invariants, proposés par Shankar et Lam, procèdent par chaînage arrière (en partant de la propriété à démontrer), et non pas par chaînage avant (en partant des conditions initiales) comme dans les procédures d'évaluation ascendante.

10. Exemple de preuve de propriété de sûreté

On considère dans cette partie l'automate AUT, décrit dans [HAL 93] et représenté figure 15. Cet automate correspond au programme :

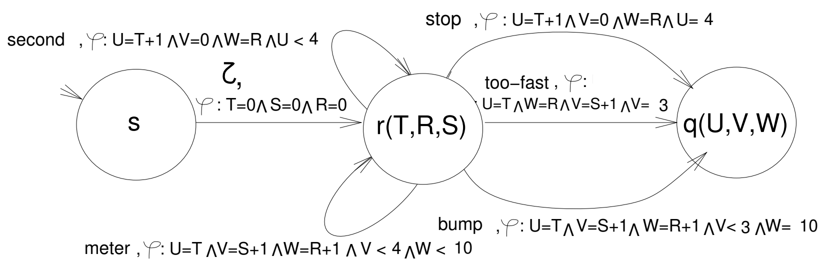

Figure 15. . *Automate AUT*

$$
\begin{array}{lcl}
aut([\,], s) & & \\
aut([\tau|L], r(T,S,R)) & \leftarrow & T=0 \wedge S=0 \wedge R=0,\ aut(L,s) \\
aut([second|L], r(U,V,W)) & \leftarrow & U=T+1 \wedge V=0 \wedge W=R \wedge U<4, \\
 & & \hfill aut(L, r(T,S,R)) \\
aut([meter|L], r(U,V,W)) & \leftarrow & U=T \wedge V=S+1 \wedge W=R+1 \wedge V<3 \wedge \\
 & & \hfill W<10,\ aut(L, r(T,S,R)) \\
aut([stop|L], q(U,V,W)) & \leftarrow & U=T+1 \wedge V=0 \wedge W=R \wedge U=4, \\
 & & \hfill aut(L, r(T,S,R)) \\
aut([bump|L], q(U,V,W)) & \leftarrow & U=T \wedge V=S+1 \wedge W=R+1 \wedge V<3 \wedge \\
 & & \hfill W=10,\ aut(L, r(T,S,R)) \\
aut([too\!-\!fast|L], q(U,V,W)) & \leftarrow & U=T \wedge W=R \wedge V=S+1 \wedge V=3, \\
 & & \hfill aut(L, r(T,S,R))
\end{array}
$$

L'objectif est de montrer la propriété de sûreté suivante : l'action *bump* ne se produit jamais. Cela revient à vérifier que les arguments d'un état $r(T,S,R)$ ne satisfont jamais la contrainte de la sixième clause du programme (associée à l'action *bump*). La propriété se formalise donc :

$$\begin{aligned} aut(L,X) \ \Rightarrow \ & (\ X = r(T,S,R) \ \Rightarrow \ \neg(\ \exists\, U,V,W\ (U = T \ \wedge \\ & V = S+1 \ \wedge \ W = R+1 \ \wedge \ V < 3 \ \wedge \ W = 10)\ )\ ) \qquad (\star\star) \end{aligned}$$

Noter que $\exists\, U,V,W\ (U = T \wedge V = S{+}1 \wedge W = R{+}1 \wedge V < 3 \wedge W = 10)$ se simplifie en $S+1 < 3 \ \wedge \ R+1 = 10$.

Un état $r(T,S,R)$ ne peut jamais être atteint si l'une des trois dernières clauses du programme est utilisée. En conséquence, pour la preuve de $(\star\star)$, on peut se limiter à l'évaluation ascendante du programme restreint à ses cinq premières clauses. En adaptant la méthode décrite en [FRI 92b], on obtient alors :

$$\mathcal{T}\!\uparrow^{\omega} = \{\ aut(s)\ \} \ \cup \ \{\ aut(r(m,0,0)) \ / \ 0 \leq m < 4\} \ \cup \ \{\ aut(r(0,m,m)) \ / \ 0 \leq m < 3\ \} \cup \{\ aut(r(m,j,n)) / \ 0 < m < 4 \wedge 0 \leq j < 3 \wedge n > 0 \wedge j \leq n \leq 2m{+}j\ \}$$

Pour montrer la formule $(\star\star)$, il suffit de montrer que les valeurs de T, R et S telles que $r(T,S,R) \ \in \ \mathcal{T}\!\uparrow^{\omega}$, ne satisfont jamais la relation $S+1 < 3 \ \wedge \ R+1 = 10$. Ceci se vérifie facilement. En conséquence, on a une preuve que l'action *bump* n'est jamais exécutée.

Remarquons que, sur cet exemple, le résultat de l'évaluation ascendante que nous avons trouvé (qui est un plus petit point-fixe) coincide avec le résultat obtenu par Halbwachs [HAL 93] (qui, *a priori*, n'est qu'une approximation de ce plus petit point-fixe).

11. Conclusion

Nous avons présenté une approche où des automates paramétrés sont exprimés sous forme de programmes logiques avec contraintes. Nous avons proposé d'utiliser des techniques d'évaluation ascendante (comme la procédure de Revesz) pour calculer les point-fixes de ces programmes, en vue de résoudre des problèmes d'inclusion de langage ou démontrer des propriétés de sûreté. Nous avons comparé notre approche avec des travaux récents similaires. Une originalité de notre travail est d'utiliser le cadre unifié de la Programmation Logique avec Contraintes à la fois pour la spécification et la vérification de systèmes concurrents. Signalons, pour terminer, que la représentation sous forme de programmes logiques permet, en sus des méthodes de preuve décrites plus haut, de faire tourner directement les automates au moyen d'interprètes comme PROLOG III [COL 90] ou CLP(R) [JAF 92], et de tester ainsi empiriquement leur comportement.

Remerciements
Nous remercions Hubert Garavel, Klaus Havelund, Michel Boyer et les membres du groupe VTT pour les nombreuses et utiles discussions que nous avons eues avec eux. La présente version de cet article a bénéficié des importantes suggestions faites par les *referees* de TSI.

12. Bibliographie

[ARN 82] ARNOLD A. et NIVAT M., « Comportements de processus », *Colloque AFCET "Les Mathématiques de l'Informatique"*, 1982, p. 35-68.

[ARN 93] ARNOLD A., « Verification and Comparison of Transition Systems », *Proc. TAPSOFT*, Lecture Notes in Computer Science 668, Springer-Verlag, 1993, p.121-135.

[BAN 86] BANCILHON F. et *al.*, « Magic Sets and Other Strange Ways to Implement Logic Programs », *Proc. ACM Symp. on Principles of Databases Systems*, Boston, 1986, p. 1-15.

[BAU 91] BAUDINET M. et *al.*, « On the Representation of Infinite Temporal Data Queries », *Proc. 10th ACM Symp. on Principles of Database Systems*, Denver, 1991, p. 280-290.

[BOE 93] BOER F.S. et *al.*, « A Paradigm for Asynchronous Communication and its Application to Concurrent Constraint Programming », *Logic Programming Languages : Constraints, Functions and Objects*, K.R. Apt, J.W. de Bakker et J.J.M.M Rutten, eds., the MIT Press, chapitre 4, 1993, p. 82-114.

[CHO 88] CHOMICKI J. et IMIELIENSKI T., « Temporal Deductive Databases and Infinite Objects », *Proc. 7th ACM Symp. on Principles of Database Systems*, Austin, 1988, p. 61-81.

[COH 90] COHEN J., « Constraint Logic Programming Languages », *Communications of the ACM*, Vol. 33, n° 7, 1990, p. 52-68.

[COL 90] COLMERAUER A., « An Introduction to Prolog III », *Communications of the ACM*, Vol. 33, n° 7, 1990, p. 69-90.

[COO 71] COOPER D.C., « Programs for Mechanical Program Verification », *Mach. Intelligence 6*, New York, 1971, p. 43-59.

[COU 78] COUSOT P. et HALBWACHS N., « Automatic Discovery of Linear Restraints among Variables of a Program », *Conference Record 5th ACM Symp. on Principles of Programming Languages*, Tucson, 1978, p. 84-96.

[EIL 74] EILENBERG S., *Automata, Languages and Machines*, Academic Press, 1974.

[FRI 92a] FRIBOURG L., « Mixing List Recursion and Arithmetic », *Proc. 7th IEEE Symp. on Logic in Computer Science*, Santa Cruz, 1992, p. 419-429.

[FRI 92b] FRIBOURG L. et VELOSO PEIXOTO M., *Bottom-up Evaluation of Datalog Programs with Arithmetic Constraints*, Rapport de Recherche LIENS-92-13, Ecole Normale Supérieure, juin 1992 (Version courte dans CADE-94, Lecture Notes in Computer Science 814, Springer-Verlag, 1994).

[FRI 93] FRIBOURG L. et VELOSO PEIXOTO M., *Concurrent Constraint Automata*, Rapport de Recherche LIENS-93-10, Ecole Normale Supérieure, mai 1993.

[HAL 93] HALBWACHS N., « Delay Analysis in Synchronus Programs », *Proc. Computer Aided Verification*, Lecture Notes in Computer Science 697, Springer-Verlag, 1993, p. 333-346.

[HIG 94] HIGASHINO T. et BOCHMANN G., « Automatic Analysis and Test Case Derivation for a Restricted Class of LOTOS Expressions with Data Parameters », *IEEE Transactions on Software Engineering*, Vol. 20, n° 1, 1994, p. 29-42.

[JAF 87] JAFFAR J. et LASSEZ J., « Constraint Logic Programming », *Proc. 14th ACM Symp. on Principles of Programming Languages*, 1987, p. 111-119.

[JAF 92] JAFFAR J. et *al.*, « The CLP(R) Language and System », *ACM Transactions on Programming Languages Systems*, Vol. 14, n° 3, 1992, p. 339-395.

[KAN 90] KANELLAKIS P. et *al.*, « Constraint Query Languages », *Proc. 9th ACM Symp. on Principles of Database Systems*, Nashville, 1990, p. 299-313.

[LLO 87] LLOYD J., *Foundations of Logic Programming*, 2nd edition, Springer-Verlag, New York, 1987.

[PRE 29] PRESBURGER M., « Uber die Vollstandidkeit eines genissen Systems der Arithmetik ganzer Axhlen », *Welshem die Addition als einzige Operation hervortritt, Sprawosdanie z I Kongresu*, 1929, p. 92-101.

[REV 93] REVESZ P., « A Closed Form For Datalog Queries with Integer (Gap)-Order Constraints », *Theoretical Computer Science*, n° 116, 1993, p. 117-149 (Version préliminaire dans *Int. Conf. on Database Theory*, Lecture Notes in Computer Science 470, Springer-Verlag, 1990, p. 187-201).

[SAR 90] SARASWAT V. et RINARD M., « Concurrent Constraint Programming », *Proc. 17th ACM Symp. on Principles of Programming Languages*, New York, 1990, p. 232-245.

[SHA 87] SHANKAR A. et LAM S., « Time-Dependent Distributed Systems : Proving Safety, Liveness and Real-Time Properties », *Distrib. Comput.*, Vol. 2, n° 2, 1987, p. 61-79.

[SHA 93] SHANKAR A., « An Introduction to Assertional Reasoning for Concurrent Systems », *ACM Computing Surveys*, Vol. 25, n° 3, 1993, p. 225-262.

[SHP 89] SHAPIRO E., « The Family of Concurrent Logic Programming Languages », *ACM Computing Surveys*, Vol. 21, n° 3, 1989, p. 412-450.

[STE 86] STERLING L.et SHAPIRO E., *The Art of Prolog: Advanced Programming Techniques*, The MIT Press, 1986.

[VAN 76] VAN EMDEN M. et KOWALSKI R., « The Semantics of Predicate Logic as a Programming Language », *J.ACM*, Vol. 23, n° 4, 1976, p. 733-742.