THESE DE DOCTORAT

présentée

A L'UNIVERSITE PARIS 7

Spécialité : Informatique Fondamentale

par

Marcos VELOSO PEIXOTO

Sujet de la thèse :

Automates à Contraintes Arithmétiques et Procédures d'Evaluation Ascendante de Programmes Logiques

Soutenue le 14 décembre 1994 devant la Commission d'examen composée de :

Président	**Guy Cousineau**
Rapporteurs	**Philippe Devienne**
	Gérard Ferrand
Examinateurs	**Pierre Deransart**
	Laurent Fribourg
	Hubert Garavel
	Irène Guessarian
	Jan Maluszyński

Remerciements

Je remercie Guy Cousineau de l'honneur qu'il me fait en présidant le jury de cette thèse.

Je remercie Gérard Ferrand et Phillipe Devienne d'avoir accepté d'être rapporteurs de ma thèse. Je les remercie pour leurs remarques, critiques et suggestions.

Je voudrais très specialement remercier mon directeur de thèse, Laurent Fribourg, tant pour la confiance qu'il m'a toujours montré, que pour son esprit critique et constructif dont mon travail a grandement profité.

Je suis aussi très reconnaissant à Irène Guessarian, avec qui j'ai travaillé pendant la première année de ma formation doctarale. Son constant appui et ses encouragements ont été un support essentiel pour l'évolution de mon travail.

Mes remerciements vont également à Pierre Deransart, Hubert Garavel et Jan Maluszyński qui m'ont fait l'honneur de participer au jury.

Introduction

Cette thèse est constituée en deux parties :

1. La modélisation et la vérification de systèmes concurrents manipulant un nombre non borné de données.
2. Le calcul par évaluation ascendante du plus petit point fixe d'un programme logique avec contraintes arithmétiques.

Nous montrerons dans la première partie que le problème de vérification de propriétés de systèmes concurrents manipulant un nombre non borné de données peut se ramener à un problème du calcul de plus petit point fixe d'un programme logique avec contraintes arithmétiques. Cela explique le lien entre les deux parties.

Dans la première partie, 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. L'utilisation de techniques d'évaluation ascendante nous permettra de montrer que, pour une certaine classe de contraintes, *le problème de l'inclusion de langages est décidable*: étant donnés deux automates généralisés AUT_1 et AUT_2, il est possible de dire si, oui ou non, toutes les suites d'actions acceptées par AUT_1 sont acceptées par AUT_2. Cela nous permettra également de prouver des propriétés de sûreté pour ces automates.

Dans la deuxième partie, nous apportons une contribution au problème du calcul par évaluation ascendante du plus petit point fixe d'un programme logique avec des contraintes arithmétiques. Nous définissons une classe de programmes logiques sur des entiers dans laquelle le processus d'évaluation ascendante engendre toujours une formule arithmétique linéaire. Un programme appartenant à cette classe contient au plus trois règles récursives, qui incrémentent leurs arguments arithmétiques selon une condition arithmétique. Nous adoptons une approche géométrique pour construire le résultat du processus d'évaluation ascendante: chaque application d'une règle récursive est représentée soit par un déplacement horizontal, soit par un déplacement vertical, soit par un déplacement transversal. Quelques exemples d'application de cette classe de programmes sont présentés à la fin de la thèse.

Première partie

Automates à Contraintes Arithmétiques

Chapitre 1

Motivation

La notion d'automate fini 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, ROY 90]. Les preuves de propriétés du système se font 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 la représentation et de l'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'actions 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.

Les résultats figurant dans cette partie ont fait l'objet d'un rapport interne LIENS [FRI 93] et d'un article à paraître dans TSI [FRI 94d].

Chapitre 2

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 avec communications 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 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 se 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 du plus petit point fixe.

Signalons également les travaux de Shankar et Lam [SHA 87, SHA 93]. Ces travaux portent aussi 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, tels 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.

Mentionnons enfin que l'approche par programmation logique avec contraintes (mais sur des domaines *finis*) a été récemment utilisée par Corsini et Rauzy [COR 94] pour modéliser des systèmes concurrents et faire de la vérification symbolique sur ces systèmes.

Chapitre 3

Préliminaires

3.1 Automates

Dans ce chapitre, 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 3.1 *Un* automate *est un quadruplet* $\ll \Xi, s_0, \Sigma, \Omega \gg$, *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. Ω *est un sous-ensemble de* $\Xi \times \Sigma \times \Xi$.

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

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 $s_0 \stackrel{a_0}{\longmapsto} s_1 \stackrel{a_1}{\longmapsto} \ldots \stackrel{a_{n-1}}{\longmapsto} s_n$, 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 3.2 . Considérons l'automate AUT_0 représenté figure 3.1. Nous avons :

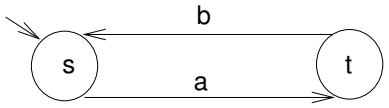


FIG. 3.1 - *Automate* AUT_0

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

3.2 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 $p(\overline{t}) :\!-p_1(\overline{t}_1), ..., p_m(\overline{t}_m)$ dans laquelle $:\!-$ 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 $:\!-p_1(\overline{t}_1), ..., p_m(\overline{t}_m)$. 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 fini donné ne pose pas de difficulté (voir, par exemple, [STE 86]). Etant donné un automate $AUT = \ll \Xi, s_0, \Sigma, \Omega \gg$, on associe à AUT, de manière standard (std), le programme Π^{std}_{AUT} suivant :

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

$$\left\{ aut^{\text{std}}([a|L], s) \ :- \ aut^{\text{std}}(L, t) \right\}$$
$$\text{pour tout} \quad < s, a, t > \in \Omega$$

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 $:\!-aut^{\text{std}}(l, s_0)$ réussit avec le programme Π^{std}_{AUT} 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^{dual}_{AUT} de la forme :

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

$$\left\{ aut^{\text{dual}}([a|L], t) \ :- \ aut^{\text{dual}}(L, s) \right\}$$
$$\text{pour tout} \quad < s, a, t > \in \Omega$$

Le but $:\!-aut^{\text{dual}}(l, X)$ réussit dans Π^{dual}_{AUT} ssi le but $:\!-aut^{\text{std}}(l^{inv}, s_0)$ réussit dans Π^{std}_{AUT} (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'atome $aut^{\text{dual}}(L, X)$ 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 3.3 Considérons l'automate AUT_0 de l'exemple 3.2. Il peut être caractérisé par le programme Π_{AUT_0} :

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

Le but $:\!-aut_0([b,a,b,a],X)$ 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. □

3.3 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 d'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 3.4 *Un* automate généralisé (avec contraintes arithmétiques) *est un quadruplet* $\ll \Xi, s_0(\overline{S}_0), \Sigma, \Omega \gg$, *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. Ω *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 coïncident, 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 :

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

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 $\ll \Xi, s_0(\overline{S}_0), \Sigma, \Omega \gg$, une *transition valide* est une instance d'une transition paramétrée $< s(\overline{S}), a(\overline{A}), t(\overline{T}), \varphi > \in \Omega$ 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 $s(\overline{\sigma}) \stackrel{a(\overline{\alpha})}{\underset{\rho(\varphi)}{\longmapsto}} t(\overline{\theta})$.

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

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

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 3.2 et la transition $< s(\overline{S}), a(\overline{A}), s(\overline{T}), \varphi >$ sera représentée par le diagramme de la figure 3.3.

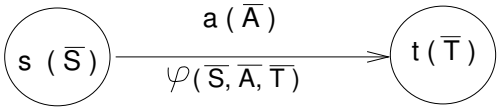


FIG. 3.2 - *Diagramme de Transition* $< s, a, t, \varphi >$

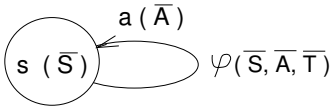


FIG. 3.3 - *Diagramme de Transition* $< s, a, s, \varphi >$

Exemple 3.5 Soit AUT_1 l'automate généralisé $\ll \{s(S), t(T)\}, s(S), \{a(A), b(B)\}, \Omega \gg$, où $\Omega = \{ < s(S), a(A), t(T), A = S >, < t(T), b(B), s(S), B = S > \}$.

Cet automate est représenté par la figure 3.4.

Le langage reconnu est :

$Rec(AUT_1) = \{[\,]\} \cup \{ [a(m)] \,/\, m \in \mathbb{Z}\} \cup \{ [a(m), b(m')] \,/\, m, m' \in \mathbb{Z}\} \cup$

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).

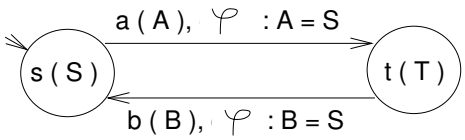


FIG. 3.4 - *Automate* AUT_1

$$\{\ [a(m), b(m'), a(m')]\ /\ m, m' \in \mathbb{Z}\ \} \cup$$
$$\{\ [a(m), b(m'), a(m'), b(m'')]\ /\ m, m', m'' \in \mathbb{Z}\ \} \cup \ldots \qquad \square$$

3.4 Programmes Logiques avec Contraintes

Nous allons à présent étendre les résultats de la section 3.2 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} :

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

$$\left\{\ aut([a(\overline{A})|L], t(\overline{T}))\ \text{:-}\ \varphi(\overline{S}, \overline{A}, \overline{T}),\ aut(L, s(\overline{S}))\ \right\}$$

pour tout $< s, a, t, \varphi > \in \Omega$

L'automate de l'exemple 3.5 est ainsi représenté par le programme :

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

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

Proposition 3.6. *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 $:\!-aut(l, X)$ 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.

En ce qui concerne le pouvoir d'expression des automates généralisés, nous avons :

Proposition 3.7 *Les automates généralisés ont les même pouvoir d'expression que les machines Turing.*

Cette proposition découle du fait[3] qu'on peut coder une machine de Minsky [MIN 67, ROU 94] par un programme logique associé à un automate généralisé et du fait que ces dernières ont le même pouvoir d'expression que les machines de Turing.

Dans les chapitres suivants, on sera amené à faire un certain nombre de manipulations sur le programme logique Π_{AUT}. 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^{tronc}_{AUT} :

$$
\begin{array}{l}
aut(s_0(\overline{S}_0)) \\
\left\{ aut(t(\overline{T}))\ :-\ \varphi(\overline{S}, \overline{A}, \overline{T}),\ aut(s(\overline{S}))\ \right\} \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{pour tout} < s, a, t, \varphi > \in \Omega
\end{array}
$$

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 $\exists \overline{A}\ \varphi(\overline{S}, \overline{A}, \overline{T})$

3. Je suis redevable à Philippe Devienne pour cette observation

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 :

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

$$\left\{ \; aut(t(\overline{T})) \;\; :\!- \;\; \varphi(\overline{S}, \overline{T}),\; aut(s(\overline{S})) \; \right\}$$
$$\text{pour tout} < s, a, t, \varphi > \in \Omega$$

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. [END 76, JAF 87]) est défini, pour un ensemble d'atomes clos (sans variables) $\mathcal{I}$, par :

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

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 :

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

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.

Chapitre 4

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 $\ll \Xi, s_0(\overline{S}_0), \Sigma, \Omega \gg$ et $\ll \Xi', s'_0(\overline{S}'_0), \Sigma', \Omega' \gg$. 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 :

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

$\left\{ \; aut([a(\overline{A})|L], t(\overline{T})) \;\; :\!- \; \varphi(\overline{S}, \overline{A}, \overline{T}), \;\; aut(L, s(\overline{S})) \; \right\}$

pour tout $< s, a, t, \varphi > \in \Omega$

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

$\left\{ \; aut'([a'(\overline{A}')|L], t'(\overline{T}')) \;\; :\!- \; \varphi'(\overline{S}', \overline{A}', \overline{T}'), \;\; aut'(L, s'(\overline{S}')) \; \right\}$

pour tout $< s', a', t', \varphi' > \in \Omega'$

4.1 Intersection

Définition 4.1 *L'*automate intersection *de AUT et AUT' est l'automate généralisé AUT^{inter} caractérisé par le programme :*

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

$$\left\{ \begin{array}{l} aut^{inter}([a(\overline{A})|L], t(\overline{T}), t'(\overline{T}')) \quad :- \quad \varphi(\overline{S}, \overline{A}, \overline{T}) \ \wedge \ \varphi'(\overline{S}', \overline{A}', \overline{T}') \ \wedge \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad A = A', \ aut^{inter}(L, s(\overline{S}), s'(\overline{S}')) \end{array} \right\}$$
$$pour\ tout\ \ < s, a, t, \varphi > \in \Omega \quad et \quad < s', a', t', \varphi' > \in \Omega'\ tels\ que\ a = a'$$

On peut facilement montrer :

Proposition 4.2 $Rec(AUT^{inter}) = Rec(AUT) \cap Rec(AUT')$

4.2 Mélange

Etant donné deux sous-ensembles A et A' des 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 4.3 *L'* automate mélange *de AUT et AUT' est l'automate généralisé AUT^{mel} caractérisé par le programme :*

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

$$\left\{ \begin{array}{l} aut^{mel}([a(\overline{A})|L], t(\overline{T}), s'(\overline{S}')) \quad :- \quad \varphi(\overline{S}, \overline{A}, \overline{T}), \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad aut^{mel}(L, s(\overline{S}), s'(\overline{S}')) \end{array} \right\}$$
$$pour\ tout\ \ < s, a, t, \varphi > \in \Omega \ et\ s' \in \Xi'$$

$$\left\{ \begin{array}{l} aut^{mel}([a'(\overline{A}')|L], s(\overline{S}), t'(\overline{T}')) \quad :- \quad \varphi'(\overline{S}', \overline{A}', \overline{T}'), \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad aut^{mel}(L, s(\overline{S}), s'(\overline{S}')) \end{array} \right\}$$
$$pour\ tout\ \ < s', a', t', \varphi' > \in \Omega' \ et\ s \in \Xi$$

On peut montrer facilement :

Proposition 4.4 $Rec(AUT^{mel}) = Rec(AUT) \sqcup\!\sqcup Rec(AUT')$.

4.3 Composition Parallèle

On peut, par analogie avec les automates 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

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

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 :

$aut([\,], s(S))$
$aut([a(A)|L], t(T)) \ :- \ A = T, \ aut(L, s(S))$
$aut([b(B)|L], s(S)) \ :- \ B = S, \ aut(L, t(T))$

$aut'([\,], w(W))$
$aut'([c(C)|L], v(V)) \ :- \ C = W, \ aut'(L, w(W))$
$aut'([d(D)|L], w(W)) \ :- \ aut'(L, v(V))$

Les automates AUT et AUT' sont représentés figure 4.1.

Supposons qu'on veuille *synchroniser* la transition étiquetée avec l'action $b(B)$ dans AUT

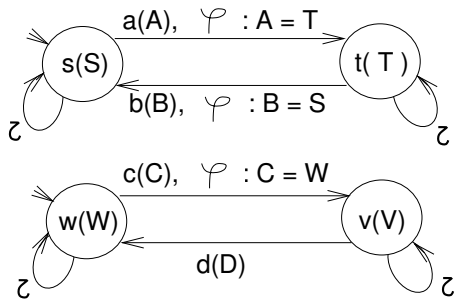


FIG. 4.1 - *Automates AUT et AUT'*

avec la transition étiquetée avec l'action $c(C)$ dans AUT'. Nous caractérisons l'automate de composition parallèle AUT^{paral} par le programme :

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

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[1].

L'automate AUT^{paral} est représenté figure 4.2.

Un exemple de composition plus réaliste (composition de processus dans un protocole de bit alterné) est donné dans le chapitre 7. Un deuxième exemple, correspondant au protocole de la fenêtre glissante, est donné dans l'annexe A.

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

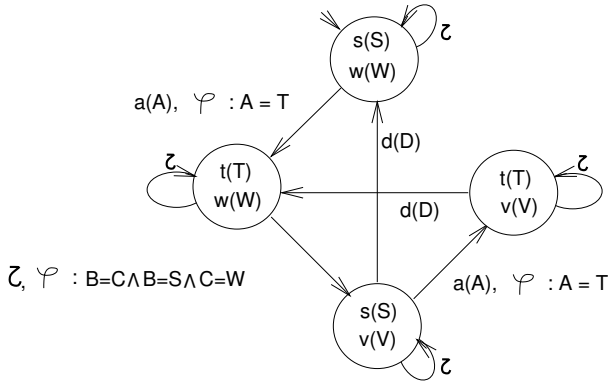


FIG. 4.2 - *Automate* AUT^{paral}

4.4 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.

Dans cette partie, on ne considérera 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[2] à un automate déterministe (voir partie 4.6).

Pour définir l'automate complété, on ajoutera un nouvel état, noté s_{erreur}, à l'ensemble des états de l'automate original. Completer un automate c'est ajouter, pour chaque transition manquante, une transition vers s_{erreur}. On a :

Définition 4.5 *L'*automate complété *de AUT est l'automate généralisé* AUT^{comp} *caractérisé par le programme :*

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

$$\left\{ aut^{comp}([a(\overline{A})|L], t(\overline{T})) \;:\!-\; \varphi(\overline{S}, \overline{A}, \overline{T}),\; aut^{comp}(L, s(\overline{S})) \right\}$$
$$pour\ tout \quad < s, a, t, \varphi > \in \Omega$$

2. dans le sense d'équivance de langage

$$\left\{ \begin{array}{l} aut^{comp}([a(\overline{A})|L], s_{erreur}) \quad :- \quad (\ \forall\ \overline{T}\ \neg\varphi(\overline{S}, \overline{A}, \overline{T})), \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad aut^{comp}(L, s(\overline{S})) \end{array} \right\}$$

pour tout $a \in \Sigma,\ s \in \Xi,$ *et* φ *tel que* $\exists\ t \in \Xi\ /\quad < s, a, t, \varphi > \in \Omega$ [3]

$$\left\{ aut^{comp}([a(\overline{A})|L], s_{erreur}) \ :- \ aut^{comp}(L, s(\overline{S})) \right\}$$

pour tout $a \in \Sigma,\ s \in \Xi$ *et* φ *tel que* $\forall\ t \in \Xi\ /\quad < s, a, t, \varphi > \notin \Omega$ [4]

$$\left\{ aut^{comp}([a(\overline{A})|L], s_{erreur}) :- \ aut^{comp}(L, s_{erreur}) \right\}$$

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

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

Il découle de la proposition 4.6 que, comme désiré, l'automate AUT^{comp} est capable de décider si, oui ou non, une liste l d'actions est reconnue par AUT.

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

4.5 Un exemple d'automate complété

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

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

Cet automate est représenté figure 4.3.

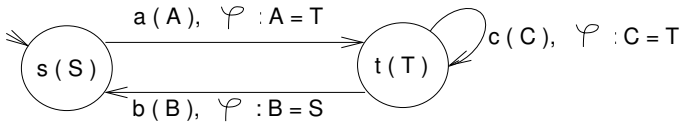


FIG. 4.3 - *Automate* AUT_2

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 4.4.

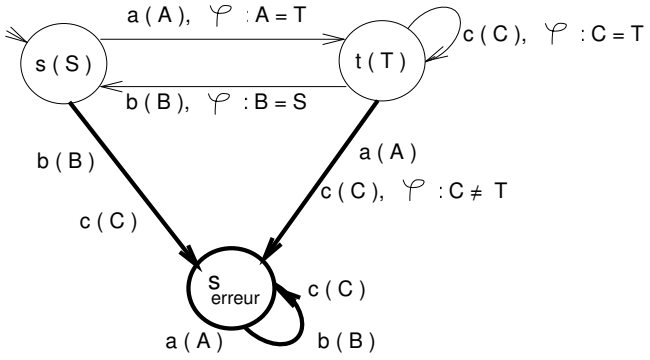


FIG. 4.4 - *Automate* aut_2^{comp}

Le programme $\Pi^{comp}_{AUT_2}$ associé à AUT^{comp}_2 est :

$aut^{comp}_2([\,], s(S))$
$aut^{comp}_2([a(A)|L], t(T)) \;:-\; A = T,\ aut^{comp}_2(L, s(S))$
$aut^{comp}_2([b(B)|L], s(S)) \;:-\; B = S,\ aut^{comp}_2(L, t(T))$
$aut^{comp}_2([c(C)|L], t(S)) \;:-\; C = T,\ aut^{comp}_2(L, t(T))$
$aut^{comp}_2([c(C)|L], s_{erreur}) \;:-\; aut^{comp}_2(L, s(S))$
$aut^{comp}_2([b(B)|L], s_{erreur}) \;:-\; aut^{comp}_2(L, s(S))$
$aut^{comp}_2([a(A)|L], s_{erreur}) \;:-\; aut^{comp}_2(L, t(T))$
$aut^{comp}_2([c(C)|L], s_{erreur}) \;:-\; C \neq T,\ aut^{comp}_2(L, t(T))$
$aut^{comp}_2([a(A)|L], s_{erreur}) \;:-\; (\forall T\ A \neq T),\ aut^{comp}_2(L, s(S)) \quad (*)$
$aut^{comp}_2([b(B)|L], s_{erreur}) \;:-\; (\forall S\ B \neq S),\ aut^{comp}_2(L, t(T)) \quad (**)$
$aut^{comp}_2([a(A)|L], s_{erreur}) \;:-\; aut^{comp}_2(L, s_{erreur})$
$aut^{comp}_2([b(B)|L], s_{erreur}) \;:-\; aut^{comp}_2(L, s_{erreur})$
$aut^{comp}_2([c(C)|L], s_{erreur}) \;:-\; aut^{comp}_2(L, s_{erreur})$

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 : $- aut^{comp}_2([b(7), c(5), a(5)], X)$ réussit dans $\Pi^{comp}_{AUT_2}$ car le but $:- aut_2([b(7), c(5), a(5)], X)$ réussit dans Π_{AUT_2}.

Le but : $- aut^{comp}_2([b(7), c(5), a(3)], s_{erreur})$ réussit dans $\Pi^{comp}_{AUT_2}$ car le but $:- aut_2([b(7), c(5), a(3)], X)$ échoue dans Π_{AUT_2}.

4.6 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 Ω de $s(\overline{S})$ vers $t(\overline{T})$, étiquetée par $a(\overline{A})$[5].

5. 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 $< s, a, t, \varphi_1 \vee \varphi_2 >$.

Soit $\Xi(u,b)$ l'ensemble $\{\ t\ \in \Xi/\ \exists\varphi\ \ <u,b,t,\varphi> \in\ \Omega\ \}$. 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 :

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

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

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

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

$$\left\{ aut'([a(A)|L], q(I, \overline{Q})) \quad :- \quad \varphi(\overline{S}, \overline{A}, \overline{Q}) \wedge \ I = j, \ \ aut'(L, s(\overline{S})) \right\}$$
pour tout $s, t_j \in \Xi$ et $a \in \Sigma$ tels que
$< s, a, t_j, \varphi > \in \Omega, \ s \notin \Xi(u, b)$ et $t \in \Xi(u, b)$

$$\left\{ \begin{array}{l} aut'([a(A)|L], q(J, \overline{R})) \quad :- \quad \varphi(\overline{Q}, \overline{A}, \overline{R}) \ \wedge \ I = i \wedge \ J = j, \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad aut'(L, q(I, \overline{Q})) \end{array} \right\}$$
pour tout $a \in \Sigma$ et $t_i, t_j \in \Xi$ tels que
$< t_i, a, t_j, \varphi > \in \Omega, \ t_i \in \Xi(u, b)$ 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 4.7 $Rec(AUT) = Rec(AUT^{det})$

Exemple 4.8 Considérons l'automate AUT_3 caractérisé par :

$aut_3([\,], s(S))$
$aut_3([a(A)|L], t(T)) \quad :- \quad A = T, \ \ aut_3(L, s(S))$
$aut_3([a(A)|L], u(U)) \quad :- \quad A = S, \ \ aut_3(L, s(S))$
$aut_3([c(C)|L], u(U)) \quad :- \quad C = T, \ \ aut_3(L, t(T))$
$aut_3([b(B)|L], v(V)) \quad :- \quad T = V, \ \ aut_3(L, t(T))$
$aut_3([b(B)|L], w(W)) \quad :- \quad B = W, \ \ aut_3(L, u(U))$
$aut_3([a(A)|L], v(V)) \quad :- \quad aut_3(L, w(W))$

Cet automate est représenté figure 4.5.

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 4.6) :

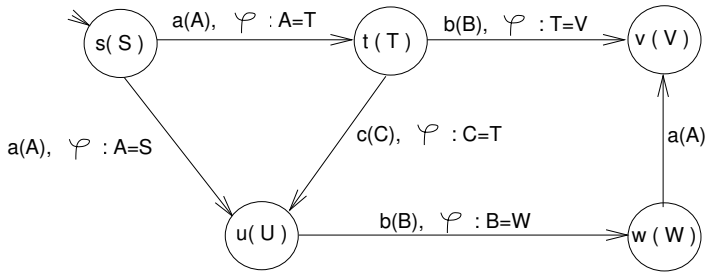


FIG. 4.5 - *Automate* AUT_3

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

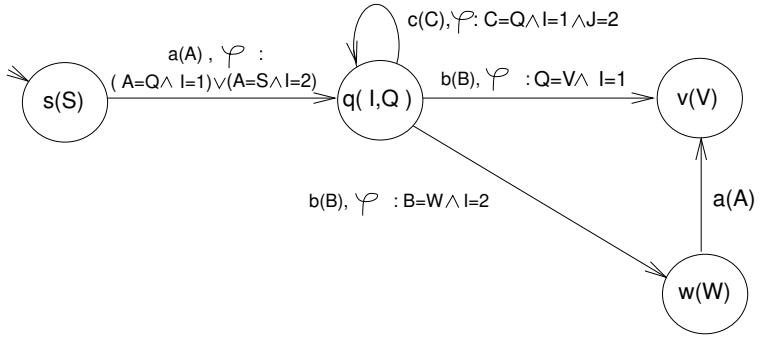


FIG. 4.6 - *Automate* AUT'_3

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 4.7) :

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

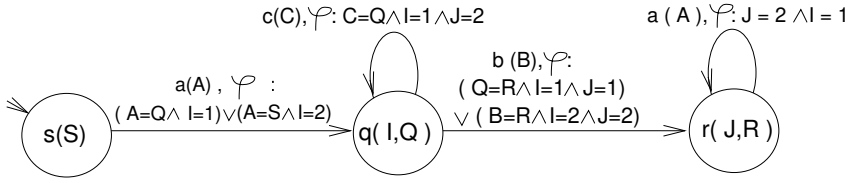


FIG. 4.7 - *Automate* AUT''_3

L'automate AUT''_3 est déterministe, et notre procédure termine ainsi en deux étapes ($AUT_3^{det} = AUT''_3$). □

Chapitre 5

Problème de l'Inclusion de Langage

Dans ce chapitre, nous étudions le problème de l'inclusion de langage pour les automates généralisés avec contraintes arithmétiques : étant donné 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 finis [EIL 74]. Nous décrivons ici une méthode pour traiter ce problème au niveau des automates généralisés avec contraintes arithmétiques. Nous allons compléter l'automate AUT_2. Soit s_{erreur} le nouvel état ajouté par l'opération de complétion. De la proposition 4.6 appliquée à AUT_2 et de la définition des prédicats aut_1 et aut_2, il découle :

Proposition 5.1 *Etant donné deux automates généralisés AUT_1 et AUT_2, $Rec(AUT_1) \subseteq Rec(AUT_2)$ ssi $\forall L\ \forall X_1\ \forall X_2\ (\ aut_1(L, X_1) \wedge\ aut_2^{comp}(L, X_2) \Rightarrow X_2 \neq s_{erreur}$*[1].

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 :

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}. Par construction, on a : $aut^{inter}(L, X_1, X_2) \Leftrightarrow$

1. L'expression $X_2 \neq s_{erreur}$ signifie que l'état s_{erreur} n'est jamais atteint.

$aut_1(L, X_1) \wedge aut_2^{comp}(L, X_2)$.

2. Construire le programme Π^{tronc} (définissant l'atome $aut^{inter}(X_1, X_2)$), 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 :

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

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

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

D'après [FRI 92], $\exists L \ aut^{inter}(L, X_1, X_2)$ est vrai ssi $aut^{inter}(X_1, X_2)$ est vrai. On peut donc raisonner sur le programme tronqué Π^{tronc} (sans l'argument de liste), à la place du programme Π. Par conséquent, 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 quand la validité des formules engendrées $\zeta_{s_1}(\overline{S}_1)$ 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 90] (qui engendre une formule d'arithmétique linéaire décidable). En conséquence, on a :

Proposition 5.2 *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 chapitre 7). 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 coïncide 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, KAN 90] et la procédure de la partie II de cette thèse), mais la terminaison n'est pas alors garantie dans le cas général.

La méthode pour prouver l'inclusion de langages reconnus par deux automates généralisés peut être appliquée pour prouver des propriétés sur le langage reconnu par un automate généralisé. Supposons qu'on désire prouver une propriété ϕ sur le langage reconnu par un automate généralisé AUT. D'abord, il faut construire un automate AUT_ϕ tel que $l \in Rec(AUT_\phi)$ ssi l satisfait la propriété ϕ. La preuve que toute liste reconnue par AUT satisfait la propriété ϕ se ramène ainsi à prouver que $REC(AUT) \subseteq REC(AUT_\phi)$.

Exemple 5.3 Considérons l'automate AUT_4 (adapté de [HIG 94]) représenté figure 5.1.

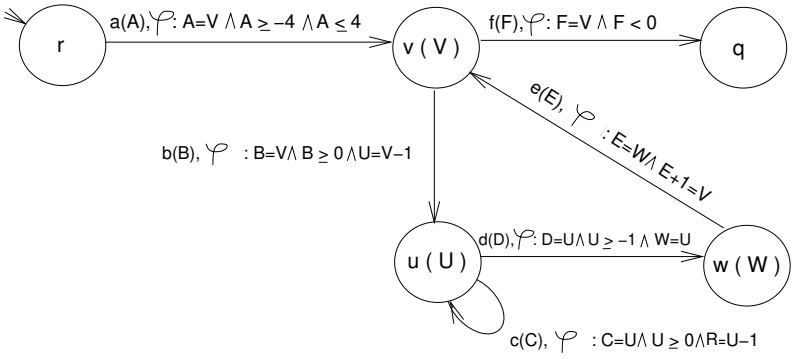


FIG. 5.1 - *Automate* AUT_4

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

Considérons à présent l'automate AUT_5 représenté figure 5.2. Cet automate reconnaît une liste d'actions ssi elle ne contient pas plus de 4 actions c consécutives.

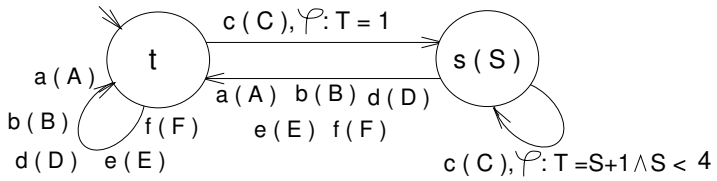


FIG. 5.2 - *Automate* AUT_5

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

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 5.3):

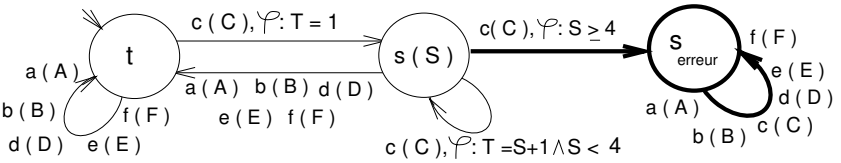


FIG. 5.3 - *Automate* AUT_5^{comp}

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

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^{inter}, 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, 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,\ 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,\ aut^{inter}(L,u(U),t) \\
aut^{inter}([e(E)|L],v(V),t) & \leftarrow & E=E' \wedge E=W \wedge E+1=V,\ aut^{inter}(L,w(W),t) \\
aut^{inter}([f(F)|L],q,t) & \leftarrow & F=F' \wedge F=V \wedge F<0,\ 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 S=1, \\
 & & \qquad 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 T=S+1 \wedge \\
 & & \qquad 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,\ 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,\ aut^{inter}(L,w(W),s(S)) \\
aut^{inter}([f(F)|L],q,t) & \leftarrow & F=F' \wedge F=V \wedge F<0,\ 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,\ 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,\ 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,\ \ 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,\ \ aut^{inter}(L,u(U),s_{erreur}) \\
aut^{inter}([e(E)|L],v(V),s_{erreur}) & \leftarrow & E=W \wedge E+1=V,\ \ 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} :

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

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

$$\mathcal{T}\uparrow^0 = \{\ aut^{inter}(r,t)\ \}$$

$$\bigcup_{0\leq j\leq 1}\mathcal{T}\uparrow^j = \mathcal{T}\uparrow^0 \cup \{aut^{inter}(v(m),t)\ /\ -4\leq m\leq 4\ \}$$

$$\bigcup_{0\leq j\leq 2}\mathcal{T}\uparrow^j = \bigcup_{0\leq j\leq 1}\mathcal{T}\uparrow^j \cup \{\ aut^{inter}(q,t)\ \} \cup \{\ aut^{inter}(u(m),t)\ /\ -1\leq m\leq 3\ \}$$

$$\bigcup_{0\leq j\leq 3}\mathcal{T}\uparrow^j = \bigcup_{0\leq j\leq 2}\mathcal{T}\uparrow^j \cup \{\ aut^{inter}(w(m),t)\ /\ -1\leq m\leq 3\ \} \cup \{\ aut^{inter}(u(m),s(n))\ /\ -1\leq m\leq 2\ \wedge\ n=1\ \}$$

$$\bigcup_{0\leq j\leq 4}\mathcal{T}\uparrow^j = \bigcup_{0\leq j\leq 3}\mathcal{T}\uparrow^j \cup \{\ aut^{inter}(u(m),s(n))\ /\ -1\leq m\leq 1\ \wedge\ n=2\ \}$$

$$\bigcup_{0\leq j\leq 5}\mathcal{T}\uparrow^j = \bigcup_{0\leq j\leq 4}\mathcal{T}\uparrow^j \cup \{\ aut^{inter}(u(m),s(n))\ /\ -1\leq m\leq 0\ \wedge\ n=3\ \}$$

$$\bigcup_{0\leq j\leq 6}\mathcal{T}\uparrow^j = \bigcup_{0\leq j\leq 5}\mathcal{T}\uparrow^j \cup \{\ aut^{inter}(u(m),s(n))\ /\ m=-1\ \wedge\ n=4\ \}$$

$$\bigcup_{0\leq j\leq 7}\mathcal{T}\uparrow^j = \bigcup_{0\leq j\leq 6}\mathcal{T}\uparrow^j$$

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 $Rec(AUT_4) \subseteq Rec(AUT_5)$. □

Chapitre 6

Propriété de Sûreté et Automate Observateur

6.1 Preuve de Propriété de Sûreté

Nous présentons ici une méthode pour prouver des propriétés de sûreté pour les automates généralisés avec contraintes arithmétiques. De manière informelle, une propriété de sûreté établit que rien de "mauvais" ne va se passer.

Plus formellement, une propriété de sûreté est une propriété qui établit des relations entre les valeurs des variables d'états [SHA 93, LAM 90, BOU 91]. La vérification qu'un automate AUT satisfait une propriété de sûreté ψ peut s'exprimer par :

$$aut(L, S) \Rightarrow \ \psi(S) \qquad (\star)$$

La formule $(\star)$ ci-dessus dit que les états accessibles de l'automate vérifient ψ. La formule $\psi(S)$ peut aussi être vue comme une assertion invariante pour le programme qui représente l'automate (cf. [LAM 90]). Nous nous intéressons ici au cas particulier où ψ est une formule arithmétique linéaire.

La propriété d'absence de blocage, la détection de non déterminisme et des états non accessibles peuvent être exprimées sous une certaine forme de ψ (voir [HIG 94]).

Nous décrivons une méthode de preuve de $(\star)$. Ceci se fait suivant les étapes ci-dessous :

1. Construire le programme Π_{AUT}^{tronc}, obtenu en tronquant l' argument de liste L et en

éliminant les paramètres d'action $\overline{A}$ des clauses de Π_{AUT}.

2. Calculer le résultat de la procédure d'évaluation ascendante pour le programme simplifié Π^{tronc}_{AUT}.

Comme dans le cas d'inclusion de langages, la procédure d'évaluation ascendante de l'étape 2 ne termine pas toujours. Lorqu'elle termine, le résultat $\mathcal{T}\uparrow^{\omega}$ est de la forme :

$$\mathcal{T}\uparrow^{w} = \bigcup_{s(\overline{S})\in\Xi}\{\ aut(s(\overline{S}))\ /\ \alpha_s(\overline{S})\ \text{ vrai}\ \}$$

où α_s est une formule arithmétique linéaire.

Rappelons que $\mathcal{T}\uparrow^{w}$ caractérise l'ensemble des états accessibles.

D'après [FRI 92], $\exists L\ aut(L,X)$ est vrai ssi $aut(X)$ est vrai. On peut donc raisonner sur le programme tronqué Π^{tronc} (sans l'argument de liste), à la place du programme Π. Par conséquent, X satisfait l'hypothèse $\exists L\ aut(L,X)$ de l'implication $aut(L,X) \Rightarrow \psi(X)$ ssi $aut(X) \in \mathcal{T}\uparrow^{w}$. Il s'ensuit que l'implication $aut(L,X) \Rightarrow \psi(X)$ est vraie ssi $\exists s \in \Xi\ \alpha_s(X) \Rightarrow \psi(X)$.

Le problème de la preuve d'une propriété de sûreté ψ est donc décidable quand la procédure d'évaluation ascendante de l'étape 2 termine. La validation de l'implication est en effet décidable, car α_s et ψ sont des formules arithmétiques linéaires.

Cela se produit pour certaines classes de programmes, comme par exemple lorsque les contraintes arithmétiques de AUT 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 90] (qui engendre une formule d'arithmétique linéaire). En conséquence, on a :

Proposition 6.1 *Le problème de la preuve d'un propriété de sûreté ψ pour un automate AUT est décidable lorsque les contraintes arithmétiques des automates ne contiennent que les symboles $=$, $\neq$, $<$.*

Notons que, même dans le cas où certaines contraintes arithmétiques dans AUT contiennent le symbole $+$, comme dans le cas d'inclusion de langages, on peut parfois résoudre la preuve des propriétés de sûreté en utilisant d'autres procédures d'évaluation ascendante.

Remarquons que nous avons exprimé le problème d'inclusion sous la forme $aut^{inter}(L,X_1,X_2) \Rightarrow X_2 \neq s_{erreur}$, ce qui correspond donc à une propriété de sûreté (l'état s_{erreur} n'est jamais atteint).

Nous présentons maintenant un exemple de preuve de propriété de sûreté.

Exemple 6.2 Considérons l'automate AUT, décrit dans [HAL 93] et représenté figure 6.1. Cet automate correspond au programme :

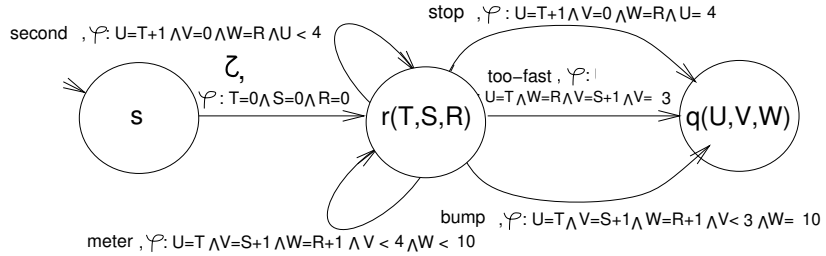


FIG. 6.1 - *Automate AUT*

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

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 :

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

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 pro-

gramme 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 dans la partie II de cette thèse, 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) coïncide avec le résultat obtenu par Halbwachs [HAL 93] (qui, *a priori*, n'est qu'une approximation de ce plus petit point-fixe).

6.2 Automate Observateur

Un *automate observateur* AUT^{obs} (voir, par exemple [DSS 85, GRO 83]) est un automate qui a pour rôle d'enregistrer des informations sur un automate AUT, sans influer sur le comportement de ce dernier, c'est-à-dire, sans restreindre le langage reconnu par AUT. Ceci permet d'étendre le champ d'expression des propriétés relatives au langage reconnu par AUT.

On peut coder un automate observateur AUT^{obs} sous la forme d'un automate généralisé à un seul état paramétré avec un argument Z (où le symbole d'état a été supprimé). Afin de ne pas influer sur le comportement de AUT, l'automate AUT^{obs} reconnaîtra n'importe quelle liste d'actions et enregistrera dans Z des informations sur AUT.

Le modèle d'observateurs que nous proposons dans cette partie est plus simple que celui proposé par Groz et Jard [GRO 83]. Leurs observateurs utilisent tous les moyens d'expression présents dans le langage du programme à vérifier (leurs observateurs peuvent avoir plusieurs états, des transitions, etc.) Pour exprimer les mêmes propriétés que [GRO 83], il suffit de considérer une intersection d'un nombre fini d'observateurs de notre modèle.

La preuve que AUT satisfait une certaine propriété sera exprimée par :

$$aut(L,X) \wedge aut^{obs}(L,Z) \Rightarrow \zeta(X,Z)$$

où AUT^{obs} est un automate observateur et ζ est une formule arithmétique. AUT^{obs} et ζ sont déterminés en fonction de la propriété à prouver.

Par exemple, si on veut prouver que :

> **Au sein d'une liste d'actions reconnues par AUT, il existe au plus 4 occurrences consécutives d'une symbole action c.**
> Alors, l'automate observateur enregistrera dans son argument Z le nombre d'occurrences consécutives de l'action c dans une liste, et $\zeta(X, Z)$ sera $Z \leq 4$.

> **Le dernier symbole d'action d'une liste reconnue par AUT est c.**
> Alors, l'automate observateur enregistrera dans son argument Z le dernier symbole d'actions de liste, et $\zeta(X, Z)$ sera $Z = q$.

La preuve de $aut(L, X) \ \wedge \ aut^{obs}(L, Z) \Rightarrow \ \zeta(X, Z)$ se ramène à la preuve de :

$$aut^{inter}(L, X, Z) \Rightarrow \ \zeta(X, Z)$$

où AUT^{inter} est l'automate résultant de l'intersection de AUT et AUT^{obs}. Notons que cette preuve correspond à la preuve d'une propriété de sûreté pour AUT^{inter} (et donc pour AUT, puisque AUT^{obs} n'influe pas sur AUT).

Notons aussi que l'automate AUT_2^{comp} de la proposition 5.1 joue le rôle d'un automate observateur ($X_2 \neq s_{erreur}$ correspond à la relation arithmétique ζ).

Exemple 6.3 Considérons l'automate AUT_4 de l'exemple 5.3 et un "observateur" qui enregistre le nombre d'occurrences consécutives du symbole d'action c. On prouvera que au sein de toute liste l reconnue par AUT_4, il existe au plus 4 symboles d'actions c consécutifs. La relation $\zeta(X, Z)$ est $Z \leq 4$.

$$
\begin{array}{lll}
aut^{obs}([\,],0) & & \\
aut^{obs}([a(A)|L],0) & :- & aut^{obs}(L,Z) \\
aut^{obs}([b(B)|L],0) & :- & aut^{obs}(L,Z) \\
aut^{obs}([c(C)|L],Z') & :- & Z+1=Z',\quad aut^{obs}(L,Z) \\
aut^{obs}([d(D)|L],0) & :- & aut^{obs}(L,Z) \\
aut^{obs}([e(E)|L],0) & :- & aut^{obs}(L,Z) \\
aut^{obs}([f(F)|L],0) & :- & aut^{obs}(L,Z)
\end{array}
$$

Notre problème se ramène à vérifier si :

$$aut_4(L,X)\ \wedge aut^{obs}(L,Z) \Rightarrow\ Z \leq 4$$

Ceci se fait en trois étapes:

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

$$
\begin{array}{lll}
aut^{inter}([\,],r,0) & & \\
aut^{inter}([a(A)|L],v(V),0) & :- & A=A' \wedge A=V \wedge A \geq -4\ \wedge A \leq 4,\ aut^{inter}(L,p,Z) \\
aut^{inter}([b(B)|L],u(U),0) & :- & B=B' \wedge B=V \wedge B \geq 0 \wedge U=V-1,\ aut^{inter}(L,v(V),Z) \\
aut^{inter}([c(C)|L],u(Y),Z') & :- & C=C' \wedge C=U \wedge U \geq 0 \wedge Z=U-1 \wedge\ Z+1=Z', \\
 & & \qquad aut^{inter}(L,u(U),Z) \\
aut^{inter}([d(D)|L],w(W),0) & :- & D=D' \wedge D=U \wedge U \geq -1 \wedge W=U,\ aut^{inter}(L,u(U),Z) \\
aut^{inter}([e(E)|L],v(V),0) & :- & E=E' \wedge E=W \wedge E+1=V,\ aut^{inter}(L,w(W),Z) \\
aut^{inter}([f(F)|L],q,0) & :- & F=F' \wedge F=V \wedge F<0,\ \ aut^{inter}(L,v(V),Z)
\end{array}
$$

If faut prouver que $aut^{inter}(L,X,Z)\ \Rightarrow\ Z \leq 4$.

2. Elimination de l'argument de liste L et élimination des paramètres des actions du programme Π^{inter}, engendrant le programme Π^{tronc} :

$$
\begin{array}{lll}
aut^{inter}(r,0) & & \\
aut^{inter}(v(V),0) & :- & V \geq -4\ \wedge V \leq 4,\quad aut^{inter}(p,Z) \\
aut^{inter}(u(U),0) & :- & V \geq 0 \wedge U=V-1,\quad aut^{inter}(v(V),Z) \\
aut^{inter}(u(Y),Z') & :- & U \geq 0 \wedge Y=U-1 \wedge Z+1=Z',\quad aut^{inter}(u(U),Z) \\
aut^{inter}(w(W),0) & :- & U \geq -1 \wedge W=U,\quad aut^{inter}(u(U),Z) \\
aut^{inter}(v(V),0) & :- & W+1=V,\quad aut^{inter}(w(W),Z) \\
aut^{inter}(q,0) & :- & V<0,\quad aut^{inter}(v(V),Z)
\end{array}
$$

If faut prouver que $aut^{inter}(X,Z) \Rightarrow Z \leq 4$.

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

$$\mathcal{T}\uparrow^0 = \{\ aut^{inter}(r,0)\ \}$$

$$\bigcup_{0\leq j\leq 1}\mathcal{T}\uparrow^j = \mathcal{T}\uparrow^0 \cup \{aut^{inter}(v(m),0) /\ -4 \leq m \leq 4\ \}$$

$$\bigcup_{0\leq j\leq 2}\mathcal{T}\uparrow^j = \bigcup_{0\leq j\leq 1}\mathcal{T}\uparrow^j \cup \{\ aut^{inter}(q,0)\ \} \cup \{\ aut^{inter}(u(m),0) /\ -1 \leq m \leq 3\ \}$$

$$\bigcup_{0\leq j\leq 3}\mathcal{T}\uparrow^j = \bigcup_{0\leq j\leq 2}\mathcal{T}\uparrow^j \cup \{\ aut^{inter}(w(m),0) /\ -1 \leq m \leq 3\ \} \cup \{\ aut^{inter}(u(m),n) /\ -1 \leq m \leq 2 \wedge n = 1\ \}$$

$$\bigcup_{0\leq j\leq 4}\mathcal{T}\uparrow^j = \bigcup_{0\leq j\leq 3}\mathcal{T}\uparrow^j \cup \{\ aut^{inter}(u(m),n) /\ -1 \leq m \leq 1 \wedge n = 2\ \}$$

$$\bigcup_{0\leq j\leq 5}\mathcal{T}\uparrow^j = \bigcup_{0\leq j\leq 4}\mathcal{T}\uparrow^j \cup \{\ aut^{inter}(u(m),n) /\ -1 \leq m \leq 0 \wedge n = 3\ \}$$

$$\bigcup_{0\leq j\leq 6}\mathcal{T}\uparrow^j = \bigcup_{0\leq j\leq 5}\mathcal{T}\uparrow^j \cup \{\ aut^{inter}(u(m),n) / m = -1 \wedge n = 4\ \}$$

$$\bigcup_{0\leq j\leq 7}\mathcal{T}\uparrow^j = \bigcup_{0\leq j\leq 6}\mathcal{T}\uparrow^j$$

Nous avons :

$$\mathcal{T}\uparrow^w = \{\ aut^{inter}(r,0)\ \} \cup \{\ aut^{inter}(q,0)\} \cup \{\ aut^{inter}(v(m),0) /\ -4 \leq m \leq 4\ \} \cup \{\ aut^{inter}(w(m),0) / -1 \leq m \leq 3\ \} \cup \{\ aut^{inter}(u(m),n) / -1 \leq m \leq (3-n) \wedge 0 \leq n \leq 4\ \}$$

On voit que $aut^{inter}(X,Z) \in \mathcal{T}\uparrow^w$ ssi $(Z = 0 \wedge (\ X = p \vee X = q \vee \exists m(X = v(m) \wedge -4 \leq m \leq 4) \vee \exists m(X = w(m) \wedge -1 \leq m \leq 3)\ )\ ) \vee \exists m(0 \leq Z \leq 4 \wedge X = u(m) \wedge -1 \leq m \leq (3-Z)\ )$.

Il s'ensuit que $aut^{inter}(X,Z) \Rightarrow Z \leq 4$. □

Chapitre 7

Le Protocole du Bit Alterné

Dans cette partie, nous appliquons l'opération de composition parallèle d'automate en vue de spécifier le *Protocole du Bit Alterné*, et nous expliquons comment utiliser la procédure de décision de la partie 6 pour démontrer des propriétés de ce protocole. La description du protocole est reprise essentiellement de [GAR 89].

Le Protocole du Bit Alterné est un processus composé de quatre entités: $SENDER$, $RECEIVER$, $MEDIUM_1$ et $MEDIUM_2$. Chaque entité peut être caractérisée par un automate généralisé. Le tableau ci-dessous (voir [GAR 89]) donne les actions qui sont exécutées (et partagées) par ces entités.

Actions	Origine	Destination	Description
$put(M)$	environnement	$SENDER$	acquisition d'un message de l'environnement
$sdt(M,B)$	$SENDER$	$MEDIUM_1$	envoi d'un message
$rdt(M,B)$	$MEDIUM_1$	$RECEIVER$	transmission d'un message
$rdte$	$MEDIUM_1$	$RECEIVER$	perte d'un message
$get(M)$	$RECEIVER$	environnement	émission d'un message vers l'environnement
$sack(B)$	$MEDIUM_2$	$SENDER$	transmission de l'acquittement
$sacke$	$MEDIUM_2$	$SENDER$	perte de l' acquittement
$rack(B)$	$RECEIVER$	$MEDIUM_1$	renvoi d'un acquittement

Le protocole du bit alterné peut être regardé comme le produit de la composition parallèle des entités $SENDER$, $RECEIVER$, $MEDIUM_1$ et $MEDIUM_2$, ainsi que le représente la figure 7.1 (voir [GAR 89]).

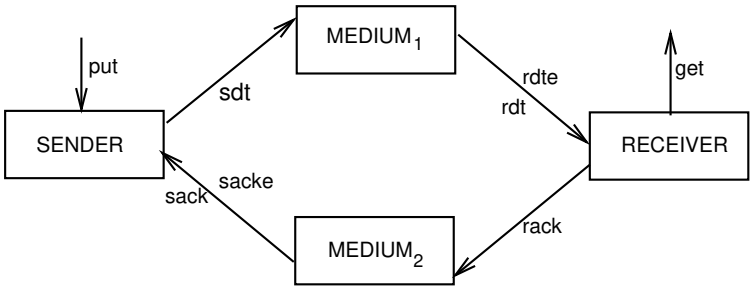


FIG. 7.1 - *Protocole du Bit Alterné*

7.1 Processus $SENDER$

Le processus $SENDER$ acquiert un message (via put), et envoie ce message après l'ajout d'un bit de contrôle B (via sdt). Ce bit est initialisé avec la valeur 0, et est incrémenté (modulo 2) avant chaque action put.

Le processus $SENDER$ renvoie un message dans les situations suivantes:

- $SENDER$ reçoit un avis de perte d'acquittement $sacke$.
- Après l'écoulement d'un certain temps sans exécution d'actions (ce qu'on représente par l'exécution de l'action τ), $SENDER$ renvoie spontanément le message.

Ce processus est caractérisé par le programme logique Π_{SENDER}:

$$
\begin{array}{lcl}
sender([\,], initsend(0)) & & \\
sender([put(M)|L], readysend(M,B)) & \leftarrow & sender(L, initsend(B)) \\
sender([sdt(M,B)|L], readysack(M,B)) & \leftarrow & sender(L, readysend(M,B)) \\
sender([sack(1)|L], readysack(M,0)) & \leftarrow & sender(L, readysack(M,0)) \\
sender([sack(0)|L], readysack(M,1)) & \leftarrow & sender(L, readysack(M,1)) \\
sender([sacke|L], readysend(M,B)) & \leftarrow & sender(L, readysack(M,B)) \\
sender([\tau|L], readysend(M,B)) & \leftarrow & sender(L, readysack(M,B)) \\
sender([sack(0)|L], initsend(1)) & \leftarrow & sender(L, readysack(N,0)) \\
sender([sack(1)|L], initsend(0)) & \leftarrow & sender(L, readysack(N,1))
\end{array}
$$

Noter que, par souci de concision, la forme des clauses du programme Π_{SENDER} a été légèrement simplifiée. Par exemple, la clause:

$$sender([put(M)|L], readysend(M',B)) \leftarrow M = M', B = B',\ sender(L, initsend(B'))$$

a été écrite sous la forme:

$sender([put(M)|L], readysend(M, B)) \leftarrow sender(L, initsend(B))$

L'automate AUT_{SENDER} est représenté figure 7.2.

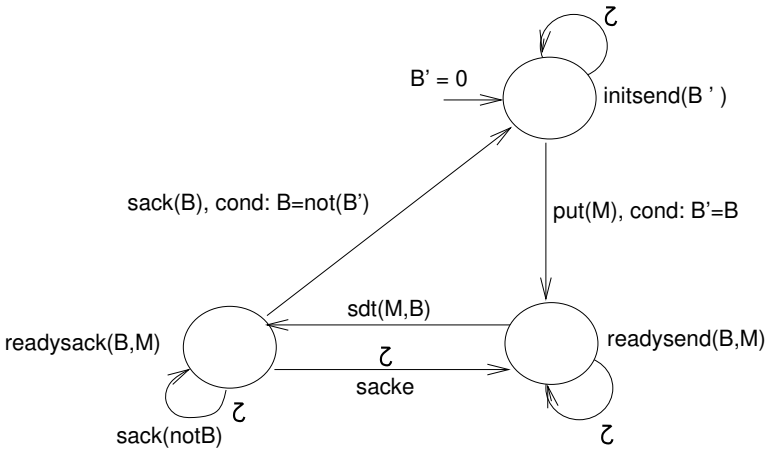


FIG. 7.2 - *Processus SENDER*

7.2 Processus $MEDIUM_1$

Le processus $MEDIUM_1$ reçoit un message (via *sdt*) avec un bit de contrôle B. Il peut se comporter alors de trois manières différentes:

- Transmission du message et du bit de contrôle, sans changement de leurs valeurs. Dans ce cas, l'action $rdt(M, B)$ est exécutée.
- Perte du message et envoi d'un avis de perte. Dans ce cas, l'action *rdte* est exécutée.
- Perte du message sans envoi d'avis de perte. Cela correspond à l'exécution de l'action silencieuse τ.

Ce processus est représenté par le programme logique Π_{MEDIUM_1}:

$$
\begin{array}{lcl}
medium_1([\,], initm1) & & \\
medium_1([sdt(M,B)|L], finalm1(M,B)) & \leftarrow & medium_1(L, initm1) \\
medium_1([rdt(M,B)|L], initm1) & \leftarrow & medium_1(L, finalm1(M,B)) \\
medium_1([rdte|L], initm1) & \leftarrow & medium_1(L, finalm1(M,B)) \\
medium_1([\tau|L], initm1) & \leftarrow & medium_1(L, finalm1(M,B))
\end{array}
$$

L'automate AUT_{MEDIUM_1} est représenté figure 7.3.

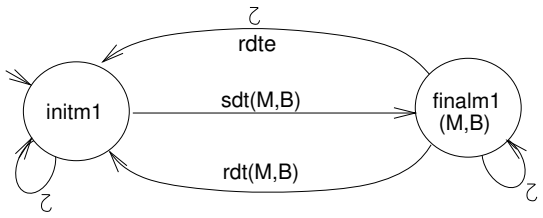


FIG. 7.3 - *Processus* $MEDIUM_1$

7.3 Processus $MEDIUM_2$

Le comportement de $MEDIUM_2$ est analogue à celui de $MEDIUM_1$. La seule différence réside dans le nom des actions qui sont exécutées.

Ce processus est caractérisé par le programme Π_{MEDIUM_2}:

$$
\begin{array}{lcl}
medium_2([\,], initm2) & & \\
medium_2([rack(B)|L], finalm2(B)) & \leftarrow & medium_2(L, initm2) \\
medium_2([sack(B)|L], initm2) & \leftarrow & medium_2(L, finalm2(B)) \\
medium_2([sacke|L], initm2) & \leftarrow & medium_2(L, finalm2(B)) \\
medium_2([\tau|L], initm2) & \leftarrow & medium_2(L, finalm2(B))
\end{array}
$$

L'automate AUT_{MEDIUM_2} est représenté figure 7.4.

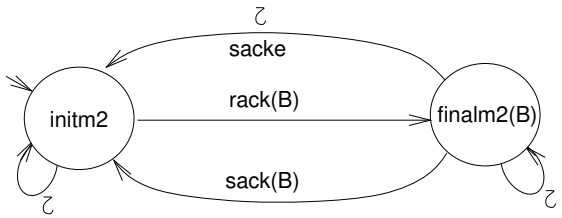


FIG. 7.4 - *Processus* $MEDIUM_2$

7.4 Processus *RECEIVER*

Le processus $RECEIVER$ se comporte comme suit:

- S'il reçoit un message (via rdt) avec un bit de contrôle B correct, il transmet le message à l'environnement (via get) et un bit de contrôle est renvoyé avec la valeur B (via $rack$).
- S'il reçoit un message (via rdt) avec un bit B incorrect, le récepteur attend qu'un autre message soit reçu.
- Si une perte de message se produit (via $rdte$), le processus renvoie un acquittement incorrect (ayant le complément de B comme bit de contrôle) via $rack$.
- Après l'écoulement d'un certain temps sans réception (ce qui est représenté par l'exécution de l'action τ), le processus renvoie un acquittement incorrect via $rack$.

Le processus est caractérisé par le programme $\Pi_{RECEIVER}$:

$$
\begin{array}{lcl}
receiver([\,], initreceive(B)) & & \\
receiver([rdt(M,B)|L], readyget(M,B)) & \leftarrow & receiver(L, initreceive(B)) \\
receiver([get(M)|L], readyrackcorrect(B)) & \leftarrow & receiver(L, readyget(M,B)) \\
receiver([rack(0)|L], initreceive(1)) & \leftarrow & receiver(L, readyrackcorrect(0)) \\
receiver([rack(1)|L], initreceive(0)) & \leftarrow & receiver(L, readyrackcorrect(1)) \\
receiver([rdt(M,1)|L], initreceive(0)) & \leftarrow & receiver(L, initreceive(0)) \\
receiver([rdt(M,0)|L], initreceive(1)) & \leftarrow & receiver(L, initreceive(1)) \\
receiver([rdte|L], readyrackincorrect(B)) & \leftarrow & receiver(L, initreceive(B)) \\
receiver([\tau|L], initreceive(B)) & \leftarrow & receiver(L, readyrack(B)) \\
receiver([rack(1)|L], initreceive(0)) & \leftarrow & receiver(L, readyrackincorrect(0)) \\
receiver([rack(0)|L], initreceive(1)) & \leftarrow & receiver(L, readyrackincorrect(1))
\end{array}
$$

L'automate $AUT_{RECEIVER}$ est représenté figure 7.5.

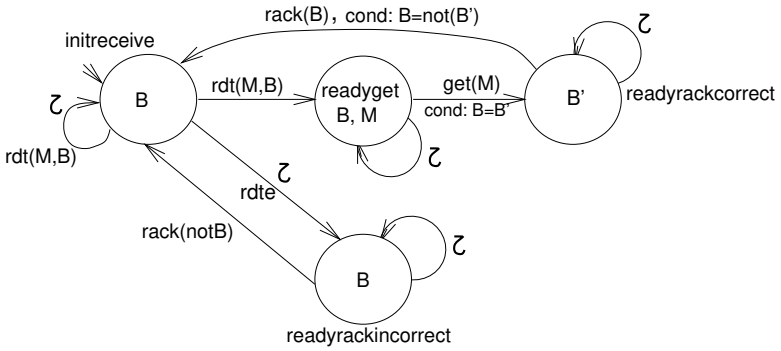


FIG. 7.5 - *Processus RECEIVER*

7.5 Composition parallèle

L'automate généralisé AUT_{BIT} associé au Protocole du Bit Alterné s'obtient en appliquant les opérations de composition parallèle aux automates associés à $SENDER$, $RECEIVER$, $MEDIUM_1$ et $MEDIUM_2$. Les actions synchronisées sont celles qui sont partagées par les différentes entités (voir figure 7.1).

On engendre ainsi le programme-composition Π_{BIT}:

$$
\begin{array}{lcl}
bit([\,], initsend(0), initm1, initreceive(0), initm2) & & \\
bit([put(M)|L], readysend(M,B), Q, R, S) & \leftarrow & bit(L, initsend(B), Q, R, S) \\
bit([\tau|L], readysack(M,0), Q, R, initm2) & \leftarrow & bit(L, readysack(M,0), Q, R, finalm2(1)) \\
bit([\tau|L], readysack(M,1), Q, R, initm2) & \leftarrow & bit(L, readysack(M,1), Q, R, finalm2(0)) \\
bit([\tau|L], readysend(M,B), Q, R, initm2) & \leftarrow & bit(L, readysack(M,B), Q, R, finalm2(B1)) \\
bit([\tau|L], initsend(1), Q, R, initm2) & \leftarrow & bit(L, readysack(M,0), Q, R, finalm2(0)) \\
bit([\tau|L], initsend(0), Q, R, initm2) & \leftarrow & bit(L, readysack(M,1), Q, R, finalm2(1)) \\
bit([\tau|L], P, initm1, readyget(M,B), S) & \leftarrow & bit(L, P, finalm1(M,B), initreceive(B), S) \\
bit([\tau|L], P, initm1, initreceive(0), S) & \leftarrow & bit(L, P, finalm1(M,1), initreceive(0), S) \\
bit([\tau|L], P, initm1, initreceive(1), S) & \leftarrow & bit(L, P, finalm1(M,0), initreceive(1), S) \\
bit([\tau|L], P, initm1, readyrackincorrect(B), S) & \leftarrow & bit(L, P, finalm1(M,B1), initreceive(B), S) \\
bit([get(M)|L], P, Q, readyrackcorrect(B), S) & \leftarrow & bit(L, P, Q, readyget(M,B), S) \\
bit([\tau|L], P, Q, initreceive(1), finalm2(0)) & \leftarrow & bit(L, P, Q, readyrackcorrect(0), initm2) \\
bit([\tau|L], P, Q, initreceive(0), finalm2(1)) & \leftarrow & bit(L, P, Q, readyrackcorrect(1), initm2) \\
bit([\tau|L], P, Q, initreceive(0), finalm2(1)) & \leftarrow & bit(L, P, Q, readyrackincorrect(0), initm2) \\
bit([\tau|L], P, Q, initreceive(1), finalm2(0)) & \leftarrow & bit(L, P, Q, readyrackincorrect(1), initm2) \\
bit([\tau|L], P, initm1, R, S) & \leftarrow & bit(L, P, finalm1(M,B1), R, S) \\
bit([\tau|L], readysend(M,B), Q, R, S) & \leftarrow & bit(L, readysack(M,B), Q, R, S) \\
bit([\tau|L], P, Q, R, initm2) & \leftarrow & bit(L, P, Q, R, finalm2(B1)) \\
bit([\tau|L], P, Q, readyrackincorrect(B), S) & \leftarrow & bit(L, P, Q, initreceive(B), S) \\
bit([\tau|L], readysack(M,B), finalm1(M,B), R, S) & \leftarrow & bit(L, readysend(M,B), initm1, R, S)
\end{array}
$$

Nous appliquons maintenant la procédure de décision du chapitre 6 pour démontrer la propriété $\psi(l)$:

> Au sein d'une liste d'actions l, il est impossible de trouver une action $put(\overline{m}_1)$ qui soit suivie d'une action $get(\overline{m}_2)$ telle que $\overline{m}_1 \neq \overline{m}_2$.

On peut facilement construire un automate AUT_ψ tel que:

$$l \in Rec(AUT_\psi) \Leftrightarrow \psi(l)$$

Cet automate est représenté figure 7.6, et est caractérisé par le programme:

$$
\begin{array}{lcl}
aut_\psi([\,], s_0(N)) & & \\
aut_\psi([put(M)|L], s_1(P)) & \leftarrow & M = P,\ aut_\psi(L, s_0(N)) \\
aut_\psi([get(M)|L], s_0(P)) & \leftarrow & M = N,\ aut_\psi(L, s_1(N)) \\
aut_\psi([\tau|L], s_1(P)) & \leftarrow & aut_\psi(L, s_1(N)) \\
aut_\psi([get(M)|L], s_0(P)) & \leftarrow & aut_\psi(L, s_0(N)) \\
aut_\psi([\tau|L], s_0(P)) & \leftarrow & aut_\psi(L, s_0(N))
\end{array}
$$

La preuve de ce que toute suite d'actions exécutée par le Protocole du Bit Alterné vérifie ψ se ramène à la preuve de: $AUT_{BIT} \subseteq AUT_\psi$.

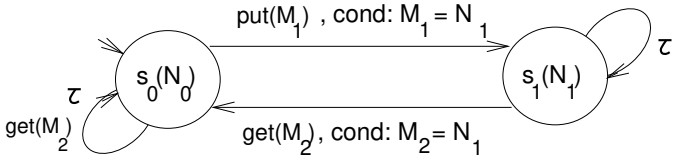


FIG. 7.6 - *Propriété ψ*

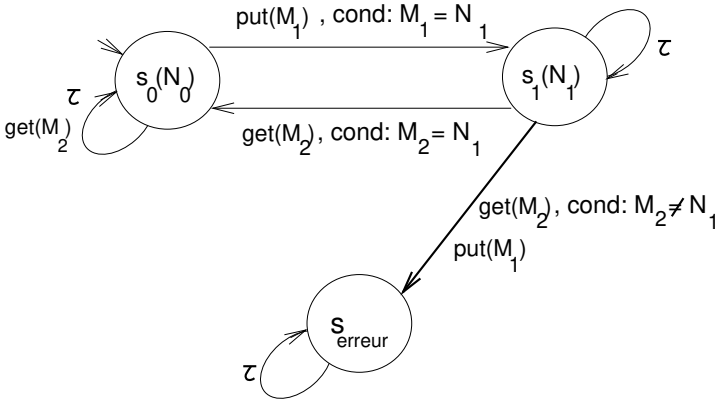


FIG. 7.7 - *Automate AUT_{ψ}^{comp}*

D'abord, il faut calculer l'automate complété.

$$
\begin{array}{lcl}
aut_{\psi}([\,], s_0(N)) & & \\
aut_{\psi}([put(M)|L], s_1(P)) & \leftarrow & M = P,\ aut_{\psi}(L, s_0(N)) \\
aut_{\psi}([get(M)|L], s_0(P)) & \leftarrow & M = N,\ aut_{\psi}(L, s_1(N)) \\
aut_{\psi}([\tau|L], s_1(P)) & \leftarrow & aut_{\psi}(L, s_1(N)) \\
aut_{\psi}([get(M)|L], s_0(P)) & \leftarrow & aut_{\psi}(L, s_0(N)) \\
aut_{\psi}([\tau|L], s_0(P)) & \leftarrow & aut_{\psi}(L, s_0(N)) \\
aut_{\psi}([get(M)|L], s_{erreur}) & \leftarrow & M \neq N,\ aut_{\psi}(L, s_1(N)) \\
aut_{\psi}([put(M)|L], s_{erreur}) & \leftarrow & aut_{\psi}(L, s_1(N)) \\
aut_{\psi}([\tau|L], s_{erreur}) & \leftarrow & aut_{\psi}(L, s_{erreur})
\end{array}
$$

L'étape suivante consiste à déterminer l'automate $AUT_{BIT} \wedge AUT_{\psi}^{comp}$. (Noter que les états s_i et les paramètres M et N de l'automate AUT_{ψ}^{comp} sont respectivement renommés en s'_i, M' et N'.)

$$
\begin{array}{lcl}
aut^{inter}([\,], initsend(0), initm1, initreceive(0), initm2, s'_0(N')) & & \\
aut^{inter}([put(M)|L], readysend(M,B), Q, R, S, s'_1(P')) & \leftarrow & M = P', aut^{inter}(L, initsend(B), Q, R, S, s'_0(N')) \\
aut^{inter}([put(M)|L], readysend(M,B), Q, R, S, s'_{erreur}) & \leftarrow & aut^{inter}(L, initsend(B), Q, R, S, s'_1(N')) \\
aut^{inter}([get(M)|L], P, Q, readyrackcorrect(B), S, s'_0(P')) & \leftarrow & M = N', aut^{inter}(L, P, Q, readyget(M,B), S, s'_1(N')) \\
aut^{inter}([get(M)|L], P, Q, readyrackcorrect(B), S, s'_0(P')) & \leftarrow & aut^{inter}(L, P, Q, readyget(M,B), S, s'_0(N')) \\
aut^{inter}([get(M)|L], P, Q, readyrackcorrect(B), S, s'_{erreur}) & \leftarrow & M \neq N', aut^{inter}(L, P, Q, readyget(M,B), S, s_1(N')) \\
aut^{inter}([\tau|L], readysack(M,0), Q, R, initm2, S') & \leftarrow & aut^{inter}(L, readysack(M,0), Q, R, finalm2(1), S') \\
aut^{inter}([\tau|L], readysack(M,1), Q, R, initm2, S') & \leftarrow & aut^{inter}(L, readysack(M,1), Q, R, finalm2(0), S') \\
aut^{inter}([\tau|L], readysend(M,B), Q, R, initm2, S') & \leftarrow & aut^{inter}(L, readysack(M,B), Q, R, finalm2(B1), S') \\
aut^{inter}([\tau|L], initsend(1), Q, R, initm2, S') & \leftarrow & aut^{inter}(L, readysack(M,0), Q, R, finalm2(0), S') \\
aut^{inter}([\tau|L], initsend(0), Q, R, initm2, S') & \leftarrow & aut^{inter}(L, readysack(M,1), Q, R, finalm2(1), S') \\
aut^{inter}([\tau|L], P, initm1, readyget(M,B), S, S') & \leftarrow & aut^{inter}(L, P, finalm1(M,B), initreceive(B), S, S') \\
aut^{inter}([\tau|L], P, initm1, initreceive(0), S, S') & \leftarrow & aut^{inter}(L, P, finalm1(M,1), initreceive(0), S, S') \\
aut^{inter}([\tau|L], P, initm1, initreceive(1), S, S') & \leftarrow & aut^{inter}(L, P, finalm1(M,0), initreceive(1), S, S') \\
aut^{inter}([\tau|L], P, initm1, readyrackincorrect(B), S, S') & \leftarrow & aut^{inter}(L, P, finalm1(M,B1), initreceive(B), S, S') \\
aut^{inter}([\tau|L], P, Q, initreceive(1), finalm2(0), S') & \leftarrow & aut^{inter}(L, P, Q, readyrackcorrect(0), initm2, S') \\
aut^{inter}([\tau|L], P, Q, initreceive(0), finalm2(1), S') & \leftarrow & aut^{inter}(L, P, Q, readyrackcorrect(1), initm2, S') \\
aut^{inter}([\tau|L], P, Q, initreceive(0), finalm2(1), S') & \leftarrow & aut^{inter}(L, P, Q, readyrackincorrect(0), initm2, S') \\
aut^{inter}([\tau|L], P, Q, initreceive(1), finalm2(0), S') & \leftarrow & aut^{inter}(L, P, Q, readyrackincorrect(1), initm2, S') \\
aut^{inter}([\tau|L], P, initm1, R, S, S') & \leftarrow & aut^{inter}(L, P, finalm1(M,B1), R, S, S') \\
aut^{inter}([\tau|L], readysend(M,B), Q, R, S, S') & \leftarrow & aut^{inter}(L, readysack(M,B), Q, R, S, S') \\
aut^{inter}([\tau|L], P, Q, R, initm2, S') & \leftarrow & aut^{inter}(L, P, Q, R, finalm2(B1), S') \\
aut^{inter}([\tau|L], P, Q, readyrackincorrect(B), S, S') & \leftarrow & aut^{inter}(L, P, Q, initreceive(B), S, S') \\
aut^{inter}([\tau|L], readysack(M,B), finalm1(M,B), R, S, S') & \leftarrow & aut^{inter}(L, readysend(M,B), initm1, R, S, S')
\end{array}
$$

Après avoir éliminé la liste d'actions L, nous engendrons le programme suivant:

$$
\begin{array}{lcl}
aut^{inter}(initsend(0), initm1, initreceive(0), initm2, s'_0(N')) & & \\
aut^{inter}(readysend(M,B), Q, R, S, s'_1(P')) & \leftarrow & M = P', aut^{inter}(initsend(B), Q, R, S, s'_0(N')) \\
aut^{inter}(readysend(M,B), Q, R, S, s'_{erreur}) & \leftarrow & aut^{inter}(initsend(B), Q, R, S, s'_1(N')) \\
aut^{inter}(P, Q, readyrackcorrect(B), S, s'_0(P')) & \leftarrow & M = N', aut^{inter}(P, Q, readyget(M,B), S, s'_1(N')) \\
aut^{inter}(P, Q, readyrackcorrect(B), S, s'_0(P')) & \leftarrow & aut^{inter}(P, Q, readyget(M,B), S, s'_0(N')) \\
aut^{inter}(P, Q, readyrackcorrect(B), S, s'_{erreur}) & \leftarrow & M \neq N', aut^{inter}(P, Q, readyget(M,B), S, s_1(N')) \\
aut^{inter}(readysack(M,0), Q, R, initm2, S') & \leftarrow & aut^{inter}(readysack(M,0), Q, R, finalm2(1), S') \\
aut^{inter}(readysack(M,1), Q, R, initm2, S') & \leftarrow & aut^{inter}(readysack(M,1), Q, R, finalm2(0), S') \\
aut^{inter}(readysend(M,B), Q, R, initm2, S') & \leftarrow & aut^{inter}(readysack(M,B), Q, R, finalm2(B1), S') \\
aut^{inter}(initsend(1), Q, R, initm2, S') & \leftarrow & aut^{inter}(readysack(M,0), Q, R, finalm2(0), S') \\
aut^{inter}(initsend(0), Q, R, initm2, S') & \leftarrow & aut^{inter}(readysack(M,1), Q, R, finalm2(1), S') \\
aut^{inter}(P, initm1, readyget(M,B), S, S') & \leftarrow & aut^{inter}(P, finalm1(M,B), initreceive(B), S, S') \\
aut^{inter}(P, initm1, initreceive(0), S, S') & \leftarrow & aut^{inter}(P, finalm1(M,1), initreceive(0), S, S') \\
aut^{inter}(P, initm1, initreceive(1), S, S') & \leftarrow & aut^{inter}(P, finalm1(M,0), initreceive(1), S, S') \\
aut^{inter}(P, initm1, readyrackincorrect(B), S, S') & \leftarrow & aut^{inter}(P, finalm1(M,B1), initreceive(B), S, S') \\
aut^{inter}(P, Q, initreceive(1), finalm2(0), S') & \leftarrow & aut^{inter}(P, Q, readyrackcorrect(0), initm2, S') \\
aut^{inter}(P, Q, initreceive(0), finalm2(1), S') & \leftarrow & aut^{inter}(P, Q, readyrackcorrect(1), initm2, S') \\
aut^{inter}(P, Q, initreceive(0), finalm2(1), S') & \leftarrow & aut^{inter}(P, Q, readyrackincorrect(0), initm2, S') \\
aut^{inter}(P, Q, initreceive(1), finalm2(0), S') & \leftarrow & aut^{inter}(P, Q, readyrackincorrect(1), initm2, S') \\
aut^{inter}(P, initm1, R, S, S') & \leftarrow & aut^{inter}(P, finalm1(M,B1), R, S, S') \\
aut^{inter}(readysend(M,B), Q, R, S, S') & \leftarrow & aut^{inter}(readysack(M,B), Q, R, S, S') \\
aut^{inter}(P, Q, R, initm2, S') & \leftarrow & aut^{inter}(P, Q, R, finalm2(B1), S') \\
aut^{inter}(P, Q, readyrackincorrect(B), S, S') & \leftarrow & aut^{inter}(P, Q, initreceive(B), S, S') \\
aut^{inter}(readysack(M,B), finalm1(M,B), R, S, S') & \leftarrow & aut^{inter}(readysend(M,B), initm1, R, S, S')
\end{array}
$$

On peut faire l'évaluation ascendante de ce programme par la méthode de Revesz, ce

qui donne:

$$\mathcal{T}\uparrow^{\omega}_{2} = \emptyset$$

et on achève la démonstration de la propriété $\psi(l)$. On peut de la même façon démontrer les propriétés suivantes:

- Au sein d'une liste d'actions l, il est impossible de trouver une action $put(\overline{m}_1)$ qui soit suivie d'une action $put(\overline{m}_2)$.
- Au sein d'une liste d'actions l, il est impossible de trouver une action $get(\overline{m}_1)$ qui soit suivie d'une action $get(\overline{m}_2)$.

Chapitre 8

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.

Deuxième partie

Evaluation Ascendante de Programmes à Contraintes Arithmétiques

Chapitre 1

Motivation

Il existe deux approches classiques pour raisonner avec des programmes logiques. L'une se fait par "évaluation descendante" : elle tente de répondre à une requête en procédant par chaînage arrière. L'autre est dite par "évaluation ascendante" : elle engendre les conséquences atomiques d'un programme par chaînage avant. La première approche est utilisée plutôt en Programmation Logique (résolution SLD) [ARN 93], tandis que la deuxième est utilisée plutôt en base de données déductives pour faire des requêtes à des programmes Datalog (c'est-à-dire, à des programmes logiques avec des constantes et des variables, mais sans symbole de fonctions) [BAN 86b, BID 92].

Nous décrivons dans cette deuxième partie de la thèse une méthode pour calculer le résultat de l'évaluation ascendante de programmes Datalog (ou programmes logiques), dont les variables sont interprétées sur le domaine des entiers relatifs. Notre motivation initiale était d'utiliser ce processus d'évaluation ascendante pour prouver des formules qui combinaient des listes avec de l'arithmétique linéaire [FRI 92]. Illustrons ce point sur un exemple qui correspond à une adaptation de l'algorithme de Majorité de Boyer-Moore [MIS 82]. Cet algorithme détermine l'élément, qui apparaît plus d'une fois sur deux dans une liste, si cet élément existe.

Soit $p(L, v, w, x, y, z)$ un atome où L est une liste d'entiers, y est la taille de la liste, v le candidat courant à être l'élément majoritaire, x est le nombre d'occurrences de v dans L, et z est le nombre d'occurrences d'un élément quelconque w dans la liste L. Le prédicat p correspondant à l'algorithme de Boyer-Moore peut être défini récursivement par le programme logique :

$$
\begin{array}{lll}
p([\,], v, w, x, y, z) & :- & x = 0,\ y = 0,\ z = 0 \\
p([u|L], v, w, x + 1, y + 1, z) & :- & u = v, u \neq w,\ p(L, v, w, x, y, z) \\
p([u|L], v, w, x, y + 1, z + 1) & :- & u \neq v, u = w, 2x > y,\ p(L, v, w, x, y, z) \\
p([u|L], v, w, x, y + 1, z) & :- & u \neq v, u \neq w, 2x > y,\ p(L, v, w, x, y, z)
\end{array}
$$

Supposons que nous voulons prouver la formule :

$$\forall\ L, v, w, x, y, z\ \ p(L, v, w, x, y, z)\ \Rightarrow\ (v \neq w\ \Rightarrow\ z \leq y\ div\ 2)$$

Il est difficile de prouver cette formule directement (par exemple, par induction), mais cella devient facile, une fois que le lemme $p(L, v, w, x, y, z) \wedge v \neq w \Rightarrow 2x \geq y \wedge\ y \geq z + x \wedge x \geq 0$ est disponible[1]. En principe, ce lemme n'est pas connu mais il peut être engendré automatiquement en appliquant une méthode d'évaluation ascendante au programme simplifié suivant Π' (obtenu à partir de Π en tronquant la liste L et son argument u, et en faisant "remonter" la contrainte $v \neq w$)

$$
\begin{array}{lll}
p'(v, w, y, z) & :- & x = 0,\ y = 0,\ z = 0 \\
p'(v, w, x + 1, y + 1, z) & :- & p'(v, w, x, y, z) \\
p'(v, w, x, y + 1, z + 1) & :- & 2x > y,\ p'(v, w, x, y, z) \\
p'(v, w, x, y + 1, z) & :- & 2x > y,\ p'(v, w, x, y, z)
\end{array}
$$

Le processus d'évaluation ascendante que nous décrivons dans cette thèse appliqué à ce programme engendre la relation $2x \geq y \wedge\ y \geq z + x \wedge x \geq 0$. La formule $p(L, v, w, x, y, z) \Longrightarrow (v \neq w \Longrightarrow z \leq y\ div\ 2)$ découle directement du résultat de l'évaluation de Π', et du fait que, par construction, $p(L, v, w, x, y, z)\ \wedge v \neq w$ implique $p'(v, w, x, y, z)$.

L'évaluation ascendante de programmes logiques interprétés sur le domaine des entiers relatifs fournit aussi un outil puissant pour prouver la terminaison de programmes logiques [BRO 91, GEL 90]. D'un point de vue plus général, l'étude du processus de l'évaluation ascendante reste toujours un domaine intéressant de recherche en lui même, ayant été étudié sous forme extensive ces dernières années. L'une des raisons de cet intérêt est liée à l'apparition du domaine de la programmation logique avec contraintes et du domaine des langages de requêtes avec contraintes (e.g. [KUP 90, KAN 90, KAB 90, LAS 90, LEV 92, MEY 91, REV 90, SRI 92b]). Une autre raison est liée à l'usage des entiers relatifs pour représenter des données sur le temps et l'usage de l'évaluation ascendante pour répondre à des requêtes temporelles [BAU 89, BAU 91, CHO 88].

1. car il suffit alors de montrer $2x \geq y \wedge\ y \geq z + x \wedge x \geq 0 \Rightarrow\ z \leq y\ div\ 2$, ce qui est vrai.

Les résultats figurant dans cette seconde partie ont été publiés dans deux rapports [FRI 92b, FRI 94], et de deux communications [FRI 94b, FRI 94e].

Chapitre 2

Comparaison avec des Travaux Récents

Des procédures d'évaluation ascendante pour des programmes logiques interprétés sur le domaine des entiers relatifs ont été développées d'une part, par des chercheurs de la communauté des base de données déductives [BAU 91, CHO 88, REV 90] et d'autre part, par des chercheurs intéressés par la preuve automatique de la terminaison de programmes en Prolog [BRO 91, GEL 90].

Revesz a élaboré une procédure qui se termine toujours pour une classe de programmes logiques interprétés sur les entiers relatifs, mais les arguments récursifs des programmes dans cette classe ne peuvent pas être incrémentés. Chomicki and Imielinski ont considéré une classe qui est utile pour modéliser le temps en base de données temporelles : le temps est représenté par un argument numérique (argument temporel), qui est décrémenté à chaque appel récursif du prédicat. La classe de programmes traitée par Chomicki and Imielinski ne permet qu'un seul argument temporel [CHO 88]. Baudinet, Niézette and Wolper considèrent une classe de programmes logiques avec plusieurs arguments temporels, mais leur procédure n'est pas assurée de terminer. En plus, les tuples dans la base de données extentionnelle (c'est-à-dire, les tuples qui satisfont la règle non récursive) sont des formules arithmétiques avec une forme restreinte (points linéaires répétés) [BAU 91].

Comme dans Baudinet, Niézette and Wolper [BAU 91], nos programmes admettent des incrémentations dans plusieurs arguments récursifs. A la différence de [BAU 91], nos programmes admettent n'importe quelle formule arithmétique pour la base de données extensives. Nous décrivons une classe de programmes pour laquelle notre procédure d'évaluation

ascendante est assurée de terminer (classe de programmes avec trois règles récursives et une inéquation par règle) .

L'idée d'engendrer automatiquement des inéquations linéaires sur les variables du programme a été exploitée pour des langages de programmations impératives par [COU 78], utilisant des techniques de programmation linéaire. Des chercheurs intéressés par l'automatisation de la preuve de la terminaison de programmes ont développé des procédures d'évaluation ascendante pour des programmes Datalog interprétés sur les entiers relatifs : étant donné une procédure en Prolog, le but est d'engendrer des inéquations sur la taille des arguments des prédicats auxiliaires appelés par la procédure, et d'utiliser ces inéquations pour prouver la terminaison de la procédure [BRO 91, GEL 90].

La méthode de [BRO 91] est capable d'engendrer des disjonctions d'inéquations, mais chaque inéquation est nécessairement de la forme $arg(i) + c > arg(j)$, ayant au plus des relations sur deux arguments. La méthode de [GEL 90] peut engendrer des relations entre plusieurs arguments (par exemple, $arg(i)+arg(j) > arg(k)+c$), mais ne peut pas engendrer des disjonctions de ces formules. Cela est aussi le cas dans le travail de [COU 78] et dans des méthodes utilisant des approches descendantes [PLU 90, ULL 88] et non ascendantes. Notre méthode n'a pas ces restrictions, et engendre une formule arithmétique qui est le plus petit point fixe du programme.

La limitation principale de notre méthode vient de la restriction syntaxique de la classe de programmes que nous pouvons traiter : le schéma de récursion est nécessairement direct et linéaire (pas d'appels mutuels, pas d'appels multiples), et dans chaque règle, les arguments récursifs sont nécessairement incrémentés par une constante.

Chapitre 3

Préliminaires

On considère des programmes Π définis par une règle non-récursive $\mathcal{R}_0$ de la forme:

$$p(x, y, z) \quad :- \quad \Theta(x, y, z). \qquad \text{règle } \mathcal{R}_0$$

où $\Theta(x, y, z)$ est une formule arithmétique linéaire,

et n règles récursives $\mathcal{R}_k$ $(1 \leq k \leq n)$ de la forme:

$$p(x + a_k, y + b_k, z + c_k) \quad :- \quad \phi_k(x, y, z),\ p(x, y, z). \qquad \text{règle } \mathcal{R}_k$$

où $a_k, b_k,\ c_k \ \in \mathbb{Z}$, et $\phi_k(x, y, z)$ est une contrainte de la forme $d_k x + e_k y + f_k z + g_k \geq 0$, avec $d_k, e_k, f_k, g_k \ \in \mathbb{Z}$.

On montrera dans les chapitres suivants (pour $n = 1, 2, 3$), que le résultat de l'évaluation ascendante de ces programmes peut être exprimé par une formule arithmétique (finie). Plus précisément, un atome $p(u, v, w)$ appartient à l'ensemble engendré par l'évaluation ascendante de Π ssi u, v, w satisfont une formule arithmétique linéaire (finie).

Dans un souci de concision, nous avons considéré que le prédicat p a pour arité 3. Les mêmes résultats sont en effet valables pour n'importe quelle arité.

Le programme Π est un programme logique avec contraintes. Son domaine d'interprétation est le domaine des entiers relatifs, au lieu du domaine de Herbrand classique. Dans un cadre de programmation logique avec contraintes, on étend les notions d'atome de Herbrand et d'interprétation de Herbrand (voir, par exemple [JAF 87, KAN 90]). Dans notre contexte, nous utiliserons la notion d'atome de Herbrand généralisé dans le sens suivant :

Un *atome généralisé de Herbrand* est défini comme une paire $(p(u,v,w)\ ;\ \Psi)$, où u, v, w sont des termes arithmétiques (construits avec $+$, des entiers relatifs et des variables) et Ψ est une formule arithmétique linéaire (ayant u, v, w comme variables libres). Notons que tout atome de Herbrand peut être exprimé par un atome généralisé de Herbrand. Par exemple, $p(6,5,7)$ est exprimé comme un atome généralisé composé de l'atome $p(u,v,w)$, et de la formule Ψ: $u = 6 \wedge v = 5 \wedge w = 7$.

Pour calculer le résultat de l'évaluation ascendante de Π, l'opérateur de conséquence immédiate pour le programme Π (voir [APT 82, END 76]) est adapté de façon à ce qu'il puisse être appliqué à des atomes généralisés de Herbrand (*cf* [JAF 87]).

Une *interprétation* est un ensemble d'atomes généralisés de Herbrand. L'*interprétation initiale* I_0 est $\{\ (p(x,y,z)\ ;\ \Theta(x,y,z))\ \}$.

Etant donné une interprétation I, on définit, pour $1 \leq k \leq n$:

$$T_k(I) \ = \ \{\ (p(u+a_k, v+b_k, w+c_k)\ ;\ \Phi_k(u,v,w) \wedge\ \Psi)\ \ /\ \ \ (p(u,v,w)\ ;\ \Psi)\ \in\ I\}$$

L'opérateur T_k peut être vu comme l'opérateur de conséquence immédiate associé à la règle $\mathcal{R}_k$ $(1 \leq k \leq n)$. L'opérateur de conséquence immédiate T associé au programme Π peut être défini par:

$$T(I) \ = \ T_1(I) \bigcup T_2(I) \bigcup \ \ldots \ \bigcup \ T_n(I)$$

On définit T^s par induction:

$$\begin{array}{lll} T^1(I) & = & T(I) \\ T^{s+1}(I) & = & T(T^s(I)), \quad \text{pour } s \geq 1 \end{array}$$

Le résultat de l'évaluation ascendante de Π est $I_0 \cup \bigcup_{s>0} T^s(I_0)$.

Ce résultat est le plus petit point fixe de T. Il est également le plus petit modèle généralisé de Π [JAF 87, END 76].

3.1 Présentation Générale de la Méthode

Dans cette partie, on présentera les notions de base sur lesquelles notre approche pour obtenir une représentation finie pour le résultat de l'évaluation ascendante de Π est basée. Ces concepts seront donnés pour le cas où Π a trois règles récursives. Le cas de deux règles

(et d'une règle) récursives peut être considéré comme un cas particulier du cas de trois règles récursives.

Nous adopterons une approche géométrique pour calculer le résultat de l'évaluation ascendante de Π. Ce résultat peut être exprimé par:

$$I_0 \cup \bigcup_{k_1,k_2,\ldots,k_m \in \{1,2,3\}, m>0} T_{k_m}...T_{k_2}T_{k_1}(I_0).$$

Nous associerons un *chemin* géométrique reliant des points de $\mathbb{N} \times \mathbb{N} \times \mathbb{N}$ à chaque composition d'opérateurs $T_{k_m}...T_{k_2}T_{k_1}$: chaque numéro 1 dans la séquence $k_m...k_2k_1$ sera associé à un déplacement horizontal, chaque numéro 2 à un déplacement vertical, et chaque numéro 3 à un déplacement transversal. Cela est schématisé dans la figure 3.1.

En abrégé, on parlera de "pas en x" pour désigner un déplacement horizontal. De même, on parlera de "pas en y" (resp. "pas en z") pour désigner un déplacement vertical (resp. transversal).

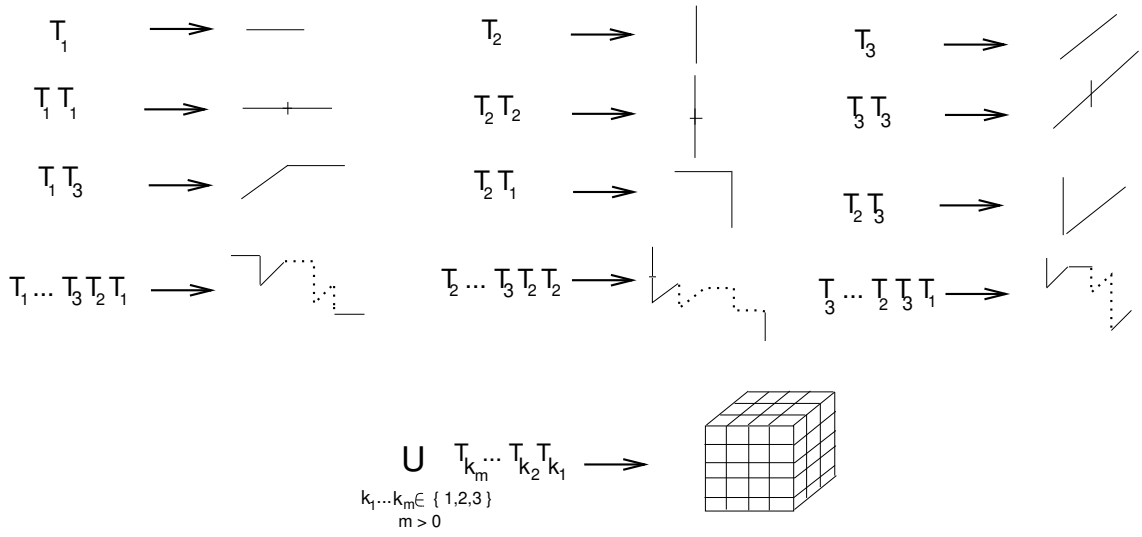


FIG. 3.1 -

Notons que dans le cas de deux règles récursives, on n'a pas de déplacements transversaux (voir figure 3.2).

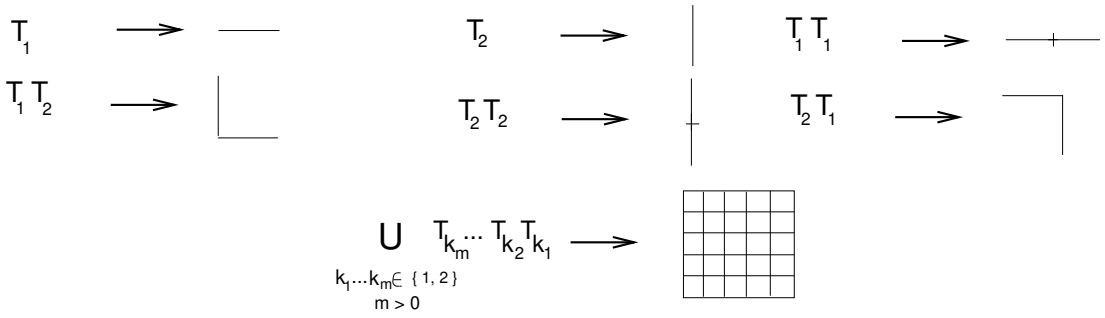


FIG. 3.2 -

Dans un chemin, l'origine (0,0,0) représente le tuple $< x, y, z >$ et chaque point M de coordonnées (α, β, γ) représente le tuple $< x + \alpha a_1 + \beta a_2 + \gamma a_3, y + \alpha b_1 + \beta b_2 + \gamma b_3, z + \alpha c_1 + \beta c_2 + \gamma c_3 >$, engendré après α applications de la règle $\mathcal{R}_1$, β applications de la règle $\mathcal{R}_2$ et γ applications de la règle $\mathcal{R}_3$.

Formellement, un *chemin* $\mathcal{P}$ *entre* $(0,0,0)$ *et* (λ, μ, ν), noté $\ll (0,0,0), ..., (\lambda, \mu, \nu) \gg$, est une séquence finie de points de $\mathbb{N} \times \mathbb{N} \times \mathbb{N}$, commençant par le point $(0,0,0)$ et finissant par le point (λ, μ, ν), tel que, si un point (α, β, γ) distinct de (λ, μ, ν) est dans $\mathcal{P}$, alors, ou $(\alpha + 1, \beta, \gamma)$, ou $(\alpha, \beta + 1, \gamma)$, ou $(\alpha, \beta, \gamma + 1)$ est en $\mathcal{P}$.

Soit (λ', μ', ν') et (λ, μ, ν) deux points, avec $(\alpha, \beta, \gamma) \neq (\alpha', \beta', \gamma')$ et $\alpha' \leq \alpha$, $\beta' \leq \beta$, $\gamma' \leq \gamma$. Un *chemin* $\mathcal{P}$ *entre* (λ', μ', ν') *et* (λ, μ, ν) est une séquence finie de points de $\mathbb{N} \times \mathbb{N} \times \mathbb{N}$, commençant par le point (λ', μ', ν') et finissant par le point (λ, μ, ν), tel que, si un point (α, β, γ) distinct de (λ, μ, ν) est dans $\mathcal{P}$, alors, ou $(\alpha+1, \beta, \gamma)$, ou $(\alpha, \beta+1, \gamma)$, ou $(\alpha, \beta, \gamma+1)$ est en $\mathcal{P}$.

Par défaut, un chemin commence par le point $(0, 0, 0)$.

A chaque segment horizontal (α, β, γ)-$(\alpha + 1, \beta, \gamma)$ d'un chemin $\mathcal{P}$ est associée une contrainte $\phi_1(x + \alpha a_1 + \beta a_2 + \gamma a_3, y + \alpha b_1 + \beta b_2 + \gamma b_3, z + \alpha c_1 + \beta c_2 + \gamma c_3)$ (abrégée en $\Phi_1(\alpha, \beta, \gamma)$). Cette contrainte représente la formule arithmétique qui doit être satisfaite pour appliquer la règle $\mathcal{R}_1$ à un point (α, β, γ).

De même, à chaque segment vertical (α, β, γ)-$(\alpha, \beta+1, \gamma)$ de $\mathcal{P}$ est associée une contrainte $\phi_2(x + \alpha a_1 + \beta a_2 + \gamma a_3, y + \alpha b_1 + \beta b_2 + \gamma b_3, z + \alpha c_1 + \beta c_2 + \gamma c_3)$ (abrégée en $\Phi_2(\alpha, \beta, \gamma)$). Cette contrainte représente la formule arithmétique qui doit être satisfaite pour appliquer la règle $\mathcal{R}_2$ à (α, β, γ).

A chaque segment transversal (α,β,γ)-$(\alpha,\beta,\gamma+1)$ de $\mathcal{P}$ est associée une contrainte $\phi_3(x+\alpha a_1+\beta a_2+\gamma a_3, y+\alpha b_1+\beta b_2+\gamma b_3, z+\alpha c_1+\beta c_2+\gamma c_3)$ (abrégée en $\Phi_3(\alpha,\beta,\gamma)$). Cette contrainte représente la formule arithmétique qui doit être satisfaite pour appliquer la règle $\mathcal{R}_3$ à (α,β,γ).

Dorénavant, la contrainte Φ_k représentera toujours la contrainte associée à la règle récursive $\mathcal{R}_k$ $(1 \leq k \leq 3)$ d'un programme Π.

Soit $M=(\alpha,\beta,\gamma)$ et $M'=(\alpha',\beta',\gamma')$ deux points de $\mathbb{N}\times\mathbb{N}\times\mathbb{N}$. Nous disons que le point M est situé avant M', et nous écrivons $M<M'$, ssi $(\alpha,\beta,\gamma)\neq(\alpha',\beta',\gamma')$ et $\alpha\leq\alpha'$, $\beta\leq\beta'$, $\gamma\leq\gamma'$. La contrainte $\Phi_k(\alpha,\beta,\gamma)$ du point $M=(\alpha,\beta,\gamma)$ sera parfois abrégée par $\Phi_k(M)$.

Nous disons que la contrainte $\Phi_k(M)$ d'un point M est *plus forte* que la contrainte $\Phi_k(M')$ d'un point M' ssi $\Phi_k(M) \Rightarrow \Phi_k(M')$. Nous disons aussi que $\Phi_k(M')$ *subsume* $\Phi_k(M)$ ou que $\Phi_k(M')$ est *plus faible* que $\Phi_k(M)$. Cela est noté, en commettant un léger abus de langage, par $\Phi_k(M)\leq\Phi_k(M')$.

Similairement, en commettant un léger abus de langage, la formule:

$$
\begin{array}{lll}
\Phi_k(M)\geq\Phi_k(M') & \text{signifie} & \Phi_k(M') \Rightarrow \Phi_k(M)\\
\Phi_k(M)=\Phi_k(M') & \text{signifie} & \Phi_k(M') \Leftrightarrow \Phi_k(M)\\
\Phi_k(M)>\Phi_k(M') & \text{signifie} & \Phi_k(M') \Rightarrow \Phi_k(M) \wedge \Phi_k(M) \not\Rightarrow \Phi_k(M')\\
\Phi_k(M)<\Phi_k(M') & \text{signifie} & \Phi_k(M) \Rightarrow \Phi_k(M') \wedge \Phi_k(M') \not\Rightarrow \Phi_k(M)\\
\Phi_k(M)\neq\Phi_k(M') & \text{signifie} & \Phi_k(M)>\Phi_k(M') \vee \Phi_k(M)<\Phi_k(M')
\end{array}
$$

Soit $d_k x+e_k y+e_k z+g_k \geq 0$ la contrainte $\phi_k(x,y,z)$ d'un programme Π et h un nombre entier relatif. L'expression $\Phi_k(M)=\Phi_k(M')+h$ signifie que $d_k\alpha+e_k\beta+f_k\gamma = d_k\alpha'+e_k\beta'+f_k\gamma'+h$. Si $h\geq 0$, on a $\Phi_k(M)\geq\Phi_k(M')$ et si $h\leq 0$, on a $\Phi_k(M)\leq\Phi_k(M')$.

Nous disons que la contrainte Φ_k est *strictement croissante* ssi $\forall M,\ \forall M'>M\ \Phi_k(M)<\Phi_k(M')$ et que la contrainte Φ_k est *strictement décroissante* ssi $\forall M,\ \forall M'>M\ \Phi_k(M)>\Phi_k(M')$. Nous disons que Φ_k est *strictement monotone* ssi Φ_k est bien *strictement croissante* ou *strictement décroissante*.

La *contrainte globale* $\Gamma_{\mathcal{P}}$ associée à un chemin $\mathcal{P}$, commençant en $(0,0,0)$ et finissant en (λ,μ,ν), est la conjonction des contraintes de tous les segments de $\mathcal{P}$. La contrainte globale $\Gamma_{\mathcal{P}}$ contient $\lambda+\mu+\nu$ atomes. Formellement, elle est définie par:

$$\Gamma_{\mathcal{P}} \stackrel{\text{déf}}{=} \Gamma_{\mathcal{P}}(\lambda,\mu,\nu)$$

où:

$$
\begin{array}{lll}
\Gamma_{\mathcal{P}}(0,0,0) & \stackrel{\text{déf}}{=} & \text{vrai.} \\
\Gamma_{\mathcal{P}}(\alpha+1,\beta,\gamma) & \stackrel{\text{déf}}{=} & \Gamma_{\mathcal{P}}(\alpha,\beta,\gamma) \wedge \Phi_1(\alpha,\beta,\gamma), \qquad \text{si } (\alpha+1,\beta,\gamma) \in \mathcal{P}. \\
\Gamma_{\mathcal{P}}(\alpha,\beta+1,\gamma) & \stackrel{\text{déf}}{=} & \Gamma_{\mathcal{P}}(\alpha,\beta,\gamma) \wedge \Phi_2(\alpha,\beta,\gamma), \qquad \text{si } (\alpha,\beta+1,\gamma) \in \mathcal{P}. \\
\Gamma_{\mathcal{P}}(\alpha,\beta,\gamma+1) & \stackrel{\text{déf}}{=} & \Gamma_{\mathcal{P}}(\alpha,\beta,\gamma) \wedge \Phi_3(\alpha,\beta,\gamma), \qquad \text{si } (\alpha,\beta,\gamma+1) \in \mathcal{P}.
\end{array}
$$

De façon analogue, on définit la contrainte globale d'un chemin $\mathcal{P}$ entre (λ',μ',ν') et (λ,μ,ν), en remplaçant $\Gamma_{\mathcal{P}}(0,0,0) \stackrel{\text{déf}}{=}$ vrai dans la définition ci-dessus par $\Gamma_{\mathcal{P}}(\lambda',\mu',\nu') \stackrel{\text{déf}}{=}$ vrai.

Etant donné un chemin $\mathcal{P}$, on notera $\Gamma^k_{\mathcal{P}}$ la composante de $\Gamma_{\mathcal{P}}$ en k. Notons que $\Gamma_{\mathcal{P}} = \Gamma^1_{\mathcal{P}} \wedge \Gamma^2_{\mathcal{P}} \wedge \Gamma^3_{\mathcal{P}}$.

On notera l'ensemble de tous les chemins $\mathcal{P}$ de l'origine $(0,0,0)$ au point (λ,μ,ν) par $\mathcal{S}(\lambda,\mu,\nu)$. Cet ensemble de chemins correspond géométriquement à un cube (voir figure 3.3).

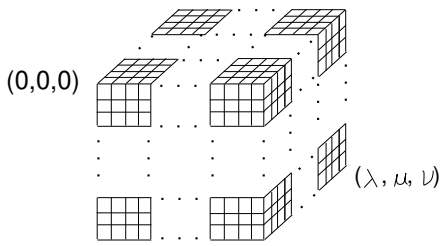


FIG. 3.3 -

La disjonction des contraintes globales de tous les chemins de $(0,0,0)$ à (λ,μ,ν) sera notée par $\Delta(\lambda,\mu,\nu)$, i.e.:

$$\Delta(\lambda,\mu,\nu) \stackrel{\text{déf}}{=} \bigvee_{\mathcal{P} \in \mathcal{S}(\lambda,\mu,\nu)} \Gamma_{\mathcal{P}}$$

En utilisant $\Delta(\lambda,\mu,\nu)$, on peut réécrire le résultat de l'évaluation ascendante de Π en:

$$\cup_{\lambda,\mu,\nu \in \mathbb{N}} \{\ p(x+\lambda a_1+\mu a_2+\nu a_3, y+\lambda b_1+\mu b_2+\nu b_3, z+\lambda c_1+\mu c_2+\nu c_3)\ /\ \Theta(x,y,z)\ \wedge\ \Delta(\lambda,\mu,\nu)\}$$

Notons que pour les programmes Π avec deux règles récursives, le résultat de l'évaluation ascendante de Π est donné par :

$$\cup_{\lambda,\mu \in \mathbb{N}} \{\ p(x+\lambda a_1+\mu a_2, y+\lambda b_1+\mu b_2, z+\lambda c_1+\mu c_2)\ /\ \Theta(x,y,z)\ \wedge\ \Delta(\lambda,\mu)\}$$

D'autre part, on peut dire qu'un atome $p(u, v, w)$ est engendré par l'évaluation ascendante de Π ssi:

$$\exists \lambda, \mu, \nu, x, y, z \quad \lambda \geq 0 \wedge \mu \geq 0 \wedge \nu \geq 0 \wedge u = x + \lambda a_1 + \mu a_2 + \nu a_3 \ \wedge \ v = y + \lambda b_1 + \mu b_2 + \nu b_3 \wedge w = z + \lambda c_1 + \mu c_2 + \nu c_3 \ \wedge \ \Theta(x, y, z) \ \wedge \ \Delta(\lambda, \mu, \nu).$$

D'après cette dernière formule, pour exprimer $p(u, v, w)$ comme une formule arithmétique linéaire, il suffit d'exprimer $\Delta(\lambda, \mu, \nu)$ comme une formule arithmétique linéaire. Cela se fera en deux étapes de simplification :

$\bigvee$-simplification: Détermination d'un sous-ensemble $\mathcal{S}'(\lambda, \mu, \nu)$ du cube $\mathcal{S}(\lambda, \mu, \nu)$, tel que tous les chemins dans $\mathcal{S}'(\lambda, \mu, \nu)$ aient une forme caractéristique et que ces chemins subsument tous les autres chemins de $\mathcal{S}(\lambda, \mu, \nu)$, c'est-à-dire :

$$\bigvee_{\mathcal{P} \in \mathcal{S}(\lambda, \mu, \nu)} \Gamma_{\mathcal{P}} \ = \ \bigvee_{\mathcal{P} \in \mathcal{S}'(\lambda, \mu, \nu)} \Gamma_{\mathcal{P}}$$

$\bigwedge$-simplification: Simplification de $\bigvee_{\mathcal{P} \in \mathcal{S}'(\lambda, \mu, \nu)} \Gamma_{\mathcal{P}}$ en une formule arithmétique. Cela se fait en réduisant la conjonction de contraintes $\Gamma_{\mathcal{P}}$, pour tout $\mathcal{P} \in \mathcal{S}'(\lambda, \mu, \nu)$.

Dans la pratique, pour trois règles récursives, le calcul de $\Delta(\lambda, \mu, \nu)$ est effectué en le décomposant en :

1. **Cas de Base** $\Delta(0, 0, 0)$ qui, par définition, vaut vrai.
2. **Composantes Rectilignes** $\Delta(\lambda, 0, 0)$, $\Delta(0, \mu, 0)$ et $\Delta(0, 0, \nu)$, avec $\lambda > 0$, $\mu > 0$ et $\nu > 0$. Ceci correspond à des chemins où il n'y a que des déplacements horizontaux $(\Delta(\lambda, 0, 0))$, ou que des déplacements verticaux $(\Delta(0, \mu, 0))$, ou que des déplacements transversaux $(\Delta(0, 0, \nu))$.
3. **Composantes Planaires** $\Delta(\lambda, \mu, 0)$, $\Delta(\lambda, 0, \nu)$ et $\Delta(\lambda, \mu, 0)$, avec $\lambda > 0$, $\mu > 0$ et $\nu > 0$. Ceci correspond a des chemins contenus dans le plans 0_{XY}, 0_{XZ}, et 0_{YZ}, respectivement.
4. **Composantes non-Planaires** $\Delta(\lambda, \mu, \nu)$, avec $\lambda > 0, \mu > 0$ et $\nu > 0$. Ceci correspond a des chemins qui contiennent au moins un déplacement horizontal, un déplacement vertical et un déplacement transversal.

Pour deux règles récursives, $\Delta(\lambda, \mu)$ est décomposé en :

1. **Cas de Base** $\Delta(0, 0)$ qui, par définition, vaut vrai.

2. **Composantes Rectilignes** $\Delta(\lambda, 0)$ et $\Delta(0, \mu)$, avec $\lambda > 0$ et $\mu > 0$. Ceci correspond a des chemins où il n'y a que des déplacements horizontaux ($\Delta(\lambda, 0)$), ou que des déplacements verticaux ($\Delta(0, \mu)$).

3. **Composantes non-rectilignes** $\Delta(\lambda, \mu)$, avec $\lambda > 0$, $\mu > 0$. Ceci correspond a des chemins qui contiennent au moins un déplacement horizontal et un déplacement vertical.

Pour des programmes avec deux règles récursives, le calcul de $\Delta(\lambda, 0)$, avec $\lambda > 0$, ne pose pas de difficultés. Il est fait comme dans le cas de 1 règle récursive.

Pour des programmes avec trois règles récursives, le calcul de $\Delta(\lambda, 0, 0)$, $\Delta(0, \mu, 0)$ et $\Delta(0, 0, \nu)$, avec $\lambda > 0$, $\mu > 0$ et $\nu > 0$, est fait comme dans le cas de 1 règle récursive. Le calcul $\Delta(\lambda, \mu, \nu)$ pour les cas planaires est fait comme dans le cas de deux règles récursives.

Dans les chapitres 6 et 7, nous étudierons les étapes de $\bigvee$-simplification et de $\bigwedge$-simplification des chemins non-linéaires pour des programmes Π avec deux règles récursives.

Dans les chapitres 9 à 12, nous étudierons l'étape de $\bigvee$-simplification des chemins non-planaires pour les programmes Π avec trois règles récursives. On s'intéressera seulement à l'étape de $\bigvee$-simplification, l'étape de $\bigwedge$-simplification ne posant pas de grandes difficultés. Elle est faite comme dans le cas de deux règles récursives.

Nous appellerons *formule caractéristique* de Π, que l'on notera ξ_Π, la formule arithmétique qui exprime le résultat de l'évaluation ascendante de Π.

3.2 Implications de Contraintes

Dans cette partie, nous montrons l'existence de certaines relations d'ordre sur les contraintes des programmes Π. Ces relations d'ordre seront appelées *implications de contraintes*. Les implications de contraintes nous fourniront une classification pour les programmes Π.

D'autre part, pour certains programmes, l'étape de $\bigvee$-simplification de notre méthode, pour exprimer le résultat de l'évalution ascendante de Π comme une formule arithmétique, est basée sur ce concept.

Dans les trois sous-parties suivantes, nous présenterons le concept d'implication de contraintes et le concept de matrice caractéristique d'un programme.

Rappelons que les règles récursives des programmes II que nous considérons sont de la forme :

$$p(x + a_k, y + b_k, z + c_k) \quad :- \quad \phi_k(x, y, z),\ p(x, y, z).$$

où $\phi_k(x, y, z)$ est une inégalité de la forme $d_k x + e_k y + f_k z + g_k \geq 0$, $1 \leq k \leq n$ avec $n = 1,\ 2,\ 3$.

3.2.1 Une Règle Récursive

Pour les programmes avec une règle récursive, les *Implications de Contraintes* sont des relations d'ordre sur les contraintes des points adjacents de N. La proposition 3.3 établit l'existence de 2 implications de contraintes pour des programmes II.

Proposition 3.1 *Soit la contrainte Φ_1 d'un programme* II. *Pour tout α, $\Phi_1(\alpha)$ est plus fort que $\Phi_1(\alpha + 1)$ ou vice-versa.*

On représentera cette implication par une flèche simple (voir figure 3.4).

FIG. 3.4 - *Implication de Contraintes* $\rightarrow \leftarrow$

Soit $a'_1 = d_1 a_1 + e_1 b_1 + f_1 c_1$.

Il existe une bijection entre le sens de l'implication de la contrainte Φ_1 et le signe du coefficient a'_1. Cette bijection est représentée dans le tableau :

$a'_1 \geq 0$	$\Phi_1(\alpha) \Rightarrow \Phi_1(\alpha + 1)$
$a'_1 \leq 0$	$\Phi_1(\alpha) \Leftarrow \Phi_1(\alpha + 1)$

Exemple 3.2 Soit π_1 le programme :

$p(x, y, z)$	:-	$\Theta(x, y, z)$.	règle $\mathcal{R}_0$
$p(x + 3, y + 2, z + 1)$	:-	$x - z > 0,\ p(x, y, z)$.	règle $\mathcal{R}_1$

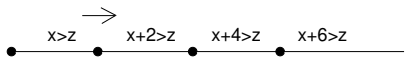


FIG. 3.5 - *Implications de Contraintes pour le programme* π_1

Soit $M = (\alpha)$. La contrainte $\Phi_1(M) \equiv \Phi_1(\alpha) \equiv \phi_1(x + 3\alpha, y + 2\alpha, z + \alpha)$ vaut $x + 3\alpha - (z + \alpha) > 0$, c'est-à-dire : $x + 2\alpha > z$.

Le sens de l'implication de contraintes est représenté dans la figure 3.5.

On a $a'_1 = d_1 a_1 + e_1 b_1 + f_1 c_1 = 1 \times 3 + 0 \times 2 + (-1) \times 1 = 2$.

□

3.2.2 Deux Règles Récursives

Pour les programmes avec deux règles récursives, les *Implications de Contraintes* sont des relations d'ordre sur les contraintes des points adjacents de $\mathbb{N} \times \mathbb{N}$. La proposition 3.3 établit l'existence de 4 implications de contraintes pour des programmes Π (2 pour chaque contrainte $\Phi_k(\alpha, \beta)$).

Proposition 3.3 *Soit la contrainte* Φ_k $(1 \leq k \leq 2)$ *d'un programme* Π. *Pour tout* α, β*:*

1. *Soit* $\Phi_k(\alpha, \beta)$ *est plus fort que* $\Phi_k(\alpha + 1, \beta)$ *ou vice-versa.*
2. *Soit* $\Phi_k(\alpha, \beta)$ *est plus fort que* $\Phi_k(\alpha, \beta + 1)$ *ou vice-versa.*

On représentera graphiquement avec des flèches simples :

– l'implication de Φ_1 par rapport aux points $(\alpha, \beta) - (\alpha + 1, \beta)$,

– l'implication de Φ_2 par rapport aux points $(\alpha, \beta) - (\alpha, \beta + 1)$.

On représentera graphiquement avec des flèches doubles les autres implications. Ceci est schématisé à la figure 3.6.

Soit $a'_k = d_1 a_k + e_1 b_k + f_1 c_k$ et $b'_k = d_2 a_k + e_2 b_k + f_2 c_k$ $(k \in \{1, 2\})$.

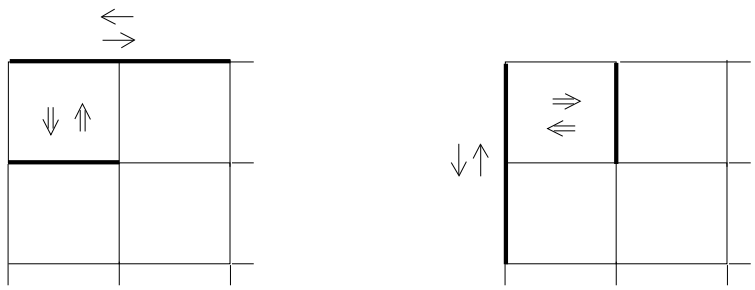


FIG. 3.6 - *Implications de Contraintes* $\Rightarrow \Leftarrow \rightarrow \leftarrow \Uparrow \Downarrow \uparrow \downarrow$

Il existe une bijection entre le sens des implications de contraintes et les signes de coefficients a'_k et b'_k. Cette bijection est représentée dans les tableaux :

$a'_1 \geq 0$	$\Phi_1(\alpha,\beta) \Rightarrow \Phi_1(\alpha+1,\beta)$
$a'_1 \leq 0$	$\Phi_1(\alpha,\beta) \Leftarrow \Phi_1(\alpha+1,\beta)$
$a'_2 \geq 0$	$\Phi_1(\alpha,\beta) \Rightarrow \Phi_1(\alpha,\beta+1)$
$a'_2 \leq 0$	$\Phi_1(\alpha,\beta) \Leftarrow \Phi_1(\alpha,\beta+1)$

$b'_1 \geq 0$	$\Phi_2(\alpha,\beta) \Rightarrow \Phi_2(\alpha+1,\beta)$
$b'_1 \leq 0$	$\Phi_2(\alpha,\beta) \Leftarrow \Phi_2(\alpha+1,\beta)$
$b'_2 \geq 0$	$\Phi_2(\alpha,\beta) \Rightarrow \Phi_2(\alpha,\beta+1)$
$b'_2 \leq 0$	$\Phi_2(\alpha,\beta) \Leftarrow \Phi_2(\alpha,\beta+1)$

On a ainsi une matrice 2×2, dont les éléments sont les signes[1] de a'_k et b'_k, que l'on appelle *matrice caractéristique* du programme, et que l'on note M_Π :

$$\begin{pmatrix} \text{signe de } a'_1 & \text{signe de } b'_1 \\ \text{signe de } a'_2 & \text{signe de } b'_2 \end{pmatrix}$$

Chaque élément de la matrice caractéristique représente un sens d'implication de contraintes.

Exemple 3.4 Soit π_2 le programme :

$p(x,y,z)$	:–	$\Theta(x,y,z).$	règle $\mathcal{R}_0$
$p(x+3,y,z+1)$	:–	$x - z > 0,\ p(x,y,z).$	règle $\mathcal{R}_1$
$p(x,y-2,z+3)$	:–	$2y - x > 0,\ p(x,y,z).$	règle $\mathcal{R}_2$

1. si $a'_k = 0$, on peut prendre l'élément dans la matrice égal à + ou −, au hasard

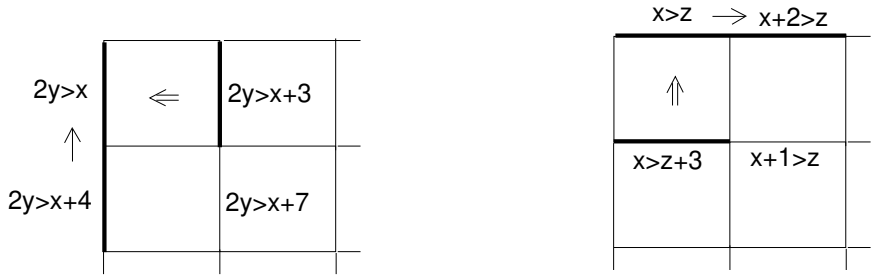


FIG. 3.7 - *Implications de Contraintes pour le programme* π_2

Soit $M = (\alpha, \beta)$. La contrainte $\Phi_1(M) \equiv \Phi_1(\alpha, \beta) \equiv \phi_1(x + 3\alpha, y - 2\beta, z + \alpha - \beta)$ vaut $x + 3\alpha - (z + \alpha + 3\beta) > 0$, c'est-à-dire : $x + 2\alpha > z + 3\beta$ et la contrainte $\Phi_2(M) \equiv \Phi_2(\alpha, \beta) \equiv \phi_2(x + 3\alpha, y - 2\beta, z + \alpha - \beta)$ vaut $2y - 4\beta - (x + 3\alpha) > 0$, c'est-à-dire : $2y > x + 3\alpha + 4\beta$.

Les sens des implications de contraintes sont représentés dans la figure 3.7.

On a :

$$\begin{array}{rcccccl} a'_1 & = & d_1a_1 + e_1b_1 + f_1c_1 & = & 1 \times 3 + 0 \times 0 + (-1) \times 1 & = & 2 \\ a'_2 & = & d_1a_2 + e_1b_2 + f_1c_2 & = & 1 \times 0 + 0 \times (-2) + (-1) \times 3 & = & -3 \\ b'_1 & = & d_2a_1 + e_2b_1 + f_2c_1 & = & -1 \times 3 + 2 \times 0 + 0 \times 1 & = & -3 \\ b'_2 & = & d_2a_2 + e_2b_2 + f_2c_2 & = & -1 \times 0 + 2 \times (-2) + 0 \times 3 & = & -4 \end{array}$$

La matrice caractéristique du programme π_2 est $\begin{pmatrix} + & - \\ - & - \end{pmatrix}$.

□

3.2.3 Trois Règles Récursives

Pour les programmes avec trois règles récursives, les *Implications de Contraintes* sont des relations d'ordre sur les contraintes des points adjacents de $\mathbb{N} \times \mathbb{N} \times \mathbb{N}$. Analogue à la proposition 3.3, nous avons:

Proposition 3.5 *Soit la contrainte* Φ_k $(1 \leq k \leq 3)$ *d'un programme* II. *Pour tout* α, β *et* γ*:*

1. *Soit* $\Phi_k(\alpha, \beta, \gamma)$ *est plus fort que* $\Phi_k(\alpha + 1, \beta, \gamma)$ *ou vice-versa.*
2. *Soit* $\Phi_k(\alpha, \beta, \gamma)$ *est plus fort que* $\Phi_k(\alpha, \beta + 1, \gamma)$ *ou vice-versa.*
3. *Soit* $\Phi_k(\alpha, \beta, \gamma)$ *est plus fort que* $\Phi_k(\alpha, \beta, \gamma + 1)$ *ou vice-versa.*

L'existence de 9 implications de contraintes pour II avec trois règles récursives découle directement de la proposition 3.5. On a 3 implications pour chaque contrainte $\Phi_k(\alpha, \beta, \gamma)$. Ces implications de contraintes sont représentées graphiquement dans la figure 3.8.

On notera avec des flèches simples :

– l'implication de Φ_1 par rapport aux points $(\alpha, \beta, \gamma) - (\alpha + 1, \beta, \gamma)$,

– l'implication de Φ_2 par rapport aux points $(\alpha, \beta, \gamma) - (\alpha, \beta + 1, \gamma)$,

– l'implication de Φ_3 par rapport aux points $(\alpha, \beta, \gamma) - (\alpha, \beta, \gamma + 1)$.

On notera avec des flèches doubles les autres implications de contraintes.

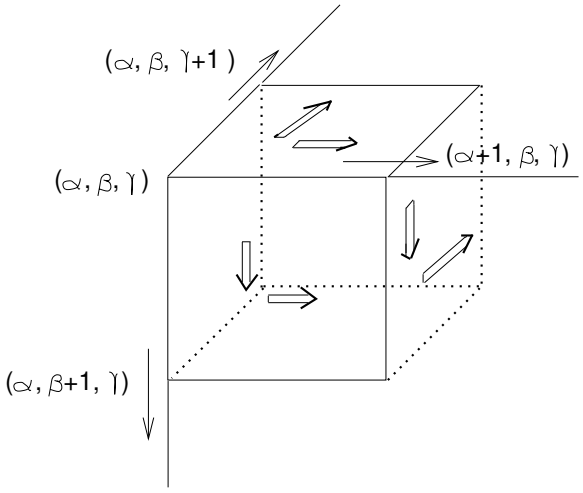


FIG. 3.8 - *Implications de Contraintes*

Soit $a'_k = d_1a_k+e_1b_k+f_1c_k$, $b'_k = d_2a_k+e_2b_k+f_2c_k$ et $c'_k = d_3a_k+e_3b_k+f_3c_k$ ($k \in \{1,2,3\}$).

Comme dans le cas de deux règles récursives, il existe une bijection entre le sens des implications de contraintes et les signes des coefficients de a'_k, b'_k et c'_k. Cette bijection est représentée dans les tableaux :

$a'_1 \geq 0$	$\Phi_1(\alpha,\beta,\gamma) \Rightarrow \Phi_1(\alpha+1,\beta,\gamma)$
$a'_1 \leq 0$	$\Phi_1(\alpha,\beta,\gamma \Leftarrow \Phi_1(\alpha+1,\beta,\gamma)$
$a'_2 \geq 0$	$\Phi_1(\alpha,\beta,\gamma) \Rightarrow \Phi_1(\alpha,\beta+1,\gamma)$
$a'_2 \leq 0$	$\Phi_1(\alpha,\beta,\gamma) \Leftarrow \Phi_1(\alpha,\beta+1,\gamma)$
$a'_3 \geq 0$	$\Phi_1(\alpha,\beta,\gamma) \Rightarrow \Phi_1(\alpha,\beta,\gamma+1)$
$a'_3 \leq 0$	$\Phi_1(\alpha,\beta,\gamma) \Leftarrow \Phi_1(\alpha,\beta,\gamma+1)$

$b'_1 \geq 0$	$\Phi_2(\alpha,\beta,\gamma) \Rightarrow \Phi_2(\alpha+1,\beta,\gamma)$
$b'_1 \leq 0$	$\Phi_2(\alpha,\beta,\gamma \Leftarrow \Phi_2(\alpha+1,\beta,\gamma)$
$b'_2 \geq 0$	$\Phi_2(\alpha,\beta,\gamma) \Rightarrow \Phi_2(\alpha,\beta+1,\gamma)$
$b'_2 \leq 0$	$\Phi_2(\alpha,\beta,\gamma) \Leftarrow \Phi_2(\alpha,\beta+1,\gamma)$
$b'_3 \geq 0$	$\Phi_2(\alpha,\beta,\gamma) \Rightarrow \Phi_2(\alpha,\beta,\gamma+1)$
$b'_3 \leq 0$	$\Phi_2(\alpha,\beta,\gamma) \Leftarrow \Phi_2(\alpha,\beta,\gamma+1)$

$c'_1 \geq 0$	$\Phi_3(\alpha,\beta,\gamma) \Rightarrow \Phi_3(\alpha+1,\beta,\gamma)$
$c'_1 \leq 0$	$\Phi_3(\alpha,\beta,\gamma \Leftarrow \Phi_3(\alpha+1,\beta,\gamma)$
$c'_2 \geq 0$	$\Phi_3(\alpha,\beta,\gamma) \Rightarrow \Phi_3(\alpha,\beta+1,\gamma)$
$c'_2 \leq 0$	$\Phi_3(\alpha,\beta,\gamma) \Leftarrow \Phi_3(\alpha,\beta+1,\gamma)$
$c'_3 \geq 0$	$\Phi_3(\alpha,\beta,\gamma) \Rightarrow \Phi_3(\alpha,\beta,\gamma+1)$
$c'_3 \leq 0$	$\Phi_3(\alpha,\beta,\gamma) \Leftarrow \Phi_3(\alpha,\beta,\gamma+1)$

On a ainsi une matrice caractéristique M_Π, 3×3, associée au programme Π, dont les éléments sont les signes de a'_k, b'_k et c'_k :

$$\begin{pmatrix} \text{signe de } a'_1 & \text{signe de } b'_1 & \text{signe de } c'_1 \\ \text{signe de } a'_2 & \text{signe de } b'_2 & \text{signe de } c'_2 \\ \text{signe de } a'_3 & \text{signe de } b'_3 & \text{signe de } c'_3 \end{pmatrix}$$

Chaque élément de la matrice caractéristique du programme représente un sens d'implication de contraintes.

Exemple 3.6 Soit π_3 le programme :

$p(x,y,z)$	:–	$\Theta(x,y,z)$.	règle $\mathcal{R}_0$
$p(x+2,y-4,z-2)$	:–	$x \geq 0,\ p(x,y,z)$.	règle $\mathcal{R}_1$
$p(x-5,y+9,z-1)$	:–	$y \geq 0,\ p(x,y,z)$.	règle $\mathcal{R}_2$
$p(x+2,y-3,z+1)$	:–	$z \geq 0,\ p(x,y,z)$.	règle $\mathcal{R}_3$

Soit $M = (\alpha, \beta, \gamma)$. La contrainte $\Phi_1(M) \equiv \Phi_1(\alpha, \beta, \gamma) \equiv \phi_1(x + 2\alpha - 5\beta + 2\gamma, y - 4\alpha + 9\beta - 3\gamma, z - 2\alpha - \beta + \gamma)$ vaut $x + 2\alpha - 5\beta + 2\gamma \geq 0$. La contrainte $\Phi_2(M) \equiv \Phi_2(\alpha, \beta, \gamma) \equiv \phi_2(x + 2\alpha - 5\beta + 2\gamma, y - 4\alpha + 9\beta - 3\gamma, z - 2\alpha - \beta + \gamma)$ vaut $x + 2\alpha - 5\beta + 2\gamma \geq 0$, et la contrainte $\Phi_3(M) \equiv \Phi_3(\alpha, \beta, \gamma) \equiv \phi_3(x + 2\alpha - 5\beta + 2\gamma, y - 4\alpha + 9\beta - 3\gamma, z - 2\alpha - \beta + \gamma)$ vaut $z - 2\alpha - \beta + \gamma \geq 0$.

Les sens des implications de contraintes de π_3 sont représentés dans la figure 3.9.

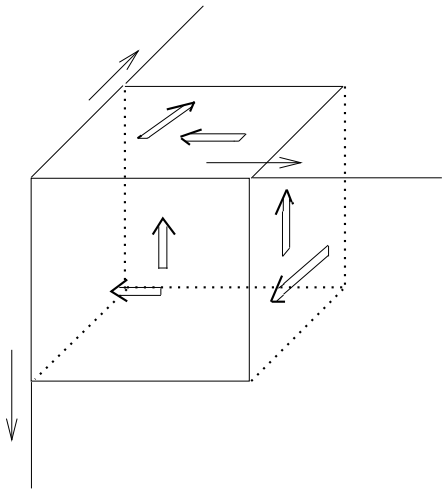


FIG. 3.9 - *Implications de Contraintes pour le programme* π_3

On a :

$$
\begin{array}{rcrcrcr}
a'_1 & = & d_1a_1 + e_1b_1 + f_1c_1 & = & 1 \times 2 + 0 \times (-4) + 0 \times (-2) & = & 2 \\
a'_2 & = & d_1a_2 + e_1b_2 + f_1c_2 & = & 1 \times (-5) + 0 \times 9 + 0 \times (-1) & = & -5 \\
a'_3 & = & d_1a_3 + e_1b_3 + f_1c_3 & = & 1 \times 2 + 0 \times (-3) + 0 \times 1 & = & 2 \\
b'_1 & = & d_2a_1 + e_2b_1 + f_2c_1 & = & 0 \times 2 + 1 \times (-4) + 0 \times (-2) & = & -4 \\
b'_2 & = & d_2a_2 + e_2b_2 + f_2c_2 & = & 0 \times (-5) + 1 \times 9 + 0 \times (-1) & = & 9 \\
b'_3 & = & d_2a_3 + e_2b_3 + f_2c_3 & = & 0 \times 2 + 1 \times (-3) + 0 \times 1 & = & -3 \\
c'_1 & = & d_3a_1 + e_3b_1 + f_3c_1 & = & 0 \times 2 + 0 \times (-4) + 1 \times (-2) & = & -2 \\
c'_2 & = & d_3a_2 + e_3b_2 + f_3c_2 & = & 0 \times (-5) + 0 \times 9 + 1 \times (-1) & = & -1 \\
c'_3 & = & d_3a_3 + e_3b_3 + f_3c_3 & = & 0 \times 2 + 0 \times (-3) + 1 \times 1 & = & 1
\end{array}
$$

La matrice caractéristique du programme π_3 est $\begin{pmatrix} + & - & - \\ - & + & - \\ + & - & + \end{pmatrix}$.

□

Nous pouvons classifier les programmes Π en fonction de leur matrice caractéristique : deux programmes de même matrice caractéristique appartiennent à une même classe de programmes.

Selon cette classification, on a 16 ($= 2^4$) classes pour les programmes avec deux règles récursives et 512 ($= 2^9$) classes pour les programmes avec trois règles récursives. Plus généralement, on a 2^{n^2} classes pour n règles récursives.

Chapitre 4

Classification Réduite de Programmes

Dans le chapitre 3, nous avons classifié les programmes II par rapport à leur matrice caractéristique : deux programmes de même matrice caractéristique appartiennent à une même classe de programmes.

Pour les programmes avec trois règles récursives, on a 512 (2^9) classes de programmes. Parmi 512 classes, nous avons :

- 410 classes où au moins 1 ligne ou 1 colonne a tous ses éléments de même signe.
- 102 classes où aucune ligne ni colonne n'a ses trois éléments de même signe.

Nous traiterons de façon identique tous les programmes dont la matrice caractéristique contient une ligne avec un même signe (**super-classe 1**). De même, nous traiterons de façon identique tous les programmes dont la matrice caractéristique contient une colonne avec un même signe (**super-classe 2**). Cela nous donne 2 premières super-classes de programmes.

Pour les 102 classes restantes, on peut définir des transformations sur ces classes de programmes, qui vont permettre également de regrouper dans une même super-classe des programmes différents. Deux programmes d'une même super-classe se déduisent l'un de l'autre en appliquant une série appropriée de deux transformations élémentaires, à savoir :

1. La permutation de l'ordre deux règles récursives.

2. L'inversion de la tête et du corps à intérieur de chaque règle récursive.

Ces transformations permettent de regrouper ces 102 classes de programmes en 11 nouvelles super-classes (**super-classes 3-13**, voir pages 89 à 92). Pour chacune de ces 11 super-classes, nous choisirons, arbitrairement, une matrice représentative pour l'étudier. Il s'agit des matrices suivantes :

$$\begin{pmatrix} - & + & + \\ + & - & - \\ - & - & + \end{pmatrix}, \begin{pmatrix} + & + & - \\ - & + & + \\ - & - & + \end{pmatrix}, \begin{pmatrix} + & + & - \\ - & - & + \\ - & - & + \end{pmatrix}, \begin{pmatrix} + & + & - \\ - & + & - \\ - & - & + \end{pmatrix}, \begin{pmatrix} + & + & - \\ - & - & + \\ - & + & + \end{pmatrix},$$

$$\begin{pmatrix} + & - & - \\ - & + & + \\ - & + & + \end{pmatrix}, \begin{pmatrix} + & - & - \\ - & + & + \\ - & + & - \end{pmatrix}, \begin{pmatrix} + & - & - \\ - & - & + \\ - & + & - \end{pmatrix}, \begin{pmatrix} + & - & - \\ - & + & - \\ - & - & + \end{pmatrix}, \begin{pmatrix} - & - & + \\ + & - & + \\ - & + & - \end{pmatrix},$$

$$\begin{pmatrix} - & - & + \\ + & + & - \\ - & + & + \end{pmatrix}.$$

Au total, on a ainsi 13 super-classes de programmes à considérer.

Dans les deux parties suivantes, nous présentons formellement les transformations pour le regroupement mentionné ci-dessus des 102 classes en 11 super-classes.

4.1 Permutation des Règles de Π

Considérons le programme Π^{perm}, obtenu à partir de Π en permutant l'ordre des règles $\mathcal{R}_1$ et $\mathcal{R}_2$. Le programme Π^{perm} est :

$p(x,y,z)$	:-	$\Theta(x,y,z).$	règle $\mathcal{R}_0$
$p(x+a_2, y+b_2, z+c_2)$	:-	$d_2x + e_2y + f_2z + h_2 \geq 0,\ p(x,y,z).$	règle $\mathcal{R}'_1 = \mathcal{R}_2$
$p(x+a_1, y+b_1, z+c_1)$	:-	$d_1x + e_1y + f_1z + h_1 \geq 0,\ p(x,y,z).$	règle $\mathcal{R}'_2 = \mathcal{R}_1$
$p(x+a_3, y+b_3, z+c_3)$	:-	$d_3x + e_3y + f_3z + h_3 \geq 0,\ p(x,y,z).$	règle $\mathcal{R}'_3 = \mathcal{R}_3$

Nous avons :

Proposition 4.1 *Les matrices caractéristiques de* Π *et* Π^{perm} *vérifient :* $M_{\Pi^{perm}} = \sigma_{1-2}(M_\Pi)$, *où* σ_{1-2} *dénote la transformation définie par :*

$$\sigma_{1-2}\left(\left(\begin{array}{ccc} m_{1,1} & m_{1,2} & m_{1,3} \\ m_{2,1} & m_{2,2} & m_{2,3} \\ m_{3,1} & m_{3,2} & m_{3,3} \end{array}\right)\right) = \left(\begin{array}{ccc} m_{2,1} & m_{2,2} & m_{2,3} \\ m_{1,1} & m_{1,2} & m_{1,3} \\ m_{3,2} & m_{3,1} & m_{3,3} \end{array}\right)$$

Comme Π et Π^{perm} représentent le même programme, Π et Π^{perm} ont la même formule caractéristique ξ :

Proposition 4.2 *Les formules caractéristiques de* Π *et* Π^{perm} *vérifient :* $\xi_{\Pi}(x, y, z) = \xi_{\Pi^{perm}}(x, y, z)$.

Par conséquent, nous pouvons regrouper dans une même super-classe deux programmes qui ont deux matrices caractéristiques identiques à la transformation σ_{1-2} près.

De même, en permutant les règles $\mathcal{R}_2$ et $\mathcal{R}_3$, nous pouvons regrouper dans une même super-classe les programmes avec matrices caractéristiques M et $\sigma_{2-3}(M)$, où:

$$\sigma_{2-3}\left(\left(\begin{array}{ccc} m_{1,1} & m_{1,2} & m_{1,3} \\ m_{2,1} & m_{2,2} & m_{2,3} \\ m_{3,1} & m_{3,2} & m_{3,3} \end{array}\right)\right) = \left(\begin{array}{ccc} m_{1,1} & m_{1,3} & m_{1,2} \\ m_{3,1} & m_{3,3} & m_{3,2} \\ m_{2,1} & m_{2,3} & m_{2,2} \end{array}\right)$$

La composition de ces deux transformations élémentaires engendre toutes les transformations possibles par permutation des règles $\mathcal{R}_1$, $\mathcal{R}_2$ et $\mathcal{R}_3$ (les permutations $(1, 2)$ et $(2, 3)$ sont une base pour le groupe de permutations de $\{1, 2, 3\}$).

Noter que deux programmes déduits l'un de l'autre par une série de permutations de règles ont même formule caractéristique et sont regroupés dans une même super-classe.

4.2 Inversion de la Tête et du Corps des Règles Π

Considérons un programme de la forme Π :

$$\begin{array}{lcl} p(x, y, z) & :- & \Theta(x, y, z) \\ p(x + a_1, y + b_1, z + c_1) & :- & \Phi_1(x, y, z),\ p(x, y, z) \\ p(x + a_2, y + b_2, z + c_2) & :- & \Phi_2(x, y, z),\ p(x, y, z) \\ p(x + a_3, y + b_3, z + c_3) & :- & \Phi_3(x, y, z),\ p(x, y, z) \end{array}$$

Soit Π_1 le programme[1] :

1. La présente formalisation nous a été suggérée par Hans Olsen, cf. [FRI 94c].

$$
\begin{array}{lcl}
p_1(x,y,z,u,v,w) & :- & x = u \wedge y = v \wedge z = w \\
p_1(x+a_1, y+b_1, z+c_1, u, v, w) & :- & \Phi_1(x,y,z),\ p_1(x,y,z,u,v,w) \\
p_1(x+a_2, y+b_2, z+c_2, u, v, w) & :- & \Phi_2(x,y,z),\ p_1(x,y,z,u,v,w) \\
p_1(x+a_3, y+b_3, z+c_3, u, v, w) & :- & \Phi_3(x,y,z),\ p_1(x,y,z,u,v,w)
\end{array}
$$

Il est facile de voir :

Proposition 4.3 $p(x,y,z) \Leftrightarrow p_1(x,y,z,u,v,w) \wedge \Theta(u,v,w)$.

Donc, le calcul de la formule caractéristique de Π se ramène au calcul de la formule caractéristique de Π_1. L'avantage de Π_1 sur Π est qu'il ne fait pas intervenir la formule de base Θ.

Considérons à présent le programme Π_1^{inv} obtenu "en gros" en inversant la tête et le corps de chaque règle de Π_1 :

$$
\begin{array}{lcl}
p_1^{inv}(x,y,z,u,v,w) & :- & x = u \wedge y = v \wedge z = w \\
p_1^{inv}(x-a_1, y-b_1, z-c_1, u, v, w) & :- & \Phi_1(x-a_1, y-b_1, z-c_1),\ p_1^{inv}(x,y,z,u,v,w) \\
p_1^{inv}(x-a_2, y-b_2, z-c_2, u, v, w) & :- & \Phi_2(x-a_2, y-b_2, z-c_2),\ p_1^{inv}(x,y,z,u,v,w) \\
p_1^{inv}(x-a_3, y-b_3, z-c_3, u, v, w) & :- & \Phi_3(x-a_3, y-b_3, z-c_3),\ p_1^{inv}(x,y,z,u,v,w)
\end{array}
$$

Il découle immédiatement de la forme du programme Π_1^{inv} :

Proposition 4.4 *Les matrices caractéristiques de* Π_1 *et* Π_1^{inv} *vérifient:* $M_{\Pi_1^{inv}} = -M_{\Pi_1}$.

D'autre part, de par la construction du programme Π_1^{inv}, le résultat de l'évaluation ascendante de Π_1^{inv} est symétrique de celui de Π_1. Ceci est formalisé par :

Proposition 4.5 *Les formules caractéristiques de* Π_1 *et* Π_1^{inv} *vérifient:* $\xi_{\Pi_1}(x,y,z,u,v,w) \Leftrightarrow \xi_{\Pi_1^{inv}}(u,v,w,x,y,z)$.

Il s'ensuit qu'on peut regrouper dans une même super-classe deux programmes ayant des matrices caractéristiques opposées.

Super-Classes 3 à 13

Super-classe 3 (12 éléments)

$$\begin{pmatrix} - & - & + \\ + & + & - \\ - & + & + \end{pmatrix} \begin{pmatrix} - & - & + \\ + & + & - \\ - & + & - \end{pmatrix} \begin{pmatrix} - & - & + \\ + & - & - \\ - & + & + \end{pmatrix} \begin{pmatrix} - & + & - \\ - & - & + \\ + & - & + \end{pmatrix}$$

$$\begin{pmatrix} - & + & - \\ - & + & + \\ + & - & + \end{pmatrix} \begin{pmatrix} - & + & - \\ - & + & + \\ + & - & - \end{pmatrix} \begin{pmatrix} + & + & - \\ - & - & + \\ + & - & - \end{pmatrix} \begin{pmatrix} + & + & - \\ - & - & + \\ + & - & + \end{pmatrix}$$

$$\begin{pmatrix} + & + & - \\ - & + & + \\ + & - & - \end{pmatrix} \begin{pmatrix} + & - & + \\ + & + & - \\ - & + & - \end{pmatrix} \begin{pmatrix} + & - & + \\ + & - & - \\ - & + & - \end{pmatrix} \begin{pmatrix} + & - & + \\ + & - & - \\ - & + & + \end{pmatrix}$$

Super-classe 4 (12 éléments)

$$\begin{pmatrix} + & + & - \\ - & + & + \\ - & - & + \end{pmatrix} \begin{pmatrix} + & - & + \\ - & + & - \\ - & + & + \end{pmatrix} \begin{pmatrix} - & + & + \\ + & - & - \\ - & + & - \end{pmatrix} \begin{pmatrix} - & - & + \\ + & - & + \\ - & + & - \end{pmatrix}$$

$$\begin{pmatrix} - & + & + \\ - & - & + \\ + & - & - \end{pmatrix} \begin{pmatrix} - & + & - \\ - & - & + \\ + & + & - \end{pmatrix} \begin{pmatrix} - & - & + \\ + & - & - \\ + & + & - \end{pmatrix} \begin{pmatrix} - & + & - \\ + & - & + \\ + & - & - \end{pmatrix}$$

$$\begin{pmatrix} + & - & - \\ - & + & + \\ + & - & + \end{pmatrix} \begin{pmatrix} + & + & - \\ - & + & - \\ + & - & + \end{pmatrix} \begin{pmatrix} + & - & - \\ + & + & - \\ - & + & + \end{pmatrix} \begin{pmatrix} + & - & + \\ + & + & - \\ - & - & + \end{pmatrix}$$

Super-classe 5 (12 éléments)

$$\begin{pmatrix} + & + & - \\ - & + & - \\ - & - & + \end{pmatrix} \quad \begin{pmatrix} + & - & - \\ - & + & - \\ - & + & + \end{pmatrix} \quad \begin{pmatrix} + & - & + \\ - & + & - \\ - & - & + \end{pmatrix} \quad \begin{pmatrix} + & - & - \\ - & + & + \\ - & - & + \end{pmatrix}$$

$$\begin{pmatrix} - & + & + \\ + & - & + \\ - & + & - \end{pmatrix} \quad \begin{pmatrix} - & + & + \\ - & - & + \\ + & + & - \end{pmatrix} \quad \begin{pmatrix} - & - & + \\ + & - & + \\ + & + & - \end{pmatrix} \quad \begin{pmatrix} - & + & + \\ + & - & + \\ + & - & - \end{pmatrix}$$

$$\begin{pmatrix} - & + & - \\ + & - & + \\ + & + & - \end{pmatrix} \quad \begin{pmatrix} - & + & + \\ + & - & - \\ + & + & - \end{pmatrix} \quad \begin{pmatrix} + & - & - \\ - & + & - \\ + & - & + \end{pmatrix} \quad \begin{pmatrix} + & - & - \\ + & + & - \\ - & - & + \end{pmatrix}$$

Super-classe 6 (12 éléments)

$$\begin{pmatrix} + & + & - \\ - & - & + \\ - & - & + \end{pmatrix} \quad \begin{pmatrix} + & - & + \\ - & + & - \\ - & + & - \end{pmatrix} \quad \begin{pmatrix} - & + & - \\ - & + & - \\ + & - & + \end{pmatrix} \quad \begin{pmatrix} - & - & + \\ + & + & - \\ - & - & + \end{pmatrix}$$

$$\begin{pmatrix} - & + & + \\ + & - & - \\ - & + & + \end{pmatrix} \quad \begin{pmatrix} - & + & + \\ - & + & + \\ + & - & - \end{pmatrix} \quad \begin{pmatrix} - & - & + \\ + & + & - \\ + & + & - \end{pmatrix} \quad \begin{pmatrix} - & + & - \\ + & - & + \\ + & - & + \end{pmatrix}$$

$$\begin{pmatrix} + & - & + \\ + & - & + \\ - & + & - \end{pmatrix} \quad \begin{pmatrix} + & + & - \\ - & - & + \\ + & + & - \end{pmatrix} \quad \begin{pmatrix} + & - & - \\ - & + & + \\ + & - & - \end{pmatrix} \quad \begin{pmatrix} + & - & - \\ + & - & - \\ - & + & + \end{pmatrix}$$

Super-classe 7 (12 éléments)

$$\begin{pmatrix} + & + & - \\ - & - & + \\ - & + & + \end{pmatrix} \quad \begin{pmatrix} + & + & - \\ - & - & + \\ - & + & - \end{pmatrix} \quad \begin{pmatrix} + & - & + \\ - & + & + \\ - & + & - \end{pmatrix} \quad \begin{pmatrix} + & - & + \\ - & - & + \\ - & + & - \end{pmatrix}$$

$$\begin{pmatrix} - & + & - \\ + & - & - \\ - & + & + \end{pmatrix} \quad \begin{pmatrix} - & - & + \\ - & + & + \\ + & - & - \end{pmatrix} \quad \begin{pmatrix} - & - & + \\ + & + & - \\ + & - & - \end{pmatrix} \quad \begin{pmatrix} - & - & + \\ + & + & - \\ + & - & + \end{pmatrix}$$

$$\begin{pmatrix} - & + & - \\ + & - & - \\ + & - & + \end{pmatrix} \quad \begin{pmatrix} - & + & - \\ + & + & - \\ + & - & + \end{pmatrix} \quad \begin{pmatrix} + & - & + \\ - & + & + \\ + & - & - \end{pmatrix} \quad \begin{pmatrix} + & + & - \\ + & - & - \\ - & + & + \end{pmatrix}$$

Super-classe 8 (6 éléments)

$$\begin{pmatrix} + & - & - \\ - & + & + \\ - & + & + \end{pmatrix} \quad \begin{pmatrix} - & + & - \\ + & - & + \\ - & + & - \end{pmatrix} \quad \begin{pmatrix} - & - & + \\ - & - & + \\ + & + & - \end{pmatrix}$$

$$\begin{pmatrix} - & + & + \\ + & - & - \\ + & - & - \end{pmatrix} \quad \begin{pmatrix} + & - & + \\ - & + & - \\ + & - & + \end{pmatrix} \quad \begin{pmatrix} + & + & - \\ + & + & - \\ - & - & + \end{pmatrix}$$

Super-classe 9 (12 éléments)

$$\begin{pmatrix} + & - & - \\ - & + & + \\ - & + & - \end{pmatrix} \quad \begin{pmatrix} + & - & - \\ - & - & + \\ - & + & + \end{pmatrix} \quad \begin{pmatrix} - & + & - \\ + & + & - \\ - & - & + \end{pmatrix} \quad \begin{pmatrix} - & + & - \\ + & - & + \\ - & + & + \end{pmatrix}$$

$$\begin{pmatrix} - & - & + \\ - & + & + \\ + & + & - \end{pmatrix} \quad \begin{pmatrix} - & - & + \\ - & + & - \\ + & - & + \end{pmatrix} \quad \begin{pmatrix} - & + & + \\ + & - & - \\ + & - & + \end{pmatrix} \quad \begin{pmatrix} - & + & + \\ + & + & - \\ + & - & - \end{pmatrix}$$

$$\begin{pmatrix} + & - & + \\ - & - & + \\ + & + & - \end{pmatrix} \quad \begin{pmatrix} + & - & + \\ - & + & - \\ + & - & - \end{pmatrix} \quad \begin{pmatrix} + & + & - \\ + & - & - \\ - & - & + \end{pmatrix} \quad \begin{pmatrix} + & + & - \\ + & - & + \\ - & + & - \end{pmatrix}$$

Super-classe 10 (6 éléments)

$$\begin{pmatrix} + & - & - \\ - & - & + \\ - & + & - \end{pmatrix} \quad \begin{pmatrix} - & + & - \\ + & - & - \\ - & - & + \end{pmatrix} \quad \begin{pmatrix} - & - & + \\ - & + & - \\ + & - & - \end{pmatrix}$$

$$\begin{pmatrix} - & + & + \\ + & + & - \\ + & - & + \end{pmatrix} \quad \begin{pmatrix} + & - & + \\ - & + & + \\ + & + & - \end{pmatrix} \quad \begin{pmatrix} + & + & - \\ + & - & + \\ - & + & + \end{pmatrix}$$

Super-classe 11 (2 éléments)

$$\begin{pmatrix} + & - & - \\ - & + & - \\ - & - & + \end{pmatrix} \quad \begin{pmatrix} - & + & + \\ + & - & + \\ + & + & - \end{pmatrix}$$

Super-classe 12 (12 éléments)

$$\begin{pmatrix} - & + & + \\ + & - & - \\ - & - & + \end{pmatrix} \quad \begin{pmatrix} - & + & - \\ + & - & + \\ - & - & + \end{pmatrix} \quad \begin{pmatrix} - & + & + \\ + & + & - \\ - & - & + \end{pmatrix} \quad \begin{pmatrix} - & + & + \\ - & + & - \\ + & - & - \end{pmatrix}$$

$$\begin{pmatrix} - & + & + \\ - & + & - \\ + & - & + \end{pmatrix} \quad \begin{pmatrix} - & - & + \\ - & + & - \\ + & + & - \end{pmatrix} \quad \begin{pmatrix} + & - & - \\ - & + & + \\ + & + & - \end{pmatrix} \quad \begin{pmatrix} + & - & + \\ - & + & - \\ + & + & - \end{pmatrix}$$

$$\begin{pmatrix} + & - & - \\ - & - & + \\ + & + & - \end{pmatrix} \quad \begin{pmatrix} + & - & - \\ + & - & + \\ - & + & + \end{pmatrix} \quad \begin{pmatrix} + & - & - \\ + & - & + \\ - & + & - \end{pmatrix} \quad \begin{pmatrix} + & + & - \\ + & - & + \\ - & - & + \end{pmatrix}$$

Super-classe 13 (4 éléments)

$$\begin{pmatrix} - & - & + \\ + & - & - \\ - & + & - \end{pmatrix} \quad \begin{pmatrix} - & + & - \\ - & - & + \\ + & - & - \end{pmatrix} \quad \begin{pmatrix} + & + & - \\ - & + & + \\ + & - & + \end{pmatrix} \quad \begin{pmatrix} + & - & + \\ + & + & - \\ - & + & + \end{pmatrix}$$

Chapitre 5

Une Règle Récursive

Dans ce chapitre, on montrera comment effectuer l'étape de $\bigwedge$-simplification pour des programmes avec une règle récursive.

Notons qu'il existe un seul chemin $\mathcal{P}$ entre le point (0) et (λ). Par conséquent, nous n'avons pas besoin ici de l'étape de $\bigvee$-simplification. Il s'ensuit que :

$$\mathcal{S}'(\lambda) = \mathcal{P} \quad \text{et} \quad \Delta(\lambda) = \Gamma_{\mathcal{P}}$$

Proposition 5.1 *La contrainte globale $\Gamma_{\mathcal{P}}$ d'un chemin entre les points (0) et (λ), avec $\lambda > 0$, équivaut à $\Phi_1(0)$ si $a'_1 \geq 0$, et équivaut à $\Phi_1(\lambda - 1)$ si $a'_1 < 0$.*

Preuve : Si $\forall \alpha \ \Phi_1(\alpha) \Rightarrow \Phi_1(\alpha + 1)$ (cas où $a'_1 \geq 0$), la contrainte la plus forte de $\mathcal{P}$ est la première contrainte du chemin, c'est-à-dire, $\Phi_1(0)$.

Si $\forall \alpha \ \Phi_1(\alpha + 1) \Rightarrow \Phi_1(\alpha + 1)$ (cas où $a'_1 \leq 0$), la contrainte la plus forte de $\mathcal{P}$ est la dernière contrainte du chemin, c'est-à-dire, $\Phi_1(\lambda)$. □

Exemple 5.2 Reprenons le programme π_1 de l'exemple 3.2 :

$p(x, y, z)$	:–	$\Theta(x, y, z)$.	règle $\mathcal{R}_0$
$p(x + 3, y + 2, z + 1)$	:–	$x - z > 0,\ p(x, y, z)$.	règle $\mathcal{R}_1$

Nous avons $a'_1 = 2$, donc $a'_1 \geq 0$.

Considérons le chemin $\mathcal{P} = \ll 0, \ldots, \lambda \gg$, avec $\lambda > 0$. On a $\Gamma_{\mathcal{P}}$ équivalent à $\Phi_1(0)$, c'est-à-dire, $x - z > 0$.

Rappelons qu'un atome $p(u, v, w)$ est engendré par l'évaluation ascendante de ce programme ssi :

$$\exists \lambda, x, y, z \quad \lambda \geq 0 \wedge \ u = x + 3\lambda \ \wedge \ v = y + 2\lambda \ \wedge \ w = z + \lambda \ \wedge \ \Theta(x, y, z) \ \wedge \ \Delta(\lambda).$$

En décomposant le cas où $\lambda = 0$ et $\lambda > 0$, et en utilisant le fait que :

- $\Delta(0)$, vaut vrai par définition.
- $\Delta(\lambda)$ avec $\lambda > 0$, vaut $x > y$.

on obtient que $p(u, v, w)$ ssi :

$$\exists \lambda, x, y, z \ ((u = x \ \wedge \ v = y \ \wedge \ w = z \ \wedge \Theta(x, y, z)) \vee$$
$$(\ \lambda > 0 \wedge \ u = x + 3\lambda \ \wedge \ v = y + 2\lambda \ \wedge \ w = z + \lambda \ \wedge \ \Theta(x, y, z) \ \wedge \ x > z \ ) \ ).$$

En éliminant les variables x, y et z, quantifiées existentiellement, on obtient $p(u, v, w)$ ssi : $\Theta(u, v, w) \ \vee \exists \lambda \ (\ \lambda > 0 \wedge \Theta(u - 3\lambda, v - 2\lambda, w - \lambda) \ \wedge u > w + 2\lambda \ )$.

□

Chapitre 6

Deux Règles Récursives - Cas Simples

Dans ce chapitre, on montrera comment effectuer des $\bigvee$-simplifications et des $\bigwedge$-simplifications pour 14 des 16 classes de programmes avec deux règles récursives. Ces classes sont appelées cas simples.

La $\bigvee$-simplification de $\Delta(\lambda,\mu)$ consiste à déterminer un sous-ensemble $\mathcal{S}'(\lambda,\mu)$ du carré $\mathcal{S}(\lambda,\mu)$, tel que tous les chemins dans $\mathcal{S}'(\lambda,\mu)$ ont une forme caractéristique et tel que :

$$\bigvee_{\mathcal{P}\in\mathcal{S}(\lambda,\mu)} \Gamma_{\mathcal{P}} \;=\; \bigvee_{\mathcal{P}\in\mathcal{S}'(\lambda,\mu)} \Gamma_{\mathcal{P}}$$

La $\bigwedge$-simplification consiste à réduire la contrainte globale de chaque chemin en $\mathcal{S}'(\lambda,\mu)$ à une formule arithmétique avec un nombre fixé d'atomes.

En appliquant ces deux étapes, on obtiendra une formule arithmétique pour exprimer le résultat de l'évaluation ascendante des programmes Π avec deux règles récursives.

Rappelons que les programmes Π que nous considérons sont de la forme :

$p(x,y,z)$	:–	$\Theta(x,y,z).$	règle $\mathcal{R}_0$
$p(x+a_1,y+b_1,z+c_1)$	:–	$\Phi_1(x,y,z),\ p(x,y,z)$	règle $\mathcal{R}_1$
$p(x+a_2,y+b_2,z+c_2)$	:–	$\Phi_2(x,y,z),\ p(x,y,z)$	règle $\mathcal{R}_2$

où $a_1,\ a_2,\ b_1,\ b_2,\ c_1,\ c_2\ \in \mathbb{Z}$, $\Theta(x,y,z)$ est une formule arithmétique linéaire et $\Phi_k(x,y,z)$ représente respectivement les contraintes $d_k x + e_k y + f_k z + g_k \geq 0$, avec $k \in \{1,2\}$ et $d_k, e_k, f_k, g_k\ \in \mathbb{Z}$.

Rappelons aussi que les signes des coefficients $a'_1,\ a'_2,\ b'_1,\ b'_2$ sont les éléments de la matrice caractéristique de Π.

6.1 $\vee$-simplification

En utilisant la notion d'implication de contraintes, pour certains programmes, nous pouvons simplifier la forme des chemins en $\mathcal{S}(\lambda,\mu)$, en considérant un ensemble des chemins $\mathcal{S}'(\lambda,\mu)$, tel que chaque chemin en $\mathcal{S}'(\lambda,\mu)$ soit contenu dans au maximum trois lignes droites et tel que :

$$\bigvee_{\mathcal{P}\in\mathcal{S}(\lambda,\mu,)} C_{\mathcal{P}} \;=\; \bigvee_{\mathcal{P}\in\mathcal{S}'(\lambda,\mu)} C_{\mathcal{P}}$$

Cette simplification est la conséquence d'un certaine combinaison de sens d'implications de contraintes du programme.

6.1.1 Cas Simple 1

Proposition 6.1 *Pour un programme Π avec $b'_1 \geq 0$ et $a'_2 \leq 0$ (correspondant à des implications de contraintes $\Rightarrow\Uparrow$), les chemins dans l'ensemble $\mathcal{S}'(\lambda,\mu)$ sont tous de la forme représentée à la figure 6.1 (à droite).*

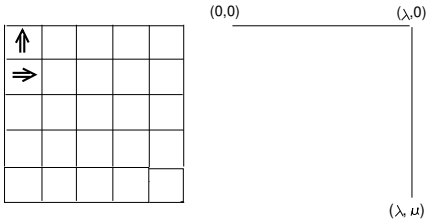


FIG. *6.1* - $b'_1 \geq 0$ et $a'_2 \leq 0$

Preuve: Considérons un chemin $\mathcal{P}$ quelconque reliant les points (0,0) et (λ, μ). Soit $\mathcal{P}'$ le chemin défini par la séquence $\ll (0,0), \ldots, (\lambda, 0), \ldots, (\lambda, \mu) \gg$. Nous prouvons ici que la contrainte globale $\Gamma_{\mathcal{P}}$ est subsumée par la contrainte globale $\Gamma_{\mathcal{P}'}$.

- Comme $\forall \alpha, \beta\ \Phi_1(\alpha, \beta+1) \Rightarrow \Phi_1(\alpha, \beta)$, chaque contrainte $\Phi_1(\rho, 0)$ de $\mathcal{P}'$ est plus faible que la contrainte $\Phi_1(\rho, \sigma)$ de $\mathcal{P}$. Il s'ensuit que $\Gamma^1_{\mathcal{P}} \Rightarrow \Gamma^1_{\mathcal{P}'}$.
- Comme $\forall \alpha, \beta\ \Phi_2(\alpha, \beta) \Rightarrow \Phi_2(\alpha+1, \beta)$, chaque contrainte $\Phi_2(\lambda, \sigma)$ de $\mathcal{P}'$ est plus faible que la contrainte $\Phi_2(\rho, \sigma)$ de $\mathcal{P}$. Il s'ensuit que $\Gamma^2_{\mathcal{P}} \Rightarrow \Gamma^2_{\mathcal{P}'}$.

Donc, $\Gamma_{\mathcal{P}} \Rightarrow \Gamma_{\mathcal{P}'}$. □

Pour un programme Π avec $b'_1 \leq 0$ et $a'_2 \geq 0$ (correspondant à des implications de contraintes $\Leftarrow\Downarrow$), les chemins dans l'ensemble $\mathcal{S}'(\lambda, \mu)$ sont de la forme représentée à la figure 6.2 (à droite).

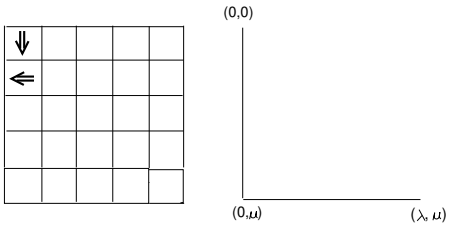


FIG. 6.2 - $b'_1 \leq 0$ *et* $a'_2 \geq 0$

La preuve est analogue à la précédente.

Exemple 6.2 Soit π_4 le programme:

$p(x, y, z)$	:–	$\Theta(x, y, z)$.	règle $\mathcal{R}_0$
$p(x+3, y, z+1)$	:–	$x - z > 0,\ p(x, y, z)$.	règle $\mathcal{R}_1$
$p(x, y+2, z+3)$	:–	$x - 2y > 0,\ p(x, y, z)$.	règle $\mathcal{R}_2$

On a:

$$
\begin{array}{rcrcrcr}
a'_1 & = & d_1a_1 + e_1b_1 + f_1c_1 & = & 1\times 3 + 0\times 0 + (-1)\times 1 & = & 2 \\
a'_2 & = & d_1a_2 + e_1b_2 + f_1c_2 & = & 1\times 0 + 0\times 2 + (-1)\times 3 & = & -3 \\
b'_1 & = & d_2a_1 + e_2b_1 + f_2c_1 & = & 1\times 3 + (-2)\times 0 + 0\times 1 & = & 3 \\
b'_2 & = & d_2a_2 + e_2b_2 + f_2c_2 & = & 1\times 0 + (-2)\times 2 + 0\times 3 & = & -4
\end{array}
$$

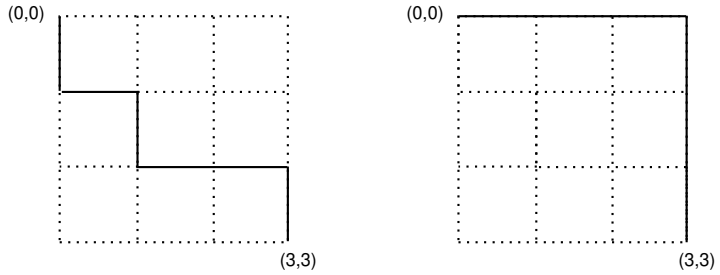


FIG. 6.3 - *Chemin* $\mathcal{P}$ *Chemin* $\mathcal{P}'$

Considérons, par exemple, le chemin $\mathcal{P} = \ll (0,0),(1,0),(1,1),(2,1),(2,2),(2,3),(3,3) \gg$ (voir figure 6.3 à gauche).

Soit $\mathcal{P}'$ le chemin $\ll (0,0),(1,0),(2,0),(3,0),(3,1),(3,2),(3,3) \gg$ (voir figure 6.3 à droite).

La contrainte globale $\Gamma_{\mathcal{P}}$ vaut $\Gamma^1_{\mathcal{P}} \wedge \Gamma^2_{\mathcal{P}}$ et la contrainte globale $\Gamma_{\mathcal{P}'}$ vaut $\Gamma^1_{\mathcal{P}'} \wedge \Gamma^2_{\mathcal{P}'}$, où :

$$\begin{array}{rcccccc}
\Gamma^1_{\mathcal{P}} & = & x-z-3>0 & \wedge & x-z-4>0 & \wedge & x-z-2>0 \\
 & & \Downarrow & & \Downarrow & & \Downarrow \\
\Gamma^1_{\mathcal{P}'} & = & x-z>0 & \wedge & x-z+2>0 & \wedge & x-z+4>0
\end{array}$$

$$\begin{array}{rcccccc}
\Gamma^2_{\mathcal{P}} & = & x-2y>0 & \wedge & x-2y-1>0 & \wedge & x-2y+1>0 \\
 & & \Downarrow & & \Downarrow & & \Downarrow \\
\Gamma^2_{\mathcal{P}'} & = & x-2y+9>0 & \wedge & x-2y+5>0 & \wedge & x-2y+1>0
\end{array}$$

Par conséquent, $\Gamma_{\mathcal{P}} \Rightarrow \Gamma_{\mathcal{P}'}$.

□

6.1.2 Cas Simple 2

Proposition 6.3 *Pour un programme* Π *avec* $a'_1 \geq 0$ *et* $b'_1 \geq 0$ *(correspondant à des implications de contraintes* $\overrightarrow{\Rightarrow}$*), les chemins dans l'ensemble* $\mathcal{S}'(\lambda,\mu)$ *sont tous de la forme représentée à la figure 6.4 (à droite).*

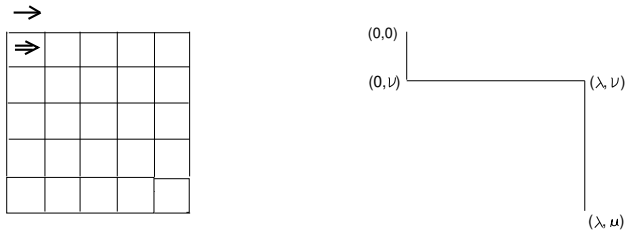


FIG. *6.4* - $a'_1 \geq 0$ et $b'_1 \geq 0$

Preuve: Considérons un chemin $\mathcal{P}$ reliant les points (0,0) et (λ, μ). Soit $N = (0, \nu)$ le point où se produit le premier pas en x de $\mathcal{P}$. Considérons le chemin $\mathcal{P}'$ défini par la séquence $\ll (0,0), \ldots, (0,\nu), \ldots, (\lambda,\nu), \ldots, (\lambda,\mu) \gg$. Nous prouvons ici que la contrainte globale $\Gamma_{\mathcal{P}}$ est subsumée par la contrainte globale $\Gamma_{\mathcal{P}'}$.

Les chemins $\mathcal{P}$ et $\mathcal{P}'$ coïncident jusqu'au point N. Après le point N, le chemin $\mathcal{P}'$ est formé par les points $\ll (0,\nu), \ldots, (\lambda,\nu), \ldots, (\lambda,\mu) \gg$.

- Comme $\forall \alpha, \beta \ \Phi_1(\alpha,\beta) \Rightarrow \Phi_1(\alpha+1,\beta)$, $\Gamma^1_{\mathcal{P}'}$ équivaut à $\Phi_1(0,\nu)$. Comme $\Phi_1(0,\nu)$ est une contrainte de $\mathcal{P}$, il s'ensuit que $\Gamma^1_{\mathcal{P}} \ \Rightarrow \Gamma^1_{\mathcal{P}'}$.
- Comme $\forall \alpha, \beta \ \Phi_2(\alpha,\beta) \Rightarrow \Phi_2(\alpha+1,\beta)$, chaque contrainte $\Phi_1(\lambda,\sigma)$ d'un point $\mathcal{P}'$ situé après le point N est plus faible que la contrainte $\Phi_2(\rho,\sigma)$ de $\mathcal{P}$. Il s'ensuit que $\Gamma^2_{\mathcal{P}} \ \Rightarrow \Gamma^2_{\mathcal{P}'}$.

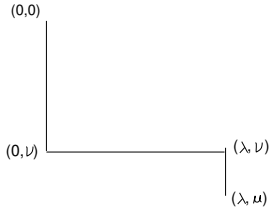
Par conséquent, $\Gamma_{\mathcal{P}} \Rightarrow \Gamma_{\mathcal{P}'}$. □

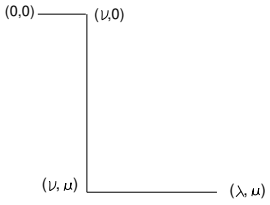
A la figure 6.5, on représente la forme des chemins dans l'ensemble $\mathcal{S}'(\lambda,\mu)$ pour les programmes avec $a'_1 \leq 0, \ b'_1 \leq 0$ (correspondant à des implications de contraintes $\overset{\leftarrow}{\Leftarrow}$), ou $a'_2 \geq 0, \ b'_2 \geq 0$ (correspondant à $\downarrow\Downarrow$), ou $a'_2 \leq 0, \ b'_2 \leq 0$ (correspondant à $\uparrow\Uparrow$). Ces preuves sont analogues à celle où $a'_1 \geq 0$ et $b'_1 \geq 0$.

Exemple 6.4 Soit π_5 le programme:

$p(x,y,z)$	:–	$\Theta(x,y,z)$.	règle $\mathcal{R}_0$
$p(x+3,y+2,z-1)$	:–	$2y-x>0, \ p(x,y,z)$.	règle $\mathcal{R}_1$
$p(x-1,y-2,z+2)$	:–	$z+x>0, \ p(x,y,z)$.	règle $\mathcal{R}_2$

On a:





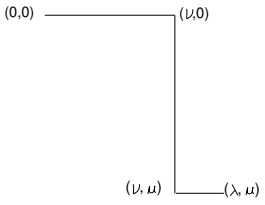


FIG. 6.5 - $a'_1 \leq 0,\ b'_1 \leq 0$ $\qquad$ $a'_2 \geq 0,\ b'_2 \geq 0$ $\qquad$ $a'_2 \leq 0,\ b'_2 \leq 0$

$$\begin{array}{rcccccl} a'_1 & = & d_1a_1 + e_1b_1 + f_1c_1 & = & -1\times 3 + 2\times 2 + 0\times(-1) & = & 1, \\ a'_2 & = & d_1a_2 + e_1b_2 + f_1c_2 & = & -1\times(-1) + 2\times(-2) + 0\times 2 & = & -3 \\ b'_1 & = & d_2a_1 + e_2b_1 + f_2c_1 & = & 1\times 3 + 0\times 2 + 1\times(-1) & = & 2 \\ b'_2 & = & d_2a_2 + e_2b_2 + f_2c_2 & = & 1\times(-1) + 0\times(-2) + 1\times 2 & = & 1 \end{array}$$

Considérons, par exemple, le chemin $\mathcal{P} = \ll (0,0),(0,1),(1,1),(1,2),(2,2),(3,2),(3,3) \gg$ (voir figure 6.6 à gauche).

Soit $\mathcal{P}'$ le chemin $\ll (0,0),(0,1),(1,1),(2,1),(3,1),(3,2),(3,3) \gg$ (voir figure 6.6 à droite).

La contrainte globale $\Gamma_{\mathcal{P}}$ vaut $\Gamma^1_{\mathcal{P}} \wedge \Gamma^2_{\mathcal{P}}$ et la contrainte globale $\Gamma_{\mathcal{P}'}$ vaut $\Gamma^1_{\mathcal{P}'} \wedge \Gamma^2_{\mathcal{P}'}$, où :

$$\begin{array}{rcl} \Gamma^1_{\mathcal{P}} & = & \underbrace{2y - x - 3 > 0 \wedge 2y - x - 5 > 0 \wedge 2y - x - 4 > 0}_{\Downarrow\ 2y - x - 3 > 0} \\ \Gamma^1_{\mathcal{P}'} & = & \overbrace{2y - x - 3 > 0 \wedge 2y - x - 2 > 0 \wedge 2y - x - 1 > 0}^{\Updownarrow} \end{array}$$

$$\begin{array}{rcccccc} \Gamma^2_{\mathcal{P}} & = & z + x > 0 & \wedge & z + x + 3 > 0 & \wedge & z + x + 8 > 0 \\ & & \Downarrow & & \Downarrow & & \Downarrow \\ \Gamma^2_{\mathcal{P}'} & = & z + x + 6 > 0 & \wedge & z + x + 7 > 0 & \wedge & z - x + 8 > 0 \end{array}$$

On a $\Gamma_{\mathcal{P}} \Rightarrow \Gamma_{\mathcal{P}'}$.

□

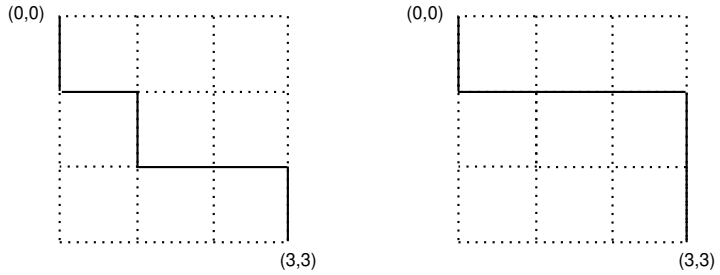


FIG. 6.6 - *Chemin* $\mathcal{P}$ *Chemin* $\mathcal{P}'$

Cas Simple 2 "optimisé"

Proposition 6.5 *Pour les programmes* Π *avec* $a'_1 \geq 0$, $a'_2 \geq 0$, $b'_1 \geq 0$ *et* $b'_2 \geq 0$ *(correspondant à des implications de contraintes* $\overrightarrow{\Rightarrow}$ $\downarrow\Downarrow$ *), on peut simplifier davantage la forme des chemins dans l'ensemble* $\mathcal{S}'(\lambda, \mu)$*, en considérant en* $\mathcal{S}'(\lambda, \mu)$ *seulement des chemins avec la forme représentée à la figure 6.7.*

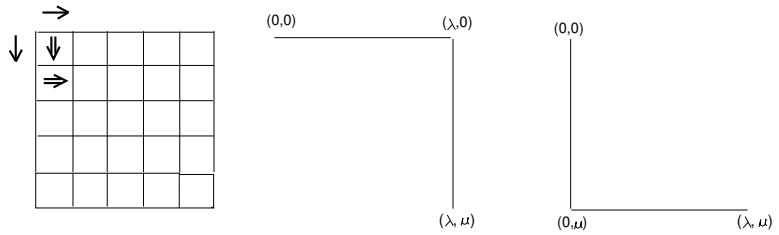


FIG. *6.7* - $a'_1 \geq 0$, $a'_2 \geq 0$, $b'_1 \geq 0$ et $b'_2 \geq 0$

Preuve: Considérons un chemin $\mathcal{P}$ reliant les points (0,0) et (λ, μ). On distinguera deux cas :

1. Le chemin $\mathcal{P}$ commence par un pas en x. Soit $\mathcal{P}'$ le chemin défini par la séquence $\ll (0,0), \ldots, (\lambda, 0), \ldots, (\lambda, \mu) \gg$. Nous prouvons ici que la contrainte globale $\Gamma_{\mathcal{P}}$ est subsumée par la contrainte globale $\Gamma_{\mathcal{P}'}$.
   - Comme $\Phi_1(\alpha, \beta) \Rightarrow \Phi_1(\alpha + 1, \beta)$, $\Gamma^1_{\mathcal{P}'}$ équivaut à $\Phi_1(0,0)$. Comme la contrainte $\Phi_1(0,0)$ est une contrainte de $\mathcal{P}$, il s'ensuit que $\Gamma^1_{\mathcal{P}} \Rightarrow \Gamma^1_{\mathcal{P}'}$.
   - Comme $\Phi_2(\alpha, \beta) \Rightarrow \Phi_2(\alpha + 1, \beta)$, chaque contrainte $\Phi_2(\lambda, \sigma)$ de $\mathcal{P}'$ est plus faible que la contrainte $\Phi_2(\rho, \sigma)$ de $\mathcal{P}$. Il s'ensuit que $\Gamma^2_{\mathcal{P}} \Rightarrow \Gamma^2_{\mathcal{P}'}$.

2. Le chemin $\mathcal{P}$ commence par un pas en y. On considère le chemin $\mathcal{P}'$ défini par la séquence $\ll (0,0),\ldots,(0,\mu),\ldots,(\lambda,\mu) \gg$. Comme dans le cas précédent, nous prouvons ici que $\Gamma_{\mathcal{P}}$ est subsumée par $\Gamma_{\mathcal{P}'}$.

   - Comme $\Phi_1(\alpha,\beta) \Rightarrow \Phi_1(\alpha,\beta+1)$, chaque contrainte $\Phi_1(\rho,\mu)$ de $\mathcal{P}'$ est plus faible que la contrainte $\Phi_1(\rho,\sigma)$ de $\mathcal{P}$. Il s'ensuit que $\Gamma^1_{\mathcal{P}} \Rightarrow \Gamma^1_{\mathcal{P}'}$.
   - Comme $\Phi_2(\alpha,\beta) \Rightarrow \Phi_2(\alpha,\beta+1)$, $\Gamma^2_{\mathcal{P}'}$ équivaut à $\Phi_2(0,0)$. Comme la contrainte $\Phi_2(0,0)$ est une contrainte de $\mathcal{P}$, il s'ensuit que $\Gamma^2_{\mathcal{P}} \Rightarrow \Gamma^2_{\mathcal{P}'}$.

Dans ces deux cas, $\Gamma_{\mathcal{P}} \Rightarrow \Gamma_{\mathcal{P}'}$. □

Pour les programmes avec $a'_1 \leq 0$, $a'_2 \leq 0$, $b'_1 \leq 0$ et $b'_2 \leq 0$ (correspondant à des implications de contraintes $\overleftarrow{\Leftarrow} \uparrow\Uparrow$), la forme des chemins en $\mathcal{S}'(\lambda,\mu)$ est la même que pour ceux avec $a'_1 \geq 0$, $a'_2 \geq 0$, $b'_1 \geq 0$ et $b'_2 \geq 0$ (correspondant à des implications de contraintes $\overrightarrow{\Rightarrow} \downarrow\Downarrow$).

6.2 $\bigwedge$-simplification

La $\bigwedge$-simplification consiste à réduire la contrainte globale de chaque chemin en $\mathcal{S}'(\lambda,\mu)$ à une formule arithmétique avec un nombre fixé d'atomes. Comme ces chemins sont contenus dans au maximum trois lignes droites, pour effectuer la $\bigwedge$-simplification de chaque contrainte globale, il suffit de regarder le sens des implications des flèches simples.

Considérons, par exemple, un programme II avec contraintes d'implications $\downarrow\leftarrow\Rightarrow\Uparrow$. De la sous-partie 6.1.1, on sait que tout chemin en $\mathcal{S}(\lambda,\mu)$ est subsumé par un chemin $\mathcal{P}'$ avec une forme représentée dans la figure 6.1. Le chemin $\mathcal{P}'$ est défini par la séquence $\ll (0,0),\ldots,(\lambda,0),\ldots,(\lambda,\mu) \gg$. Comme $\Phi_1(\alpha+1,\beta) \Rightarrow \Phi_1(\alpha,\beta)$ (flèche $\leftarrow$), $\Gamma^1_{\mathcal{P}'}$ équivaut à $\Phi_1(\lambda-1,0)$. Comme $\Phi_2(\alpha,\beta) \Rightarrow \Phi_2(\alpha,\beta+1)$ (flèche $\downarrow$), $\Gamma^2_{\mathcal{P}'}$ équivaut à $\Phi_2(\lambda,0)$. Il en résulte : $\Delta(\lambda,\mu) \equiv \Phi_1(\lambda-1,0) \wedge \Phi_2(\lambda,0)$.

Pour les 14 classes de programmes qui peuvent être traitées comme des cas "simples", la forme des chemins en $\mathcal{S}'(\lambda,\mu)$ est représentée aux figures 6.8, 6.9 et 6.10 . On a représenté par un trait plus épais les contraintes les plus fortes de ces chemins. Ces contraintes sont déterminées en fonction des sens des flèches simples.

Pour ces classes de programmes, la disjonction $\Delta(\lambda,\mu)$, pour $\lambda,\mu \in \mathbb{N}^*$, est équivalente à une formule arithmétique, donnée dans le tableau suivant.

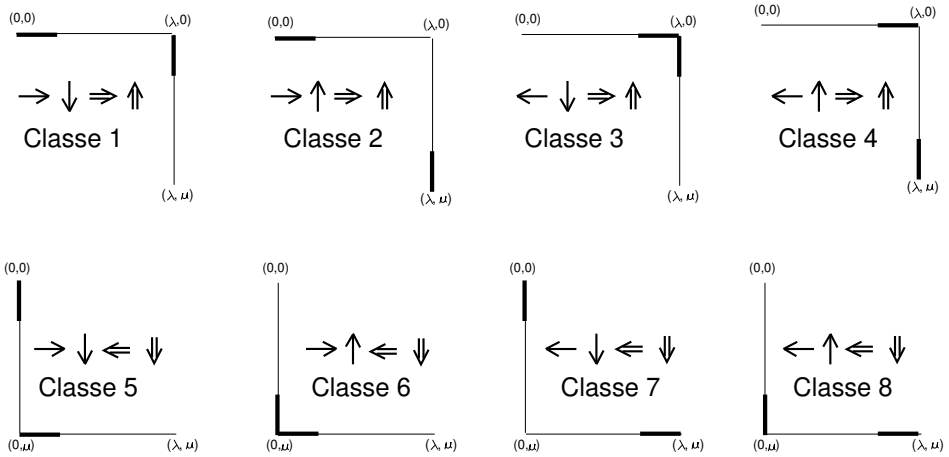


FIG. 6.8 - *Cas simples 1*

FIG. 6.9 - *Cas Simples 2 - "optimisé"*

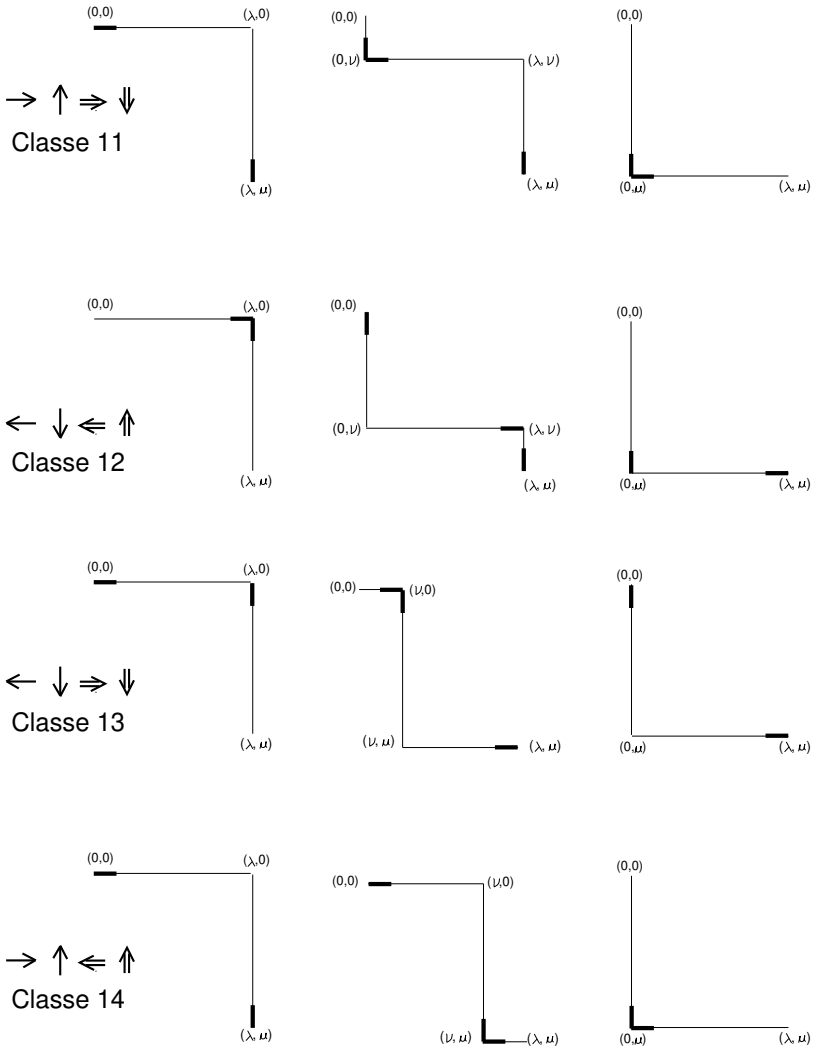


FIG. 6.10 - *Cas Simples 2*

Classe	$\Delta(\lambda,\mu)$, avec $\lambda,\mu>0$.
1	$\Phi_1(0,0)\wedge\Phi_2(\lambda,0)$
2	$\Phi_1(0,0)\wedge\Phi_2(\lambda,\mu-1)$
3	$\Phi_1(\lambda-1,0)\wedge\Phi_2(\lambda,0)$
4	$\Phi_1(\lambda-1,0)\wedge\Phi_2(\lambda,\mu-1)$
5	$\Phi_1(0,\mu)\wedge\Phi_2(0,0)$
6	$\Phi_1(0,\mu)\wedge\Phi_2(0,\mu-1)$
7	$\Phi_1(0,0)\wedge\Phi_2(\lambda-1,\mu)$
8	$\Phi_1(\lambda-1,\mu)\wedge\Phi_1(0,\mu-1)$
9	$(\Phi_1(0,0)\wedge\Phi_2(\lambda,0))\vee(\Phi_1(0,\mu)\wedge\Phi_2(0,0))$
10	$(\Phi_1(\lambda-1,0)\wedge\Phi_2(\lambda,\mu-1))\vee(\Phi_1(\lambda-1,\mu)\wedge\Phi_2(0,\mu-1))$
11	$\exists\nu\,((\Phi_1(0,0)\wedge\Phi_2(\lambda,\mu-1))\vee(\Phi_1(0,\mu)\wedge\Phi_2(0,\mu-1))\vee(\Phi_1(0,\nu)\wedge\Phi_2(0,\nu-1)\wedge\Phi_2(\lambda,\mu-1)))$
12	$\exists\nu\,((\Phi_1(\lambda-1,0)\wedge\Phi_2(\lambda,0))\vee(\Phi_1(\lambda-1,\mu)\wedge\Phi_2(0,0))\vee(\Phi_1(\lambda-1,\nu)\wedge\Phi_2(0,0)\wedge\Phi_2(\lambda,\nu)))$
13	$\exists\nu\,((\Phi_1(\lambda-1,\mu)\wedge\Phi_2(0,0))\vee(\Phi_1(\lambda-1,0)\wedge\Phi_2(\lambda,0))\vee(\Phi_1(\nu-1,0)\wedge\Phi_1(\lambda-1,\mu)\wedge\Phi_2(\nu,0)))$
14	$\exists\nu\,((\Phi_1(0,\mu-1)\wedge\Phi_2(0,\mu-1))\vee(\Phi_1(0,0)\wedge\Phi_2(\lambda,\mu-1))\vee(\Phi_1(0,0)\wedge\Phi_1(\nu,\mu)\wedge\Phi_2(\nu,\mu-1)))$

Pour les classes 11 et 12, dans la formule de $\Delta'(\lambda,\mu)$, la lettre ν désigne une variable quantifiée existentiellement qui satisfait à $0<\nu<\mu$. Chaque valeur de ν correspond à un chemin de $\mathcal{S}'(\lambda,\mu)$.

Pour les classes 13 et 14, la lettre ν désigne une variable quantifiée existentiellement qui satisfait à $0<\nu<\lambda$. Chaque valeur de ν correspond aussi à un chemin de $\mathcal{S}'(\lambda,\mu)$.

Rappelons qu'un atome $p(u,v,w)$ est engendré par l'évaluation ascendante de Π ssi :

$$\exists\lambda,\mu,x,y,z\quad \lambda\geq 0\wedge\mu\geq 0\wedge u=x+\lambda a_1+\mu a_2\ \wedge\ v=y+\lambda b_1+\mu b_2\ \wedge w=z+\lambda c_1+\mu c_2 \wedge\ \Theta(x,y,z)\ \wedge\ \Delta(\lambda,\mu).$$

En décomposant $\Delta(\lambda,\mu)$ suivant : $\lambda=0$ et $\mu=0$, $\lambda>0$ et $\mu=0$, $\lambda=0$ et $\mu>0$, et $\lambda>0$ et $\mu>0$, on obtient :

$$\begin{aligned}\exists\lambda,\mu,x,y,z\ \ (\ u=x\ \wedge\ v=y\ \wedge w=z\ \wedge\ \Theta(x,y,z)\ )\vee\\ (\ \lambda>0\wedge u=x+\lambda a_1\ \wedge\ v=y+\lambda b_1\ \wedge w=z+\lambda c_1\ \wedge\ \Theta(x,y,z)\ \wedge\ \Delta(\lambda,0)\ )\ \vee\\ (\ \mu>0\wedge u=x+\mu a_2\ \wedge\ v=y+\mu b_2\ \wedge w=z+\mu c_2\ \wedge\ \Theta(x,y,z)\ \wedge\ \Delta(0,\mu)\ )\ \vee\\ (\ \lambda>0\wedge\mu>0\wedge u=x+\lambda a_1+\mu a_2\ \wedge\ v=y+\lambda b_1+\mu b_2\ \wedge w=z+\lambda c_1+\mu c_2\ \wedge\\ \Theta(x,y,z)\ \wedge\ \Delta(\lambda,\mu)\ ).\end{aligned}$$

Le calcul de $\Delta(\lambda,\mu)$, pour $\lambda = 0$ ou $\mu = 0$, est fait comme dans le cas d'une règle récursive (voir chapitre 5).

Par conséquent, nous avons obtenu une une représentation par une formule arithmétique finie du résultat de l'évaluation ascendante de Π.

Exemple 6.6 (suite exemple 6.2) Comme $a'_2 \leq 0$ et $b'_1 \geq 0$, les chemins en $\mathcal{S}'(\lambda,\mu)$ pour π_1 ont la forme représentée à la figure 6.11. Soit $\mathcal{P}'$ le chemin $\ll (0,0),\ldots,(\lambda,0),\ldots,(\lambda,\mu) \gg$. Comme $\forall\alpha,\beta\ \Phi_1(\alpha,\beta) \Rightarrow \Phi_1(\alpha+1,\beta)$ (car $a'_1 \geq 0$), la contrainte horizontale la plus forte de $\mathcal{P}'$ est $\Phi_1(0,0)$. Donc $\Gamma^1_{\mathcal{P}'}$ équivaut à $x - z > 0$. Comme $\forall\alpha,\beta\ \Phi_2(\alpha,\beta+1) \Rightarrow \Phi_2(\alpha,\beta)$ (car $b'_2 \leq 0$), la contrainte verticale la plus forte de $\mathcal{P}'$ est $\Phi_2(\lambda,\mu-1)$, c'est-à-dire $2y - 3x - 3\lambda > 0$. Donc $\Gamma^2_{\mathcal{P}'}$ équivaut à $x + 3\lambda - 2y - 4\mu + 4 > 0$. Les contraintes les plus fortes de $\mathcal{P}'$ sont marquées avec un trait plus épais.

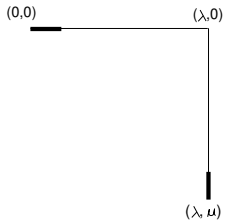


FIG. 6.11 - *Chemin $\mathcal{P}'$*

Il s'ensuit que, la disjonction $\Delta(\lambda,\mu)$, pour $\lambda > 0$ et $\mu > 0$, équivaut à :

$x - z > 0 \wedge x + 3\lambda - 2y - 4\mu + 4 > 0$.

Il est facile de vérifier que $\Delta(\lambda,0)$, pour $\lambda > 0$, équivaut à $x - z > 0$ et que $\Delta(0,\mu)$, pour $\mu > 0$, équivaut à $x - 2y - 4\mu + 4 > 0$.

Rappelons qu'un atome $p(u,v,w)$ est engendré par l'évaluation ascendante de π_1 ssi :

$\exists\lambda,\mu,x,y,z\ \ \lambda \geq 0 \wedge \mu \geq 0 \wedge u = x+3\lambda \wedge v = y+2\mu \wedge w = z+\lambda+3\mu \wedge \Theta(x,y,z) \wedge \Delta(\lambda,\mu)$.

En décomposant $\Delta(\lambda,\mu)$ avec $\lambda \geq 0$ et $\mu \geq 0$ en :

- $\Delta(0,0)$,
- $\Delta(\lambda,0)$ avec $\lambda > 0$,
- $\Delta(0,\mu)$ avec $\mu > 0$,

– $\Delta(\lambda, \mu)$ avec $\lambda > 0$ et $\mu > 0$,

on obtient :

$$\exists \lambda, \mu, x, y, z \; ((u = x \wedge v = y \wedge w = z \wedge \Theta(x, y, z)) \vee (\lambda > 0 \wedge u = x + 3\lambda \wedge v = y \wedge w = z + \lambda \wedge \Theta(x, y, z) \wedge x - z > 0) \vee (\mu > 0 \wedge u = x \wedge v = y + 2\mu \wedge w = z + 3\mu \wedge \Theta(x, y, z) \wedge x - 2y - 4\mu + 4 > 0) \vee (\lambda > 0 \wedge \mu > 0 \wedge u = x + 3\lambda \wedge v = y + 2\mu \wedge w = z + \lambda + 3\mu \wedge \Theta(x, y, z) \wedge x - z > 0 \wedge x + 3\lambda - 2y - 4\mu + 4 > 0)).$$

□

Exemple 6.7 (suite exemple 6.4) Comme $a'_1 \geq 0$ et $b'_1 \geq 0$, les chemins $\mathcal{P}'$ en $\mathcal{S}'(\lambda, \mu)$ pour π_2 ont les formes représentées à la figure 6.12. Comme $\forall \alpha, \beta \; \Phi_1(\alpha, \beta) \Rightarrow \Phi_1(\alpha + 1, \beta)$ (car $a'_1 \geq 0$), la contrainte horizontale la plus forte est $\Phi_1(0, \nu)$, c'est-à-dire $2y - x - 3\nu > 0$. Donc $\Gamma^1_{\mathcal{P}'}$ équivaut à $2y - x - 3\nu > 0$. Comme $\forall \alpha, \beta \; \Phi_2(\alpha, \beta) \Rightarrow \Phi_2(\alpha, \beta + 1)$ (car $b'_2 \geq 0$), pour la contrainte verticale, selon la valeur de ν, on a :

– Pour $\nu = 0$, la contrainte verticale la plus forte est $\Phi_2(\lambda, 0)$, qui vaut $z - \lambda + (x + 3\lambda) > 0$, c'est-à-dire, $z + x + 2\lambda > 0$. Dans ce cas, $\Gamma^2_{\mathcal{P}'}$ équivaut à $z + x + 2\lambda > 0$.

– Pour $0 < \nu < \mu$, la contrainte verticale la plus forte est $\Phi_2(0, 0)$ (qui vaut $z + x > 0$) ou $\Phi_2(\lambda, \nu)$ (qui vaut $z - \lambda + 2\nu + (x + 3\lambda + \nu) > 0$, c'est-à-dire, $z + x + 2\lambda + 3\nu > 0$). Dans ce cas, $\Gamma^2_{\mathcal{P}'}$ équivaut à $z + x > 0 \wedge z + x + 2\lambda + 3\nu > 0$, c'est-à-dire, $z + x > 0$.

– Pour $\nu = \mu$, la contrainte verticale la plus forte est $\Phi_2(0, 0)$, c'est à dire $z + x > 0$. Dans ce cas, $\Gamma^2_{\mathcal{P}'}$ équivaut à $z + x > 0$.

Il s'ensuit que, la disjonction $\Delta(\lambda, \mu)$, pour $\lambda > 0$ et $\mu > 0$, équivaut à :

$$\exists \, \nu \; (\; (\; 2y - x - 3\nu > 0 \wedge z + x + 2\lambda > 0 \wedge \nu = 0 \;) \vee \; (2y - x - 3\nu > 0 \wedge z - 2\nu + 2 - x > 0 \wedge z + x > 0 \wedge 0 < \nu < \mu \;) \;) \vee \; (\; 2y - x - 3\nu > 0 \wedge z + x > 0 \wedge \nu = \mu) \;)$$

En éliminant la variable ν, quantifiée existentiellement, on obtient, pour $\lambda > 0$ et $\mu > 0$, $\Delta(\lambda, \mu) \equiv \; (\; (\; 2y - x > 0 \wedge z + x + 2\lambda > 0 \;) \; \vee \; (\; 2y - x - 3 > 0 \wedge z + x > 0) \;)$.

Il est facile de vérifier que $\Delta(\lambda, 0)$, pour $\lambda > 0$, équivaut à $2y - x > 0$ et que $\Delta(0, \mu)$, pour $\mu > 0$, équivaut à $z + x > 0$.

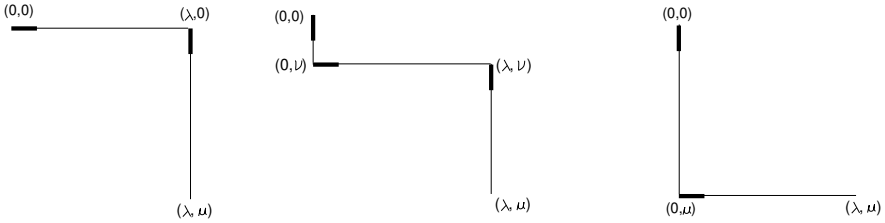


FIG. 6.12 - $\nu = 0$ $0 < \nu < \mu$ $\nu = \mu$

Rappelons qu'un atome $p(u, v, w)$ est engendré par l'évaluation ascendante de π_2 ssi :

$\exists \lambda, \mu, x, y, z \quad \lambda \geq 0 \wedge \mu \geq 0 \wedge u = x + 3\lambda - \mu \ \wedge \ v = y + 2\lambda - 2\mu \ \wedge \ w = z - \lambda + 2\mu \ \wedge \ \Theta(x, y, z) \ \wedge \ \Delta(\lambda, \mu)$.

En décomposant $\Delta(\lambda, \mu)$ avec $\lambda \geq 0$ et $\mu \geq 0$ en :

- $\Delta(0, 0)$,
- $\Delta(\lambda, 0)$ avec $\lambda > 0$,
- $\Delta(0, \mu)$ avec $\mu > 0$,
- $\Delta(\lambda, \mu)$ avec $\lambda > 0$ et $\mu > 0$,

on obtient :

$$\begin{aligned}\exists \lambda, \mu, x, y, z \ ((u = x \ \wedge \ v = y \ \wedge \ w = z \ \wedge \Theta(x, y, z)) \vee \\ (\ \lambda > 0 \ \wedge \ u = x + 3\lambda \ \wedge \ v = y + 2\lambda \ \wedge \ w = z - \lambda \ \wedge \ \Theta(x, y, z) \ \wedge \ 2y - x > 0) \ \vee \\ (\ \mu > 0 \ \wedge \ u = x - \mu \ \wedge \ v = y - 2\mu \ \wedge \ w = z + 2\mu \ \wedge \ \Theta(x, y, z) \ \wedge \ z + x > 0 \ ) \vee \\ (\ \lambda > 0 \wedge \mu > 0 \wedge u = x + 3\lambda - \mu \wedge v = y + 2\lambda - 2\mu \wedge w = z - \lambda + 2\mu \wedge \Theta(x, y, z) \wedge \\ (\ (\ 2y - x > 0 \ \wedge \ z + x + 2\lambda > 0 \ ) \ \vee \ (\ 2y - x - 3 > 0 \ \wedge \ z + x > 0) \ ) \ ).\end{aligned}$$

□

Chapitre 7

Deux Règles Récursives - Cas Difficiles

Dans ce chapitre, on montrera comment effectuer des $\bigvee$-simplifications et des $\bigwedge$-simplifications pour les programmes des cas restants, c'est-à-dire, pour les programmes où $a'_1 \geq 0,\ a'_2 \leq 0,\ b'_1 \leq 0,\ b'_2 \geq 0$ (correspondant à des implications de contraintes $\overrightarrow{\Leftarrow}\ \downarrow\Uparrow$) ou $a'_1 \leq 0,\ a'_2 \geq 0,\ b'_1 \geq 0,\ b'_2 \leq 0$ (correspondant à des implications de contraintes $\overleftarrow{\Rightarrow}\ \uparrow\Downarrow$).

Observons au préalable que les cas où $a'_1 = 0$, $a'_2 = 0$, $b'_1 = 0$ ou $b'_2 = 0$ peuvent toujours être traités comme un des cas simples du chapitre 6. Par exemple, si $a'_1 = 0$, on peut considérer que le signe de $a'_1 = 0$ soit le même que le signe de b'_2. Dans ce cas, on se retrouve dans le cas simple 1 du chapitre 6.

Dans ce chapitre, nous supposerons donc toujours que les coefficients a'_1, a'_2, b'_1 et b'_2 du programme II considéré sont non nuls.

Pour ces programmes, les chemins en $\mathcal{S}(\lambda, \mu)$ ne sont plus contenus dans un nombre fixe de lignes droites. Les $\bigvee$-simplifications et les $\bigwedge$-simplifications seront faites en utilisant la notion de *contrainte périodique* et la notion de *motif* associé à cette contrainte.

Rappelons d'abord que $\phi_k(x, y, z)$ est une contrainte de la forme $d_k x + e_k y + f_k z + g_k \geq 0$, que a'_k note $d_1 a_k + e_1 b_k + f_1 c_k$ et que b'_k note $d_2 a_k + e_2 b_k + f_2 c_k$ $(k \in \{1, 2\})$.

Il découle que, pour tout $\nu_1, \nu_2, \alpha, \beta \in \mathbb{N}$:

$$\Phi_1(\nu_1 + \alpha, \nu_2 + \beta) = \Phi_1(\nu_1, \nu_2) + \alpha a'_1 + \beta a'_2$$

$\Phi_2(\nu_1 + \alpha, \nu_2 + \beta) = \Phi_2(\nu_1, \nu_2) + \alpha b'_1 + \beta b'_2$

Ces résultats seront utilisés plusieurs fois dans ce chapitre.

Comme d'habitude, on notera par $|x|$ la valeur absolue d'un entier relatif x.

7.1 Période

De manière informelle, une contrainte Φ_k (pour $k = 1$ ou 2) est périodique si elle se répète après un certain nombre d'applications des règles $\mathcal{R}_1$ et $\mathcal{R}_2$. Plus formellement:

Définition 7.1 (Périodicité) *Une contrainte Φ_k (pour $k = 1$ ou 2) est* périodique *ssi il existe un vecteur non-nul $\vec{\sigma}$ d'entiers naturels de la forme (σ_1, σ_2), tel que $\Phi_k(0,0) = \Phi_k(\sigma_1, \sigma_2)$.*

On appelle période *de Φ_k un vecteur (τ_1, τ_2) non nul d'entiers naturels premiers entre eux tel que $\Phi_k(0,0) = \Phi_k(\tau_1, \tau_2)$.*

Notons que si $\Phi_k(0,0) = \Phi_k(\sigma_1, \sigma_2)$ alors $\forall \alpha, \beta\ \Phi_k(\alpha, \beta) = \Phi_k(\alpha + n\sigma_1, \beta + n\sigma_2)$, pour tout $n \in \mathbb{N}$.

Exemple 7.2 Reprenons le programme de l'exemple 3.4 :

$p(x, y, z)$	:–	$\Theta(x, y, z)$.	règle $\mathcal{R}_0$
$p(x+3, y, z+1)$	:–	$x - z > 0,\ p(x, y, z)$.	règle $\mathcal{R}_1$
$p(x, y-2, z+3)$	:–	$2y - x > 0,\ p(x, y, z)$.	règle $\mathcal{R}_2$

La contrainte $\Phi_1(\alpha, \beta)$ vaut $x + 3\alpha - (z + \alpha + 3\beta) > 0$, c'est-à-dire : $x + 2\alpha > z + 3\beta$.

La contrainte $\Phi_1(0,0)$ vaut $x > z$ et la contrainte $\Phi_1(3,2)$ vaut $x + 2 \times 3 > z + 3 \times 2$, c'est-à-dire $x > z$. Par conséquent, $\Phi_1(0,0) = \Phi_1(3,2)$. On en conclut que Φ_1 est périodique.

La contrainte $\Phi_2(\alpha, \beta)$ vaut $2y - 4\beta - (x + 3\alpha) > 0$, c'est-à-dire : $2y > x + 3\alpha + 4\beta$.

La contrainte $\Phi_2(0,0)$ vaut $2y > x$. Par conséquent, $\Phi_2(0,0) < \Phi_2(\alpha, \beta)$, pour tout $(\alpha, \beta) \neq (0,0)$. Donc, Φ_2 n'est pas périodique.

□

Proposition 7.3 (Périodicité de Φ_1) *Pour un programme* II *donné, la contrainte* Φ_1 *est périodique ssi* $a'_1 \times a'_2 < 0$. *De plus, la période associée à* Φ_1 *vaut* $(\frac{|a'_2|}{pgcd\{|a'_2|,|a'_1|\}}, \frac{|a'_1|}{pgcd\{|a'_2|,|a'_1|\}})$.

Preuve : On a :

$\forall \alpha, \beta\ \Phi_1(\alpha,\beta) = \Phi_1(0,0) + \alpha a'_1 + \beta a'_2$. Donc Φ_1 est périodique ssi l'équation $\alpha a'_1 + \beta a'_2 = 0$ admet une solution solution non triviale d'entiers naturels en (α, β). Cela est le cas ssi $a'_1 \times a'_2 < 0$.

La solution minimale non triviale pour $\alpha a'_1 + \beta a'_2 = 0$ nous donne la période de Φ_1. Cette solution est $(\frac{|a'_2|}{\text{pgcd}\{|a'_2|,|a'_1|\}}, \frac{|a'_1|}{\text{pgcd}\{|a'_2|,|a'_1|\}})$. □

Remarquons que la période associée à une contrainte périodique Φ_1 est unique.

Exemple 7.4 (suite exemple 7.2) Comme $a'_1 = 2$ et $a'_2 = -3$ (voir exemple 3.4) et pgcd{3,2}=1, la période pour Φ_1 est (3,2). □

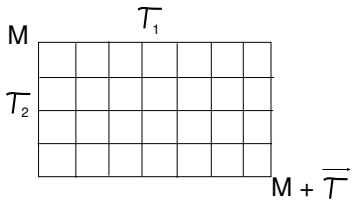
De même, pour la contrainte Φ_2 :

Proposition 7.5 (Périodicité de Φ_2) *Pour un programme* II *donné, la contrainte* Φ_2 *est périodique ssi* $b'_1 \times b'_2 < 0$. *De plus, la période associée à* Φ_2 *vaut* $(\frac{|b'_2|}{pgcd\{|b'_2|,|b'_1|\}}, \frac{|b'_1|}{pgcd\{|b'_2|,|b'_1|\}})$.

7.2 Motif et Chemin-Motif

Soit M un point de coordonnées (ν_1, ν_2) et soit $\vec{\sigma}$ le vecteur (σ_1, σ_2). L'expression $M + \vec{\sigma}$ représente le vecteur $(\nu_1 + \sigma_1, \nu_2 + \sigma_2)$.

Définition 7.6 (Motif) *Soit* Φ_k *une contrainte périodique de période* $\vec{\tau}$. *Le motif associé à* Φ_k *est l'ensemble de tous les chemins reliant un point donné* M *au point* $M + \vec{\tau}$. *(voir figure 7.1).*



$$\Phi_k(M) = \Phi_k(M + \vec{\tau})$$

FIG. *7.1* - Motif pour Φ_k

Etant donné la période $\vec{\tau}$ pour Φ_k et un vecteur $\vec{\tau'}$ ($\vec{0} \leq \vec{\tau'} \leq \vec{\tau}$), l'ensemble de tous les chemins reliant un point donné M au point $M + \vec{\tau'}$ est appelé *sous-motif* de $\vec{\tau}$.

Proposition 7.7 *Pour un programme* Π *avec une contrainte périodique* Φ_1*, tous les points d'une même contrainte* Φ_1 *sont situés sur une ligne droite de coefficient angulaire positif de valeur* $\frac{-a'_1}{a'_2}$.

Preuve : Considérons deux points M et $M + (\nu_1, \nu_2)$ avec une même contrainte horizontale. Nous avons :

$\Phi_1(M + (\nu_1, \nu_2)) = \Phi_1(M) + \nu_1 a'_1 + \nu_2 a'_2$.

Comme $\Phi_1(M + (\nu_1, \nu_2)) = \Phi_1(M)$, il s'ensuit que $\nu_1 a'_1 + \nu_2 a'_2 = 0$, et par conséquent :

$$\frac{\nu_2}{\nu_1} = \frac{-a'_1}{a'_2}$$

Il s'ensuit que le rapport $\frac{\nu_2}{\nu_1}$ est constant : tous les points de même contrainte Φ_1 sont donc situés sur une ligne droite de coefficient angulaire égal à cette constante, c'est-à-dire, $\frac{-a'_1}{a'_2}$. Comme Φ_1 est périodique, on a d'après la proposition 7.3 : $a'_1 \times a'_2 < 0$. Le coefficient angulaire $\frac{-a'_1}{a'_2}$ est donc positif. □

Proposition 7.8 *Soit* Π *un programme avec une contrainte périodique* Φ_1 *et* M *un point de* $\mathbb{N} \times \mathbb{N}$*. La ligne droite qui relie tous les points de contrainte horizontale équivalente à* $\Phi_1(M)$ *divise le plan* $\mathbb{N} \times \mathbb{N}$ *en deux parties :*

- *L'une où toutes les contraintes horizontales sont strictement plus fortes que* $\Phi_1(M)$.

– *l'autre où toutes les contraintes horizontales sont strictement plus faibles que* $\Phi_1(M)$.

De plus, si $a'_1 > 0$ *(resp.* $a'_1 < 0$*), la partie où on trouve les points de contraintes strictement plus fortes est celle située strictement au-dessus (resp. strictement en dessous) de cette ligne droite.*

Exemple 7.9 Pour le programme de l'exemple 7.2, les points avec même contrainte horizontale sont situés sur une ligne droite de coefficient $\frac{2}{3}$ (voir figure 7.2), car $a'_1 = 2$ et car $a'_2 = -3$ (voir exemple 3.4).

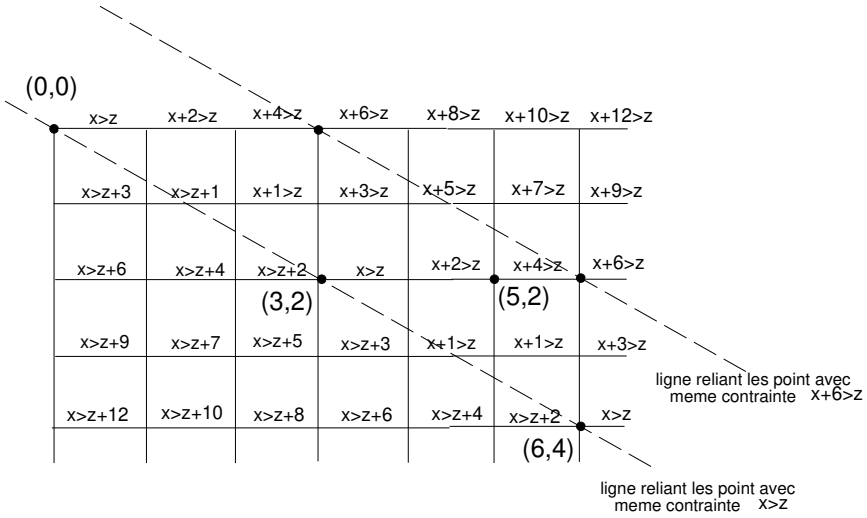


FIG. 7.2 - *Points avec même contrainte horizontale*

Le point (5,2) est situé en dessous de la ligne droite que relie les points de contrainte horizontale $x + 6 > z$. En conformité avec la proposition 7.8 (cas $a'_1 > 0$), on vérifie que la contrainte $\Phi_1(5, 2)$, qui vaut $x + 4 > z$ est plus forte que $x + 6 > z$. D'autre part, ce même point est au-dessus de la ligne droite que relie les points de contrainte horizontale $x > z$. On vérifie de même que la contrainte $\Phi_1(5, 2)$ est en plus faible que $x > z$. □

De même, pour la contrainte Φ_2 :

Proposition 7.10 *Pour un programme* II *avec une contrainte périodique* Φ_2*, tous les points de même contrainte* Φ_2 *sont situés sur une ligne droite de coefficient angulaire positif de valeur* $\frac{-b'_1}{b'_2}$.

Proposition 7.11 *Soit* Π *un programme avec une contrainte périodique* Φ_2 *et* M *un point de* $\mathbb{N}\times\mathbb{N}$. *La ligne droite qui relie tous les points de contrainte verticale équivalente à* $\Phi_2(M)$ *divise le plan* $\mathbb{N}\times\mathbb{N}$ *en deux parties:*

- *L'une où toutes les contraintes verticales sont strictement plus fortes que* $\Phi_2(M)$.
- *l'autre où toutes les contraintes verticales sont strictement plus faibles que* $\Phi_2(M)$.

Un motif $\vec{\tau}$ représente l'ensemble des chemins $\mathcal{P}$ reliant un point M donné au point $M+\vec{\tau}$. La proposition 7.12 établit l'existence d'une "meilleure façon" de relier les points M et $M+\vec{\tau}$, lorsque l'on ne passe pas par des points de contraintes plus fortes que $\Phi_k(M)$. Par meilleure façon, on entend la façon de relier les points M et $M+\vec{\tau}$ qui rend la conjonction $\Gamma_{\mathcal{P}}$ la plus faible possible. Pour cela, il faut rendre la deuxième contrainte Φ_j $(j\neq k)$ aussi faible que possible.

Proposition 7.12 (Chemin-Motif) *Soit* Π *un programme avec une contrainte* Φ_k *(*$k=1$ *ou* 2*) périodique,* $\vec{\tau}$ *le motif de* Φ_k *et* M *un chemin quelconque. Il existe un chemin, noté* $\mathcal{P}_{motif}$, *commençant en un point* M *et finissant en un point* $M+\vec{\tau}$, *qui satisfait les deux conditions dessous:*

1. $\Gamma^k_{\mathcal{P}_{motif}} = \Phi_k(M)$.
2. *Pour tout chemin* $\mathcal{P}$ *entre* M *et* $M+\vec{\tau}$ *tel que* $\Gamma^k_{\mathcal{P}} \Rightarrow \Phi_k(M)$, *on a* $\Gamma^j_{\mathcal{P}} \Rightarrow \Gamma^j_{\mathcal{P}_{motif}}$ $(j\neq k)$.

Ce chemin est dit *chemin-motif* de $\vec{\tau}$. Sa forme ne dépend pas du point M considéré.

On appelle *préfixe d'un chemin-motif* le début d'un chemin-motif et on appelle *suffixe d'un chemin-motif* la fin d'un chemin-motif.

Nous présentons maintenant un algorithme pour construire un chemin-motif $\mathcal{P}_{\text{motif}}$ associé au motif $\vec{\tau}$ pour la contrainte Φ_1. Un chemin-motif est déterminé de façon unique par ce algorithme. Il y a deux cas à considérer : $a'_1 > 0$ et $a'_1 < 0$.

Pour le cas où $a'_1 > 0$, on considère un chemin, reliant les point M et $M+(\tau_1,\tau_2)$, composé par une séquence de τ_1 pas en x, suivi d'une séquence de τ_2 pas en y. Ensuite :

- si le programme a l'implication de contrainte $\Rightarrow (b_1 < 0)$, les contraintes Φ_2 sont déjà aussi faibles que possible (voir figure 7.3).

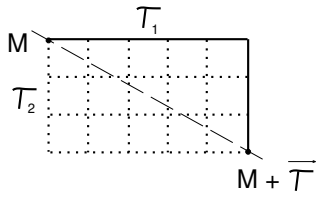


FIG. 7.3 - *Chemin-Motif pour* $a'_1 > 0$ *et* $b'_1 > 0$

– si le programme a l'implication de contrainte $\Leftarrow (b_1 < 0)$, on transforme itérativement ce chemin dans un autre, en se rapprochant le plus possible de la ligne droite joignant les points M et $M + (\tau_1, \tau_2)$, sans la couper (voir figure 7.5).

La non coupure de la droite M-$M + (\tau_1, \tau_2)$ garantit que le chemin est contenu dans la partie du plan où les contraintes sont plus faibles que $\Phi_1(M)$ (voir proposition 7.8). Chaque transformation élémentaire (voir figure 7.4) a pour effet d'affaiblir la contrainte verticale : à cause de l'implication $\Leftarrow (b_1 < 0)$, après transformation, la contrainte verticale $\Phi_2(\alpha - 1, \beta)$ est strictement plus faible que la contrainte verticale initiale $\Phi_2(\alpha, \beta)$.

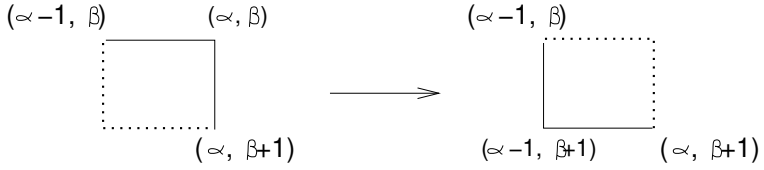


FIG. 7.4 - *Transformation élémentaire pour* $a'_1 > 0$ *et* $b'_1 < 0$

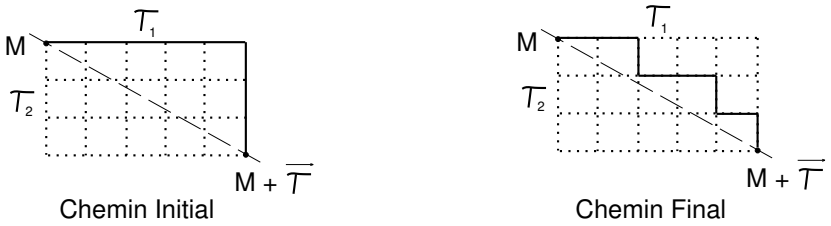


FIG. 7.5 - *Construction Chemin-Motif pour* $a'_1 > 0$ *et* $b'_1 < 0$

Plus formellement, pour le cas où $a'_1 > 0$, l'algorithme pour l'obtention d'un chemin-motif associé à Φ_1 est :

1. Si $b'_1 > 0$:
   $\mathcal{P}_{\text{motif}} = \ll M, M+(1,0), \ldots, M+(\tau_1,0), M+(\tau_1,1), \ldots, M+(\tau_1,\tau_2) \gg$. fin
2. Si $b'_1 < 0$ (voir exemple 7.13):
   - Initialiser $\mathcal{Q}$ avec le chemin:
     $\ll M, M+(1,0), \ldots, M+(\tau_1,0), M+(\tau_1,1), \ldots, M+(\tau_1,\tau_2) \gg$.
   - Tant qu'il existe un fragment de chemin $(\alpha-1,\beta),(\alpha,\beta),(\alpha,\beta+1)$ dans $\mathcal{Q}$ tel que $\Phi_1(\alpha-1,\beta+1) \geq \Phi_1(M)$, transformer le fragment en $(\alpha-1,\beta),(\alpha-1,\beta+1),(\alpha,\beta+1)$ (voir figure 7.5).
   - $\mathcal{P}_{\text{motif}} = \mathcal{Q}$. fin

Ce algorithme fait un choix non-déterministe ("don't care") du fragment où on applique la transformation élémentaire, au cas où il en existe plusieurs. Toutefois, il est facile de voir que le système de transformation est confluent et que l'algorithme détermine de façon unique un chemin-motif.

Par construction du chemin-motif, on a facilement (cf. proposition 7.8) :

Proposition 7.13 *Soit* Π *un programme avec* $a'_1 > 0$ *et* $a'_2 < 0$, *et* $\vec{\tau}$ *le motif de la contrainte* Φ_1. *Soit* $\mathcal{P}_{motif}$ *le chemin-motif entre un point* M *quelconque et* $M+\vec{\tau}$, *déterminé par l'algorithme ci-dessus. Considérons un point* M', *tel que* $M < M' < M+\vec{\tau}$. *On a:*

- *Si le point* M' *est situé strictement au-dessous de* $\mathcal{P}_{motif}$, *alors* $\Phi(M')$ *est strictement plus faible que* $\Phi_1(M)$.
- *Si le point* M' *est situé strictement au-dessus de* $\mathcal{P}_{motif}$, *alors* $\Phi(M')$ *est strictement plus fort que* $\Phi_1(M)$.

Exemple 7.14 Reprenons le programme de l'exemple 3.4. Pour ce programme, nous avons $a'_1 = 2 > 0$ et $b'_1 = -3 < 0$. Des exemples 7.2 et 7.4, nous savons que ϕ_1 est périodique avec motif (3,2).

Considérons un point $M = (\alpha, \beta)$. Nous voulons obtenir le chemin-motif $\mathcal{P}_{\text{motif}}$ qui relie le point M au point $M+(3,2)$.

Le chemin $\mathcal{Q}$ est initialisé avec :

$\ll M, M+(1,0), M+(2,0), M+(3,0), M+(3,1), M+(3,2) \gg$.

Dans $\mathcal{Q}$, on remplace le point $M+(3,0)$ par $M+(3-1,0+1) \equiv M+(2,1)$. Ceci est possible car $\Phi_1(M+(2,1)) \geq \Phi_1(M)$. En effet :

$$\begin{array}{lcl} \Phi_1(M+(2,1)) & \equiv & x+2\alpha+4 > z+3\beta \\ & & \Uparrow \\ \Phi_1(M) & \equiv & x+2\alpha > z+3\beta \end{array}$$

On obtient ainsi le chemin $\mathcal{Q}' = \ll M, M+(1,0), M+(2,0), M+(2,1), M+(3,1), M+(3,2) \gg$.

Montrons qu'on ne peut pas appliquer davantage la transformation élémentaire de la figure 7.4.

- On ne peut pas remplacer $M+(2,0)$ par $M+(1,1)$, car $\Phi_1(M+(1,1)) < \Phi_1(M)$. En effet :

$$\begin{array}{lcl} \Phi_1(M+(1,1)) & \equiv & x+2\alpha > z+3\beta+1 \\ & & \Downarrow \\ \Phi_1(M) & \equiv & x+2\alpha > z+3\beta \end{array}$$

- On ne peut pas remplacer $M+(3,1)$ par $M+(2,2)$, car $\Phi_1(M+(2,2)) < \Phi_1(M)$. En effet :

$$\begin{array}{lcl} \Phi_1(M+(2,2)) & \equiv & x+2\alpha > z+3\beta+2 \\ & & \Downarrow \\ \Phi_1(M) & \equiv & x+2\alpha > z+3\beta \end{array}$$

Donc, aucune transformation élémentaire ne peut être effectuée. Géométriquement, ceci correspond au fait que toute nouvelle transformation ferait couper la droite M-$M+(3,2)$.

Le chemin-motif $\mathcal{P}_{\text{motif}}$ (voir figure 7.6) est ainsi égal à $\ll M, M+(1,0), M+(2,0), M+(2,1), M+(3,1), M+(3,2) \gg$.

□

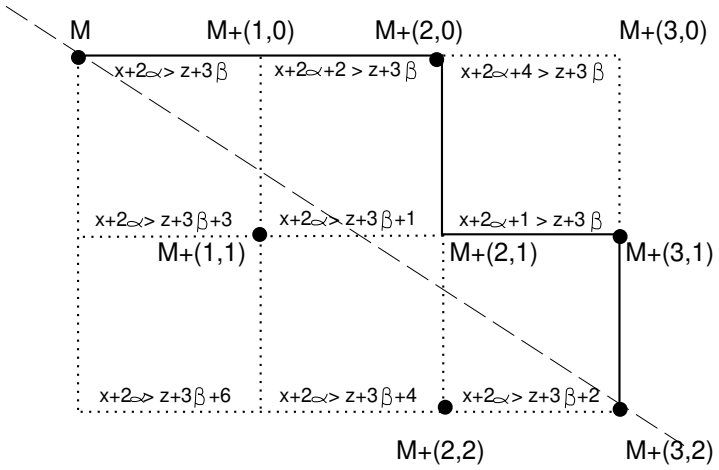


FIG. 7.6 - *Chemin-motif*

Pour le cas où $a'_1 < 0$, l'algorithme pour l'obtention d'un chemin-motif associé à Φ_1 est :

1. Si $b'_1 < 0$:
   $\mathcal{P}_{\text{motif}} = \ll M, M+(1,0), M+(1,1), \ldots, M+(1,\tau_2), M+(2,\tau_2), \ldots, M+(\tau_1,\tau_2) \gg$.
   fin

2. Si $b'_1 > 0$:

   - Initialiser $\mathcal{Q}$ avec le chemin:
     $\ll M, M+(1,0), M+(1,1), \ldots, M+(1,\tau_2), M+(2,\tau_2), \ldots, M+(\tau_1,\tau_2) \gg$.
   - Tant qu'il existe un fragment de chemin $(\alpha,\beta-1),(\alpha,\beta),(\alpha+1,\beta) \in \mathcal{Q}$ tel que $\Phi_1(\alpha,\beta-1) \geq \Phi_1(M)$, transformer le fragment en $(\alpha,\beta-1),(\alpha+1,\beta-1),(\alpha+1,\beta)$.
   - $\mathcal{P}_{\text{motif}} = \mathcal{Q}$. fin

7.3 $\bigvee$-simplification

Dans cette partie, nous présentons une méthode basée sur la notion de motif pour effectuer l'étape de $\bigvee$-simplification. On se limitera ici à la classe $a'_1 > 0,\ a'_2 < 0,\ b'_1 < 0,\ b'_2 > 0$. Des résultats symétriques peuvent être établis pour la classe $a'_1 < 0,\ a'_2 > 0,\ b'_1 > 0,\ b'_2 < 0$.

D'après la proposition 7.3, comme $a'_1 > 0$ et $a'_2 < 0$, la contrainte Φ_1 est périodique et le motif associé à cette contrainte est $(\frac{-a'_2}{\text{pgcd}\{-a'_2, a'_1\}}, \frac{a'_1}{\text{pgcd}\{-a'_2, a'_1\}})$.

On notera $a''_1 = \frac{a'_1}{\text{pgcd}\{-a'_2, a'_1\}}$ et $a''_2 = \frac{a'_2}{\text{pgcd}\{-a'_2, a'_1\}}$. Donc, le motif associé à Φ_1 et $(-a''_2, a''_1)$.

Le chemin-motif $\mathcal{P}_{\text{motif}}$ associé à ce motif est donné par l'algorithme de la sous-partie 7.2.

Lemme 7.15 *Soit* Π *un programme avec coefficients* $a'_1 < 0,\ a'_2 > 0,\ b'_1 > 0,\ b'_2 < 0$. *Etant donné un chemin* $\mathcal{P}$, *soit* $M_h = (\nu_1, \nu_2)$ *le point où on trouve la contrainte horizontale la plus forte de* $\mathcal{P}$. *On a* $\nu_1 a'_1 + \nu_2 a'_2 \geq 0$.

Preuve : Soit $M' = (0, \nu)$ le point de $\mathcal{P}$ où le premier pas en x de $\mathcal{P}$ apparaît. On a :

$$\left.\begin{array}{l} \Phi_1(\nu_1, \nu_2) = \Phi_1(0,0) + \nu_1 a'_1 + \nu_2 a'_2. \\ \Phi_1(0,0) \geq \Phi_1(0,\nu) \quad (\text{ car } a'_2 < 0). \end{array}\right\} \Longrightarrow \Phi_1(\nu_1, \nu_2) \geq \Phi_1(0,\nu) + \nu_1 a'_1 + \nu_2 a'_2.$$

Comme $\Phi_1(\nu_1, \nu_2)$ est la contrainte horizontale la plus forte de $\mathcal{P}$, il s'ensuit que $\nu_1 a'_1 + \nu_2 a'_2 \geq 0$. □

Corollaire 7.16 *Soit* Π *un programme avec coefficients* $a'_1 < 0,\ a'_2 > 0,\ b'_1 > 0,\ b'_2 < 0$ *et* $M_h = (\nu_1, \nu_2)$ *le point où on trouve la contrainte horizontale la plus forte d'un chemin* $\mathcal{P}$. *La ligne droite que relie les points avec contrainte* $\Phi_1(M_h)$ *coupe l'axe* $0Y$, *passant par* $(0,0)$, *en un point de coordonnées* $(0, y_0)$, *avec* $y_0 \geq 0$.

Preuve : Soit $M = (\nu_1, \nu_2)$. Le coefficient angulaire de la ligne qui relie les points avec contrainte $\Phi_1(M_h)$ est $\frac{-a'_2}{a'_1}$. D'après le lemme 7.15, on a $\nu_1 a'_1 + \nu_2 a'_2 \geq 0$. Il s'ensuit que la ligne qui relie les points avec contrainte $\Phi_1(M_h)$ coupe l'axe $0Y$, passant par le point $(0,0)$, en un point de coordonnées $(0, y_0)$, avec $y_0 \geq 0$. Remarquons que y_0 n'est pas nécessairement un entier. □

Théorème 7.17 *Pour un programme* Π *avec* $a'_1 > 0,\ a'_2 < 0,\ b'_1 < 0,\ b'_2 > 0$, *la forme des chemins en* $\mathcal{S}'(\lambda, \mu)$ *est représentée dans la figure 7.7.*

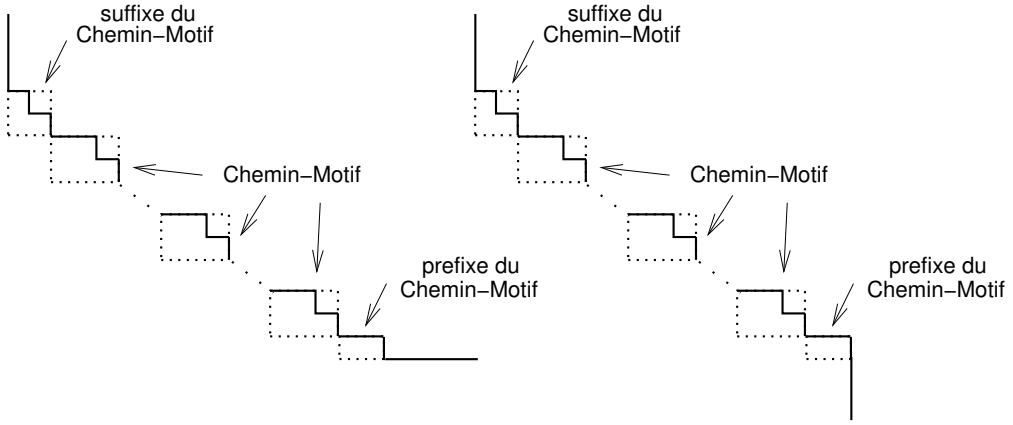


FIG. 7.7 - *Forme des chemins en* $\mathcal{S}'(\lambda, \mu)$

Preuve : Etant donné un chemin quelconque $\mathcal{P}$ de $\mathcal{S}(\lambda, \mu)$, on construira un chemin $\mathcal{P}'$ de la forme représentée dans la figure 7.7, tel que la contrainte globale de $\mathcal{P}'$ subsume la contrainte globale de $\mathcal{P}$.

Construction du chemin $\mathcal{P}'$:

D'après la proposition 7.3, la contrainte Φ_1 de la règle $\mathcal{R}_1$ de Π est périodique avec motif $(-a_2'', a_1'')$. Soit $\mathcal{P}_{\text{motif}}$ le chemin-motif associé à cette contrainte et soit M_h le point de $\mathcal{P}$ où la contrainte horizontale la plus forte de $\mathcal{P}$ apparaît.

Le chemin $\mathcal{P}'$ est obtenu (voir figure 7.10) de la façon suivante :

- Au-dessus du point M_h, on compose le chemin-motif $\mathcal{P}_{\text{motif}}$, en partant du point M_h et en remontant vers l'origine (0,0). Cela détermine le point N quand on rencontre l'axe $0y$ (voir figure 7.8 à gauche) ou l'axe $0x$ (voir figure 7.8 à droite).

  Toutefois, par le corollaire 7.16, la situation illustrée dans la figure 7.8 à droite ne se produit jamais. Par conséquent, N est le point de $\mathcal{P}'$ où on trouve le premier pas en x.

- On relie le point (0,0) et le point N (si $N \neq (0,0)$) par des pas en y.

- En dessous du point M_h, on compose également le chemin-motif $\mathcal{P}_{\text{motif}}$, en partant du point M_h et en descendant vers le point (λ, μ). Cela détermine le point Q quand on

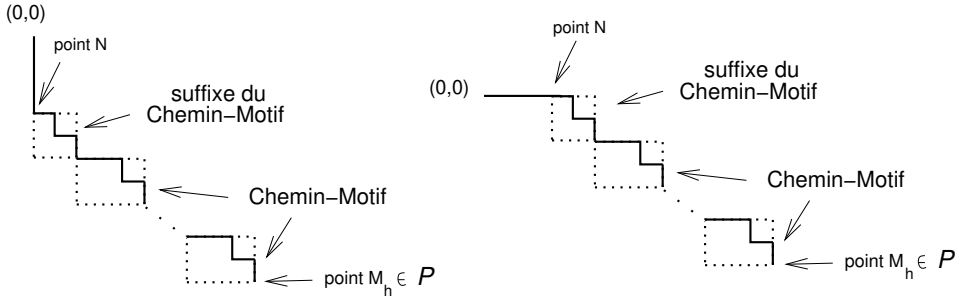


FIG. 7.8 - *Partie supérieure de $\mathcal{P}'$*

rencontre la ligne droite Oy qui passe par (λ, μ) (voir figure 7.9 à droite), ou la ligne droite $0x$ qui passe par (λ, μ) (voir figure 7.9 à gauche).

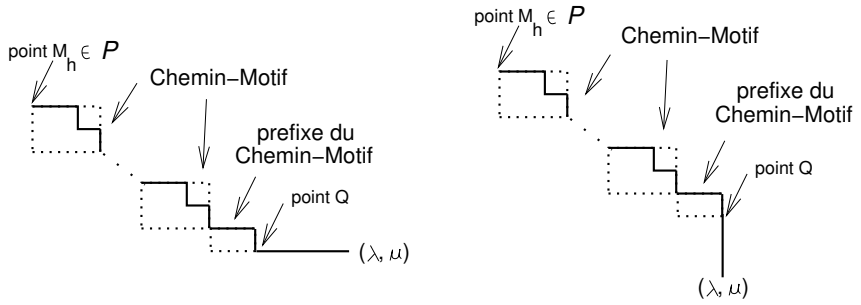


FIG. 7.9 - *Partie inférieure de $\mathcal{P}'$*

Le point Q peut être ainsi situé soit à gauche, soit au-dessus du point (λ, μ).

– Si le point Q est situé à gauche de (λ, μ), on relie ces deux points par des pas en x (voir figure 7.10 à gauche). Si le point Q est situé au-dessus de (λ, μ), on relie ces deux points par des pas en y (voir figure 7.10 à droite).

On notera par $\mathcal{P}'_1$ la partie de $\mathcal{P}'$ entre les points (0,0) et le point N, par $\mathcal{P}'_2$ la partie de $\mathcal{P}'$ entre les points N et Q et le point N et par $\mathcal{P}'_3$ la partie de $\mathcal{P}'$ entre les points Q et (λ, μ).

Preuve de $\Gamma_{\mathcal{P}} \Rightarrow \Gamma_{\mathcal{P}'}$:

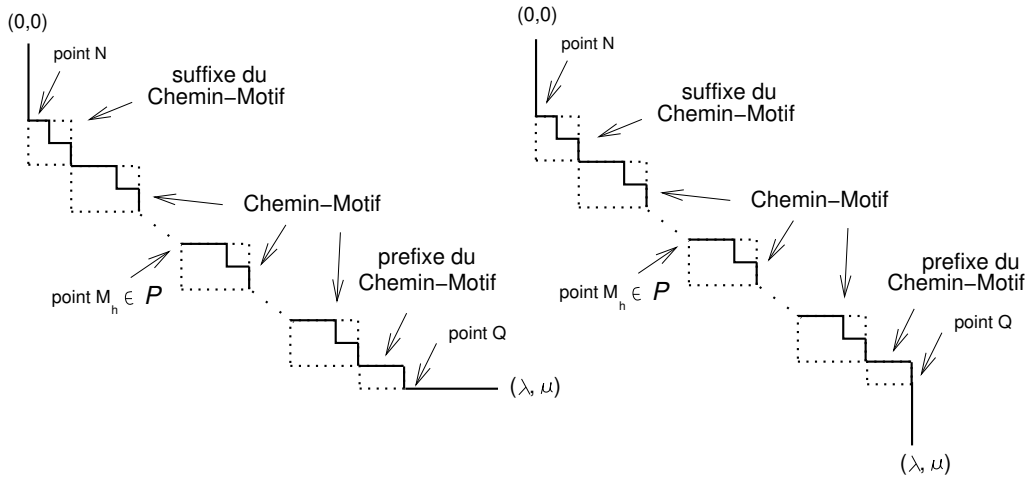


FIG. 7.10 - *Chemin $\mathcal{P}'$ dans sa totalité*

Nous montrons ensuite que la contrainte globale de $\mathcal{P}$ est subsumée par la contrainte globale de $\mathcal{P}'$. Cela est fait en montrant que :

1. $\Gamma^1_{\mathcal{P}'} = \Phi_1(M_h)$.
2. $\Gamma^2_{\mathcal{P}} \Rightarrow \Gamma^2_{\mathcal{P}'}$

– **Preuve de $\Gamma^1_{\mathcal{P}'} = \Phi_1(M_h)$.** Entre les points N et Q, la conjonction $\Gamma^1_{\mathcal{P}'_2}$ équivaut à $\Phi_1(M_h)$, par la propriété caractéristique du chemin-motif (d'après la proposition 7.12).

Si le point Q est situé au-dessus du point (λ, μ), tous les pas en x de $\mathcal{P}'$ se trouvent entre les points N et Q. Il s'ensuit que $\Gamma^1_{\mathcal{P}'}$ équivaut à $\Phi_1(M_h)$.

Si le point Q est situé à gauche du point (λ, μ), après le point Q on trouve une séquence de pas en x. Comme $\forall \alpha, \beta\ \Phi_1(\alpha, \beta) \Rightarrow \Phi_1(\alpha, \beta + 1)$ $(a'_1 \geq 0)$, la conjonction $\Gamma^1_{\mathcal{P}'_3}$ équivaut à $\Phi_1(Q)$. Il s'ensuit que $\Gamma^1_{\mathcal{P}'}$ équivaut à $\Phi_1(M_h) \wedge \Phi_1(Q)$, qui équivaut lui-même à $\Phi_1(M_h)$ (car M_h et Q appartiennent à un même chemin-motif, et $\Phi_1(M_h)$ est la contrainte horizontale la plus forte par construction).

Comme $\Phi_1(M_h)$ est une contrainte de $\mathcal{P}$, il en resulte que $\Gamma^1_{\mathcal{P}} \Rightarrow \Gamma^1_{\mathcal{P}'}$.

– **Preuve de $\Gamma^2_{\mathcal{P}} \Rightarrow \Gamma^2_{\mathcal{P}'}$.** Comme $\forall \alpha, \beta\ \Phi_2(\alpha + 1, \beta) \Rightarrow \Phi_2(\alpha, \beta)$ $(b'_1 \leq 0)$, si toutes les contraintes verticales de $\mathcal{P}$ sont situées à droite du chemin $\mathcal{P}'$, la conjonction

$\Gamma^2_{\mathcal{P}'}$ subsume $\Gamma^2_{\mathcal{P}}$ (tout segment vertical de $\mathcal{P}$ aura une contrainte plus forte que sa projection sur $\mathcal{P}'$).

On montre à présent qu'en effet : les points de $\mathcal{P}$ où il y a des pas en y sont situés à droite de $\mathcal{P}'$. Supposons, par l'absurde, qu'il existe un point R de $\mathcal{P}$ où il y a des pas en y et que R soit situé à gauche de $\mathcal{P}'$.
Soit R' le point où on trouve le premier pas en x de $\mathcal{P}$ après le point R. On distinguera deux cas :

1. R' est situé à l'intérieur d'un motif (voir figure 7.11). Soit M'_h le point d'où le chemin-motif associé part ($\Phi_1(M'_h) = \Phi_1(M_h)$).
   D'après la proposition 7.13, comme R' est situé au-dessous d'un chemin-motif, $\Phi_1(R') < \Phi_1(M'_h) = \Phi_1(M_h)$. D'autre part, comme M_h est le point où on trouve la contrainte horizontale la plus forte de $\mathcal{P}$, on a $\Phi_1(R') \geq \Phi_1(M_h)$. Ces deux faits sont en contradiction.
2. R' n'est pas situé à l'intérieur d'un motif (voir figure 7.12). Dans ce cas, nous considèrerons un point R'' qui est situé à droite et sur le même axe $0x$ que R', et tel que R'' soit situé à l'intérieur d'un chemin-motif. Soit M'_h le point d'où le chemin-motif associé part ($\Phi_1(M'_h) = \Phi_1(M_h)$). Nous avons $\Phi_1(R') < \Phi_1(R'')$ ($a'_1 > 0$).
   D'après la proposition 7.13, $\Phi_1(R'') < \Phi_1(M'_h) = \Phi_1(M_h)$. Par conséquent, $\Phi_1(R') < \Phi_1(M_h)$. D'autre part, comme M_h est le point où on trouve la contrainte horizontale la plus forte de $\mathcal{P}$, on a $\Phi_1(R') \geq \Phi_1(M_h)$. Ces deux faits sont en contradiction.

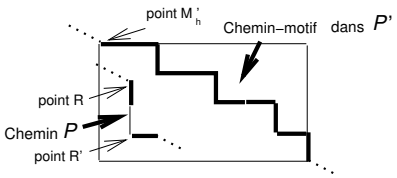


FIG. 7.11 -

□

Proposition 7.18 *Soit P' un chemin de l'une des formes représentées dans la figure 7.7. Le chemin P' peut être représenté aussi par une des formes données dans la figure 7.13 : un*

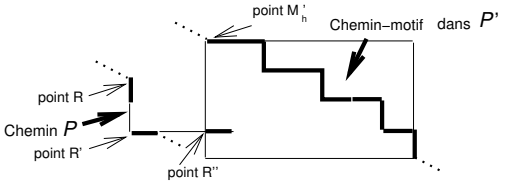


FIG. 7.12 -

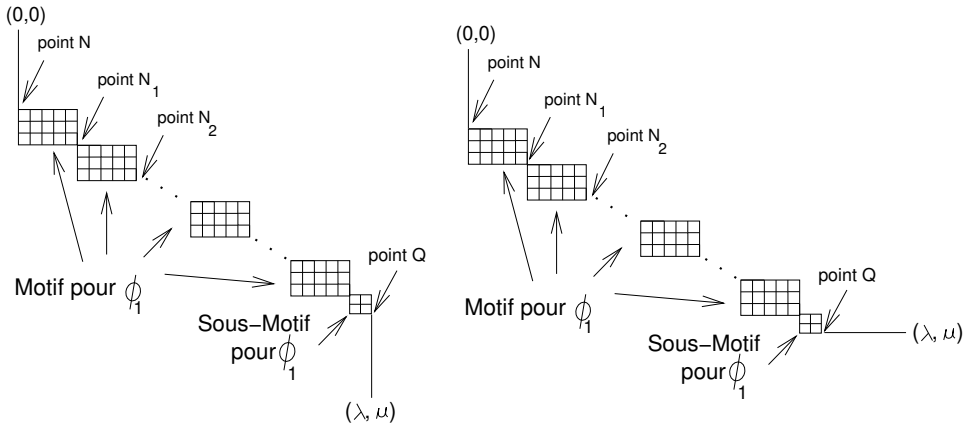


FIG. *7.13* - Représentation des chemins $\mathcal{S}'(\lambda, \mu)$

chemin vertical, suivi d'une série de motifs, suivie d'un sous motif et d'un chemin horizontal (ou vertical).

Notons que le sous-motif n'apparait plus qu'une seule fois (en fin de chemin de $\mathcal{P}'$).

Preuve : Soit N le point d'où est issu le premier pas en x de $\mathcal{P}'$ et N_i le point $N + i(-a''_2, a''_1)$, où $(-a''_2, a''_1)$ est la période de la contrainte Φ_1. Pour que $\mathcal{P}'$ ait une des formes données dans la figure 7.13, il suffit de prouver que les points N_1, N_2, ... appartiennent à $\mathcal{P}'$. Ceci vient du fait qu'on peut toujours faire "commuter" le suffixe d'un chemin-motif avec une série de motifs (voir figure 7.14).

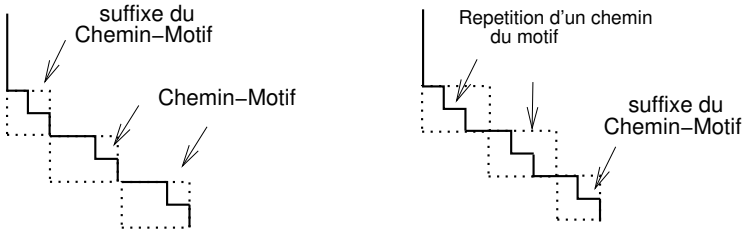


FIG. 7.14 - *"Commutation" des chemins*

□

De façon analogue :

Proposition 7.19 *Soit P' un chemin de l'une des formes représentées dans la figure 7.7. Le chemin P' peut aussi être représenté par l'une des formes données dans la figure 7.15 : un chemin vertical, suivi d'un sous-motif, suivi d'une série de motifs, et d'un chemin horizontal (ou vertical).*

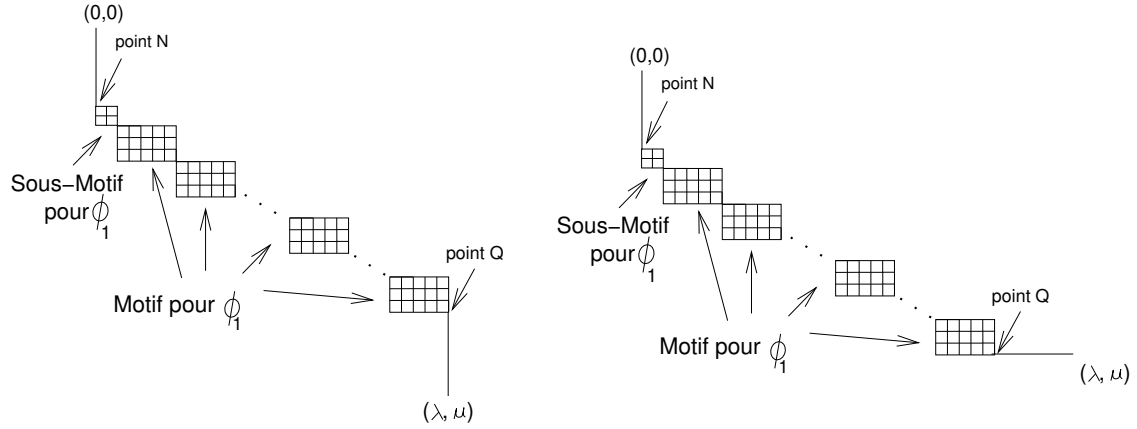


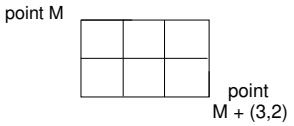
FIG. *7.15* - Représentation des chemins $\mathcal{S}'(\lambda, \mu)$

Notons que dans ce cas, le sous-motif apparait uniquement au début du chemin.

Exemple 7.20 Soit π_6 le programme :

$p(x,y,z)$	:–	$\Theta(x,y,z).$	règle $\mathcal{R}_0$
$p(x+2,y-3,z-2)$	:–	$x \geq 0,\ p(x,y,z)$	règle $\mathcal{R}_1$
$p(x-3,y+5,z+1)$	:–	$y \geq 0,\ p(x,y,z)$	règle $\mathcal{R}_2$

La contrainte en Φ_1 est périodique avec motif (3,2) (voir figure 7.16).



$$\Phi_1(M) = \Phi_1(M + (3,2))$$

FIG. 7.16 - *Motif pour* Φ_1

Tout chemin entre les points $(0,0)$ et (λ,μ) est subsumé par un chemin de la forme représentée dans la figure 7.17.

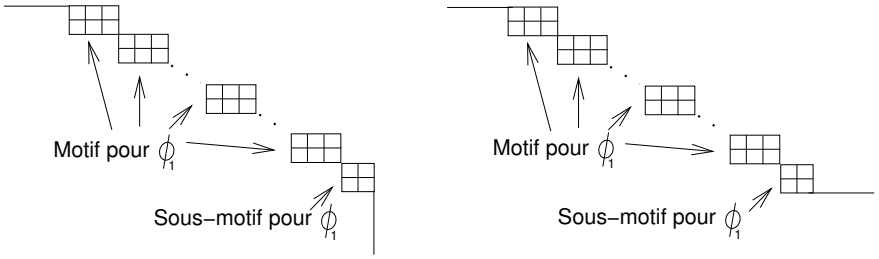


FIG. 7.17 -

7.4 $\bigwedge$-simplification

Dans cette partie, nous effectuons l'étape de $\bigwedge$-simplification pour un programme dans la classe $a'_1 > 0,\ a'_2 < 0,\ b'_1 < 0,\ b'_2 > 0$. Des résultats symétriques peuvent être établis pour la classe $a'_1 < 0,\ a'_2 > 0,\ b'_1 > 0,\ b'_2 < 0$.

Pour effectuer la $\bigwedge$-simplification pour les cas "difficiles", il faut d'abord étudier le comportement de la contrainte Φ_2 de la règle $\mathcal{R}_2$ par rapport au motif de la contrainte périodique Φ_1.

Définition 7.21 (Monotonicité Périodique) *Soit $\vec{\tau}$ une période pour la contrainte Φ_k. Nous disons que la contrainte Φ_j ($k \neq j$) est* périodiquement croissante *(resp.* périodiquement décroissante*) ssi $\Phi_j(M) \leq \Phi_j(M+\vec{\tau})$ (resp. $\Phi_j(M) \geq \Phi_j(M+\vec{\tau})$). La contrainte Φ_j est* périodiquement monotone *ssi Φ_j est périodiquement croissante ou périodiquement décroissante.*

Pour un programme Π avec $a'_1 > 0,\ a'_2 < 0,\ b'_1 < 0,\ b'_2 > 0$, nous avons :

Proposition 7.22 (Monotonicité Périodique de Φ_2)
- *Ou bien* $\forall\ \alpha, \beta\ \Phi_2(\alpha, \beta) \geq \Phi_2(\alpha - a'_2, \beta + a'_1)$.
- *Ou bien* $\forall\ \alpha, \beta\ \Phi_2(\alpha, \beta) < \Phi_2(\alpha - a'_2, \beta + a'_1)$.

Dans le premier cas, Φ_2 est périodiquement croissante et dans le deuxième périodiquement strictement décroissante.

La périodicité monotone est caractérisée algébriquement par :

Proposition 7.23
- Φ_2 *est périodiquement croissante pour le motif de* Φ_1 *ssi* $a'_1 b'_2 - a'_2 b'_1 \geq 0$.
- Φ_2 *est périodiquement strictement décroissante pour le motif de* Φ_1 *ssi* $a'_1 b'_2 - a'_2 b'_1 < 0$.

Grâce à la définition de périodicité monotonique, il est facile d'effectuer l'étape de $\bigwedge$-simplification. Ceci est formalisé par les propositions suivantes :

Proposition 7.24 *Soit Π un programme de contrainte périodique Φ_1, et de contrainte Φ_2 périodiquement croissante par rapport à Φ_1 ($a'_1 > 0,\ a'_2 < 0,\ b'_1 < 0,\ b'_2 > 0,\ a'_1 b'_2 - a'_2 b'_1 \geq 0$). La contrainte horizontale la plus forte d'un chemin $\mathcal{P}$ en $\mathcal{S}'(\lambda, \mu)$ est située en un point du premier motif de ce chemin. La contrainte verticale la plus forte d'un chemin $\mathcal{P}$ en $\mathcal{S}'(\lambda, \mu)$ est située en (0,0) ou en un point du premier motif de ce chemin. Les positions possibles pour ces contraintes sont renforcées par un trait plus épais dans la figure 7.18.*

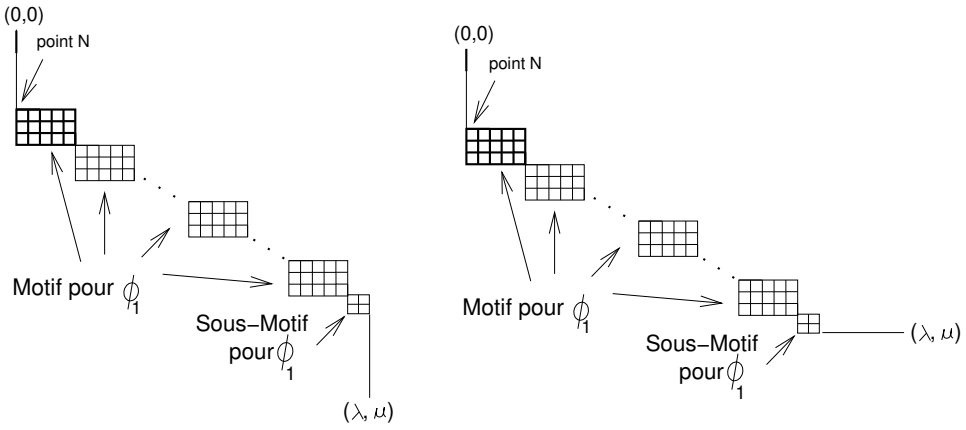


FIG. *7.18* - Contraintes plus fortes des chemins $\mathcal{S}'(\lambda, \mu)$

Proposition 7.25 *Soit* Π *un programme de contrainte périodique* Φ_1*, et de contrainte* Φ_2 *périodiquement srictement décroissante par rapport à* Φ_1 *(* $a'_1 > 0$, $a'_2 < 0$, $b'_1 < 0$, $b'_2 > 0$, $a'_1 b'_2 - a'_2 b'_1 < 0$*). La contrainte horizontale la plus forte d'un chemin* $\mathcal{P}$ *en* $\mathcal{S}'(\lambda, \mu)$ *est située en un point du dernier motif de ce chemin. La contrainte verticale la plus forte d'un chemin* $\mathcal{P}$ *en* $\mathcal{S}'(\lambda, \mu)$ *est située en* $(0, 0)$ *ou en un point du dernier motif de ce chemin. Les positions possibles pour ces contraintes sont renforcées par un trait plus épais dans la figure 7.19.*

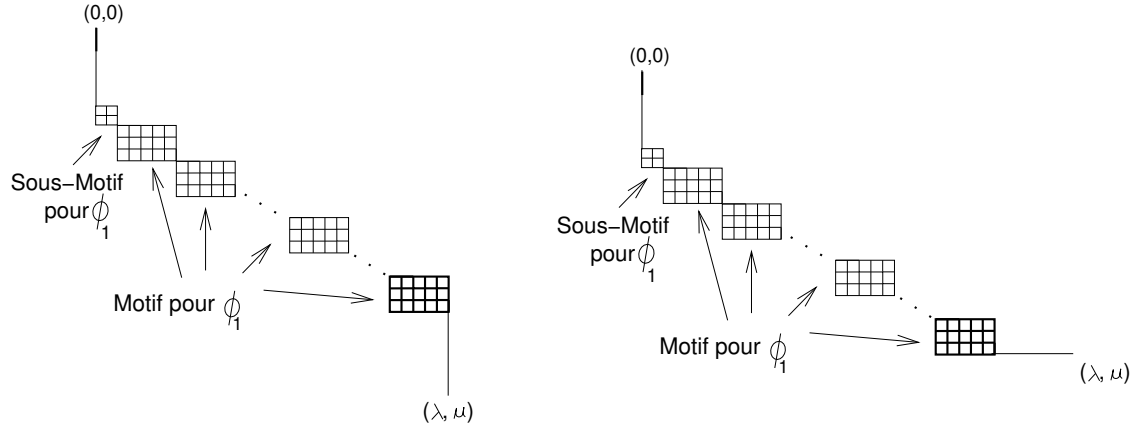


FIG. *7.19* - Contraintes plus fortes des chemins $\mathcal{S}'(\lambda, \mu)$

Montrons ensuite qu'une formalisation arithmétique pour $\Delta(\lambda, \mu)$ découle des propositions 7.24 et 7.25. Nous nous limiterons au cas où Φ_2 est périodiquement croissante par

rapport à Φ_1 ($a'_1 b'_2 - a'_2 b'_1 \geq 0$). Le cas où où Φ_2 est périodiquement strictement décroissante par rapport à Φ_1 ($a'_1 b'_2 - a'_2 b'_1 < 0$) est analogue.

On notera par $\widehat{\Delta}[N\_\_N']$ la disjonction des contraintes globales de tous les chemins entre les points N et N'. Remarquons que $\Delta(N) = \widehat{\Delta}[(0,0)\_\_N]$.

Théorème 7.26 *Soit* Π *un programme avec une contrainte périodique* Φ_1*, et une contrainte* Φ_2 *périodiquement croissante par rapport à* Φ_1 *(*$a'_1 > 0$, $a'_2 < 0$, $b'_1 < 0$, $b'_2 > 0$, $a'_1 b'_2 - a'_2 b'_1 \geq 0$*). La formule* $\Delta(\lambda, \mu)$ *est équivalente à une formule arithmétique.*

Preuve : Par définition, $\Delta(\lambda, \mu) = \bigvee_{\mathcal{P} \in \mathcal{S}(\lambda,\mu)} \Gamma_{\mathcal{P}}$.

Par le théorème 7.17 et la proposition 7.18, $\Delta(\lambda, \mu) \equiv \bigvee_{\mathcal{P} \in \mathcal{S}'(\lambda,\mu)} \Gamma_{\mathcal{P}}$, où les chemins en $\mathcal{S}'(\lambda, \mu)$ ont la forme représentée dans la figure 7.13.

Soit $\mathcal{P}$ un chemin quelconque de $\mathcal{S}'(\lambda, \mu)$ et soit $N = (0, \nu)$ le point de $\mathcal{P}$ où on trouve le premier pas de x. Le motif associé à Φ_1 est (τ_1, τ_2), avec $\tau_1 = -a''_2$ et $\tau_2 = a''_1$.

Pour le calcul de $\Delta(\lambda, \mu)$, on distinguera deux situations :

1. On peut placer un chemin-motif entre les points N et (λ, μ). Algébriquement, ceci correspond à : $N + (\tau_1, \tau_2) \leq (\lambda, \mu)$
2. On ne peut pas placer un chemin-motif entre les points N et (λ, μ). Cela est le cas quand il y a moins de τ_1 pas en x entre N et (λ, μ), ou moins de τ_2 pas en y entre ces mêmes points. Algébriquement, ceci correspond à : $\tau_1 < \lambda$ ou $\tau_2 + \nu < \mu$.

Situation 1 : D'après la proposition 7.24, la contrainte horizontale la plus forte de $\mathcal{P}$ est située en un pont entre N et $N + (\tau_1, \tau_2)$. La contrainte verticale la forte de $\mathcal{P}$ est située en (0,0) ou bien en un point entre N et $N + (\tau_1, \tau_2)$ (voir figure 7.18).

Formellement :

$$\Delta(\lambda, \mu) = \exists \nu \geq 0 \ (\widehat{\Delta}[(0,0)\_\_(0,\nu)] \ \wedge \ \widehat{\Delta}[(0,\nu)\_\_(\tau_1, \tau_2 + \nu)].$$

La composante $\widehat{\Delta}[(0,0)\_\_(0,\nu)]$ représente la contrainte du chemin entre (0,0) et N. La composante $\widehat{\Delta}[(0,\nu)\_\_(\tau_1, \tau_2 + \nu)]$ représente la contrainte des chemins entre N et $N + (\tau_1, \tau_2)$.

En décomposant $\nu \geq 0$ en $\nu = 0$ et $\nu > 0$, et en tenant compte de ce que :

$\widehat{\Delta}[(0,0)\_\_(0,0)] = vrai$, et
$\widehat{\Delta}[(0,0)\_\_(0,\nu)] = \Phi_2(0,0)$, pour tout $\nu > 0$,
on a :

$$\Delta(\lambda,\mu) = \widehat{\Delta}[(0,0)\_\_(\tau_1,\tau_2)] \ \vee \ (\Phi_2(0,0) \ \wedge \ \exists \nu > 0 \ (\widehat{\Delta}[(0,\nu)\_\_(\tau_1,\tau_2+\nu)] \ ) \ ). \qquad (\star)$$

La formule $\exists \nu > 0 \ (\widehat{\Delta}[(0,\nu)\_\_(\tau_1,\tau_2+\nu)] \ )$ est bornée, car il existe $\dfrac{(\tau_2+\tau_1)!}{\tau_2! \times \tau_1!}$ chemins entre les points $(0,\nu)$ et $(\tau_1,\tau_2+\nu)$.

Par conséquent, la disjonction $\Delta(\lambda,\mu)$ des contraintes globales de tous les chemins entre $(0,0)$ et (λ,μ) est équivalente à une formule arithmétique finie.

Situation 2 : Cette situation correspond au cas où il y a moins de τ_1 pas en x entre N et (λ,μ) ou moins de τ_2 pas en y entre ces points. Algébriquement ceci correspond à : $\tau_1 < \lambda$ ou $\tau_2 + \nu < \mu$. Le chemin $\mathcal{P}$ aura donc une des formes représentées dans la figure 7.20.

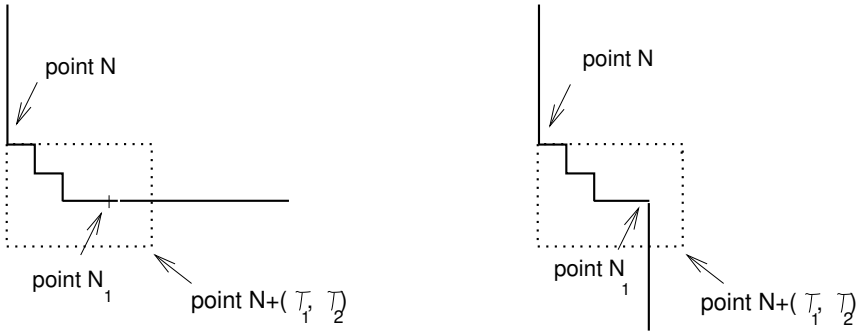


FIG. 7.20 -

Soit (τ_1',τ_2') un sous motif de (τ_1,τ_2), et soit $N_1 = N + (\tau_1',\tau_2')$.

Le calcul de $\Delta(\lambda,\mu)$ se fait de façon analogue à celui de la situation 1 :

$$\Delta(\lambda,\mu) = \exists \nu, \tau_1', \tau_2' \geq 0 \ \ (\ (\tau_1' < \tau_1 \vee \tau_2' < \tau_2 \ ) \wedge \widehat{\Delta}[(0,0)\_\_(0,\nu)] \wedge \ \widehat{\Delta}[(0,\nu)\_\_(\tau_1',\tau_2'+\nu)] \ ).$$

En décomposant $\nu \geq 0$ en $\nu = 0$ et $\nu > 0$, et en tenant compte de ce que :
$\widehat{\Delta}[(0,0)\_\_(0,0)] = vrai$ et
$\widehat{\Delta}[(0,0)\_\_(0,\nu)] = \Phi_2(0,0)$, pour $\nu > 0$,
on a :

$$\begin{aligned}\Delta(\lambda,\mu) = \exists \tau_1', \tau_2' \geq 0 \ (\ (\tau_1' < \tau_1 \vee \tau_2' < \tau_2) \wedge (\ \widehat{\Delta}[(0,0)\_\_(\tau_1',\tau_2')] \vee \\ (\Phi_2(0,0) \ \wedge \ \exists \nu > 0 \ \widehat{\Delta}[(0,\nu)\_\_(\tau_1',\tau_2'+\nu)] \ ) \ ) \ ). \qquad (\star\star)\end{aligned}$$

Pour des raisons analogues au cas de la situation 1, l'expression $\Delta(\lambda, \mu)$ est équivalente à une formule arithmétique finie.

□

Dans la pratique, pour la situation 2, on calculera la contrainte globale de tous les chemins correspondant à différentes valeurs de τ_1' et τ_2', et ensuite on prendra la disjonction de ces contraintes. Ceci est illustré dans l'exemple 7.27.

Exemple 7.27 (Suite de l'exemple 7.20) Considérons le programme π_6. Nous avons prouvé que tout chemin entre (0,0) et (λ, μ) est subsumé par un chemin ayant une forme représentée dans la figure 7.17.

Soit $\mathcal{P}$ un de ces chemins et le $N = (0, \nu)$ le premier pas en x de $\mathcal{P}$.

Les situations suivantes peuvent se produire :

1. On peut placer un chemin-motif associé au motif (3,2), entre $(0, \nu)$ et (λ, μ). Algébriquement, ceci correspond au cas où $(3, 2 + \nu) \leq (\lambda, \mu)$.

2. Il n'y a pas assez de pas en x ou assez de pas en y entre $(0, \nu)$ et (λ, μ), pour placer un chemin du motif (3,2). Algébriquement, ceci correspond au cas où $(\lambda < 3 \vee 2 + \nu < \mu)$.

Dans le premier cas, le chemin $\mathcal{P}$ peut avoir une des formes représentées dans la figure 7.21. A la figure 7.22, nous représentons les contraintes entre les points $(0, \nu)$ et $(3, 2 + \nu)$ des différentes formes possibles de $\mathcal{P}$.

Dans le deuxième cas, le chemin $\mathcal{P}$ peut avoir une des formes représentées dans la figure 7.23 ou 7.24.

Soit $\mathcal{P}_i$ un chemin de la forme i, representé au figures 7.21, 7.23 et 7.24, et $\Gamma_{\mathcal{P}_i}$ la contrainte du chemin $\mathcal{P}_i$ entre N et (λ, μ).

Nous avons donc, en utilisant le fait que $\Phi_2(0, 0) \equiv y \geq 0$ et en utilisant les formules $(\star)$ et $(\star\star)$ ci-dessus :

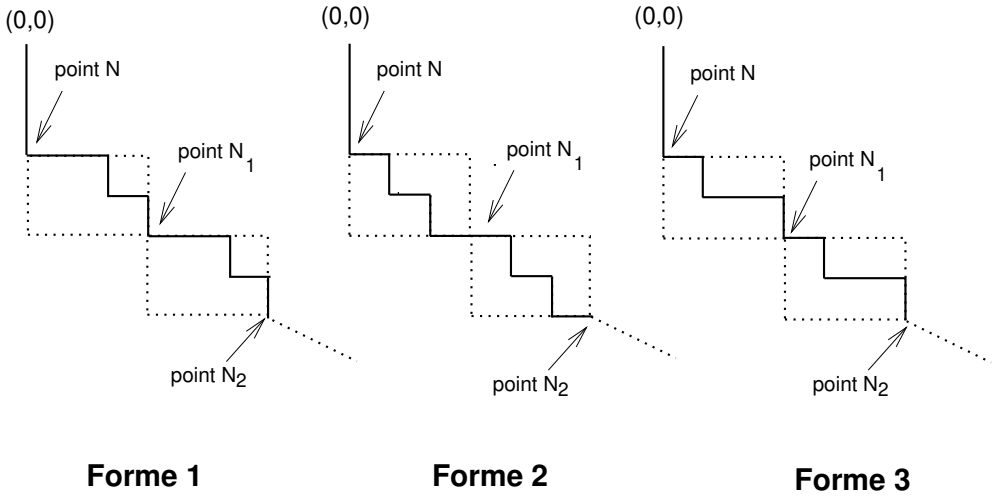


FIG. 7.21 - *Différentes formes de $\mathcal{P}$ pour la situation 1*

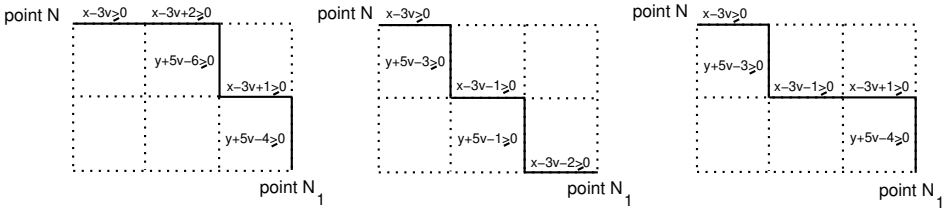


FIG. 7.22 - *Différentes formes de $\mathcal{P}$ entre les points $N = (0, \nu)$ et $N_1 = (3, 2 + \nu)$*

$$
\begin{aligned}
\Gamma_{\mathcal{P}_1} &= y + 5\nu - 6 \geq 0 \wedge\ x - 3\nu \geq 0 \\
\Gamma_{\mathcal{P}_2} &= y + 5\nu - 3 \geq 0 \wedge x - 3\nu - 2 \geq 0 \\
\Gamma_{\mathcal{P}_3} &= y + 5\nu - 4 \geq 0 \wedge x - 3\nu - 1 \geq 0 \\
\Gamma_{\mathcal{P}_4} &= x - 3\nu \geq 0 \wedge \nu = \mu \\
\Gamma_{\mathcal{P}_5} &= y + 5\nu - 6 \geq 0 \wedge x - 3\nu \geq 0 \wedge \nu + 1 = \mu \\
\Gamma_{\mathcal{P}_6} &= y + 5\nu - 3 \geq 0 \wedge x - 3\nu - 1 \geq 0 \wedge \nu + 1 = \mu \\
\Gamma_{\mathcal{P}_7} &= \lambda = 0 \\
\Gamma_{\mathcal{P}_8} &= y + 5\nu - 3 \geq 0 \wedge x - 3\nu \geq 0 \wedge \lambda = 1 \\
\Gamma_{\mathcal{P}_9} &= y + 5\nu - 6 \geq 0 \wedge x - 3\nu \geq 0 \wedge \lambda = 2 \\
\Gamma_{\mathcal{P}_{10}} &= y + 5\nu - 3 \geq 0 \wedge x - 3\nu - 1 \geq 0 \wedge \lambda = 2
\end{aligned}
$$

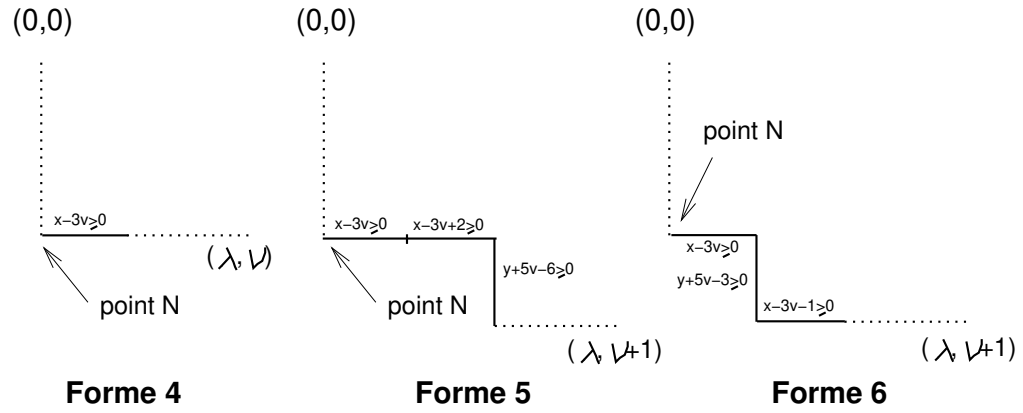


FIG. 7.23 - *Différentes formes de $\mathcal{P}$ pour la situation 2*

La formule $\Delta(\lambda, \mu)$, pour $\lambda > 0$ et $\mu > 0$, est équivalente à :

- Pour $(3, 2+\nu) \leq (\lambda, \mu)$:
  $\Delta(\lambda, \mu) \equiv \exists \nu \ (\ (\nu = 0 \ \vee \ (\ \nu > 0 \ \wedge \ y \geq 0) \ ) \ \wedge \ (\Gamma_{\mathcal{P}_1} \vee \Gamma_{\mathcal{P}_2} \ \vee \Gamma_{\mathcal{P}_3} \ ) \ ).$
- Pour $\lambda < 3 \vee 2 + \nu < \mu$:
  $\Delta(\lambda, \mu) \equiv \exists \nu \ (\ (\nu = 0 \vee (\nu > 0 \wedge y \geq 0) \ ) \wedge (\ (\ \Gamma_{\mathcal{P}_4} \vee \Gamma_{\mathcal{P}_5} \vee \Gamma_{\mathcal{P}_6} \vee \Gamma_{\mathcal{P}_7} \vee \Gamma_{\mathcal{P}_8} \vee \Gamma_{\mathcal{P}_9} \vee \Gamma_{\mathcal{P}_{10}} \ ) \ ) \ ).$

□

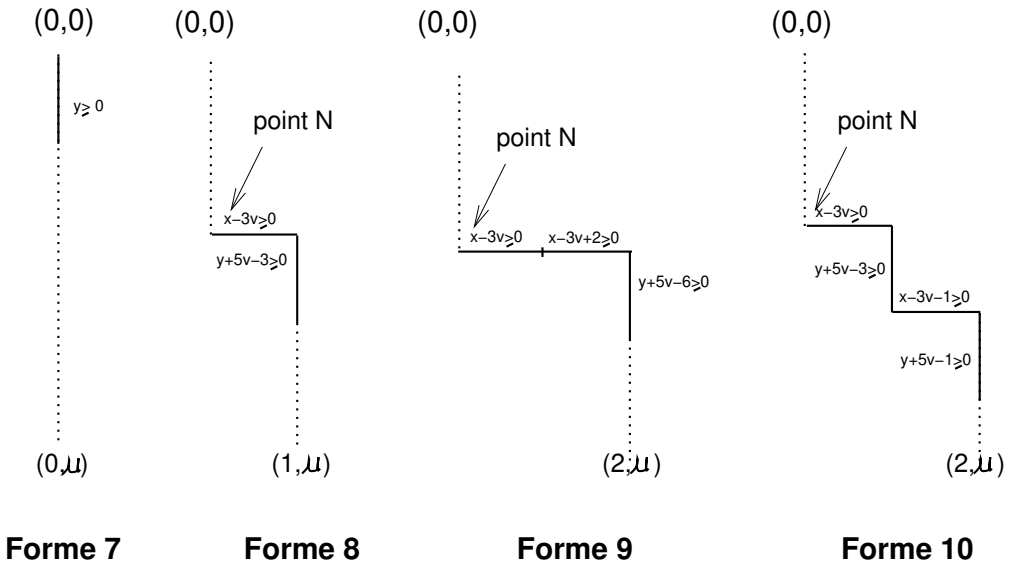


FIG. 7.24 - *Différentes formes de $\mathcal{P}$ pour la situation 2*

Chapitre 8

Deux Règles Récursives avec Egalité

Dans ce chapitre, on expliquera comment adapter notre méthode pour exprimer le résultat de l'évaluation ascendante de programmes avec contraintes d'égalité, par une formule arithmétique (finie).

Les programmes que nous considérons sont ainsi de la forme :

$p(x,y,z)$	:–	$\Theta(x,y,z).$	règle $\mathcal{R}_0$
$p(x+a_1, y+b_1, z+c_1)$	:–	$\Phi_1(x,y,z),\ p(x,y,z)$	règle $\mathcal{R}_1$
$p(x+a_2, y+b_2, z+c_2)$	:–	$\Phi_2(x,y,z),\ p(x,y,z)$	règle $\mathcal{R}_2$

où $a_1,\ a_2,\ b_1,\ b_2,\ c_1,\ c_2\ \in \mathbb{Z}$, $\Theta(x,y,z)$ est une formule arithmétique linéaire, $\Phi_1(x,y,z)$ représente la contrainte $d_1x + e_1y + f_1z + g_1 \geq 0$ et $\Phi_2(x,y,z)$ représente la contrainte $d_2x + e_2y + f_2z + g_2 = 0$, avec $d_k, e_k, f_k, g_k\ \in \mathbb{Z}$ et $k \in \{1,2\}$.

Le cas où les deux contraintes contiennent simultanément des signes d'égalité se traite de façon analogue.

Le calcul de $\Delta(\lambda,\mu)$, pour $\lambda = 0$ ou $\mu = 0$, ne pose pas de grandes difficultés. Pour le cas où $\lambda \neq 0$ et $\mu \neq 0$, nous considérerons quatre classes de programmes, selon une propriété d'invariance de la contrainte.

Définition 8.1 *Soit* Π *un programme de la forme ci-dessous. La contrainte* Φ_2 *est* inva-

riante *par rapport à la règle* $\mathcal{R}_1$ *(resp.*$\mathcal{R}_2$*) ssi* $d_2a_1+e_2b_1+f_2c_1 = 0$ *(resp.* $d_2a_2+e_2b_2+f_2c_2 = 0$*).*

Rappelons que $b'_1 = d_2a_1 + e_2b_1 + f_2c_1$ et $b'_2 = d_2a_2 + e_2b_2 + f_2c_2$.

Les quatre classes de programmes que nous considérons sont :

1. Programmes où la contrainte Φ_2 est invariante par rapport à R_2, mais pas par rapport à R_1 ($b'_2 = 0$ et $b'_1 \neq 0$).
2. Programmes où la contrainte Φ_2 est invariante par rapport à R_1, mais pas par rapport à R_2 ($b'_1 = 0$ et $b'_2 \neq 0$).
3. Programmes où la contrainte Φ_2 est simultanément invariante par rapport à R_1 et R_2 ($b'_1 = 0$ et $b'_2 = 0$).
4. Programmes où la contrainte Φ_2 n'est ni invariante par rapport à R_1 ni invariante par rapport à R_2 ($b'_1 \neq 0$ et $b'_2 \neq 0$).

Nous montrerons comment appliquer notre approche géométrique sur un exemple de la première classe. Notre approache est appliquée de façon similaire pour les autres classes.

Exemple 8.2 Soit π_7 le programme :

$$\begin{array}{lll} p(x,y,z) & :- & \Theta(x,y,z). \\ p(x+1,y,z) & :- & x \geq z,\ p(x,y,z). \\ p(x+1,y,z+1) & :- & x = z,\ p(x,y,z). \end{array}$$

On a :

$$\begin{array}{rcccccl} a'_1 & = & d_1a_1 + e_1b_1 + f_1c_1 & = & 1\times 1 + 0\times 0 + (-1)\times 0 & = & 1 \\ a'_2 & = & d_1a_2 + e_1b_2 + f_1c_2 & = & 1\times 1 + 0\times 0 + (-1)\times 1 & = & 0 \\ b'_1 & = & d_2a_1 + e_2b_1 + f_2c_1 & = & 1\times 1 + 0\times 0 + -1\times 0 & = & 1 \\ b'_2 & = & d_2a_2 + e_2b_2 + f_2c_2 & = & 1\times 1 + 0\times 0 + -1\times 1 & = & 0 \end{array}$$

La contrainte $x = z$ est invariante par rapport à $\mathcal{R}_2$ (car $b'_1 \neq 0$ et $b'_2 = 0$), mais pas par rapport à $\mathcal{R}_1$. Cela signifie qu'une fois qu'on a appliqué la règle $\mathcal{R}_1$ (après une application de la règle $\mathcal{R}_2$), la contrainte $x = z$ ne sera plus jamais satisfaite et qu'on ne pourra donc

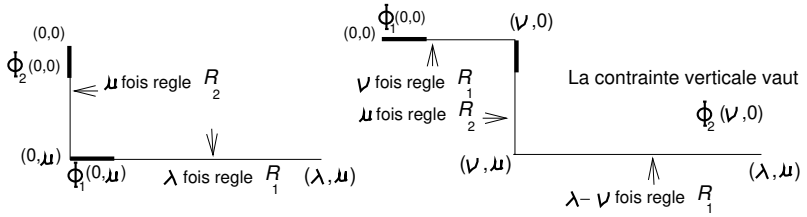


FIG. 8.1 -

plus appliquer la règle R_2 à nouveau. Ceci signifie que toutes les séquences d'application des règles $\mathcal{R}_1$ et $\mathcal{R}_2$ sont de la forme (voir figure 8.1) :

$$\mathcal{R}_1^{\nu}\mathcal{R}_2^{\mu}\mathcal{R}_1^{\lambda-\nu}$$

Pour appliquer la règle $\mathcal{R}_2$ après ν applications de $\mathcal{R}_1$, la contrainte $\Phi_2(\nu,0)$ (c'est-à-dire, $x+\nu=z$) doit être satisfaite. En plus, comme pour Φ_1 les implications de contraintes sont $\rightarrow\Downarrow$ (car $a'_1 \geq 0$ et $a'_2 \geq 0$), la contrainte Φ_1 est monotoniquement croissante. Donc, la contrainte Φ_1 la plus forte est celle du premier pas en x du chemin.

Soit $\mathcal{P}$ un chemin entre les points (0,0) et (λ,μ). Il y a deux cas à considérer :

Cas 1 : Le chemin $\mathcal{P}$ commence par un pas en x (voir figure 8.1, à gauche).

Cas 2 : Le chemin $\mathcal{P}$ commence par un pas en y (voir figure 8.1, à droite).

Evaluation de $\Delta(\lambda,\mu)$, pour $\lambda > 0$ et $\mu > 0$:

Cas 1 : $\Delta(\lambda,\nu) \equiv \Phi_1(0,\mu) \wedge \Phi_2(0,0)$, c'est-à-dire, ici : $x \geq z \wedge x = z$.

Cas 2 : $\Delta(\lambda,\nu) \equiv \Phi_1(0,0) \wedge \exists\nu\ (0 < \nu \leq \lambda \wedge \Phi_2(\nu,0))$, c'est-à-dire, ici : $x \geq z \wedge \exists\nu\ (0 < \nu \leq \lambda \wedge x+\nu = z)$.

Il s'ensuit que $\Delta(\lambda,\mu)$, pour $\lambda > 0$ et $\mu > 0$, équivaut à $(\Phi_1(0,\mu) \wedge \Phi_2(0,0)) \vee (\ \Phi_1(0,0) \wedge \exists\nu\ (0 < \nu \leq \lambda \wedge \Phi_2(\nu,0)\ )\ )$, c'est-à-dire, $(x \geq z \wedge x = z) \vee (\ x \geq z \wedge \exists\nu\ (0 < \nu \leq \lambda \wedge x+\nu = z)\ )$. Cette expression se simplifie en $x \geq z$.

Evaluation de $\Delta(\lambda, \mu)$, pour $\lambda = 0$ ou $\mu = 0$:

$\Delta(0,0) \equiv vrai.$

$\Delta(\lambda, 0) \equiv x \geq z$, pour $\lambda > 0$.

$\Delta(0, \nu) \equiv x = z$, pour $\nu > 0$.

Rappelons qu'un atome $p(u, v, w)$ est engendré par l'évaluation ascendante ssi :

$$\exists\ \lambda, \mu, x, y, z\ \ (\ \lambda \geq 0 \wedge \mu \geq 0 \wedge \nu \geq 0 \wedge u = x + \lambda a_1 + \mu a_2\ \ \wedge\ \ v = y + \lambda b_1 + \mu b_2\ \wedge \\ w = z + \lambda c_1 + \mu c_2 + \ \ \wedge\ \Theta(x, y, z)\ \ \wedge\ \ \Delta(\lambda, \mu).$$

Ici $a_1 = 1$, $a_2 = 0$, $a_3 = 0$, $b_1 = 1$, $b_2 = 0$ et $b_3 = 1$, donc $p(u, v, w)$ ssi :

$$\exists \lambda, \mu, x, y, z\quad \lambda \geq 0 \wedge \mu \geq 0\ \ \wedge\ \ u = x + \lambda + \nu\ \ \wedge\ \ v = y \wedge\ \ w = z + \mu\ \ \wedge\ \ \Theta(x, y, z)\ \ \wedge\ \ \Delta(\lambda, \mu).$$

En remplaçant $\Delta(\lambda, \mu)$ par sa valeur et en éliminant les variables x, y et z, quantifiées existentiellement, on obtient $p(u, v, w)$ ssi :

$$\Theta(u, v, w) \vee \exists \lambda\ (\ \ \lambda > 0 \wedge \Theta(u - \lambda, v, w)\ \wedge u \geq w + \lambda) \vee \exists \mu\ (\ \ \mu > 0 \wedge \Theta(x - \mu, v, w - \mu) \wedge u = \\ w) \vee \exists \lambda, \mu\ (\ \ \ \ \lambda > 0 \wedge \mu > 0 \wedge \Theta(x - \lambda - \nu, u, v, w - \mu)\ \ \wedge u \geq w + \lambda)$$

Ou de façon plus concise :

$$\Theta(u, v, w)\ \ \vee\ \ \exists \lambda, \mu\ (\ \ \ \ \lambda \geq 0 \wedge \mu \geq 0 \wedge \Theta(x - \lambda - \nu, u, v, w - \mu)\ \ \wedge u \geq w + \lambda)$$

□

Chapitre 9

Trois Règles Récursives - Cas Simples

Dans ce chapitre, on montrera comment effectuer des $\bigvee$-simplifications pour certaines classes de programmes avec trois règles récursives.

Comme dans les cas simples de deux règles récursives, la $\bigvee$-simplification sera la conséquence d'une certaine combinaison de sens d'implications de contraintes.

Dans les cinq parties suivantes, nous considérerons des combinaisons de sens d'implications de contraintes pour lesquelles la forme des chemins de $\mathcal{S}'(\lambda, \mu, \nu)$ ne dépend que de la matrice caractéristique du programme, c'est-à-dire, des signes de a'_k, b'_k et c'_k. Nous verrons dans les chapitres 10, 11 et 12, qu'en général, la forme des chemins de $\mathcal{S}'(\lambda, \mu, \nu)$ sera déterminée par des relations plus complexes sur a'_k, b'_k et c'_k.

Les cas que nous étudierons dans ce chapitre ne sont pas disjoints : ainsi il pourra se trouver que la forme des chemins en $\mathcal{S}'(\lambda, \mu, \nu)$ pour un programme puisse parfois être obtenue de plusieurs manières différentes.

9.1 Cas Simple 1

Proposition 9.1 *Pour un programme* Π *avec* $a'_2 \leq 0$, $a'_3 \leq 0$, $b'_1 \geq 0$ *et* $c'_1 \geq 0$ *(correspondant à des implications de contraintes comme celles représentées dans la figure 9.1 à*

gauche), les chemins dans l'ensemble $\mathcal{S}'(\lambda, \mu, \nu)$ sont tous de la forme représentée à la figure 9.1 à droite.

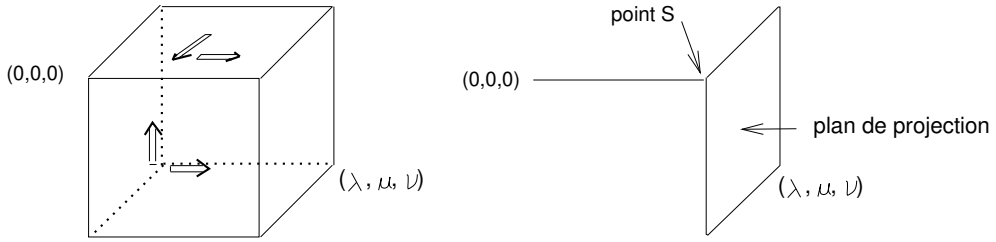


FIG. *9.1* - Cas où $a'_2 \leq 0$, $a'_3 \leq 0$, $b'_1 \geq 0$ et $c'_1 \geq 0$

Preuve : Considérons un chemin $\mathcal{P}$ reliant les points (0,0,0) et (λ, μ, ν). Nous voulons construire un chemin $\mathcal{P}'$, de la forme représentée à la figure 9.1 à droite, tel que la contrainte globale $\Gamma_{\mathcal{P}}$ soit subsumée par la contrainte globale $\Gamma_{\mathcal{P}'}$.

Construction du chemin $\mathcal{P}'$:

Le chemin $\mathcal{P}'$ est obtenu de la façon suivante :

- En partant de $(0, 0, 0)$, on effectue une séquence de pas en x jusqu'à rencontrer le plan 0_{YZ} qui passe par le point (λ, μ, ν). Cette intersection détermine le point S.
- On projette le chemin $\mathcal{P}$ sur le plan 0_{YZ} qui passe par le point (λ, μ, ν). Cette projection relie les points S et (λ, μ, ν).

On obtient ainsi le chemin $\mathcal{P}'$.

Preuve de $\Gamma_{\mathcal{P}} \Rightarrow \Gamma_{\mathcal{P}'}$:

Nous montrons ensuite que la contrainte globale de $\mathcal{P}$ est subsumée par la contrainte globale de $\mathcal{P}'$. Cela se fait de la manière suivante :

1. $\Gamma^1_{\mathcal{P}} \Rightarrow \Gamma^1_{\mathcal{P}'}$

2. $\Gamma^2_{\mathcal{P}} \Rightarrow \Gamma^2_{\mathcal{P}'}$
3. $\Gamma^3_{\mathcal{P}} \Rightarrow \Gamma^3_{\mathcal{P}'}$

- **Preuve de $\Gamma^1_{\mathcal{P}} \Rightarrow \Gamma^1_{\mathcal{P}'}$:** Comme $\forall \alpha, \beta, \gamma\ \Phi_1(\alpha, \beta+1, \gamma) \Rightarrow \Phi_1(\alpha, \beta, \gamma)$ $(a'_2 \leq 0)$ et $\forall \alpha, \beta, \gamma\ \Phi_1(\alpha, \beta, \gamma+1) \Rightarrow \Phi_1(\alpha, \beta, \gamma+1)$ $(a'_3 \leq 0)$, chaque contrainte $\Phi_1(\rho, 0, 0)$ de $\mathcal{P}'$ est plus faible que la $\Phi_1(\rho, \sigma, \tau)$ de $\mathcal{P}$. Par conséquent, $\Gamma^1_{\mathcal{P}} \Rightarrow \Gamma^1_{\mathcal{P}}$.
- **Preuve de $\Gamma^2_{\mathcal{P}} \Rightarrow \Gamma^2_{\mathcal{P}'}$:** Comme $\forall \alpha, \beta, \gamma\ \Phi_2(\alpha, \beta, \gamma) \Rightarrow \Phi_2(\alpha+1, \beta, \gamma)$ $(b'_1 \geq 0)$, en projetant chaque segment vertical de $\mathcal{P}$ sur le plan $0YZ$ qui passe par (λ, μ, ν), on obtient un segment vertical de $\mathcal{P}'$ de contrainte plus faible. Il s'ensuit que les contraintes verticales de $\mathcal{P}'$ sont plus faibles que celles de $\mathcal{P}$. Par conséquent, $\Gamma^2_{\mathcal{P}} \Rightarrow \Gamma^2_{\mathcal{P}'}$.
- **Preuve de $\Gamma^3_{\mathcal{P}_1} \Rightarrow \Gamma^3_{\mathcal{P}'_1}$:** De façon analogue, comme $\forall \alpha, \beta, \gamma\ \Phi_3(\alpha, \beta, \gamma) \Rightarrow \Phi_3(\alpha+1, \beta, \gamma)$ $(c'_1 \geq 0)$, en projetant chaque segment transversal de $\mathcal{P}$ sur le plan $0YZ$ qui passe par (λ, μ, ν), on obtient un segment transversal de $\mathcal{P}'$ de contrainte plus faible. Il s'ensuit les contraintes transversales de $\mathcal{P}'$ sont plus faibles que celles de $\mathcal{P}$. Par conséquent, $\Gamma^3_{\mathcal{P}} \Rightarrow \Gamma^3_{\mathcal{P}'}$

Donc, $\Gamma_{\mathcal{P}} \Rightarrow \Gamma_{\mathcal{P}'}$, avec $\mathcal{P}'$ de la forme représentée à la figure 9.1 à droite. □

Il y a 5 autres combinaisons d'implications de contraintes pour lesquelles on peut faire un raisonnement analogue à celui-là. Les implications de contraintes pour ces classes sont représentées dans la figure 9.2.

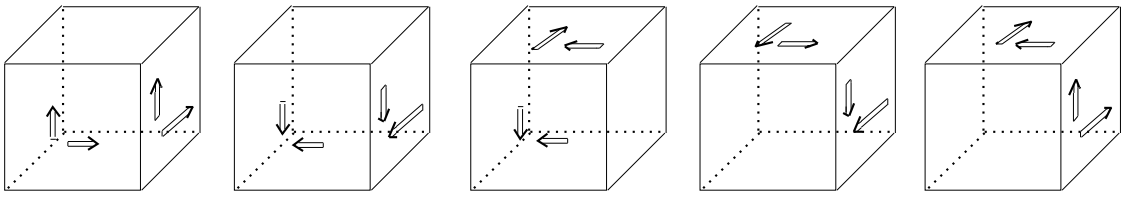


FIG. 9.2 - *Cas analogues au cas où $a'_2 \leq 0$, $a'_3 \leq 0$, $b'_1 \geq 0$ et $c'_1 \geq 0$*

9.2 Cas Simple 2

Proposition 9.2 *Pour un programme* II *avec $a'_1 \geq 0$, $b'_1 \geq 0$ et $c'_1 \geq 0$ (correspondant à des implications de contraintes comme celles représentées à la figure 9.3 à gauche), les chemins dans l'ensemble $\mathcal{S}'(\lambda, \mu, \nu)$ sont tous de la forme représentée à la figure 9.3 à droite.*

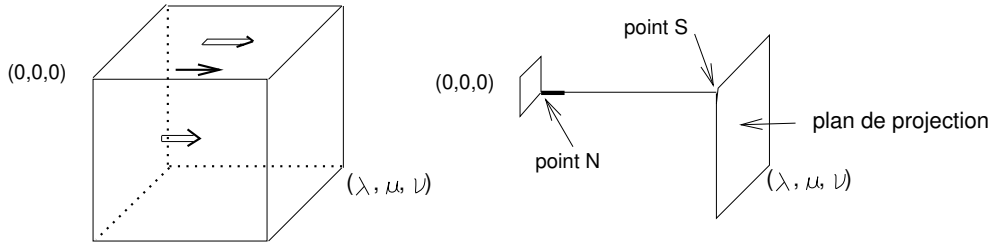


FIG. *9.3* - Cas où $a'_1 \geq 0$, $b'_1 \geq 0$ et $c'_1 \geq 0$

Preuve : Considérons un chemin $\mathcal{P}$ reliant les points (0,0,0) et (λ, μ, ν). Nous voulons construire un chemin $\mathcal{P}'$, de la forme représentée dans la figure 9.3 à droite, tel que la contrainte globale $\Gamma_{\mathcal{P}}$ soit subsumée par la contrainte globale $\Gamma_{\mathcal{P}'}$.

Construction du chemin $\mathcal{P}'$:

Soit N le point où on trouve le premier pas en x de $\mathcal{P}$. Le chemin $\mathcal{P}'$ coïncide avec $\mathcal{P}$ jusqu'au point N.

Après le point N, le chemin $\mathcal{P}'$ est obtenu de la façon suivante :

- En partant de N, on effectue une séquence de pas en x jusqu'à rencontrer le plan 0_{YZ} qui passe par le point (λ, μ, ν). Cette intersection détermine le point S.
- On projette la partie du chemin $\mathcal{P}$ située après N sur le plan 0_{YZ} qui passe par le point (λ, μ, ν). Cette projection relie les points S et (λ, μ, ν).

On obtient ainsi le chemin $\mathcal{P}'$.

Preuve de $\Gamma_{\mathcal{P}} \Rightarrow \Gamma_{\mathcal{P}'}$:

Nous montrons ensuite que la contrainte globale de $\mathcal{P}$ est subsumée par la contrainte globale de $\mathcal{P}'$.

Soit $\mathcal{P}_1$ (resp. $\mathcal{P}'_1$) la partie de chemin $\mathcal{P}$ (resp. $\mathcal{P}'$) située après le point N. Comme $\mathcal{P}$ et $\mathcal{P}'$ coïncident jusqu'au point N, il suffit de démontrer que $\Gamma_{\mathcal{P}_1} \Rightarrow \Gamma_{\mathcal{P}'_1}$. D'autre part, comme

$\Phi_1(N)$ est une contrainte de $\mathcal{P}$ (puisque dans $\mathcal{P}$, il y a un segment horizontal qui part de N), il suffit de prouver que :

1. $\Gamma^1_{\mathcal{P}'_1} = \Phi_1(N)$.
2. $\Gamma^2_{\mathcal{P}_1} \Rightarrow \Gamma^2_{\mathcal{P}'_1}$
3. $\Gamma^3_{\mathcal{P}_1} \Rightarrow \Gamma^3_{\mathcal{P}'_1}$

- **Preuve de $\Gamma^1_{\mathcal{P}'_1} = \Phi_1(N)$:** Comme $\forall \alpha, \beta, \gamma\ \Phi_1(\alpha, \beta, \gamma) \Rightarrow\ \Phi_1(\alpha+1, \beta, \gamma)\,(a'_1 \geq 0)$, la contrainte horizontale la plus forte après le point N de $\mathcal{P}'$ est $\Phi_1(N)$. Par conséquent, $\Gamma^1_{\mathcal{P}'_1} = \Phi_1(N)$.
- **Preuve de $\Gamma^2_{\mathcal{P}_1} \Rightarrow \Gamma^2_{\mathcal{P}'_1}$:** Comme $\forall \alpha, \beta, \gamma\ \Phi_2(\alpha, \beta, \gamma) \Rightarrow\ \Phi_2(\alpha+1, \beta, \gamma)\ (b'_1 \geq 0)$, en projetant chaque segment vertical de $\mathcal{P}$ sur le plan $0YZ$ qui passe par (λ, μ, ν), on obtient un segment vertical de $\mathcal{P}'$ de contrainte plus faible. Il s'ensuit, qu'après le point N, chaque contrainte verticale de $\mathcal{P}'$ est plus faible qu'une contrainte de $\mathcal{P}$. Par conséquent, $\Gamma^2_{\mathcal{P}_1} \Rightarrow \Gamma^2_{\mathcal{P}'_1}$.
- **Preuve de $\Gamma^3_{\mathcal{P}_1} \Rightarrow \Gamma^3_{\mathcal{P}'_1}$:** Comme $\forall \alpha, \beta, \gamma\ \Phi_3(\alpha, \beta, \gamma) \Rightarrow\ \Phi_3(\alpha+1, \beta, \gamma)\ (c'_1 \geq 0)$, en projetant chaque segment transversal de $\mathcal{P}$ sur le plan $0YZ$ qui passe par (λ, μ, ν), on obtient un segment transversal de $\mathcal{P}'$ de contrainte plus faible. Il s'ensuit, qu'après le point N, chaque contrainte transversale de $\mathcal{P}'$ est plus faible qu'une contrainte de $\mathcal{P}$. Par conséquent, $\Gamma^3_{\mathcal{P}_1} \Rightarrow \Gamma^3_{\mathcal{P}'_1}$

Donc, $\Gamma_{\mathcal{P}} \Rightarrow \Gamma_{\mathcal{P}'}$, avec $\mathcal{P}'$ de la forme représentée à la figure 9.3 à droite. □

Il y a 5 autres combinaisons d'implications de contraintes pour lesquelles on peut faire un raisonnement analogue à celui-là (voir figure 9.4).

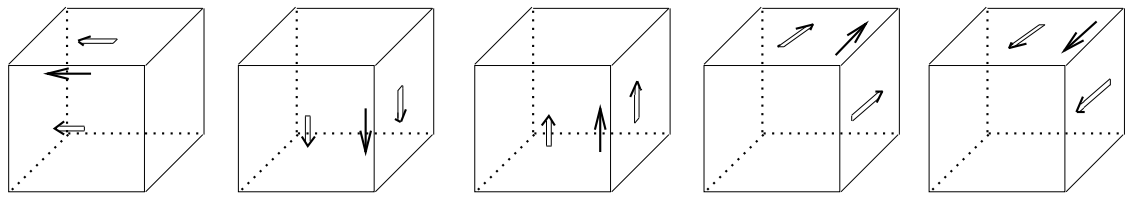


FIG. 9.4 - *Cas analogues à $a'_1 \geq 0$, $b'_1 \geq 0$ et $c'_1 \geq 0$*

9.3 Cas Simple 3

Proposition 9.3 *Pour un programme* Π *avec* $a'_1 \geq 0$, $b'_1 \geq 0$, $c'_2 \leq 0$ *e* $c'_3 \leq 0$ *(correspondant à des implications de contraintes comme celles représentées à la figure 9.5 à gauche), les chemins dans l'ensemble* $\mathcal{S}'(\lambda, \mu, \nu)$ *sont tous de la forme représentée à la figure 9.5 à droite.*

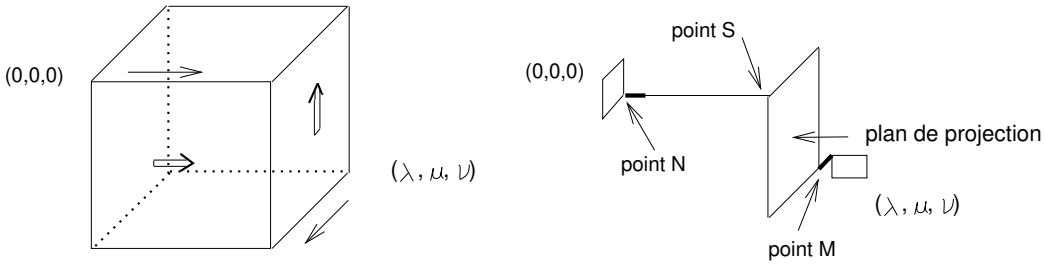


FIG. *9.5* - Cas où $a'_1 \geq 0$, $b'_1 \geq 0$, $c'_2 \leq 0$ e $c'_3 \leq 0$

Preuve : Considérons un chemin $\mathcal{P}$ reliant les points (0,0,0) et (λ, μ, ν). Soit N le point où on trouve le premier pas en x de $\mathcal{P}$ et M le point où on trouve le dernier pas en z de $\mathcal{P}$. Comme dans les cas simples 1 et 2, nous voulons construire un chemin $\mathcal{P}'$, de la forme représentée dans la figure 9.5, tel que la contrainte globale $\Gamma_{\mathcal{P}}$ soit subsumée par la contrainte globale $\Gamma_{\mathcal{P}'}$.

Construction du chemin $\mathcal{P}'$:

Le chemin $\mathcal{P}'$ coïncide avec $\mathcal{P}$ jusqu'au point N et après le point M.

Entre les points N et M, le chemin $\mathcal{P}'$ est obtenu de la façon suivante :

- En partant de N, on effectue une séquence de pas en x jusqu'à rencontrer le plan 0_{YZ} qui passe par le point M. Cette intersection détermine le point S.
- On projette la partie du chemin $\mathcal{P}$ située entre les points N et M sur le plan 0_{YZ} qui passe par le point (λ, μ, ν). Cette projection relie les points S et M.

On obtient ainsi un chemin $\mathcal{P}'$.

Preuve de $\Gamma_{\mathcal{P}} \Rightarrow \Gamma_{\mathcal{P}'}$:

Nous montrons ensuite que la contrainte globale de $\mathcal{P}$ est subsumée par la contrainte globale de $\mathcal{P}'$.

Soit $\mathcal{P}_1$ (resp. $\mathcal{P}'_1$) la partie de chemin $\mathcal{P}$ (resp. $\mathcal{P}'$) située entre les points N et M. Comme $\mathcal{P}$ et $\mathcal{P}'$ coïncident jusqu'au point N et après le point M, il suffit de démontrer que $\Gamma_{\mathcal{P}_1} \Rightarrow \Gamma_{\mathcal{P}'_1}$. D'autre part, comme $\Phi_1(N)$ et $\Phi_3(M)$ sont des contraintes de $\mathcal{P}$ (puisque dans $\mathcal{P}$, il y a un segment horizontal qui part de N et un segment transversal qui part de M), il suffit de prouver que :

1. $\Gamma^1_{\mathcal{P}'_1} = \Phi_1(N)$.
2. $\Gamma^2_{\mathcal{P}_1} \Rightarrow \Gamma^2_{\mathcal{P}'_1}$
3. $\Gamma^3_{\mathcal{P}_1} = \Phi_3(M)$

- **Preuve de $\Gamma^1_{\mathcal{P}'_1} = \Phi_1(N)$:** Comme $\forall \alpha, \beta, \gamma\ \Phi_1(\alpha, \beta, \gamma) \Rightarrow\ \Phi_1(\alpha + 1, \beta, \gamma)\ (a'_1 \geq 0)$, la contrainte horizontale la plus forte de $\mathcal{P}'$ entre les points N et M est $\Phi_1(N)$. Par conséquent, $\Gamma^1_{\mathcal{P}'_1} = \Phi_1(N)$.
- **Preuve de $\Gamma^2_{\mathcal{P}_1} \Rightarrow \Gamma^2_{\mathcal{P}'_1}$:** Comme $\forall \alpha, \beta, \gamma\ \Phi_2(\alpha, \beta, \gamma) \Rightarrow\ \Phi_2(\alpha + 1, \beta, \gamma)\ (b'_1 \geq 0)$ et comme $\forall \alpha, \beta, \gamma\ \Phi_2(\alpha, \beta, \gamma + 1) \Rightarrow\ \Phi_2(\alpha, \beta, \gamma)\ (b'_3 \leq 0)$, chaque segment vertical de $\mathcal{P}'$ a une contrainte plus faible que celle du segment de $\mathcal{P}$ qui la projette. Par conséquent, $\Gamma^2_{\mathcal{P}_1} \Rightarrow \Gamma^2_{\mathcal{P}'_1}$.
- **Preuve de $\Gamma^3_{\mathcal{P}_1} = \Phi_3(M)$:** Comme $\forall \alpha, \beta, \gamma\ \Phi_3(\alpha, \beta, \gamma + 1) \Rightarrow\ \Phi_3(\alpha, \beta, \gamma)\ (c'_3 \leq 0)$, la contrainte transversale la plus forte de $\mathcal{P}'$ entre les points N et M est $\Phi_3(M)$. Par conséquent, $\Gamma^3_{\mathcal{P}'_1} = \Phi_3(M)$.

Donc, $\Gamma_{\mathcal{P}} \Rightarrow \Gamma_{\mathcal{P}'}$, avec $\mathcal{P}'$ de la forme représentée à la figure 9.5 à droite. □

Il y a 5 autres combinaisons d'implications de contraintes pour lesquelles on peut faire un raisonnement analogue à ce cas. Ces combinaisons d'implications peuvent être obtenues par les transformations données au chapitre 4.

9.4 Cas Simple 4

Proposition 9.4 *Pour un programme* Π *avec* $a'_1 \geq 0$, $a'_2 \geq 0$, $b'_3 \leq 0$ *et* $c'_3 \leq 0$ *(correspondant à des implications de contraintes comme celles représentées à la figure 9.6 à gauche),*

les chemins dans l'ensemble $\mathcal{S}'(\lambda, \mu, \nu)$ sont tous de la forme représentée à la figure 9.6 à droite.

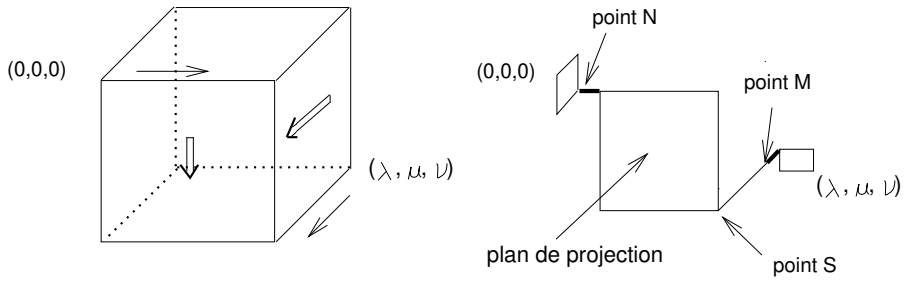


FIG. *9.6* - Cas où $a'_1 \geq 0$, $a'_2 \geq 0$, $b'_3 \leq 0$ et $c'_3 \leq 0$

Preuve : Considérons un chemin $\mathcal{P}$ reliant les points (0,0,0) et (λ, μ, ν). Soit N le point où on trouve le premier pas en x de $\mathcal{P}$ et M le point où on trouve le dernier pas en z de $\mathcal{P}$. Comme dans les cas simples 1 et 2, nous voulons construire un chemin $\mathcal{P}'$, de la forme représentée dans la figure 9.5, tel que la contrainte globale $\Gamma_{\mathcal{P}}$ soit subsumée par la contrainte globale $\Gamma_{\mathcal{P}'}$.

Construction du chemin $\mathcal{P}'$:

Le chemin $\mathcal{P}'$ coïncide avec $\mathcal{P}$ jusqu'au point N et après le point M.

Entre les points N et M, le chemin $\mathcal{P}'$ est obtenu de la façon suivante :

- On projette la partie du chemin $\mathcal{P}$ située entre les points N et M sur le plan $0YZ$ qui passe par le point N. Soit S le dernier point du chemin projeté. Cette projection relie les points N et S.
- On relie les points S et M par des pas en z.

On obtient ainsi un chemin $\mathcal{P}'$.

Preuve de $\Gamma_{\mathcal{P}} \Rightarrow \Gamma_{\mathcal{P}'}$:

Nous montrons ensuite que la contrainte globale de $\mathcal{P}$ est subsumée par la contrainte globale de $\mathcal{P}'$.

Soit $\mathcal{P}_1$ (resp. $\mathcal{P}'_1$) la partie de chemin $\mathcal{P}$ (resp. $\mathcal{P}'$) située entre les points N et M. Comme $\mathcal{P}$ et $\mathcal{P}'$ coïncident jusqu'au point N et après le point M, il suffit de démontrer que $\Gamma_{\mathcal{P}_1} \Rightarrow \Gamma_{\mathcal{P}'_1}$. D'autre part, comme $\Phi_1(N)$ et $\Phi_3(M)$ sont contraintes de $\mathcal{P}$ (puisque dans $\mathcal{P}$, il y a un segment horizontal qui part de N et un segment transversal qui part de M), il suffit de prouver que :

1. $\Gamma^1_{\mathcal{P}'_1} = \Phi_1(N)$.
2. $\Gamma^2_{\mathcal{P}_1} \Rightarrow \Gamma^2_{\mathcal{P}'_1}$
3. $\Gamma^3_{\mathcal{P}_1} = \Phi_3(M)$

- **Preuve de $\Gamma^1_{\mathcal{P}'_1} = \Phi_1(N)$:** Comme $\forall \alpha, \beta, \gamma\ \Phi_1(\alpha, \beta, \gamma) \Rightarrow\ \Phi_1(\alpha + 1, \beta, \gamma)\ (a'_1 \geq 0)$ et $\forall \alpha, \beta, \gamma\ \Phi_1(\alpha, \beta, \gamma) \Rightarrow\ \Phi_1(\alpha, \beta + 1, \gamma)$, la contrainte horizontale la plus forte de $\mathcal{P}'$ entre les points N et M est $\Phi_1(N)$.
- **Preuve de $\Gamma^2_{\mathcal{P}_1} \Rightarrow \Gamma^2_{\mathcal{P}'_1}$:** Comme $\forall \alpha, \beta, \gamma\ \Phi_2(\alpha, \beta, \gamma + 3) \Rightarrow\ \Phi_2(\alpha, \beta, \gamma)\ (b'_3 \leq 0)$, en projetant chaque segment vertical de $\mathcal{P}$ sur le plan $0yz$ qui passe par N, on obtient un segment vertical de $\mathcal{P}'$ de contrainte plus faible. Il s'ensuit les contraintes verticales de $\mathcal{P}'$ sont plus faibles que celles de $\mathcal{P}$.
- **Preuve de $\Gamma^3_{\mathcal{P}_1} = \Phi_3(M)$:** Comme $\forall \alpha, \beta, \gamma\ \Phi_3(\alpha, \beta, \gamma + 1) \Rightarrow\ \Phi_3(\alpha, \beta, \gamma)\ (c'_3 \leq 0)$, la contrainte transversale la plus forte de $\mathcal{P}'$ entre les points N et M est $\Phi_3(M)$. Par conséquent, $\Gamma^3_{\mathcal{P}'_1} = \Phi_3(M)$.

Donc, $\Gamma_{\mathcal{P}} \Rightarrow \Gamma_{\mathcal{P}'}$, avec $\mathcal{P}'$ de la forme représentée à la figure 9.6 à droite. □

Des résultats analogues peuvent être établis pour 5 autres combinaisons d'implications de contraintes. Ces combinaisons d'implications peuvent être obtenues par les transformations données au chapitre 4.

9.5 Cas Simple 5

Proposition 9.5 *Pour un programme* Π *avec* $a'_1 \geq 0$, $b'_1 \geq 0$, $b'_3 \leq 0$ *et* $c'_3 \leq 0$ *(correspondant à des implications de contraintes comme celles représentées à la figure 9.7 à gauche),*

les chemins dans l'ensemble $\mathcal{S}'(\lambda, \mu, \nu)$ sont tous de la forme représentée à la figure 9.7 à droite.

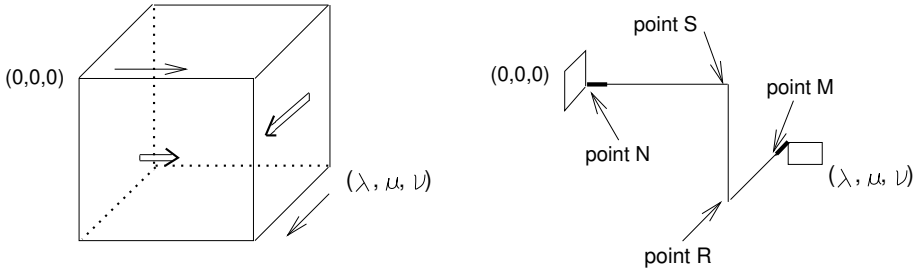


FIG. *9.7* - Cas où $a'_1 \geq 0$, $b'_1 \geq 0$, $b'_3 \leq 0$ et $c'_3 \leq 0$

Preuve : Considérons un chemin $\mathcal{P}$ reliant les points (0,0,0) et (λ, μ, ν). Soit N le point où on trouve le premier pas en x de $\mathcal{P}$ et M le point où on trouve le dernier pas en z de $\mathcal{P}$. Nous voulons construire un chemin $\mathcal{P}'$, de la forme représentée dans la figure 9.7, tel que la contrainte globale $\Gamma_{\mathcal{P}}$ est subsumée par la contrainte globale $\Gamma_{\mathcal{P}'}$.

Construction du chemin $\mathcal{P}'$:

Le chemin $\mathcal{P}'$ coïncide avec $\mathcal{P}$ jusqu'au point N et après le point M.

Entre les points N et M, le chemin $\mathcal{P}'$ est obtenu de la façon suivante :

- En partant de N, on effectue une séquence de pas en x jusqu'à rencontrer le plan $0yz$ qui passe par le point M. Cette intersection détermine le point S.
- En partant de S, on effectue une séquence de pas en y jusqu'à rencontrer le plan $0xz$ qui passe par le point M. Cette intersection détermine le point R.
- On relie les points R et M par des pas en z.

On obtient ainsi un chemin $\mathcal{P}'$.

Preuve de $\Gamma_{\mathcal{P}} \Rightarrow \Gamma_{\mathcal{P}'}$:

Nous montrons ensuite que la contrainte globale de $\mathcal{P}$ est subsumée par la contrainte globale de $\mathcal{P}'$.

Soit $\mathcal{P}_1$ (resp. $\mathcal{P}'_1$) la partie de chemin $\mathcal{P}$ (resp. $\mathcal{P}'$) située entre les points N et M. Comme $\mathcal{P}$ et $\mathcal{P}'$ coïncident jusqu'au point N et après le point M, il suffit de démontrer que $\Gamma_{\mathcal{P}_1} \Rightarrow \Gamma_{\mathcal{P}'_1}$. D'autre part, comme $\Phi_1(N)$ et $\Phi_3(M)$ sont contraintes de $\mathcal{P}$ (puisque dans $\mathcal{P}$, il y a un segment horizontal qui part de N et un segment transversal qui part de M), il suffit de prouver que :

1. $\Gamma^1_{\mathcal{P}'_1} = \Phi_1(N)$.
2. $\Gamma^2_{\mathcal{P}_1} \Rightarrow \Gamma^2_{\mathcal{P}'_1}$
3. $\Gamma^3_{\mathcal{P}_1} = \Phi_3(M)$

- **Preuve de $\Gamma^1_{\mathcal{P}'_1} = \Phi_1(N)$:** Comme $\forall \alpha, \beta, \gamma\ \Phi_1(\alpha, \beta, \gamma) \Rightarrow\ \Phi_1(\alpha + 1, \beta, \gamma)\ (a'_1 \geq 0)$, la contrainte horizontale la plus forte de $\mathcal{P}'$ entre les points N et M est $\Phi_1(N)$. Par conséquent, $\Gamma^1_{\mathcal{P}'_1} = \Phi_1(N)$.
- **Preuve de $\Gamma^2_{\mathcal{P}_1} \Rightarrow \Gamma^2_{\mathcal{P}'_1}$:** Comme $\forall \alpha, \beta, \gamma\ \Phi_2(\alpha, \beta, \gamma) \Rightarrow\ \Phi_2(\alpha + 1, \beta, \gamma)\ (b'_1 \geq 0)$ et comme $\forall \alpha, \beta, \gamma\ \Phi_2(\alpha, \beta, \gamma + 1) \Rightarrow\ \Phi_2(\alpha, \beta, \gamma)\ (b'_3 \leq 0)$, chaque segment de $\mathcal{P}'$ a une contrainte verticale plus faible qu'un segment de $\mathcal{P}$. Par conséquent, $\Gamma^2_{\mathcal{P}_1} \Rightarrow \Gamma^2_{\mathcal{P}'_1}$.
- **Preuve de $\Gamma^3_{\mathcal{P}_1} \Rightarrow \Gamma^3_{\mathcal{P}'_1}$:** Comme $\forall \alpha, \beta, \gamma\ \Phi_3(\alpha, \beta, \gamma + 1) \Rightarrow\ \Phi_3(\alpha, \beta, \gamma)\ (c'_3 \leq 0)$, la contrainte transversale la plus forte de $\mathcal{P}'$ entre les points N et M est $\Phi_3(M)$. Par conséquent, $\Gamma^3_{\mathcal{P}'_1} = \Phi_3(M)$.

Donc, $\Gamma_{\mathcal{P}} \Rightarrow \Gamma_{\mathcal{P}'}$, avec $\mathcal{P}'$ de la forme représentée à la figure 9.7 à droite. □

Il y a 5 autres combinaisons d'implications de contraintes pour lesquelles on peut faire un raisonnement analogue (voir chapitre 4).

9.6 Obtention de $\Delta(\lambda, \mu, \nu)$

Les chemins de $\mathcal{S}'(\lambda, \mu, \nu)$, obtenus par des $\bigvee$-simplications pour les cinq cas simples, ont des parties contenues dans des plans 0_{XY}, 0_{XY} et 0_{YZ} (voir figures 9.1,9.3, 9.5, 9.6 et 9.7).

Chacun de ces plans est engendré par des applications itératives de deux règles récursives du programme (le plan $0XY$ par l'application itérative des règles $\mathcal{R}_1$ et $\mathcal{R}_2$, le plan $0XZ$ par l'application itérative de $\mathcal{R}_1$ et $\mathcal{R}_3$ et le plan $0YZ$ par l'application itérative de $\mathcal{R}_2$ et $\mathcal{R}_3$).

Par conséquent, ces parties du chemin $\mathcal{P}'$ (contenues dans des plans $0XY$, $0XY$ ou $0YZ$) peuvent être remplacées par des parties avec des formes plus simples qui les subsument (voir chapitres 6 et 7). Cela se fait en appliquant notre méthode pour deux règles récursives.

Ainsi, on pourra exprimer $\Delta(\lambda,\mu,\nu)$ par une formule arithmétique. Cela est illustré dans l'exemple suivant.

Exemple 9.6 Soit π_8 le programme:

$$\begin{array}{lcl} p(x,y,z) & :- & \Theta(x,y,z). \\ p(x+6,y+2,z-1) & :- & x-y\geq 0,\ p(x,y,z). \\ p(x-1,y+1,z+2) & :- & y\geq 0,\ p(x,y,z). \\ p(x-4,y-3,z+4) & :- & y+z\geq 0,\ p(x,y,z). \end{array}$$

On a :

$$\begin{array}{rclcrcl} a'_1 & = & d_1a_1+e_1b_1+f_1c_1 & = & 1\times 6+(-1)\times 2+0\times(-1) & = & 4 \\ a'_2 & = & d_1a_2+e_1b_2+f_1c_2 & = & 1\times(-1)+(-1)\times 1+0\times 2 & = & -2 \\ a'_3 & = & d_1a_3+e_1b_3+f_1c_3 & = & 1\times(-4)+(-1)\times(-3)+0\times 4 & = & -1 \\ b'_1 & = & d_2a_1+e_2b_1+f_2c_1 & = & 0\times 6+1\times 2+0\times(-1) & = & 2 \\ b'_2 & = & d_2a_2+e_2b_2+f_2c_2 & = & 0\times(-1)+1\times 1+0\times 2 & = & 1 \\ b'_3 & = & d_2a_3+e_2b_3+f_2c_3 & = & 0\times(-4)+1\times(-3)+0\times 4 & = & -3 \\ c'_1 & = & d_3a_1+e_3b_1+f_3c_1 & = & 0\times 6+1\times 2+1\times(-1) & = & 1 \\ c'_2 & = & d_3a_2+e_3b_2+f_3c_2 & = & 0\times(-1)+1\times 1+1\times 2 & = & 3 \\ c'_3 & = & d_3a_3+e_3b_3+f_3c_3 & = & 0\times(-4)+1\times(-3)+1\times 4 & = & 1 \end{array}$$

Comme $a'_2\leq 0$, $a'_3\leq 0$, $b'_1\geq 0$ et $c'_1\geq 0$, la forme des chemins en $\Delta(\lambda,\mu,\nu)$ peut être calculée par le cas simple 1 (voir figure 9.8).

En plus, comme $b'_3\leq 0$ et $c'_3\geq 0$, la contrainte globale de tout chemin entre les points $(\lambda,0,0)$ et (λ,μ,ν) est subsumée par la contrainte globale du chemin $\ll(\lambda,0,0),\ldots,(\lambda,\mu,0),\ldots,(\lambda,\mu,\nu)\gg$ (voir figure 9.9).

La $\bigwedge$-simplification s'effectue en fonction des flèches simples ($a'_1\geq 0$, $b'_1\geq 0$ et $c'_1\geq 0$). Les contraintes plus fortes sont representées par un trait plus épais dans la figure 9.10.

Il s'ensuit que, pour $\lambda>0$, $\mu>0$ et $\nu>0$, $\Delta(\lambda,\mu,\nu)$ équivaut à $\Phi_1(0,0,0)\wedge\Phi_2(\lambda,0,0)\wedge$

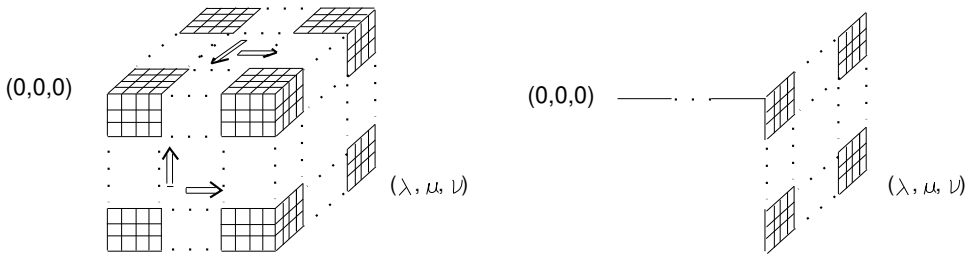


FIG. 9.8 - $\bigvee$-*simplification - étape 1*

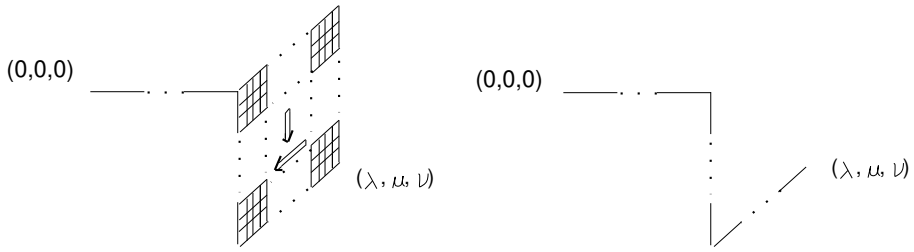


FIG. 9.9 - $\bigvee$-*simplification - étape 2*

$\Phi_3(\lambda, \nu, 0)$, c'est-à-dire $x - y \geq 0 \wedge y + 2\lambda \geq 0 \wedge y + z + \lambda + 3\nu \geq 0$.

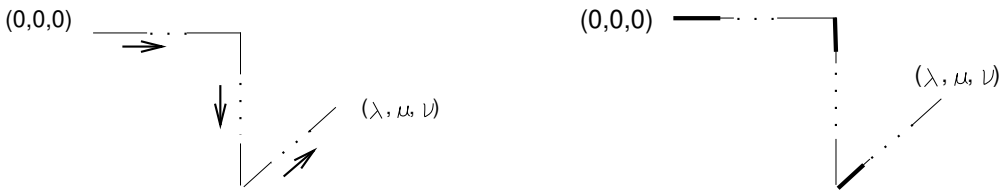


FIG. 9.10 - $\bigwedge$-*simplification*

Chapitre 10

Classification des Cas Difficiles (Trois Règles)

Nous considérons dans ce chapitre des programmes II dont les implications de contraintes ont des sens différents de ceux indiqués au chapitre 9: il s'agit des cas difficiles. Comme dans le cas de deux règles récursives, l'étape de $\bigvee$-simplifications pour ces cas difficiles est faite en utilisant la notion de *contrainte périodique* et la notion de *motif* associé à des contraintes du programme. Nous présenterons d'abord des résultats liés à ces notions pour des programmes avec trois règles récursives.

Nous classifierons ces cas difficiles en deux :

- Cas difficile 1 : correspondant aux programmes avec deux contraintes *co-périodiques*.

- Cas difficile 2 : correspondant aux programmes avec une contrainte *bi-périodique* et chacune des autres bi-périodiquement strictement croissante ou décroissante.

Ce chapitre a 2 parties. La première donne les définitions de base concernant les notions de période et motif. La deuxième donne une caractérisation algébrique pour l'existence de telles périodes.

10.1 Périodes et Motifs

Soit M un point de coordonnées (α, β, γ) et soit $\vec{\sigma}$ le vecteur $(\sigma_1, \sigma_2, \sigma_3)$. L'expression $M + \vec{\sigma}$ représente le vecteur $(\alpha + \sigma_1, \beta + \sigma_2, \gamma + \sigma_3)$.

Une contrainte Φ_k est périodique ssi il existe un vecteur $\vec{\sigma}$, tel que, pour tout M: $\Phi_k(M) = \Phi_k(M + \vec{\sigma})$.

Les notions de *période*, *motif* et *sous-motif* sont analogues à celles données dans la sous-partie 7.2.

Définition 10.1 (Co-périodicité) *Deux contraintes Φ_k et $\Phi_{k'}$ ($k \neq k'$ et $k, k' \in \{1, 2, 3\}$) sont* co-périodiques *ssi il existe un vecteur non-nul $(\sigma_1, \sigma_2, \sigma_3)$ d' entiers naturels, tel que:*

- $\Phi_k(0, 0, 0) = \Phi_k(\sigma_1, \sigma_2, \sigma_3)$.

  et

- $\Phi_{k'}(0, 0, 0) = \Phi_{k'}(\sigma_1, \sigma_2, \sigma_3)$.

On appelle co-période *de Φ_k e $\Phi_{k'}$ un vecteur (τ_1, τ_2, τ_3) non nul d'entiers naturels premiers entre eux*[1] *tel que $\Phi_k(0, 0) = \Phi_k(\tau_1, \tau_2, \tau_3)$ et $\Phi_{k'}(0, 0) = \Phi_{k'}(\tau_1, \tau_2, \tau_3)$.*

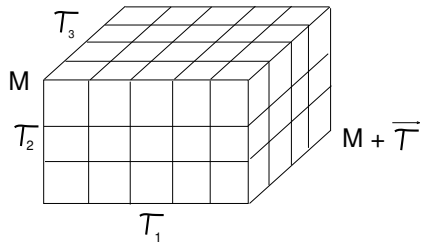
Notons que si $\Phi_i(0, 0, 0) = \Phi_i(\sigma_1, \sigma_2, \sigma_3)$ alors $\forall \alpha, \beta, \gamma\ \Phi_i(\alpha, \beta, \gamma) = \Phi_i(\alpha + n\sigma_1, \beta + n\sigma_2 + n, \gamma + n\sigma_3)$, pour tout $n \in \mathbb{N}$.

Le *co-motif* associé à cette co-période (et aussi noté par $\vec{\tau}$) est l'ensemble de tous les chemins reliant un point donné M au point $M + \vec{\tau}$ (voir figure 10.1).

Etant donné une co-période $\vec{\tau}$ pour deux contraintes Φ_k et $\Phi_{k'}$ ($k \neq k'$ et $k, k' \in \{1, 2, 3\}$), nous disons qu'une contrainte $\Phi_{k''}$ ($k'' \neq k$, $k'' \neq k'$, et $k'' \in \{1, 2, 3\}$ est *co-périodiquement croissante* (resp. *co-périodiquement décroissante*) ssi $\Phi_{k''}(\vec{0}) \leq \Phi_{k''}(\vec{\tau})$ (resp. $\Phi_{k''}(\vec{0}) \geq \Phi_{k''}(\vec{\tau})$).

Exemple 10.2 Soit π_9 le programme :

1. c'est-à-dire, tels que $\text{pgcd}(\tau_1, \tau_2, \tau_3) = 1$.



$\Phi_k(M) = \Phi_k(M + \vec{\tau})$ $\Phi_{k'}(M) = \Phi_{k'}(M + \vec{\tau})$

FIG. 10.1 - *Co-motif pour Φ_k et $\Phi_{k'}$*

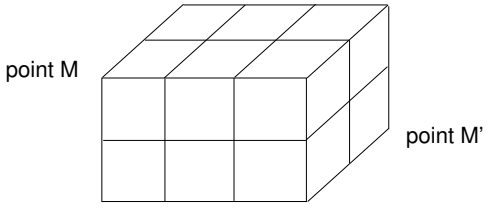
$p(x, y, z)$	:-	$\Theta(x, y, z)$.	règle $\mathcal{R}_0$
$p(x + 2, y - 4, z - 2)$	:-	$x \geq 0,\ p(x, y, z)$	règle $\mathcal{R}_1$
$p(x - 5, y + 9, z - 1)$	:-	$y \geq 0,\ p(x, y, z)$	règle $\mathcal{R}_2$
$p(x + 2, y - 3, z + 1)$	:-	$z \geq 0,\ p(x, y, z)$	règle $\mathcal{R}_3$

Le point (α, β, γ) représente le tuple $< x+2\alpha-5\beta+2\gamma, y-4\alpha+9\beta-3\gamma, z-2\alpha-\beta+\gamma >$.

Le point $(0, 0, 0)$ représente le tuple $< x, y, z >$ et le point $(3, 2, 2)$ représente le tuple $< x, y, z - 6 >$.

Il s'ensuit que $\Phi_1(0, 0, 0) = \Phi_1(3, 2, 2)$ et que $\Phi_2(0, 0, 0) = \Phi_2(3, 2, 2)$.

Les contraintes Φ_1 et Φ_2 sont ainsi co-périodiques, avec co-motif (3,2,2) (voir figure 10.2).



$\Phi_1(M) = \Phi_1(M')$ $\Phi_2(M) = \Phi_2(M')$

FIG. 10.2 - *Co-motif pour Φ_1 et Φ_2*

La contrainte Φ_3 est co-périodiquement croissante pour le motif (3,2,2): comme $\Phi_3(3, 2, 2) = \Phi_3(0, 0, 0) - 6$, il s'ensuit que $\Phi_3(0, 0, 0) \leq \Phi_3(3, 2, 2)$.

□

Définition 10.3 (Bi-périodicité) *Une contrainte* Φ_k *est* bi-périodique *ssi il existe deux vecteurs non-nuls* $(\sigma_1, \sigma_2, \sigma_3)$ *et* (ρ_1, ρ_2, ρ_3) *d'entiers naturels tels que:*

- $\exists i, j \in \{1,2,3\}\ \ i \neq j \wedge \sigma_i = \rho_j = 0$
  et
- $\Phi_k(0,0,0) = \Phi_k(\sigma_1, \sigma_2, \sigma_3)\ \wedge\ \Phi_k(0,0,0) = \Phi_k(\rho_1, \rho_2, \rho_3)$.

Les relations ci-dessous établissent que la contrainte Φ_k *est périodique en deux des plans* $0XY$, $0XZ$, *et* $0YZ$. *Soit* $\vec{\tau^1}$ *et* $\vec{\tau^2}$ *sont des périodes de* Φ_k *dans ces plans. La* bi-période *de* Φ_k *est la paire* $\{\vec{\tau^1}, \vec{\tau^2}\}$.

Un bi-motif *associé* $\{\vec{\tau^1}, \vec{\tau^2}\}$ *est la paire de motifs associé à* $\vec{\tau^1}$ *et* $\vec{\tau^2}$ *(voir figure 10.3, pour le cas où* $i = 1$ *et* $j = 3$*).*

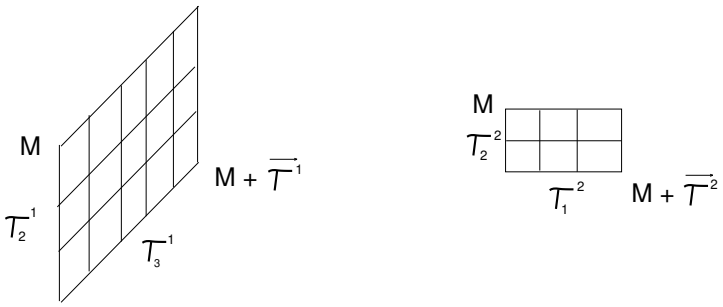


FIG. *10.3* - Motifs pour Φ_k

Etant donné une bi-période $\{\vec{\tau^1}, \vec{\tau^2}\}$ pour une contrainte Φ_k, nous disons qu'une contrainte $\Phi_{k'}$ ($k' \neq k$ et $k' \in \{1,2,3\}$) est *bi-périodiquement strictement croissante* (resp. *bi-périodiquement strictement décroissante*) ssi $\Phi_{k'}(\vec{0}) < \Phi_{k'}(\vec{\tau^1}) \wedge \Phi_{k'}(\vec{0}) < \Phi_{k'}(\vec{\tau^2})$ (resp. $\Phi_{k'}(\vec{0}) > \Phi_{k'}(\vec{\tau^1}) \wedge \Phi_{k'}(\vec{0}) > \Phi_{k'}(\vec{\tau^2})$).

Exemple 10.4 Soit π_{10} le programme:

$p(x,y,z)$	:-	$\Theta(x,y,z)$.	règle $\mathcal{R}_0$
$p(x+3, y+2, z-2)$	:-	$x \geq 0,\ p(x,y,z)$	règle $\mathcal{R}_1$
$p(x-4, y+1, z+5)$	:-	$y \geq 0,\ p(x,y,z)$	règle $\mathcal{R}_2$
$p(x-2, y-1, z+2)$	:	$z \geq 0,\ p(x,y,z)$	règle $\mathcal{R}_3$

Le point (α, β, γ) représente le tuple $< x+3\alpha-4\beta-2\gamma, y+2\alpha+\beta-\gamma, z-2\alpha+5\beta+2\gamma >$.

Le point $(4,3,0)$ représente le tuple $< x, y+11, z+7 >$ et le point $(2,0,3)$ représente le tuple $< x, y+1, z+2 >$.

Il s'ensuit que $\Phi_1(0,0,0) = \Phi_1(4,3,0) \wedge \Phi_1(0,0,0) = \Phi_1(2,0,3)$.

La contrainte Φ_1 est ainsi bi-périodique, avec motifs (4,3,0) et (2,0,3). Ces deux motifs sont situés dans deux plans différents (voir figure 10.4).

$\Phi_1(M) = \Phi_1(M')$ et $\Phi_1(N) = \Phi_1(N')$

FIG. 10.4 - *Bi-périodicité de* Φ_1

Nous avons:

$$\begin{aligned}
\Phi_2(4,3,0) &= \Phi_2(0,0,0) + 11 \\
\Phi_3(4,3,0) &= \Phi_2(0,0,0) + 1 \\
\Phi_2(2,0,3) &= \Phi_2(0,0,0) + 7 \\
\Phi_2(2,0,3) &= \Phi_2(0,0,0) + 2
\end{aligned}$$

La contrainte Φ_2 et la contrainte Φ_3 sont ainsi bi-périodiquement strictement croissantes pour le bi-motif (4,3,0) et (2,0,3) de Φ_1.

□

10.2 Caractérisation Algébrique des Cas Difficiles

Dans cette partie, nous donnons une caractérisation algébrique pour que:

– Il ait deux contraintes co-périodiques.

– Il ait une contrainte Φ_i bi-périodique, avec Φ_j $(j \neq i)$ bi-périodiquement strictement croissante ou décroissante.

Ces deux types de programmes définirons deux classes de programmes difficiles.

Considérons les équations ci-dessous:

$$\begin{array}{lll} a'_1\alpha + a'_2\beta + a'_3\gamma = 0 & & \text{équation } \mathcal{E}_1 \\ b'_1\alpha + b'_2\beta + b'_3\gamma = 0 & & \text{équation } \mathcal{E}_2 \\ c'_1\alpha + c'_2\beta + c'_3\gamma = 0 & & \text{équation } \mathcal{E}_3 \end{array}$$

où α, β et γ sont les inconnues (variables), a'_k note la constante $d_1a_k + e_1b_k + f_1c_k$, b'_k note $d_2a_k + e_2b_k + f_2c_k$, and c'_k note $d_3a_k + e_3b_k + f_3c_k$ $(k \in \{1,2,3\})$.

L'équation $\mathcal{E}_1$ énonce que la contrainte Φ_1 de la règle $\mathcal{R}_1$ n'a pas changé après α applications de $\mathcal{R}_1$, β applications de $\mathcal{R}_2$ et γ applications of $\mathcal{R}_3$. L'équation $\mathcal{E}_2$ énonce le même pour la contrainte Φ_2 de la règle $\mathcal{R}_2$ et $\mathcal{E}_3$ pour la contrainte Φ_3 de la règle $\mathcal{R}_3$.

Soit Σ_1, Σ_2 et Σ_3 les systèmes:

$$\Sigma_1 \quad \left\{ \begin{array}{lll} b'_1\alpha + b'_2\beta + b'_3\gamma = 0 & & \text{équation } \mathcal{E}_2 \\ c'_1\alpha + c'_2\beta + c'_3\gamma = 0 & & \text{équation } \mathcal{E}_3 \end{array} \right.$$

$$\Sigma_2 \quad \left\{ \begin{array}{lll} a'_1\alpha + a'_2\beta + a'_3\gamma = 0 & & \text{équation } \mathcal{E}_1 \\ c'_1\alpha + c'_2\beta + c'_3\gamma = 0 & & \text{équation } \mathcal{E}_3 \end{array} \right.$$

$$\Sigma_3 \quad \left\{ \begin{array}{lll} a'_1\alpha + a'_2\beta + a'_3\gamma = 0 & & \text{équation } \mathcal{E}_1 \\ b'_1\alpha + b'_2\beta + b'_3\gamma = 0 & & \text{équation } \mathcal{E}_2 \end{array} \right.$$

Considérons les situations suivantes :

1. Il existe une solution non triviale $(\sigma_1, \sigma_2, \sigma_3)$ d'entiers naturels pour le système Σ_i $(1 \leq i \leq 3)$.
2. Aucun des systèmes Σ_1, Σ_2 et Σ_3 n'admet une solution non triviale d'entiers naturels, mais l'équation $\mathcal{E}_i$ admet une solution non triviale d'entiers naturels $(1 \leq i \leq 3)$.

On a :

Proposition 10.5 *La situation 1 se produit ssi Φ_j et Φ_k sont co-périodiques, pour $j \neq k$ et $j,k \neq i$, la situation 2 se produit ssi Φ_i est bi-périodique et Φ_j (pour $j \neq i$) est bi-périodiquement strictement croissante ou décroissante.*

De cette manière, on a une caractérisation algébrique pour que Π ait deux contraintes co-périodiques ou une contrainte bi-périodique avec chaque autre contrainte bi-périodiquement strictement croissante ou décroissante.

Dans la suite, on détaillera cette caractérisation algébrique.

10.2.1 Co-périodicité

Soit Π un programme avec contraintes Φ_1 et Φ_2 co-périodiques.

D'une part, par la proposition 10.5, il existe une solution non triviale $(\sigma_1, \sigma_2, \sigma_3)$ d' entiers naturels pour le système Σ_1. D'autre part, il existe une solution non triviale d'entiers naturels pour le système Σ_1 ssi la ligne d'intersection des plans définis par les équations $\mathcal{E}_1$ et $\mathcal{E}_2$ passent par le 1^{er} octant (où toutes les coordonnées sont positives) et par le 7^{eme} octant (où toutes les coordonnées sont négatives).

La pente de la ligne d'intersection de $\mathcal{E}_1$ et $\mathcal{E}_2$ est donnée par le déterminant:

$$\begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ a'_1 & a'_2 & a'_3 \\ b'_1 & b'_2 & b'_3 \end{vmatrix},$$

qui vaut $(a'_2b'_3 - a'_3b'_2)\vec{i} + (a'_3b'_1 - a'_1b'_3)\vec{j} + (a'_1b'_2 - a'_2b'_1)\vec{k}$.

Il en résulte que Σ_1 admet une solution non triviale d'entiers naturels dans les cas suivants:

1. $a'_2b'_3 - a'_3b'_2 \geq 0$, $a'_3b'_1 - a'_1b'_3 \geq 0$ et $a'_1b'_2 - a'_2b'_1 \geq 0$, mais pas $a'_2b'_3 - a'_3b'_2 = 0$, $a'_3b'_1 - a'_1b'_3 = 0$ et $a'_1b'_2 - a'_2b'_1 = 0$ simultanément.
2. $a'_2b'_3 - a'_3b'_2 \leq 0$, $a'_3b'_1 - a'_1b'_3 \leq 0$ et $a'_1b'_2 - a'_2b'_1 \leq 0$, mais pas $a'_2b'_3 - a'_3b'_2 = 0$, $a'_3b'_1 - a'_1b'_3 = 0$ et $a'_1b'_2 - a'_2b'_1 = 0$ simultanément.
3. $a'_2b'_3 - a'_3b'_2 = 0$, $a'_3b'_1 - a'_1b'_3 = 0$ et $a'_1b'_2 - a'_2b'_1 = 0$, mais ni $a'_i \geq 0$ pour tout i, ni $a'_i \leq 0$ pour tout i.

On notera la conjonction $a'_2b'_3 - a'_3b'_2 = 0 \ \wedge \ a'_3b'_1 - a'_1b'_3 = 0 \ \wedge \ a'_1b'_2 - a'_2b'_1 = 0$ par $\mathcal{C}_1$.

L'étude du comportement de la contrainte Φ_3 par rapport au co-motif de Φ_1 et Φ_2 conduit à l'examen des sous-cas suivants :

1. $a'_2b'_3 - a'_3b'_2 \geq 0$, $a'_3b'_1 - a'_1b'_3 \geq 0$, $a'_1b'_2 - a'_2b'_1 \geq 0$, mais pas $\mathcal{C}_1$:
   (a) Φ_3 **périodiquement croissante :** $c'_1(a'_2b'_3 - a'_3b'_2) + c'_2(a'_3b'_1 - a'_1b'_3) + c'_3(a'_1b'_2 - a'_2b'_1) \geq 0$.
   (b) Φ_3 **périodiquement décroissante :** $c'_1(a'_2b'_3 - a'_3b'_2) + c'_2(a'_3b'_1 - a'_1b'_3) + c'_3(a'_1b'_2 - a'_2b'_1) \leq 0$.
2. $a'_2b'_3 - a'_3b'_2 \leq 0$, $a'_3b'_1 - a'_1b'_3 \leq 0$ et $a'_1b'_2 - a'_2b'_1 \leq 0$, mais pas $\mathcal{C}_1$:
   (a) Φ_3 **périodiquement croissante :** $c'_1(a'_2b'_3 - a'_3b'_2) + c'_2(a'_3b'_1 - a'_1b'_3) + c'_3(a'_1b'_2 - a'_2b'_1) \leq 0$.
   (b) Φ_3 **périodiquement décroissante :** $c'_1(a'_2b'_3 - a'_3b'_2) + c'_2(a'_3b'_1 - a'_1b'_3) + c'_3(a'_1b'_2 - a'_2b'_1) \geq 0$.
3. $\mathcal{C}_1$, mais ni $\forall i,\ a'_i \geq 0$, ni $\forall i, a'_i \leq 0$:
   (a) Φ_3 **périodiquement croissante :** $a'_1 > 0,\ a'_2 \leq 0,\ a'_3 \leq 0,\ -c'_1a'_2 + a'_1c'_2 \geq 0$
   (b) Φ_3 **périodiquement décroissante :** $a'_1 > 0,\ a'_2 \leq 0,\ a'_3 \leq 0,\ -c'_1a'_2 + a'_1c'_2 \leq 0$
   (c) Φ_3 **périodiquement croissante :** $a'_1 < 0,\ a'_2 \geq 0,\ a'_3 \geq 0,\ c'_1a'_2 - a'_1c'_2 \geq 0$
   (d) Φ_3 **périodiquement décroissante :** $a'_1 < 0,\ a'_2 \geq 0,\ a'_3 \geq 0,\ c'_1a'_2 - a'_1c'_2 \leq 0$
   (e) Φ_3 **périodiquement croissante :** $a'_1 \leq 0,\ a'_2 > 0,\ a'_3 \leq 0,\ c'_1a'_2 - a'_1c'_2 \geq 0$
   (f) Φ_3 **périodiquement décroissante :** $a'_1 \leq 0,\ a'_2 > 0,\ a'_3 \leq 0,\ c'_1a'_2 - a'_1c'_2 \leq 0$
   (g) Φ_3 **périodiquement croissante :** $a'_1 \geq 0,\ a'_2 < 0,\ a'_3 \geq 0,\ -c'_1a'_2 + a'_1c'_2 \geq 0$
   (h) Φ_3 **périodiquement décroissante :** $a'_1 \geq 0,\ a'_2 < 0,\ a'_3 \geq 0,\ -c'_1a'_2 + a'_1c'_2 \leq 0$
   (i) Φ_3 **périodiquement croissante :** $a'_1 \leq 0,\ a'_2 \leq 0,\ a'_3 > 0,\ c'_1a'_3 - a'_1c'_3 \geq 0$
   (j) Φ_3 **périodiquement décroissante :** $a'_1 \leq 0,\ a'_2 \leq 0,\ a'_3 > 0,\ c'_1a'_3 - a'_1c'_3 \leq 0$
   (k) Φ_3 **périodiquement croissante :** $a'_1 \geq 0,\ a'_2 \geq 0,\ a'_3 < 0,\ -c'_1a'_3 + a'_1c'_3 \geq 0$
   (l) Φ_3 **périodiquement décroissante :** $a'_1 \geq 0,\ a'_2 \geq 0,\ a'_3 < 0,\ -c'_1a'_3 + a'_1c'_3 \geq 0$

Pour les sous-cas (1) et (2), le vecteur $(|a'_2b'_3 - a'_3b'_2|, |a'_3b'_1 - a'_1b'_3|, |a'_1b'_2 - a'_2b'_1|)$ est un multiple scalaire du co-motif pour Φ_1 et Φ_2. Le co-motif (unique) est obtenu en divisant ce vecteur par le $\mathrm{pgcd}\{|a'_2b'_3 - a'_3b'_2|, |a'_3b'_1 - a'_1b'_3|, |a'_1b'_2 - a'_2b'_1|\}$.

Pour les sous-cas (3a), ..., (3h), le vecteur $(|a'_2|, |a'_1|, 0)$ est un multiple scalaire d'un co-motif pour Φ_1 et Φ_2. Le co-motif est obtenu en divisant ce vecteur par le plus grand commun diviseur de ses composants.

Pour les sous-cas (3i), (3j), (3k) et (3l), le vecteur $(|a'_1|, 0, |a'_3|)$ est un multiple scalaire d'un co-motif. Le co-motif est obtenu comme dans le sous cas précédent.

Une caractérisation algébrique analogue peut être obtenue pour la situation où Φ_1 et Φ_3 sont co-périodiques (étude du système Σ_2) et pour la situation où Φ_2 et Φ_3 (étude du système Σ_3) sont co-périodiques.

10.2.2 Bi-périodicité

Considérons maintenant la situation où Φ_1 est bi-périodique et Φ_2 et Φ_3 sont bi-périodiquement strictement monotones. Cela implique que l'équation $\mathcal{E}_1$ a une solution non-triviale d'entiers naturels. Les cas où Φ_2 est bi-périodique (l'équation $\mathcal{E}_2$ admet une solution positive) et où Φ_3 est bi-périodique (l'équation $\mathcal{E}_2$ admet une solution positive) sont traités de façon analogue.

$a'_1 > 0,\ a'_2 \leq 0,\ a'_3 \leq 0,\ \frac{a'_3b'_1 - a'_1b'_3}{a'_1b'_2 - a'_2b'_1} < 0,\ \frac{a'_3c'_1 - a'_1c'_3}{a'_1c'_2 - a'_2c'_1} < 0$	bi-périodicité de Φ_1 avec $\tau_2 = 0$ et $\nu_3 = 0$
$a'_1 < 0,\ a'_2 \geq 0,\ a'_3 \geq 0,\ \frac{a'_3b'_1 - a'_1b'_3}{a'_1b'_2 - a'_2b'_1} < 0,\ \frac{a'_3c'_1 - a'_1c'_3}{a'_1c'_2 - a'_2c'_1} < 0$	bi-périodicité de Φ_1 avec $\tau_2 = 0$ et $\nu_3 = 0$
$a'_1 \leq 0,\ a'_2 > 0,\ a'_3 \leq 0,\ \frac{a'_2b'_3 - a'_3b'_2}{a'_1b'_2 - a'_2b'_1} < 0,\ \frac{a'_2c'_3 - a'_3c'_2}{a'_1c'_2 - a'_2c'_1} < 0$	bi-périodicité de Φ_1 avec $\tau_1 = 0$ et $\nu_3 = 0$
$a'_1 \geq 0,\ a'_2 < 0,\ a'_3 \geq 0,\ \frac{a'_2b'_3 - a'_3b'_2}{a'_1b'_2 - a'_2b'_1} < 0,\ \frac{a'_2c'_3 - a'_3c'_2}{a'_1c'_2 - a'_2c'_1} < 0$	bi-périodicité de Φ_1 avec $\tau_1 = 0$ et $\nu_3 = 0$
$a'_1 \leq 0,\ a'_2 \leq 0,\ a'_3 > 0,\ \frac{a'_2b'_3 - a'_3b'_2}{a'_3b'_1 - a'_1b'_3} < 0,\ \frac{a'_2c'_3 - a'_3c'_2}{a'_3c'_1 - a'_1c'_3} < 0$	bi-périodicité de Φ_1 avec $\tau_1 = 0$ et $\nu_2 = 0$
$a'_1 \geq 0,\ a'_2 \geq 0,\ a'_3 < 0,\ \frac{a'_2b'_3 - a'_3b'_2}{a'_3b'_1 - a'_1b'_3} < 0,\ \frac{a'_2c'_3 - a'_3c'_2}{a'_3c'_1 - a'_1c'_3} < 0$	bi-périodicité de Φ_1 avec $\tau_1 = 0$ et $\nu_2 = 0$

TAB. 10.1 - *Bi-périodicité de* Φ_1

On analysera maintenant le comportement de Φ_2 et Φ_3 par rapport à ce bi-motif.

Si Φ_1 est bi-périodique avec motifs $\vec{\tau}$ et $\vec{\nu}$, tel que $\tau_2 = 0$ et $\nu_3 = 0$ (ceci correspond aux programmes dont les coefficients satisfont les conditions trouvées dans les deux premières

lignes du tableau 10.1), on a :

$\frac{a'_1 b'_3 - a'_3 b'_1}{a'_1} > 0$	Φ_2 strictement périodiquement croissante
$\frac{a'_1 b'_3 - a'_3 b'_1}{a'_1} < 0$	Φ_2 strictement périodiquement décroissante
$\frac{a'_1 c'_3 - a'_3 c'_1}{a'_1} > 0$	Φ_3 strictement périodiquement croissante
$\frac{a'_1 c'_3 - a'_3 c'_1}{a'_1} < 0$	Φ_3 strictement périodiquement décroissante

Si Φ_1 est bi-périodique avec motifs $\vec{\tau}$ et $\vec{\nu}$, tel que $\tau_1 = 0$ et $\nu_3 = 0$ (correspondant aux troisième et quatrième lignes du tableau 10.1), on a :

$\frac{a'_2 b'_3 - a'_3 b'_2}{a'_2} > 0$	Φ_2 strictement périodiquement croissante
$\frac{a'_2 b'_3 - a'_3 b'_2}{a'_2} < 0$	Φ_2 strictement périodiquement décroissante
$\frac{a'_2 c'_3 - a'_3 c'_2}{a'_2} > 0$	Φ_3 strictement périodiquement croissante
$\frac{a'_2 c'_3 - a'_3 c'_2}{a'_2} < 0$	Φ_3 strictement périodiquement croissante

Si Φ_1 est bi-périodique avec motifs $\vec{\tau}$ et $\vec{\nu}$, tel que $\tau_1 = 0$ et $\nu_2 = 0$ (correspondant aux

deux dernières lignes du tableau 10.1), on a :

$\frac{a'_2 b'_3 - a'_3 b'_2}{a'_3} < 0$	Φ_2 strictement périodiquement croissante
$\frac{a'_2 b'_3 - a'_3 b'_2}{a'_3} > 0$	Φ_2 strictement périodiquement décroissante
$\frac{a'_2 c'_3 - a'_3 c'_2}{a'_3} < 0$	Φ_3 strictement périodiquement croissante
$\frac{a'_2 c'_3 - a'_3 c'_2}{a'_3} > 0$	Φ_3 strictement périodiquement croissante

Chapitre 11

Trois Règles Récursives - Co-périodicité

Dans ce chapitre, on montrera comment effectuer l'étape de $\bigvee$-simplifications pour les programmes avec deux contraintes co-périodiques. Le raisonnement utilisé ici est analogue à celui du cas difficile de deux règles récursives (expliqué au chapitre 7).

Le version pour trois règles récursives du théorème 7.17 est :

Théorème 11.1 (Programmes Co-périodiques) *Pour un programme avec contraintes Φ_i et Φ_j co-périodiques, la forme des chemins en $\mathcal{S}'(\lambda, \mu, \nu)$ est représentée à la figure 11.1.*

Noter que pour le cas de deux règles récursives, nous avions une séquence de pas en x ou en y avant et après la composition des motifs. Pour trois règles récursives, nous avons une séquence de pas contenue dans l'un des plans : 0_{XY}, ou 0_{XY}, ou 0_{YZ}.

Comme dans le cas de deux règles récursives, la propriété de monotonicité de Φ_k $(k \neq i, j)$ sera utilisée pour effectuer l'étape de $\bigwedge$-simplification.

Exemple 11.2 (Suite exemple 10.2) Pour le programme π_9, la forme des chemins en $\mathcal{S}'(\lambda, \mu, \nu)$ est représentée à la figure 11.2.

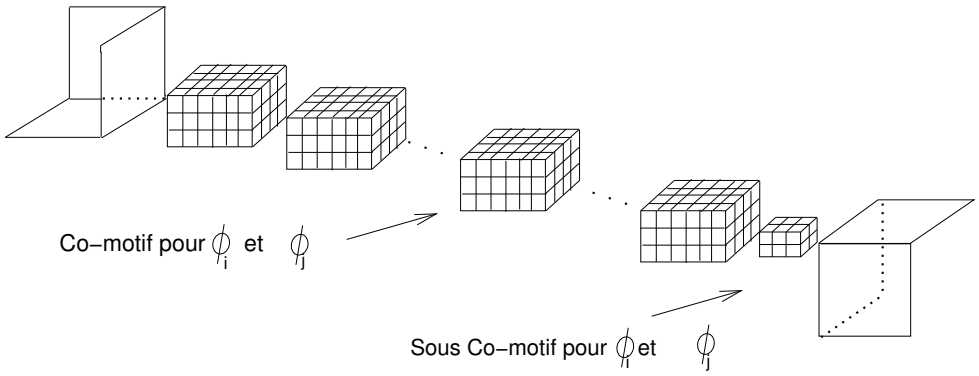


FIG. *11.1* - Forme des chemins en $\mathcal{S}'(\lambda, \mu, \nu)$

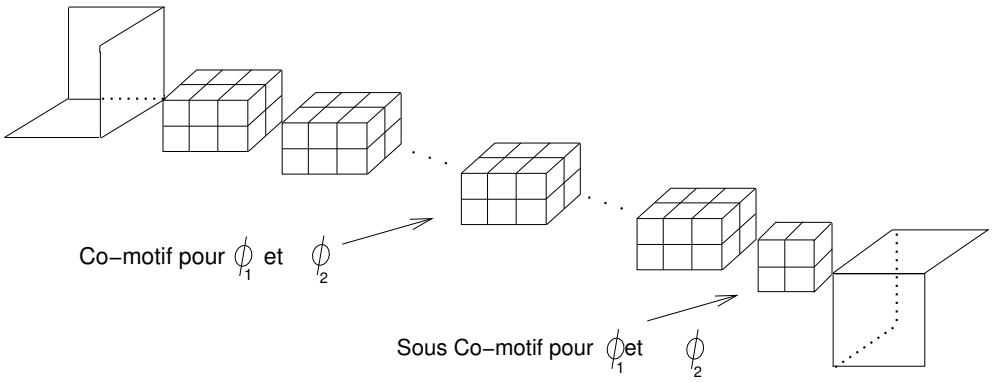


FIG. 11.2 -

Chapitre 12

Trois Règles Récursives - Cas Difficiles 2

Dans ce chapitre, on montrera comment effectuer l'étape de $\bigvee$-simplifications pour les programmes avec une contrainte bi-périodique et les deux autres bi-périodiquement strictement croissante ou décroissante.

On se limitera ici à la classe :

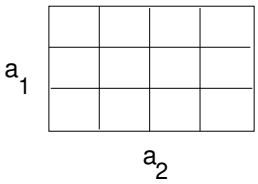
$$\begin{pmatrix} + & + & - \\ - & + & + \\ - & + & + \end{pmatrix}$$

Notons d'abord que cette classe ne tombe dans aucun des cinq cas simples. Un programme avec une matrice caractéristique comme cela est dans la super-classe 2 (voir chapitre 4).

De la classification algébrique de Π (voir chapitre 10), la contrainte Φ_1 est bi-périodique, avec motifs $(\frac{-a'_2}{\text{pgcd}\{-a'_2, a'_1\}}, \frac{a_1}{\text{pgcd}\{-a'_2, a'_1\}}, 0)$ (dans le plan $0xy$) et $(\frac{-a'_3}{\text{pgcd}\{-a'_3, a'_1\}}, 0, \frac{a_1}{\text{pgcd}\{-a'_3, a'_1\}})$ (dans le plan $0xz$).

Dans un souci de simplicité, on supposera que $\text{pgcd}\{-a'_2, a'_1\} = 1$ et que $\text{pgcd}\{-a'_3, a'_1\} = 1$. La bi-périodicité de Φ_1 s'exprime par :

Proposition 12.1 (Bi-périodicité de Φ_1)
$\forall\alpha,\forall\beta,\forall\gamma \quad \Phi_1(\alpha,\beta,\gamma) = \Phi_1(\alpha - a'_2, \beta + a'_1, \gamma) \; = \Phi_1(\alpha - a'_3, \beta, \gamma + a'_1)$.



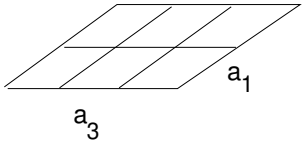


FIG. 12.1 - *Motif dans le plan* $0xy$ *Motif dans le plan* $0xz$

Soit $\mathcal{P}^1_{\text{motif}}$ le chemin-motif pour Φ_1 situé dans le plan $0xy$ et $\mathcal{P}^2_{\text{motif}}$ le chemin motif pour ϕ_1 situé dans le plan $0xz$. Ils sont obtenus en applicant l'algorithme donné à la fin de la partie 7.2.

Comme le chemin-motif $\mathcal{P}^1_{\text{motif}}$ (resp. $\mathcal{P}^2_{\text{motif}}$) est un certain chemin du motif $(-a'_2, a'_1, 0)$ (resp. $(-a'_3, 0, a'_1)$), il sera représenté dans les figures de ce chapitre par ce motif. De même les préfixes et les suffixes des chemins-motifs seront représentés par des sous-motifs.

Du fait que $b'_1 \geq 0$, $b'_2 \geq 0$ et $b'_3 \geq 0$, il s'ensuit que la contrainte Φ_2 est croissante dans les plans $0xy$ et $0xz$. Donc la contrainte Φ_2 est bi-périodiquement strictement croissante.

La contrainte Φ_3 est ou bien bi-périodiquement strictement croissante ou bien bi-périodiquement strictement décroissante pour le motif de Φ_1.

Nous avons ainsi deux situations à analyser :

- Φ_2 et Φ_3 sont bi-périodiquement strictement croissantes pour le bi-motif de Φ_1.
- Φ_2 est bi-périodiquement strictement croissante pour le bi-motif de Φ_1 et Φ_3 est bi-périodiquement strictement décroissante.

12.1 Φ_3 bi-périodiquement strictement croissante

Dans cette partie, on analysera le cas où Φ_2 et Φ_3 sont bi-périodiquement strictement croissantes pour le bi-motif de Φ_1.

Pour que la contrainte Φ_3 soit strictement croissante pour le bi-motif de Φ_1 (voir chapitre 10), il faut que a'_1, a'_2, a'_3 , c'_1, c'_2, c'_3 satisfassent à :

$$a'_2 \times c'_1 \ < \ a'_1 \times c'_2 \qquad\qquad a'_3 \times c'_1 \ < \ a'_1 \times c'_3$$

La relation $a'_2 \times c'_1 \ < \ a'_1 \times c'_2$ établit la croissance de Φ_3 par rapport au motif $(-a'_2, a'_1, 0)$, c'est-à-dire : $\forall \alpha, \beta, \gamma, \ \Phi_3(\alpha, \beta, \gamma) < \Phi_3(\alpha - a'_2, \beta + a'_1, \gamma)$.

La relation $a'_3 \times c'_1 \ < \ a'_1 \times c'_3$ établit la croissance de Φ_3 par rapport au motif $(-a'_3, 0, a'_1)$, c'est-à-dire : $\forall \alpha, \beta, \gamma \ \Phi_3(\alpha, \beta, \gamma) < \Phi_3(\alpha - a'_3, \beta, \gamma + a'_1)$.

On prouvera que la contrainte globale de tout chemin $\mathcal{P}$ en $\mathcal{S}(\lambda, \mu, \nu)$ est subsumée par la contrainte globale d'un chemin $\mathcal{P}'$ contenu dans un nombre fixe de plans.

Soit M, N et Q des points du chemin $\mathcal{P}$ (voir figure 12.2), tels que:

- En M, on trouve le premier pas en z de $\mathcal{P}$ qui soit précédé d'un pas en y de $\mathcal{P}$. Le dernier pas en y avant M apparaît au point Q.

- En N, on trouve le premier pas en x de $\mathcal{P}$ qui soit situé après M.

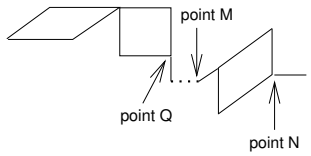


FIG. 12.2 -

Soit α, β et γ, respectivement, le nombre de pas en x, pas en y et pas en z de $\mathcal{P}$ situés après le point M.

Pour la construction de $\mathcal{P}'$ après le point M, on distinguera le cas où (α div $\mid a'_2 \mid \geq$ β div $\mid a'_1 \mid$) du cas où (α div $\mid a'_2 \mid \ < \ \beta$ div $\mid a'_1 \mid$).

1. Considérons d'abord que (α div $\mid a'_2 \mid \geq \ \beta$ div $\mid a'_1 \mid$). Cela correspond à un manque relatif de pas en y par rapport au nombre de pas en x.

Construction du chemin $\mathcal{P}'$:

Le chemin $\mathcal{P}'$ coïncide avec $\mathcal{P}$ jusqu'au point M. Après le point M, le chemin $\mathcal{P}'$ est obtenu de la façon suivante :

- En partant de M, on compose le chemin-motif $\mathcal{P}^1_{\text{motif}}$ (dans le plan $0xy$), comme illustré à la figure 12.3, jusqu'à utiliser tous les β pas en y, situés après le point M. L'intersection avec le plan $0xz$ qui passe par (λ, μ, ν) détermine le point R. A la fin du plan $0xy$, nous trouvons un préfixe du chemin motif $\mathcal{P}^1_{\text{motif}}$.

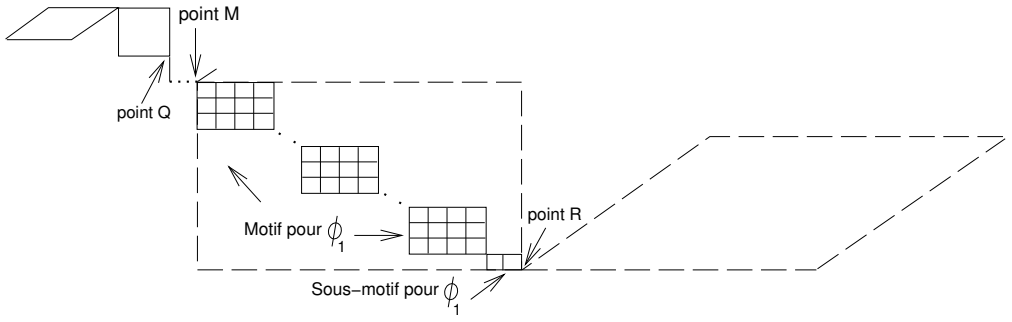


FIG. 12.3 - *Composition de chemin-motif $\mathcal{P}^1_{motif}$*

- En partant de R, on compose le chemin-motif $\mathcal{P}^2_{\text{motif}}$ (dans le plan $0xz$), comme illustré à la figure 12.4. Le point S est déterminé quand on rencontre l'axe $0x$ ou l'axe $0z$ qui passe par le point (λ, μ, ν). A la fin du plan $0xz$, nous trouvons un préfixe du chemin motif $\mathcal{P}^2_{\text{motif}}$.
- On relie le point S et le point (λ, μ, ν) par une séquence de pas en x ou une séquence de pas en z.

On obtient ainsi le chemin $\mathcal{P}'$.

Preuve de $\Gamma_{\mathcal{P}} \Rightarrow \Gamma_{\mathcal{P}'}$ ou du fait que $\mathcal{P}$ est déjà contenu dans un nombre fixe de plans :

Nous montrons ensuite que **la contrainte globale de $\mathcal{P}$ est subsumée par la contrainte globale de $\mathcal{P}'$** ou que **$\mathcal{P}$ est déjà contenu dans un nombre fixe de plans**.

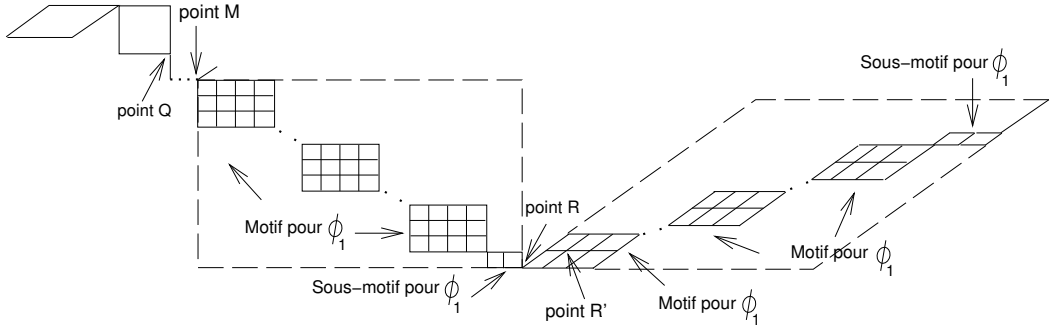


FIG. 12.4 - *Composition de chemin-motif* $\mathcal{P}^2_{motif}$

Soit $\mathcal{P}_1$ (resp. $\mathcal{P}'_1$) la partie de chemin $\mathcal{P}$ (resp. $\mathcal{P}'$) située après le point M. Comme $\mathcal{P}$ et $\mathcal{P}'$ coïncident jusqu'au point M et comme $\Phi_2(Q)$ et $\Phi_3(M)$ sont des contraintes de $\mathcal{P}$ (puisque dans $\mathcal{P}$, il y a un segment vertical qui part de Q et un segment transversal qui part de M), pour prouver que la contrainte globale de $\mathcal{P}$ est subsumée par la contrainte globale de $\mathcal{P}'$, il suffit de démontrer que :

(a) $\Gamma^1_{\mathcal{P}_1} \Rightarrow \Gamma^1_{\mathcal{P}'_1}$.

(b) $\Phi_2(Q) \Rightarrow \Gamma^2_{\mathcal{P}'_1}$.

(c) $\Phi_3(M) \Rightarrow \Gamma^3_{\mathcal{P}'_1}$

- **Preuve de** $\Gamma^1_{\mathcal{P}_1} \Rightarrow \Gamma^1_{\mathcal{P}'_1}$ **:** Entre les points M et S, par la propriété de base de chemin-motif (proposition 7.12), la conjonction des contraintes Φ équivaut à $\Phi_1(M)$.

  Comme $\forall \alpha, \beta, \gamma \ \Phi_1(\alpha, \beta, \gamma) \Rightarrow \ \Phi_1(\alpha + 1, \beta, \gamma)$ $(a'_1 \geq 0)$, après le point S, les contraintes Φ_1 sont plus faibles que $\Phi_1(S)$.

  Par conséquent, $\Gamma^1_{\mathcal{P}'_1}$ équivaut à $\Phi_1(M)$.

  Du fait que $\forall \alpha, \beta, \gamma \ \Phi_1(\alpha, \beta+1, \gamma) \Rightarrow \ \Phi_1(\alpha, \beta, \gamma)$ $(a'_2 \leq 0)$ et $\forall \alpha, \beta, \gamma \ \Phi_1(\alpha, \beta, \gamma+1) \Rightarrow \ \Phi_1(\alpha, \beta, \gamma)$ $(a'_3 \leq 0)$, il s'ensuit que $\Phi_1(N) \Rightarrow \Phi_1(M)$.

  Comme dans $\mathcal{P}$, il y a un segment horizontal qui part de N, $\Phi_1(N)$ est une contrainte de $\mathcal{P}$.

  Donc $\Gamma^1_{\mathcal{P}_1} \Rightarrow \Gamma^1_{\mathcal{P}'_1}$.

- **Preuve de** $\Phi_2(Q) \Rightarrow \Gamma^2_{\mathcal{P}'_1}$ **:** Comme la contrainte Φ_2 est croissante dans les plan $0xy$ et $0xz$, la contrainte Φ_2 la plus forte de $\mathcal{P}'$ après le point M est plus faible que $\Phi_2(Q)$.

– **Preuve de $\Phi_3(M) \Rightarrow \Gamma^3_{\mathcal{P}'_1}$:** Comme le plan $0xy$ ne contient pas de pas en z et comme Φ_3 est périodiquement strictement croissante dans le $0xz$, la contrainte Φ_3 la plus forte après M est située dans le premier motif de $0xz$. Soit R' le point où on trouve cette contrainte. On a:

$\Phi_3(R') = \Phi_3(R) - \alpha' \mid c'_1 \mid + \gamma' \mid c_3 \mid$, avec $\alpha' \leq \mid a'_3 \mid$ et $\gamma' \leq \mid a'_1 \mid$.

Comme $\alpha' \leq \mid a'_3 \mid$ et $\gamma' \leq \mid a'_1 \mid$, il s'ensuit que $-\alpha' \mid c'_1 \mid + \gamma' \mid c_3 \mid \geq - \mid a'_3 \times c'_1 \mid$.

Donc :

$\Phi_3(R') \geq \Phi_3(R) - \mid a'_3 \times c'_1 \mid$.

Soit n le nombre de chemins-motifs $\mathcal{P}^1_{\text{motif}}$ (motifs $(-a'_2, a'_1, 0)$) dans le plan $0xy$ et $(\alpha_1, \beta_1, 0)$ le dernier sous-motif dans ce plan. On a :

$\Phi_3(R) = \Phi_3(M) + n(a'_1 \times c'_2 - a'_2 \times c'_1) - \alpha_1 \mid c'_1 \mid + \beta_1 \mid c'_2 \mid$, avec $0 < \alpha_1 \leq \mid a'_2 \mid$, $0 < \beta_1 \leq \mid a'_1 \mid$.

Donc:

$\Phi_3(R') = \Phi_3(M) + n(a'_1 \times c'_2 - a'_2 \times c'_1) - \alpha_1 \mid c'_1 \mid + \beta_1 \mid c'_2 \mid - \mid a'_3 \times c'_1 \mid$, avec $\alpha_1 \leq \mid a'_2 \mid$, $\beta_1 \leq \mid a'_1 \mid$.

Comme $a'_1 c'_2 - a'_2 c'_1 > 0$ (car Φ_3 est périodiquement strictement croissante pour le motif $(-a'_2, a'_1, 0)$) et comme $-\alpha_1 \mid c'_1 \mid + \beta_1 \mid c'_2 \mid \geq - \mid a'_2 c'_1 \mid + \mid c'_2 \mid$ (car $\alpha_1 \leq \mid a'_2 \mid$, $\beta_1 \leq \mid a'_1 \mid$), il s'ensuit que:

$\Phi_3(R') \geq \Phi_3(M) + n(a'_1 c'_2 - a'_2 c'_1) - \mid a'_2 \times c'_1 \mid + \mid c'_2 \mid - \mid a'_3 \times c'_1 \mid$.

Par conséquent :

$\exists n_0$, indépendant de x, y et z (on prend n_0 le premier entier tel que $n_0 \geq \dfrac{\mid -a'_2 c'_1 + c'_2 - a'_3 c'_1 \mid}{\mid a'_1 c'_2 - a'_2 c'_1 \mid}$), tel que pour $n \geq n_0$, $\Phi_3(R') \geq \Phi_3(M)$. Cela signifie que toute contrainte Φ_3 après le point M est plus faible que $\Phi_3(M)$, pour $n \geq n_0$.

Comme il y a un manque de pas en y par rapport au nombre de pas en x, la relation $n \geq n_0$ est toujours satisfaite sauf pour le cas où le nombre de pas en y après M est borné. Dans ce cas, $\mathcal{P}$ est déjà contenu dans un nombre fini (borné) de plans (voir figure 12.5).

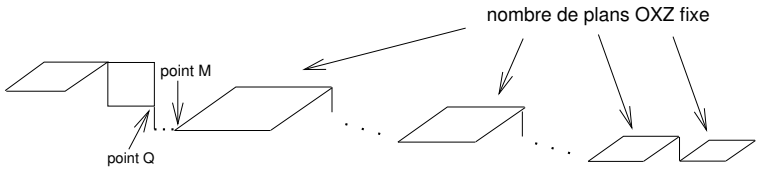


FIG. 12.5 - *Nombre de pas en y après M borné*

Nous avons donc prouvé que $\Gamma_{\mathcal{P}} \Rightarrow \Gamma_{\mathcal{P}'}$ ou que $\mathcal{P}$ est déjà contenu dans un nombre fixe de plans.

2. Considérons maintenant que (α div $\mid a'_2 \mid <$ β div $\mid a'_1 \mid$). Cela correspond à un manque relatif de pas en x par rapport au nombre de pas en y.

Construction du chemin $\mathcal{P}'$:

Le chemin $\mathcal{P}'$ coïncide avec $\mathcal{P}$ jusqu'au point M. Après le point M, le chemin $\mathcal{P}'$ est obtenu en projectant $\mathcal{P}$ sur le plan $0XY$ qui passe par M et ensuite en effectuant une séquence γ pas en z (voir figure 12.6).

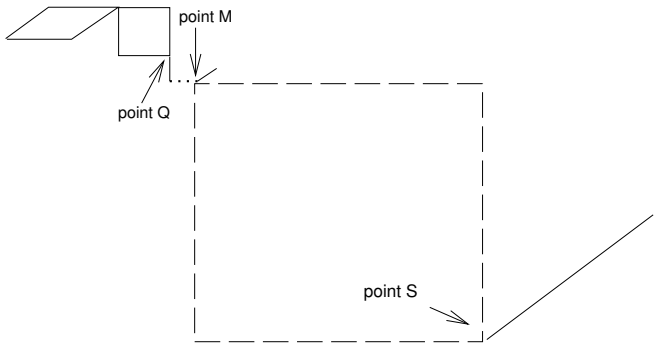


FIG. 12.6 -

Soit S la projection du dernier point de $\mathcal{P}$ sur le plan $0XY$ qui passe par M.

Preuve de $\Gamma_{\mathcal{P}} \Rightarrow \Gamma_{\mathcal{P}'}$:

Nous montrons ensuite que la contrainte globale de $\mathcal{P}$ est subsumée par la contrainte globale de $\mathcal{P}'$.

Soit $\mathcal{P}_1$ (resp. $\mathcal{P}'_1$) la partie de chemin $\mathcal{P}$ (resp. $\mathcal{P}'$) située après le point M. Comme $\mathcal{P}$ et $\mathcal{P}'$ coïncident jusqu'au point M et comme $\Phi_2(Q)$ et $\Phi_3(M)$ sont des contraintes de $\mathcal{P}$ (puisque dans $\mathcal{P}$, il y a un segment vertical qui part de Q et un segment transversal qui part de M), il suffit de démontrer que :

(a) $\Gamma^1_{\mathcal{P}_1} \Rightarrow \Gamma^1_{\mathcal{P}'_1}$.
(b) $\Phi_2(Q) \Rightarrow \Gamma^2_{\mathcal{P}'_1}$.
(c) $\Phi_3(M) \Rightarrow \Gamma^3_{\mathcal{P}'_1}$

- **Preuve de $\Gamma^1_{\mathcal{P}_1} \Rightarrow \Gamma^1_{\mathcal{P}'_1}$:** Comme $\forall \alpha, \beta, \gamma\ \Phi_1(\alpha, \beta, \gamma+1) \Rightarrow\ \Phi_1(\alpha, \beta, \gamma)\,(a'_3 \leq 0)$, en projetant chaque segment horizontal de $\mathcal{P}_1$ sur le plan $0XY$ qui passe par M, on obtient un segment horizontal de $\mathcal{P}'_1$ de contrainte plus faible. Il s'ensuit que les contraintes horizontales de $\mathcal{P}'_1$ sont plus faibles que celles de $\mathcal{P}_1$. Par conséquent, $\Gamma^1_{\mathcal{P}_1} \Rightarrow \Gamma^1_{\mathcal{P}'_1}$.

- **Preuve de $\Phi_2(Q) \Rightarrow \Gamma^2_{\mathcal{P}'_1}$:** Comme la contrainte Φ_2 est croissante dans le plan $0XY$, tous les contraintes verticales de $\mathcal{P}'$ sont plus faibles que $\Phi_2(Q)$. Par conséquent, $\Phi_2(Q) \Rightarrow \Gamma^2_{\mathcal{P}'_1}$.
- **Preuve de $\Phi_3(M) \Rightarrow \Gamma^3_{\mathcal{P}'_1}$:** Comme $\forall \alpha, \beta, \gamma\ \Phi_3(\alpha, \beta, \gamma) \Rightarrow \Phi_3(\alpha + 1, \beta, \gamma)$ $(c'_3 \geq 0)$, la contrainte tranversale après M la plus forte est $\Phi_3(S)$. Mais :
  $\Phi_3(S) = \Phi_3(M) - \alpha \mid c'_1 \mid + \beta \mid c'_2 \mid$
  Comme $\alpha \mid a'_1 \mid < \beta \mid a'_2 \mid$ (car α div $\mid a'_2 \mid <$ β div $\mid a'_1 \mid$) et $a'_1 c'_2 - a'_2 c'_1 > 0$ (car Φ_3 périodiquement strictement croissante pour le motif $(-a'_2, a'_1, 0)$), il s'ensuit que $\alpha \mid c'_1 \mid < \beta \mid c'_2 \mid$ et que $\Phi_3(S) \Rightarrow \Phi_3(M)$. Par conséquence, $\Phi_3(M) \Rightarrow \Gamma^3_{\mathcal{P}'_1}$.

En résumé, tout chemin $\mathcal{P}$ est subsumé par un chemin ayant une des formes représentées dans la figure 12.7.

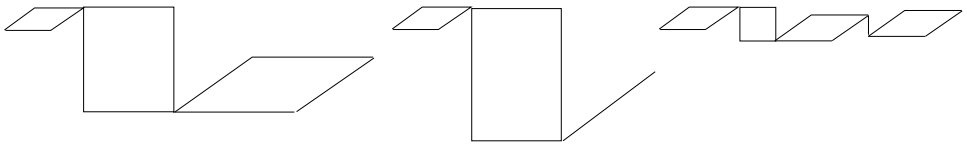


FIG. 12.7 - *Formes des chemins en $\mathcal{S}'(\lambda, \mu, \nu)$*

12.2 Φ_3 bi-périodiquement strictement décroissante

Dans cette partie, on analysera le cas où Φ_2 est bi-périodiquement strictement croissante pour le bi-motif de Φ_1 et Φ_3 est bi-périodiquement strictement décroissante.

Pour que la contrainte Φ_3 soit strictement décroissante pour le bi-motif de Φ_1 (voir chapitre 10), il faut que a'_1, a'_2, a'_3, c'_1, c'_2, c'_3 satisfassent à :

$$a'_2 \times c'_1 > a'_1 \times c'_2 \qquad\qquad a'_3 \times c'_1 > a'_1 \times c'_3$$

Comme dans la partie 12.1, on prouvera que la contrainte globale de tout chemin $\mathcal{P}$ en $\mathcal{S}(\lambda, \mu, \nu)$ est subsumée par la contrainte globale un chemin $\mathcal{P}'$ contenu dans un nombre fixe de plans.

Soit L, M, N et Q des points de $\mathcal{P}$ (voir figure 12.8), tel que:

- En M, on trouve le premier pas en z de $\mathcal{P}$, tel qu'il existe au moins un pas en y en $\mathcal{P}$ avant M. Le premier pas en y avant M part du point Q.

– En Q, on trouve le dernier pas en z de $\mathcal{P}$, tel qu'il existe au moins un pas en x en $\mathcal{P}$ après Q. Le premier pas en x après Q part du point L.

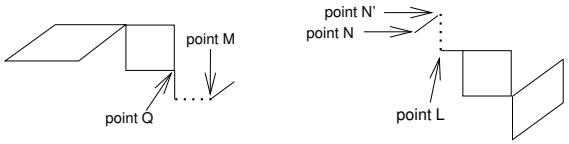


FIG. 12.8 -

Soit N' le point situé un pas en z après N ($N' = N + (0,0,1)$). Soit α, β et γ, respectivement, le nombre de pas en x, pas en y et pas en z de $\mathcal{P}$ situés entre les points M et N'.

Pour la construction de $\mathcal{P}'$ après le point M, on distinguera le cas où (α div $\mid a'_2 \mid \geq \beta$ div $\mid a'_1 \mid$) du cas où (α div $\mid a'_2 \mid < \beta$ div a'_1).

1. Considérons d'abord que (α div $\mid a'_2 \mid \geq \beta$ div $\mid a'_1 \mid$). Cela correspond à un manque relatif de pas en y par rapport au nombre de pas en x.

Construction du chemin $\mathcal{P}'$:

Le chemin $\mathcal{P}'$ coïncide avec $\mathcal{P}$ avant le point M et après le point N'. Entre les points M et N', le chemin $\mathcal{P}'$ est obtenu de la façon suivante :

– En partant de N' et en remontant vers le point M, on compose le chemin-motif $\mathcal{P}^1_{\text{motif}}$ (dans le plan $0xy$), comme illustré à la figure 12.9, jusqu'à utiliser tous les β pas en y situés entre les points M et N'. L'intersection avec le plan $0xz$ qui passe par M détermine le point R. Au début du plan $0xy$, nous trouvons un suffixe du chemin motif $\mathcal{P}^1_{\text{motif}}$.

– En partant de R et se dirigeant vers le point M, on compose le chemin-motif $\mathcal{P}^2_{\text{motif}}$ (dans le plan $0xz$), comme illustré à la figure 12.10. Le point S est déterminé quand on rencontre l'axe $0x$ ou l'axe $0z$ qui passent par M. Au début du plan $0xz$, nous trouvons un suffixe du chemin motif $\mathcal{P}^2_{\text{motif}}$.

– On relie le point S et le point M par une séquence de pas en x ou une séquence de pas en z.

On obtient ainsi le chemin $\mathcal{P}'$.

Preuve de $\Gamma_{\mathcal{P}} \Rightarrow \Gamma_{\mathcal{P}'}$ ou que $\mathcal{P}$ est déjà contenu dans un nombre fixe de plans :

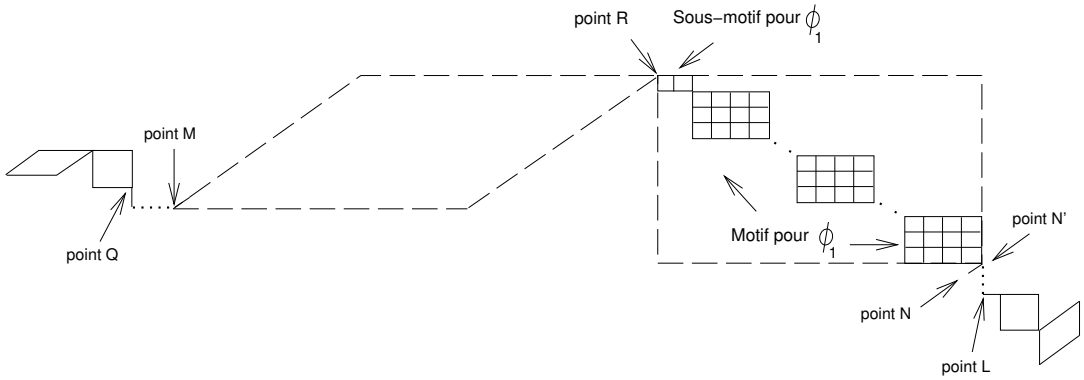


FIG. 12.9 - *Composition de chemin-motif* $\mathcal{P}^1_{motif}$

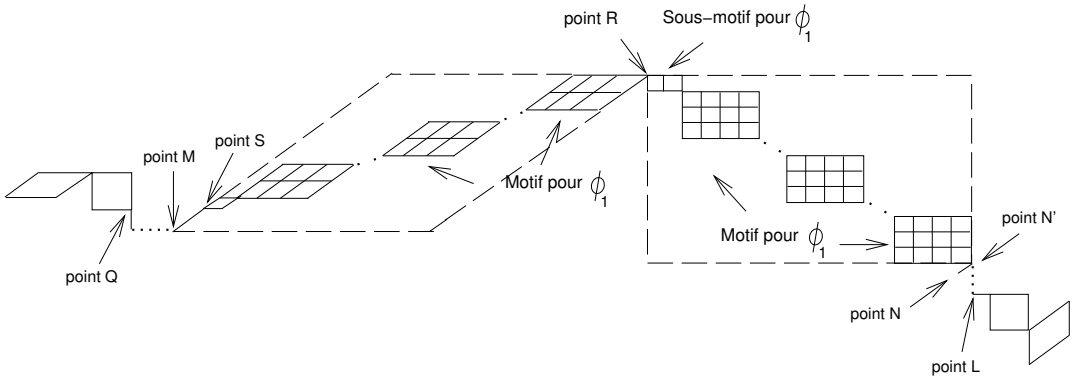


FIG. 12.10 - *Composition de chemin-motif* $\mathcal{P}^2_{motif}$

La preuve que **la contrainte globale de $\mathcal{P}$ est subsumée par la contrainte globale de $\mathcal{P}'$** ou que **$\mathcal{P}$ est déjà contenu dans un nombre fixe de plans**. est analogue à celle donnée à la section 12.1.

2. Considérons maintenant que (α div $\mid a'_2 \mid <$ β div $\mid a'_1 \mid$). Cela correspond à un manque relatif de pas en x par rapport au nombre de pas en y.

Construction du chemin $\mathcal{P}'$:

Le chemin $\mathcal{P}'$ coïncide avec $\mathcal{P}$ avant le point M et après le point N'. Entre les points M et N', le chemin $\mathcal{P}'$ est obtenu de la façon suivante :

– En partant de N' et en remontant vers le point M, on compose le chemin-motif

$\mathcal{P}^1_{\text{motif}}$ (dans le plan $0xy$), comme illustré à la figure 12.11, jusqu'à utiliser tous les α pas en x situés entre M et N'. L'intersection avec le plan $0yz$ qui passe par M détermine le point R. Au début du plan $0xy$, nous trouvons un suffixe du chemin motif $\mathcal{P}^1_{\text{motif}}$.

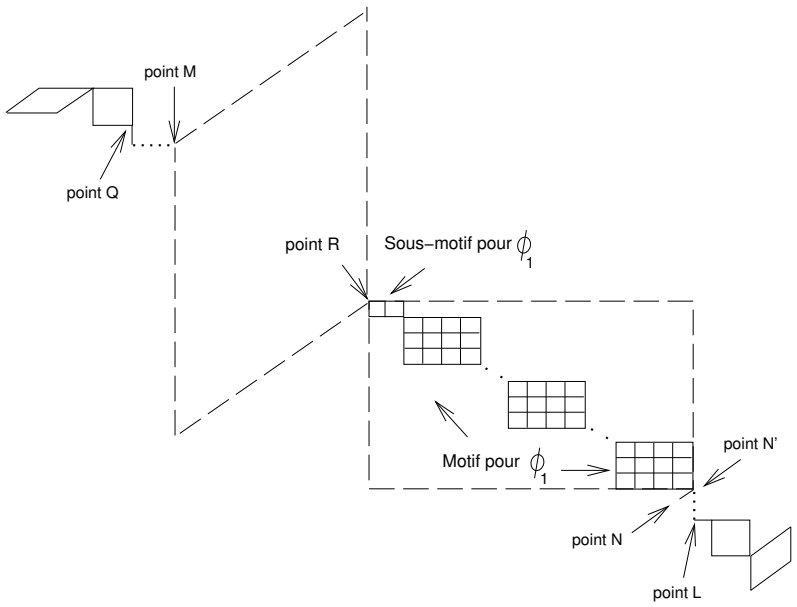


FIG. 12.11 - *Composition de chemin-motif* $\mathcal{P}^1_{motif}$

– En partant de R et en remontant vers le point M, on exécute une séquence de pas en y jusqu'à rencontrer le plan $0xz$ qui passe par M. Cela détermine le point S.

– On relie M et S par des pas en x (voir figure 12.12).

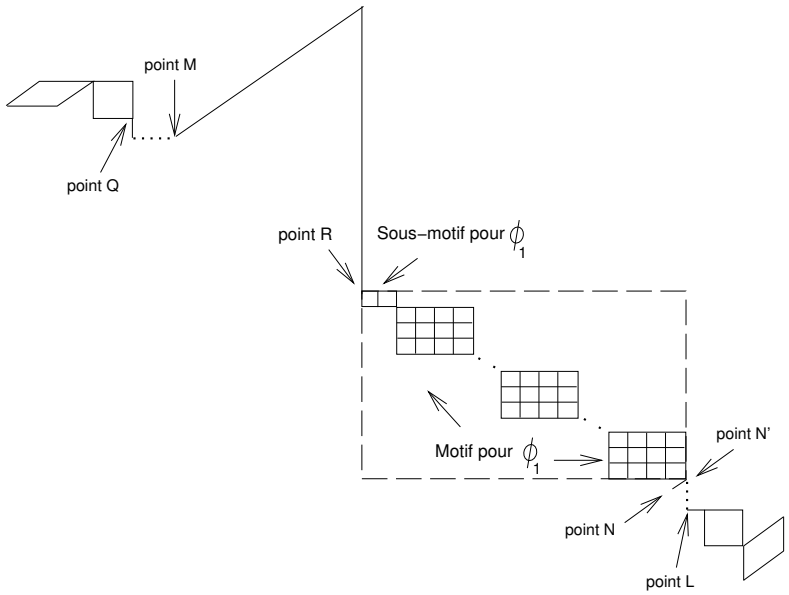


FIG. 12.12 - *Chemin $\mathcal{P}'$*

On obtient ainsi le chemin $\mathcal{P}'$.

Preuve de $\Gamma_{\mathcal{P}} \Rightarrow \Gamma_{\mathcal{P}'}$:

Nous montrons ensuite que la contrainte globale de $\mathcal{P}$ est subsumée par la contrainte globale de $\mathcal{P}'$

Soit $\mathcal{P}_1$ (resp. $\mathcal{P}'_1$) la partie de chemin $\mathcal{P}$ (resp. $\mathcal{P}'$) située entre les points M et N'. Comme $\mathcal{P}$ et $\mathcal{P}'$ coïncident jusqu'au point M et après le point N' et comme $\Phi_2(L)$, $\Phi_2(Q)$ et $\Phi_3(M)$ sont des contraintes de $\mathcal{P}$ (puisque dans $\mathcal{P}$, il y a un segment horizontal qui part de P, un segment vertical qui part de Q et un segment transversal qui part de M), il suffit de démontrer que :

(a) $\Phi_1(L) \Rightarrow \Gamma^1_{\mathcal{P}'_1}$.

(b) $\Phi_2(Q) \Rightarrow \Gamma^2_{\mathcal{P}'_1}$.

(c) $\Gamma^3_{\mathcal{P}'_1} = \Phi_3(M)$

- **Preuve de $\Gamma^1_{\mathcal{P}_1} \Rightarrow \Gamma^1_{\mathcal{P}'_1}$:** Entre les points S et N', par la propriété de base de chemin-motif (proposition 7.12), la conjonction des contraintes Φ équivaut à $\Phi_1(N')$.

  Comme $\forall \alpha, \beta, \gamma \ \Phi_1(\alpha, \beta + 1, \gamma) \Rightarrow \ \Phi_1(\alpha, \beta, \gamma)$ $(a'_2 \leq 0)$, entre les points S et N', les contraintes Φ_1 sont plus faibles que $\Phi_1(L)$.

  Par conséquent, $\Phi_1(L) \Rightarrow \Gamma^1_{\mathcal{P}'_1}$.

- **Preuve de $\Phi_2(Q) \Rightarrow \Gamma^2_{\mathcal{P}'_1}$.:** Comme la contrainte est croissante dans les plan $0xy$ et $0xz$, la contrainte Φ_2 la plus forte de $\mathcal{P}'$ entre les points M et N' est plus faible que $\Phi_2(Q)$. Par conséquent, $\Phi_2(Q) \Rightarrow \Gamma^2_{\mathcal{P}'_1}$.

- **Preuve de $\Gamma^3_{\mathcal{P}'_1} = \Phi_3(M)$:** Comme $\forall \alpha, \beta, \gamma \ \Phi_3(\alpha, \beta, \gamma) \Rightarrow \ \Phi_3(\alpha + 1, \beta, \gamma)$ $(c'_1 \geq 0)$, la contrainte tranversale la plus forte de $\mathcal{P}'$ entre les points M et N' est $\Phi_3(M)$. Par conséquent, $\Gamma^3_{\mathcal{P}'_1} = \Phi_3(M)$.

En résumé, tout chemin $\mathcal{P}$ est subsumé par un chemin ayant une des formes représentées dans la figure 12.13.

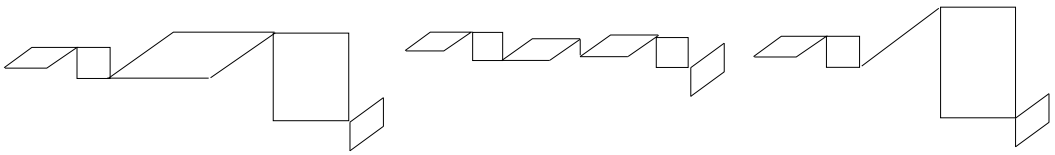


FIG. 12.13 - *Formes des chemins en $\mathcal{S}'(\lambda, \mu, \nu)$*

Dans les deux cas (Φ_3 bi-périodiquement croissante ou décroissante), tout chemin de $\mathcal{S}(\lambda, \mu, \nu)$ est subsumé par un chemin contenu dans un nombre fixe de plans.

Chapitre 13

Trois Règles Récursives - Cas Difficiles 2 bis

Dans ce chapitre, on montrera comment effectuer l'étape de $\bigvee$-simplifications pour les programmes appartenant à la super-classe 4, et qui ont une contrainte bi-périodique et les deux autres bi-périodiquement strictement croissante ou décroissante.

On considérera des programmes avec matrice caractéristique :

$$\begin{pmatrix} + & + & - \\ - & + & + \\ - & - & + \end{pmatrix}$$

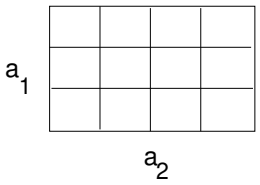
Rappelons (voir chapitre 4) que cette matrice a été choisie comme représentative de la super-classe 4.

La différence principale entre cette super-classe et celle traitée dans le chapitre précédent est qu'ici aucune contrainte n'est monotone. Dans le chapitre précédent, la contrainte de la règle $\mathcal{R}_2$ était monotone.

De la classification algébrique de Π (voir chapitre 10), la contrainte Φ_1 est bi-périodique, avec motifs $(\frac{-a'_2}{\text{pgcd}\{-a'_2, a'_1\}}, \frac{a'_1}{\text{pgcd}\{-a'_2, a'_1\}}, 0)$ (dans le plan $0xy$) et $(\frac{-a'_3}{\text{pgcd}\{-a'_3, a'_1\}}, 0, \frac{a'_1}{\text{pgcd}\{-a'_3, a'_1\}})$ (dans le plan $0xz$).

Dans une souci de simplicité, on supposera que pgcd$\{-a'_2, a'_1\} = 1$ et que pgcd$\{-a'_3, a'_1\} = 1$. La bi-périodicité de Φ_1 s'exprime par :

Proposition 13.1 (Bi-périodicité de Φ_1)
$\forall\alpha, \forall\beta, \forall\gamma \quad \Phi_1(\alpha, \beta, \gamma) = \Phi_1(\alpha - a'_2, \beta + a'_1, \gamma) \; = \Phi_1(\alpha - a'_3, \beta, \gamma + a'_1)$.



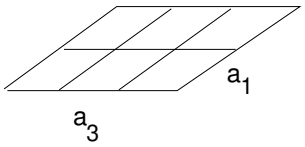


FIG. 13.1 - *Motif dans le plan* $0xy$ *Motif dans le plan* $0xz$

Soit $\mathcal{P}^1_{\text{motif}}$ le chemin-motif pour Φ_1 situé dans le plan $0xy$ et $\mathcal{P}^2_{\text{motif}}$ le chemin motif pour Φ_1 situé dans le plan $0xz$. Ils sont obtenus en appliquant l'algorithme donné à la fin de la partie 7.2.

Comme le chemin-motif $\mathcal{P}^1_{\text{motif}}$ (resp. $\mathcal{P}^2_{\text{motif}}$) est chemin qui appartient au motif $(-a'_2, a'_1, 0)$ (resp. $(-a'_3, 0, a'_1)$), il sera représenté dans les figures de ce chapitre par ce motif. De même les préfixes et les suffixes des chemins-motifs seront représentés par des sous-motifs.

Du fait que $b'_1 \geq 0$ et $b'_2 \geq 0$, il s'ensuit que la contrainte Φ_2 est croissante dans le plan $0xy$. Par conséquent, la contrainte Φ_2 est bi-périodiquement strictement croissante pour le bi-motif de Φ_1.

La contrainte Φ_3 est ou bien bi-périodiquement strictement croissante ou bien bi-périodiquement strictement décroissante pour le bi-motif de Φ_1.

Nous avons ainsi deux situations à analyser :

- Φ_2 et Φ_3 sont bi-périodiquement strictement croissantes pour le bi-motif de Φ_1.

- Φ_2 est bi-périodiquement strictement croissante pour le bi-motif de Φ_1 et Φ_3 est bi-périodiquement strictement décroissante pour le bi-motif de Φ_1.

13.1 Φ_3 bi-périodiquement strictement croissante

Dans cette partie, on analysera le cas où Φ_2 et Φ_3 sont bi-périodiquement strictement croissantes pour le bi-motif de Φ_1.

Pour que la contrainte Φ_2 soit strictement croissante pour le bi-motif de Φ_1 (voir chapitre 10), il faut que a'_1, a'_3, b'_1, b'_3 satisfassent à :

$$a'_3 \times b'_1 \ < \ a'_1 \times b'_3$$

Cette relation établit la croissance de Φ_2 par rapport au motif $(-a'_3, 0, a'_1)$, c'est-à-dire : $\forall \alpha, \beta, \gamma,\ \Phi_2(\alpha, \beta, \gamma) < \Phi_2(\alpha - a'_3, \beta, \gamma + a'_1)$.

Pour que la contrainte Φ_3 soit strictement croissante pour le bi-motif de Φ_1 (voir chapitre 10), il faut que a'_1, a'_2, a'_3, c'_1, c'_2, c'_3 satisfassent à :

$$a'_2 \times c'_1 \ < \ a'_1 \times c'_2 \qquad\qquad a'_3 \times c'_1 \ < \ a'_1 \times c'_3$$

La relation $a'_2 \times c'_1 \ < \ a'_1 \times c'_2$ établit la croissance de Φ_3 par rapport au motif $(-a'_2, a'_1, 0)$, c'est-à-dire : $\forall \alpha, \beta, \gamma,\ \Phi_3(\alpha, \beta, \gamma) < \Phi_3(\alpha - a'_2, \beta + a'_1, \gamma)$.

La relation $a'_3 \times c'_1 \ < \ a'_1 \times c'_3$ établit la croissance de Φ_3 par rapport au motif $(-a'_3, 0, a'_1)$, c'est-à-dire : $\forall \alpha, \beta, \gamma\ \Phi_3(\alpha, \beta, \gamma) < \Phi_3(\alpha - a'_3, \beta, \gamma + a'_1)$.

On prouvera que la contrainte globale de tout chemin $\mathcal{P}$ en $\mathcal{S}(\lambda, \mu, \nu)$ est subsumée par la contrainte globale d'un chemin $\mathcal{P}'$ contenu dans un nombre fixe de plans.

Soit M, N et Q des points du chemin $\mathcal{P}$ (voir figure 13.2), tels que:

- En M, on trouve le premier pas en z de $\mathcal{P}$ qui soit précédé d'un pas en y de $\mathcal{P}$. Le dernier pas en y avant M apparaît au point Q.
- En N, on trouve le premier pas en x de $\mathcal{P}$ qui soit situé après M.

Soit α, β et γ, respectivement, le nombre de pas en x, pas en y et pas en z de $\mathcal{P}$ situés après le point M.

Pour la construction de $\mathcal{P}'$ après le point M, on distinguera le cas où (α div $\mid a'_2 \mid \geq$ β div $\mid a'_1 \mid$) du cas où (α div $\mid a'_2 \mid <$ β div $\mid a'_1 \mid$).

1. Considérons d'abord que α div $\mid a'_2 \mid \ \geq \ \beta$ div $\mid a'_1 \mid$). Cela correspond à un manque relatif de pas en y par rapport au nombre de pas en x.

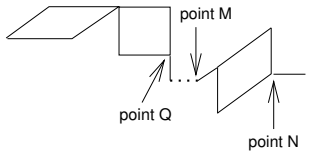


FIG. 13.2 -

Construction du chemin $\mathcal{P}'$:

Le chemin $\mathcal{P}'$ coïncide avec $\mathcal{P}$ jusqu'au point M. Après le point M, le chemin $\mathcal{P}'$ est obtenu de la façon suivante :

- En partant de M, on compose le chemin-motif $\mathcal{P}^1_{\text{motif}}$ (dans le plan $0xy$), comme illustré à la figure 13.3, jusqu'à utiliser tous les β pas en y, situés après le point M. L'intersection avec le plan $0xz$ qui passe par (λ, μ, ν) détermine le point R. A la fin du plan $0xy$, nous trouvons un préfixe du chemin motif $\mathcal{P}^1_{\text{motif}}$.

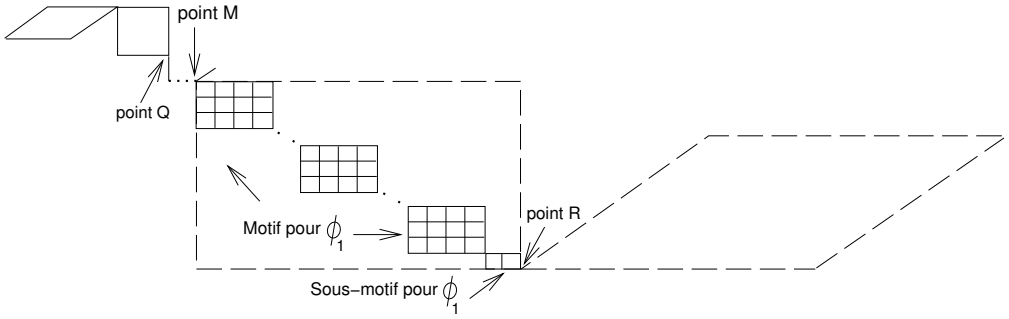


FIG. 13.3 - *Composition de chemin-motif* $\mathcal{P}^1_{motif}$

- En partant de R, on compose le chemin-motif $\mathcal{P}^2_{\text{motif}}$ (dans le plan $0xz$), comme illustré à la figure 13.4. Le point S est déterminé quand on rencontre l'axe $0x$ ou l'axe $0z$ qui passent par le point (λ, μ, ν). A la fin du plan $0xz$ et avant le point S, nous trouvons un préfixe du chemin motif $\mathcal{P}^2_{\text{motif}}$.
- On relie le point S et le point (λ, μ, ν) par une séquence de pas en x ou une séquence de pas en z.

On obtient ainsi le chemin $\mathcal{P}'$.

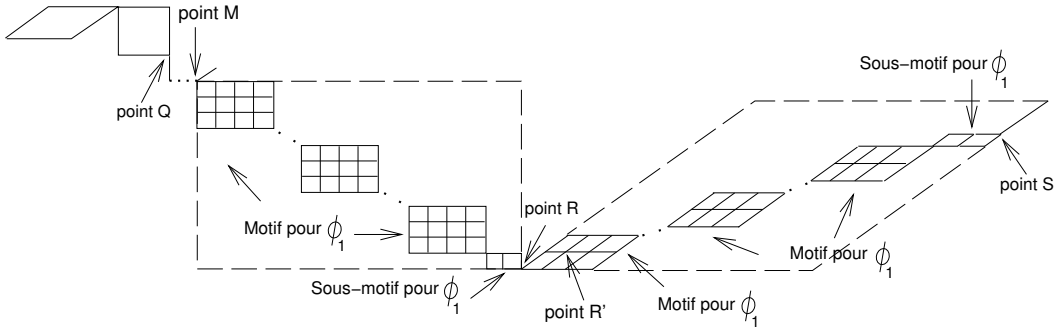


FIG. 13.4 - *Composition de chemin-motif* $\mathcal{P}^2_{motif}$

Preuve de $\Gamma_{\mathcal{P}} \Rightarrow \Gamma_{\mathcal{P}'}$ ou du fait que $\mathcal{P}$ est déjà contenu dans un nombre fixe de plans :

Nous montrons ensuite que **la contrainte globale de $\mathcal{P}$ est subsumée par la contrainte globale de $\mathcal{P}'$** ou que **$\mathcal{P}$ est déjà contenu dans un nombre fixe de plans**.

Soit $\mathcal{P}_1$ (resp. $\mathcal{P}'_1$) la partie de chemin $\mathcal{P}$ (resp. $\mathcal{P}'$) située après le point M. Comme $\mathcal{P}$ et $\mathcal{P}'$ coïncident jusqu'au point M et comme $\Phi_2(Q)$ et $\Phi_3(M)$ sont des contraintes de $\mathcal{P}$ (puisque dans $\mathcal{P}$, il y a un segment vertical qui part de Q et un segment transversal qui part de M), pour prouver que la contrainte globale de $\mathcal{P}$ est subsumée par la contrainte globale de $\mathcal{P}'$, il suffit de démontrer que :

(a) $\Gamma^1_{\mathcal{P}_1} \Rightarrow \Gamma^1_{\mathcal{P}'_1}$.

(b) $\Phi_2(Q) \Rightarrow \Gamma^2_{\mathcal{P}'_1}$.

(c) $\Phi_3(M) \Rightarrow \Gamma^3_{\mathcal{P}'_1}$

– **Preuve de $\Gamma^1_{\mathcal{P}_1} \Rightarrow \Gamma^1_{\mathcal{P}'_1}$:** Entre les points M et S, par la propriété de base de chemin-motif (proposition 7.12), la conjonction des contraintes Φ_1 équivaut à $\Phi_1(M)$.

Comme $\forall \alpha, \beta, \gamma \ \Phi_1(\alpha, \beta, \gamma) \Rightarrow \ \Phi_1(\alpha + 1, \beta, \gamma)$ $(a'_1 \geq 0)$, après le point S, les contraintes Φ_1 sont plus faibles que $\Phi_1(S)$.

Par conséquent, $\Gamma^1_{\mathcal{P}'_1}$ équivaut à $\Phi_1(M)$.

Du fait que $\forall \alpha, \beta, \gamma\ \Phi_1(\alpha, \beta+1, \gamma) \Rightarrow\ \Phi_1(\alpha, \beta, \gamma)\ (a'_2 \leq 0)$ et
$\forall \alpha, \beta, \gamma\ \Phi_1(\alpha, \beta, \gamma+1) \Rightarrow\ \Phi_1(\alpha, \beta, \gamma)\ (a'_3 \leq 0)$, il s'ensuit que $\Phi_1(N) \Rightarrow \Phi_1(M)$.
Comme dans le chemin $\mathcal{P}$ il y a un segment horizontal qui part de N, $\Phi_1(N)$ est une contrainte de $\mathcal{P}$.

Donc $\Gamma^1_{\mathcal{P}_1} \Rightarrow \Gamma^1_{\mathcal{P}'_1}$.

– **Preuve de $\Phi_2(Q) \Rightarrow \Gamma^2_{\mathcal{P}'_1}$:** Comme la contrainte Φ_2 est croissante dans le plan $0xy$ et, comme après le point R nous n'avons plus de pas en y, la contrainte Φ_2 la plus forte de $\mathcal{P}'$ après le point M est plus faible que $\Phi_2(Q)$.

– **Preuve de $\Phi_3(M) \Rightarrow \Gamma^3_{\mathcal{P}'_1}$:** Comme le plan $0xy$ ne contient pas de pas en z et comme Φ_3 est périodiquement strictement croissante dans le $0xz$, la contrainte Φ_3 la plus forte après M est située dans le premier motif de $0xz$. Soit R' le point où on trouve cette contrainte. On a:

$\Phi_3(R') = \Phi_3(R)\ - \alpha' \mid c'_1 \mid\ + \gamma' \mid c_3 \mid$, avec $\alpha' \leq \mid a'_3 \mid$ et $\gamma' \leq \mid a'_1 \mid$.

Comme $\alpha' \leq \mid a'_3 \mid$ et $\gamma' \leq \mid a'_1 \mid$, il s'ensuit que $-\alpha' \mid c'_1 \mid\ + \gamma' \mid c_3 \mid \geq - \mid a'_3 \times c'_1 \mid$.
Donc :

$\Phi_3(R') \geq \Phi_3(R) - \mid a'_3 \times c'_1 \mid$.

Soit n le nombre de chemins-motifs $\mathcal{P}^1_{\text{motif}}$ (motifs $(-a'_2, a'_1, 0)$) dans le plan $0xy$ et $(\alpha_1, \beta_1, 0)$ le dernier sous-motif dans ce plan. On a :

$\Phi_3(R) = \Phi_3(M) + n(a'_1 c'_2 - a'_2 c'_1) - \alpha_1 \mid c'_1 \mid\ + \beta_1 \mid c'_2 \mid$, avec $0 < \alpha_1 \leq \mid a'_2 \mid$, $0 < \beta_1 \leq \mid a'_1 \mid$.

Donc :

$\Phi_3(R') \geq \Phi_3(M) + n(a'_1 c'_2 - a'_2 c'_1) - \alpha_1 \mid c'_1 \mid + \beta_1 \mid c'_2 \mid\ - \mid a'_3 \times c'_1 \mid$, avec $\alpha_1 \leq \mid a'_2 \mid$, $\beta_1 \leq \mid a'_1 \mid$.

Comme $a'_1 c'_2 - a'_2 c'_1 > 0$ (car Φ_3 est périodiquement strictement croissante pour le motif $(-a'_2, a'_1, 0)$) et comme $-\alpha_1 \mid c'_1 \mid\ + \beta_1 \mid c'_2 \mid \geq\ - \mid a'_2 \times c'_1 \mid\ + \mid c'_2 \mid$ (car $\alpha_1 \leq \mid a'_2 \mid$, $\beta_1 \leq \mid a'_1 \mid$), il s'ensuit que:

$\Phi_3(R') \geq \Phi_3(M) + n(a'_1 c'_2 - a'_2 c'_1) - \mid a'_2 \times c'_1 \mid\ + \mid c'_2 \mid - \mid a'_3 \times c'_1 \mid$.

Par conséquent :

$\exists n_0$, indépendant de x, y et z (on prend n_0 le premier entier supérieur à $\dfrac{\mid -a'_2 c'_1 + c'_2 - a'_3 c'_1 \mid}{\mid a'_1 c'_2 - a'_2 c'_1 \mid}$), tel que pour $n \geq n_0$, $\Phi_3(R') \geq \Phi_3(M)$. Cela signifie que toute contrainte Φ_3 après le point M est plus faible que $\Phi_3(M)$, pour $n \geq n_0$.

Comme il y a un manque de pas en y par rapport au nombre de pas en x, la relation $n \geq n_0$ est toujours satisfaite. sauf pour le cas où le nombre de pas en y après M est borné. Dans ce cas, $\mathcal{P}$ est déjà contenu dans un nombre fixe (borné) de plans (voir figure 13.5).

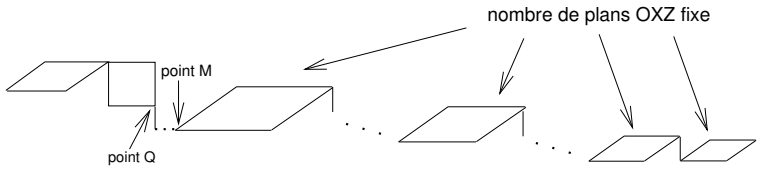


FIG. 13.5 - *Nombre de pas en y après M borné*

Nous avons donc prouvé que $\Gamma_{\mathcal{P}} \Rightarrow \Gamma_{\mathcal{P}'}$ ou que $\mathcal{P}$ est déjà contenu dans un nombre fixe de plans.

2. Considérons maintenant que (α div $\mid a'_2 \mid <\ \beta$ div $\mid a'_1 \mid$). Cela correspond à un manque relatif de pas en x par rapport au nombre de pas en y.

Construction du chemin $\mathcal{P}'$:

Le chemin $\mathcal{P}'$ coïncide avec $\mathcal{P}$ jusqu'au point M. Après le point M, le chemin $\mathcal{P}'$ est obtenu en projetant $\mathcal{P}$ sur le plan $0xy$ qui passe par M et ensuite en effectuant une séquence de γ pas en z (voir figure 13.6).

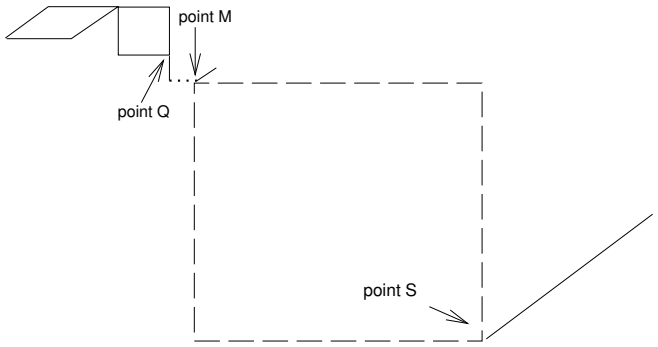


FIG. 13.6 -

Soit S la projection du dernier point de $\mathcal{P}$ sur le plan $0xy$ qui passe par M.

Preuve de $\Gamma_{\mathcal{P}} \Rightarrow \Gamma_{\mathcal{P}'}$:

Nous montrons ensuite que la contrainte globale de $\mathcal{P}$ est subsumée par la contrainte globale de $\mathcal{P}'$.

Soit $\mathcal{P}_1$ (resp. $\mathcal{P}'_1$) la partie de chemin $\mathcal{P}$ (resp. $\mathcal{P}'$) située après le point M. Comme $\mathcal{P}$ et $\mathcal{P}'$ coïncident jusqu'au point M et comme $\Phi_3(M)$ est une contrainte de $\mathcal{P}$ (puisque dans $\mathcal{P}$, il y a un segment transversal qui part de M), il suffit de démontrer que :

(a) $\Gamma^1_{\mathcal{P}_1} \Rightarrow \Gamma^1_{\mathcal{P}'_1}$.

(b) $\Gamma^2_{\mathcal{P}_1} \Rightarrow \Gamma^2_{\mathcal{P}'_1}$.

(c) $\Phi_3(M) \Rightarrow \Gamma^3_{\mathcal{P}'_1}$

- **Preuve de** $\Gamma^1_{\mathcal{P}_1} \Rightarrow \Gamma^1_{\mathcal{P}'_1}$ **:** Comme $\forall \alpha, \beta, \gamma\ \Phi_1(\alpha, \beta, \gamma+1) \Rightarrow\ \Phi_1(\alpha, \beta, \gamma)\,(a'_3 \leq 0)$, en projetant chaque segment horizontal de $\mathcal{P}_1$ sur le plan $0xy$ qui passe par M, on obtient un segment horizontal de $\mathcal{P}'_1$ de contrainte plus faible. Il s'ensuit que les contraintes horizontales de $\mathcal{P}'_1$ sont plus faibles que celles de $\mathcal{P}_1$. Par conséquent, $\Gamma^1_{\mathcal{P}_1} \Rightarrow \Gamma^1_{\mathcal{P}'_1}$.
- **Preuve de** $\Gamma^2_{\mathcal{P}_1} \Rightarrow \Gamma^2_{\mathcal{P}'_1}$ **:** Comme $\forall \alpha, \beta, \gamma\ \Phi_2(\alpha, \beta, \gamma+1) \Rightarrow\ \Phi_2(\alpha, \beta, \gamma)\,(b'_3 \leq 0)$, en projetant chaque segment vertical de $\mathcal{P}_1$ sur le plan $0xy$ qui passe par M, on obtient un segment vertical de $\mathcal{P}'_1$ de contrainte plus faible. Il s'ensuit que les contraintes verticales de $\mathcal{P}'_1$ sont plus faibles que celles de $\mathcal{P}_1$. Par conséquent, $\Gamma^2_{\mathcal{P}_1} \Rightarrow \Gamma^2_{\mathcal{P}'_1}$.
- **Preuve de** $\Phi_3(M) \Rightarrow \Gamma^3_{\mathcal{P}'_1}$ **:** Comme $\forall \alpha, \beta, \gamma\ \Phi_3(\alpha, \beta, \gamma) \Rightarrow\ \Phi_3(\alpha+1, \beta, \gamma)$ $(c'_3 \geq 0)$, la contrainte tranversale après M la plus forte est $\Phi_3(S)$. Mais :
  $\Phi_3(S) = \Phi_3(M) - \alpha \mid c'_1 \mid\ + \beta \mid c'_2 \mid$
  Comme $\alpha \mid a'_1 \mid < \beta \mid a'_2 \mid$ (car α div $\mid a'_2 \mid <\ \beta$ div $\mid a'_1 \mid$) et $a'_1 c'_2 - a'_2 c'_1 > 0$ (car Φ_3 périodiquement strictement croissante pour le motif $(-a'_2, a'_1, 0)$), il s'ensuit que $\alpha \mid c'_1 \mid < \beta \mid c'_2 \mid$ et que $\Phi_3(S)\ \Rightarrow \Phi_3(M)$. Par conséquent, $\Phi_3(M) \Rightarrow \Gamma^3_{\mathcal{P}'_1}$.

En résumé, tout chemin $\mathcal{P}$ est subsumé par un chemin ayant une des formes représentées dans la figure 13.7.

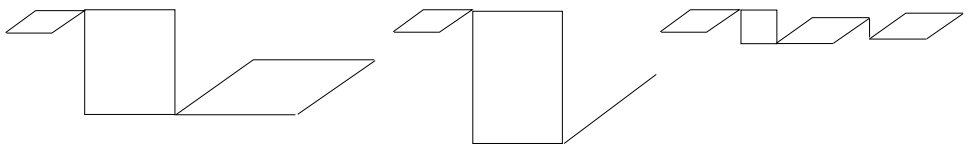


FIG. 13.7 - *Formes des chemins en* $\mathcal{S}'(\lambda, \mu, \nu)$

13.2 Φ_3 bi-périodiquement strictement décroissante

Dans cette partie, on analysera le cas où Φ_2 est bi-périodiquement strictement croissante pour le bi-motif de Φ_1 et Φ_3 est bi-périodiquement strictement décroissante pour ce bi-motif.

Pour que la contrainte Φ_2 soit strictement croissante pour le bi-motif de Φ_1 (voir chapitre 10), il faut que a'_1, a'_3, b'_1, b'_3 satisfassent à :

$$a'_3 \times b'_1 \ > \ a'_1 \times b'_3$$

Pour que la contrainte Φ_3 soit strictement décroissante pour le bi-motif de Φ_1 (voir chapitre 10), il faut que a'_1, a'_2, a'_3, c'_1, c'_2, c'_3 satisfassent à :

$$a'_2 \times c'_1 \ > \ a'_1 \times c'_2 \qquad\qquad a'_3 \times c'_1 \ > \ a'_1 \times c'_3$$

Comme dans la partie 13.1, on prouvera que la contrainte globale de tout chemin $\mathcal{P}$ en $\mathcal{S}(\lambda, \mu, \nu)$ est subsumée par la contrainte globale d'un chemin $\mathcal{P}'$ contenu dans un nombre fixe de plans.

Soit L, M, N et Q des points de $\mathcal{P}$ (voir figure 13.8), tel que:

- En M, on trouve le premier pas en z de $\mathcal{P}$, tel qu'il existe au moins un pas en y en $\mathcal{P}$ avant M. Le premier pas en y avant M part du point Q.
- En Q, on trouve le dernier pas en z de $\mathcal{P}$, tel qu'il existe au moins un pas en x en $\mathcal{P}$ après Q. Le premier pas en x après Q part du point L.

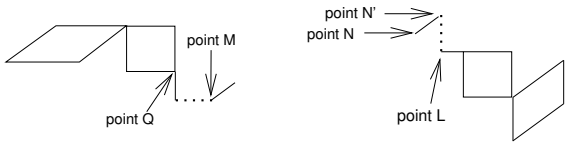


FIG. 13.8 -

Soit N' le point situé un pas en z après N ($N' = N + (0, 0, 1)$). Soit α, β et γ, respectivement, le nombre de pas en x, pas en y et pas en z de $\mathcal{P}$ situés entre les points M et N'.

En partant de N' et en remontant dans le plan $0xy$ qui passe par N', on compose le chemin-motif $\mathcal{P}^1_{\text{motif}}$ (dans le plan $0xy$), comme illustré à la figure 13.9, jusqu'à obtenir un point S tel que :

$$\Phi_3(S) \; > \; \Phi_3(N') \; + \; a'_1 \times c'_3$$

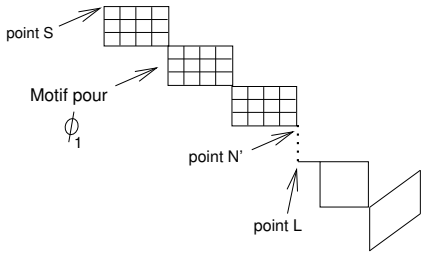


FIG. 13.9 -

Le nombre de motifs entre S et N' ne depend pas du chemin $\mathcal{P}$, cars la contrainte Φ_3 est strictement décroissante pour le motif $(-a'_2, a'_1, 0)$ de Φ_1. Si le point S ne se situe pas en bas, le chemin $\mathcal{P}$ est déjà contenu dans un nombre fixe de plans. Considérons le cas où $M \; < S$.

Soit α, β et γ, respectivement, le nombre de pas en x, pas en y et pas en z de $\mathcal{P}$ entre le point M et le point S.

Pour la construction de $\mathcal{P}'$, on distinguera le cas où $(\alpha \text{ div } \mid a'_3 \mid \geq \; \gamma \text{ div } \mid a'_1 \mid)$ du cas où $(\alpha \text{ div } \mid a'_3 \mid < \; \gamma \text{ div } \mid a'_1 \mid)$.

1. Considérons d'abord que $(\alpha \text{ div } \mid a'_3 \mid \geq \; \gamma \text{ div } \mid a'_1 \mid)$. Cela correspond à un manque relatif de pas en z par rapport au nombre de pas en x.

Construction du chemin $\mathcal{P}'$:

Le chemin $\mathcal{P}'$ coïncide avec $\mathcal{P}$ avant le point M et après le point N'. Le pont S appartient à $\mathcal{P}'$, et les point S et N' sont reliés comme illustré à la figure 13.9. Entre les points M et S, le chemin $\mathcal{P}'$ est obtenu de la façon suivante :

– En partant de S et en remontant vers le point M, on compose le chemin-motif $\mathcal{P}^2_{\text{motif}}$ (dans le plan $0xz$), comme illustré à la figure 13.10, jusqu'à utiliser tous les γ pas en z situés entre les points M et N'. L'intersection avec le plan $0xy$ qui passe par M détermine le point R. Au début du plan $0xz$, nous trouvons un suffixe du chemin motif $\mathcal{P}^2_{\text{motif}}$.

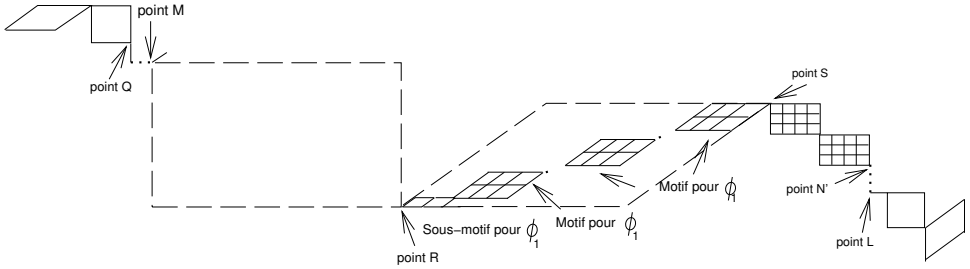


FIG. 13.10 - *Composition de chemin-motif* $\mathcal{P}^2_{motif}$

– En partant de R et se dirigeant vers le point M, on compose le chemin-motif $\mathcal{P}^1_{\text{motif}}$ (dans le plan $0xy$), comme illustré à la figure 13.11. Le point T est déterminé quand on rencontre l'axe $0x$ ou l'axe $0z$ qui passent par M. Au début du plan $0xy$, nous trouvons un suffixe du chemin motif $\mathcal{P}^1_{\text{motif}}$.

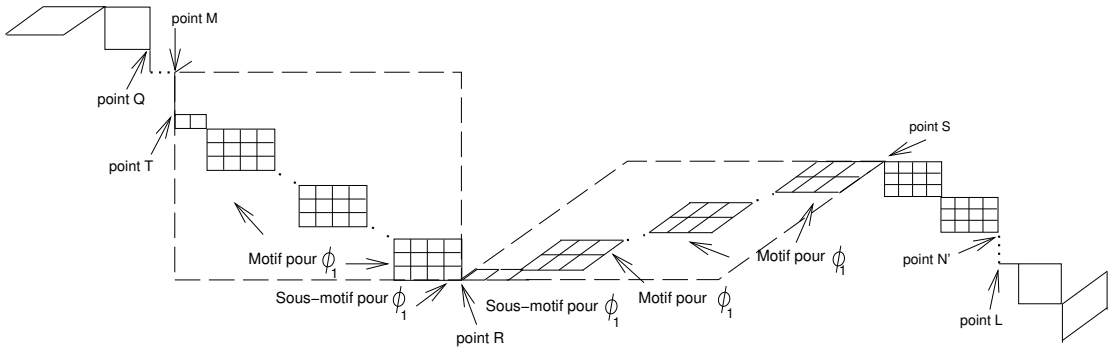


FIG. 13.11 - *Composition de chemin-motif* $\mathcal{P}^1_{motif}$

– On relie le point T et le point M par une séquence de pas en x ou une séquence de pas en y.

On obtient ainsi le chemin $\mathcal{P}'$.

Preuve de $\Gamma_{\mathcal{P}} \Rightarrow \Gamma_{\mathcal{P}'}$ ou que
$\mathcal{P}$ est déjà contenu dans un nombre fixe de plans :

Nous montrons ensuite que **la contrainte globale de $\mathcal{P}$ est subsumée par la contrainte globale de $\mathcal{P}'$** ou que **$\mathcal{P}$ est déjà contenu dans un nombre fixe de plans**.

Soit $\mathcal{P}_1$ (resp. $\mathcal{P}'_1$) la partie de chemin $\mathcal{P}$ (resp. $\mathcal{P}'$) située entre le le point M et le point N'. Comme $\mathcal{P}$ et $\mathcal{P}'$ coïncident jusqu'au point M et après le point N' et comme $\Phi_2(Q)$ est une contrainte de $\mathcal{P}$ (puisque dans $\mathcal{P}$, il y a un segment vertical qui part de Q), pour prouver que la contrainte globale de $\mathcal{P}$ est subsumée par la contrainte globale de $\mathcal{P}'$, il suffit de démontrer que :

(a) $\Gamma^1_{\mathcal{P}_1} \Rightarrow \Gamma^1_{\mathcal{P}'_1}$.

(b) $\Phi_2(Q) \Rightarrow \Gamma^2_{\mathcal{P}'_1}$.

(c) $\Gamma^3_{\mathcal{P}_1} \Rightarrow \Gamma^3_{\mathcal{P}'_1}$

- **Preuve de $\Gamma^1_{\mathcal{P}_1} \Rightarrow \Gamma^1_{\mathcal{P}'_1}$:** Entre les points T et N', par la propriété de base de chemin-motif (proposition 7.12), la conjonction des contraintes Φ_1 équivaut à $\Phi_1(N')$.
  Comme $\forall \alpha, \beta, \gamma\ \Phi_1(\alpha, \beta+1, \gamma) \Rightarrow\ \Phi_1(\alpha, \beta, \gamma)\ (a'_2 \leq 0)$, $\Phi_1(L)\ \Rightarrow\ \Phi_1(N')$.
  Si les points M et T sont reliés par des pas en x, $\Gamma^1_{\mathcal{P}'_1}$ équivaut à $\Phi_1(M) \wedge \Phi_1(N')$.
  Soit M' le point où on touve le premier pas en x de $\mathcal{P}$ après le point M. Du fait que $\forall \alpha, \beta, \gamma\ \Phi_1(\alpha, \beta+1, \gamma) \Rightarrow\ \Phi_1(\alpha, \beta, \gamma)\ (a'_2 \leq 0)$ et
  $\forall \alpha, \beta, \gamma\ \Phi_1(\alpha, \beta, \gamma+1) \Rightarrow\ \Phi_1(\alpha, \beta, \gamma)\ (a'_3 \leq 0)$, il s'ensuit que $\Phi_1(M') \Rightarrow \Phi_1(M)$.
  Donc, $\Phi_1(M') \wedge \Phi_1(L)\ \Rightarrow\ \Phi_1(M) \wedge \Phi_1(N')\ (\equiv \Gamma^1_{\mathcal{P}'_1})$.
  Si les points M et T sont reliés par des pas en y, $\Gamma^1_{\mathcal{P}'_1}$ équivaut à $\Phi_1(N')$.
  Donc, $\Phi_1(L)\ \Rightarrow\ \Phi_1(N')\ (\equiv \Gamma^1_{\mathcal{P}'_1})$.
  Comme dans le chemin $\mathcal{P}$ il y a un segment horizontal qui part de L et un autre qui part de M', $\Phi_1(L)$ et $\Phi_1(M')$ sont des contraintes de $\mathcal{P}$.
  Par conséquent, $\Gamma^1_{\mathcal{P}_1} \Rightarrow \Gamma^1_{\mathcal{P}'_1}$.
- **Preuve de $\Phi_2(Q) \Rightarrow \Gamma^2_{\mathcal{P}'_1}$:** Comme la contrainte Φ_2 est croissante dans le plan $0xy$ et entre les points R et S nous n'avons plus de pas en y, la contrainte Φ_2 la plus forte de $\mathcal{P}'$ entre M et S est plus faible que $\Phi_2(Q)$. Entre le point S et N', la contrainte la plus forte est celle du premier pas en y après le point S. Mais comme Φ_2 esp périodiquement strictement décroissante dans le plan $0xz$, cette contrainte est necessairement plus faible que $\Phi_2(R)$ (si on peut placer un nombre suffissant grand de motifs entre R et S), et par transitivité, plus faible que $\Phi_2(Q)$.
- **Preuve de $\Gamma^3_{\mathcal{P}_1} \Rightarrow \Gamma^3_{\mathcal{P}'_1}$:** Comme le plan $0xy$ ne contient pas de pas en z et comme Φ_3 est périodiquement strictement décroissante dans le $0xz$, la contrainte Φ_3 la plus forte entre les point M et N' est située dans le dernier motif de $0xz$. Soit S' le point où on trouve cette contrainte. On a:
  $\Phi_3(S) = \Phi_3(S') - \alpha' \mid c'_1 \mid + \gamma' \mid c'_3 \mid$, avec $\alpha' \leq \mid a'_3 \mid$ et $\gamma' \leq \mid a'_1 \mid$.

Comme $\alpha' \leq \mid a'_3 \mid$ et $\gamma' \leq \mid a'_1 \mid$, il s'ensuit que $\Phi_3(S') > \Phi_3(S) - a'_1 \times c'_3 + \mid c'_3 \mid$. Mais le pont S est tel que $\Phi_3(S) > \Phi_3(N') + a'_1 \times c'_3$. Il s'en suit que $\Phi_3(S') > \Phi_3(N') + c'_3 = \Phi_3(N)$.
Comme dans le chemin $\mathcal{P}$ il y a un segment transversal qui part de N, $\Phi_3(N)$ est une contrainte de $\mathcal{P}$, on a $\Gamma^3_{\mathcal{P}_1} \Rightarrow \Gamma^3_{\mathcal{P}'_1}$.

2. Considérons maintenant que (α div $\mid a'_3 \mid <$ γ div $\mid a'_1 \mid$). Cela correspond à un manque relatif de pas en x par rapport au nombre de pas en z.

Construction du chemin $\mathcal{P}'$:

Le chemin $\mathcal{P}'$ coïncide avec $\mathcal{P}$ avant le point M et après le point N'. Le pont S appartient à $\mathcal{P}'$, et les point S et N' sont reliés comme illustré à la figure 13.9. Entre les points M et S, le chemin $\mathcal{P}'$ est obtenu de la façon suivante :

– En partant de S et en remontant vers le point M, on compose le chemin-motif $\mathcal{P}^2_{\text{motif}}$ (dans le plan $0xz$), comme illustré à la figure 13.12, jusqu'à utiliser tous les pas en x situés entre les points M et N'. L'intersection avec le plan $0yz$ qui passe par M détermine le point R. Au début du plan $0xz$, nous trouvons un suffixe du chemin motif $\mathcal{P}^2_{\text{motif}}$.

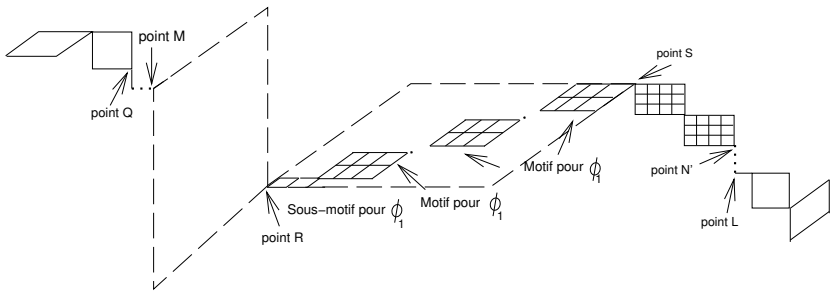


FIG. 13.12 - *Composition de chemin-motif $\mathcal{P}^2_{motif}$*

– En partant de M et en se dirigeant vers le point R, on execute une séquence de β pas en y, jusqu'à rencontrer le plan $0xz$ qui passe par R. Cette intersection dètermine le point T. On relie le point T au point R par des pas en z. Ceci est illustré à la figure 13.13.

On obtient ainsi le chemin $\mathcal{P}'$.

Preuve de $\Gamma_{\mathcal{P}} \Rightarrow \Gamma_{\mathcal{P}'}$ ou que $\mathcal{P}$ est déjà contenu dans un nombre fixe de plans :

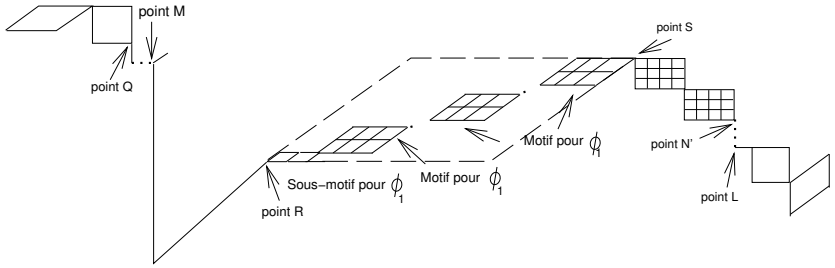


FIG. 13.13 -

Nous montrons ensuite que **la contrainte globale de $\mathcal{P}$ est subsumée par la contrainte globale de $\mathcal{P}'$** ou que **$\mathcal{P}$ est déjà contenu dans un nombre fixe de plans**.

Soit $\mathcal{P}_1$ (resp. $\mathcal{P}'_1$) la partie de chemin $\mathcal{P}$ (resp. $\mathcal{P}'$) située entre le le point M et le point N'. Comme $\mathcal{P}$ et $\mathcal{P}'$ coïncident jusqu'au point M et après le point N' et comme $\Phi_2(Q)$ est une contrainte de $\mathcal{P}$ (puisque dans $\mathcal{P}$, il y a un segment vertical qui part de Q), pour prouver que la contrainte globale de $\mathcal{P}$ est subsumée par la contrainte globale de $\mathcal{P}'$, il suffit de démontrer que :

(a) $\Gamma^1_{\mathcal{P}_1} \Rightarrow \Gamma^1_{\mathcal{P}'_1}$.

(b) $\Phi_2(Q) \Rightarrow \Gamma^2_{\mathcal{P}'_1}$.

(c) $\Gamma^3_{\mathcal{P}_1} \Rightarrow \Gamma^3_{\mathcal{P}'_1}$

- **Preuve de $\Gamma^1_{\mathcal{P}_1} \Rightarrow \Gamma^1_{\mathcal{P}'_1}$:** Entre les points M et R, nous n'avons pas de pas en x. Entre les points R et N', par la propriété de base de chemin-motif (proposition 7.12), la conjonction des contraintes Φ_1 équivaut à $\Phi_1(N')$.
  Donc $\Gamma^1_{\mathcal{P}'_1}$ équivaut à $\Phi_1(N')$.
  Comme $\forall \alpha, \beta, \gamma\ \Phi_1(\alpha, \beta+1, \gamma) \Rightarrow \Phi_1(\alpha, \beta, \gamma)$ $(a'_2 \leq 0)$, $\Phi_1(L) \Rightarrow \Phi_1(N')$ $(\equiv \Gamma^1_{\mathcal{P}'_1})$.
  Comme dans le chemin $\mathcal{P}$ il y a un segment horizontal qui part de L et un autre qui part de M', $\Phi_1(L)$ est une contrainte de $\mathcal{P}$, on a $\Gamma^1_{\mathcal{P}_1} \Rightarrow \Gamma^1_{\mathcal{P}'_1}$.
- **Preuve de $\Phi_2(Q) \Rightarrow \Gamma^2_{\mathcal{P}'_1}$:** Entre les points M et S, la contrainte la plus forte est $\Phi_2(M)$ ($b'_2 \geq 0$). Mais $\Phi_2(Q) \Rightarrow \Phi_2(M)$, car les points Q et M sont situés dans un même plan $0XY$ et Φ_2 est croissante dans le plan $0XY$.
- **Preuve de $\Gamma^3_{\mathcal{P}_1} \Rightarrow \Gamma^3_{\mathcal{P}'_1}$:** La contrainte la plus forte Φ_3 la plus forte du chemin $\mathcal{P}'$ entre les points M et N' est ou $\Phi_3(T)$ ou elle est situé dans le dernier motif

de $0xz$. Dans le premier cas, cette contrainte est plus faible que $\Phi_3(M)$ (qui est une contrainte de $\mathcal{P}$). Dans le deuxième cas, la preuve que $\Gamma^3_{\mathcal{P}_1} \Rightarrow \Gamma^3_{\mathcal{P}'_1}$ est faite comme dans le cas où (α div $\mid a'_3 \mid \geq \gamma$ div $\mid a'_1 \mid$).

En résumé, tout chemin $\mathcal{P}$ est subsumé par un chemin ayant une des formes représentées dans la figure 13.14.

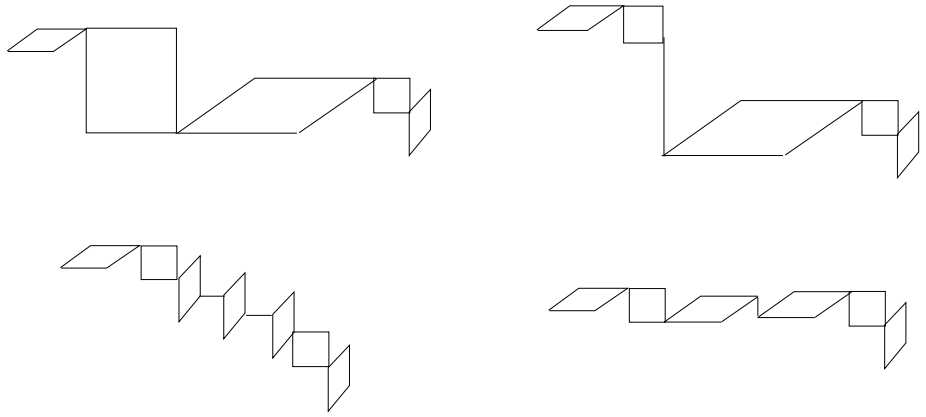


FIG. 13.14 - *Formes des chemins en* $\mathcal{S}'(\lambda, \mu, \nu)$

Dans les deux cas (Φ_3 bi-périodiquement croissante ou décroissante), tout chemin de $\mathcal{S}(\lambda, \mu, \nu)$ est subsumé par un chemin contenu dans un nombre fixe de plans.

Chapitre 14

Applications

14.1 Génération Automatique de Lemmes

La motivation initiale de ce travail (pour représenter par une formule arithmétique le résultat de l'évaluation ascendante de programmes logiques) était de fournir une méthodologie pour prouver des formules qui combinaient des listes et des formules arithmétiques.

Plus concrètement, étant donné un programme logique définissant un prédicat $p(L, x, y, x)$, où L est une liste d'entiers relatifs et x, y and z sont des variables sur le domaine des entiers relatifs, supposons que nous voulons prouver une propriété de la forme $\Psi(x, y, z)$.

La méthodologie pour prouver $\Psi(x, y, z)$ consiste en trois étapes :

1. Transformer le prédicat $p(L, x, y, z)$ dans un prédicat $p'(x, y, z)$ (voir [FRI 92]).

2. Calculer par évaluation ascendante la formule caractéristique ξ' associé à p'.

3. Démontrer que $\xi' \Rightarrow \Psi$.

Ceci sera détaillé sur l'exemple de l'algorithme de Boyer-Moore, à la partie 14.4.

14.2 Terminaison de Programmes Logiques

On peut utiliser cette méthode d'évaluation ascendante pour prouver la terminaison de programmes logiques. Pour cela, on construit un programme remplaçant les paramètres du programme donné par leur taille (ceci peut être vu comme une forme "d'interprétation abstraite" [COU 78]). Nous illustrons cette application pour le programme (extrait de [PLU 90]):

$$
\begin{array}{lcl}
split([\,],y,[\,],[\,]). & & \\
split([u|L],y,[u|M],N) & :- & u \leq y, split(L,y,M,N). \\
split([u|L],y,M,[u|N]) & :- & u > y, split(L,y,M,N).
\end{array}
$$

En remplaçant les arguments de liste par leur taille, et en enlevant les contraintes auxiliaires $u \leq y$ and $u > y$, on transforme le programme ci-dessous en :

$$
\begin{array}{lcl}
split'(x,y,z,v) & :- & x = 0, z = 0, v = 0. \\
split'(x+1,y,z+1,v) & :- & split'(x,y,z,v). \\
split'(x+1,y,z,v+1) & :- & split'(x,y,z,v).
\end{array}
$$

Notre méthode appliquée à $split'$ engendre une formule arithmétique équivalant à $x = z + v$. Cela signifie que la taille du quatrième argument de $split$ est égal à la somme de la taille du premier et du troisième argument. Cette relation entre les arguments de $split$ peut être utilisée pour prouver la terminaison du programme $quicksort$ (voir [PLU 90]). Notons que la procédure utilisée par Plümer (basée sur une évaluation descendante) engendre la relation $x \geq z + v$, qui est moins fine que la relation engendrée par notre méthode.

14.3 Bases de Données Déductives Temporelles

Un programme Datalog est un programme logique sans symbole de fonction. Un programme Datalog Temporel est un programme logique où des symboles de fonction (par exemple, $succ$, $+$) ne sont admis que pour les arguments "temporels".

Etant donné un programme Datalog Temporel et une requête $p(x,y,z)$, le traitement d'une requête consiste à déterminer s'il existe ou non une substitution close σ tel que $\sigma(p(x,y,z))$ soit une conséquence logique du programme (et à déterminer toutes les substitutions closes, s'il en existe une). Il est bien connu que ce problème est indécidable pour un programme Datalog quelconque, mais il est décidable pour des programmes avec un

seul argument temporel sur lequel on fait des incrémentations, à condition qu'il n'y ait pas de contrainte arithmétique dans le corps [CHO 88]. Notre méthode d'évaluation ascendante rend le problème du traitement d'une requête décidable pour une nouvelle classe de programmes : celle où les programmes ont un nombre *arbitraire* d'arguments temporels sur lesquels on effectue des incrémentations conditionnelles. Par contre, nos programmes ne contiennent pas plus de trois règles récursives.

14.4 Exemple de Génération Automatique de Lemmes

Le problème du candidat majoritaire consiste à désigner le candidat ayant au moins la moitié des votes dans une urne, si un tel candidat existe. Si nous modélisons l'urne par une liste L de taille y, ce problème se ramène à déterminer si un item v a eu une occurrence dans la liste L supérieure ou égale à $(y\ div\ 2)$ fois. L'algorithme de Majoration de Boyer-Moore (voir [MIS 82]) fournit une solution à ce problème, en déterminant un item v qui est un candidat majoritaire *possible*, c'est-à -dire, un item tel qu'aucun autre item n'apparaisse $(y\ div\ 2)$ fois dans L. Mis sous la forme d'un programme logique, cet algorithme donne:

$$\begin{array}{lll} p_1([\,], v, x, y) & :- & x = 0,\ y = 0 \\ p_1([u|L], v, x+1, y+1) & :- & u = v,\ p_1(L, v, x, y) \\ p_1([u|L], v, x, y+1) & :- & u \neq v,\ 2x > y,\ p_1(L, v, x, y) \\ p_1([u|L], u, x+1, y+1) & :- & u \neq v, 2x = y,\ p_1(L, v, x, y) \end{array}$$

La correction de Π_1 peut se formaliser :

$$p_1(L, v, x, y)\ \wedge\ p_2(L, w, z)\ \Longrightarrow\ (v \neq w\ \Longrightarrow\ z \leq y\ div\ 2) \qquad (*)$$

où $p_2(L, w, z)$ signifie que le nombre d'occurrences de l'item w dans la liste L est z. Le prédicat p_2 est défini par :

$$\begin{array}{lll} p_2([\,], w, z) & :- & z = 0 \\ p_2([u|L], w, z+1) & :- & u = w,\ p_2(L, w, z) \\ p_2([u|L], w, z) & :- & u \neq w,\ p_2(L, w, z) \end{array}$$

En appliquant des opérations de pliage/dépliage, nous pouvons remplacer la conjonction $p_1(L, v, x, y)\ \wedge\ p_2(L, w, z)$ par l'atome $p_3(L, v, w, x, y, z)$:

$$
\begin{array}{lll}
p_3([\,], v, w, x, y, z) & :- & x = 0,\ y = 0,\ z = 0. \\
p_3([u|L], v, w, x+1, y+1, z+1) & :- & u = v, v = w,\ p_3(L, v, w, x, y, z). \\
p_3([u|L], v, w, x+1, y+1, z) & :- & u = v, u \neq w,\ p_3(L, v, w, x, y, z). \\
p_3([u|L], v, w, x, y+1, z+1) & :- & u \neq v, u = w, 2x > y,\ p_3(L, v, w, x, y, z). \\
p_3([u|L], v, w, x, y+1, z) & :- & u \neq v, u \neq w, 2x > y,\ p_3(L, v, w, x, y, z). \\
p_3([u|L], u, w, x+1, y+1, z+1) & :- & u \neq v, u = w, 2x = y,\ p_3(L, v, w, x, y, z). \\
p_3([u|L], u, w, x+1, y+1, z) & :- & u \neq v, u \neq w, 2x = y,\ p_3(L, v, w, x, y, z).
\end{array}
$$

Après l'élimination de la liste L et du paramètre u, on obtient le programme Π_3':

$$
\begin{array}{lllll}
p_3'(v, w, x, y, z) & :- & x = 0,\ y = 0,\ z = 0 & & \text{règle } \mathcal{R}_0 \\
p_3'(v, w, x+1, y+1, z+1) & :- & v = w,\ p_3'(v, w, x, y, z) & & \text{règle } \mathcal{R}_1 \\
p_3'(v, w, x+1, y+1, z) & :- & v \neq w,\ p_3'(v, w, x, y, z) & & \text{règle } \mathcal{R}_2 \\
p_3'(v, w, x, y+1, z+1) & :- & v \neq w, 2x > y,\ p_3'(v, w, x, y, z) & & \text{règle } \mathcal{R}_3 \\
p_3'(v, w, x, y+1, z) & :- & 2x > y,\ p_3'(v, w, x, y, z) & & \text{règle } \mathcal{R}_4 \\
p_3'(w, w, x+1, y+1, z+1) & :- & v \neq w, 2x = y,\ p_3'(v, w, x, y, z) & & \text{règle } \mathcal{R}_5 \\
p_3'(u, w, x+1, y+1, z) & :- & u \neq v, u \neq w, 2x = y,\ p_3'(v, w, x, y, z) & & \text{règle } \mathcal{R}_6
\end{array}
$$

La preuve de la correction de l'expression $(*)$ se réduit à la preuve de : $(p_3'(v, w, x, y, z) \wedge v \neq w) \Rightarrow z \leq y\ div\ 2$. Cela sera fait en montrant que si un atome $p_3'(v, w, x, y, z)$ appartient au résultat de l'évaluation ascendante de Π_3', alors v, w, z et y satisfont à $(v \neq w \Rightarrow z \leq y\ div\ 2)$.

A priori, notre méthode pour calculer le résultat de l'évaluation ascendante ne s'applique pas au programme Π_3', car il contient 6 règles récursives et les contraintes de ces règles contiennent plus d'une inégalité.

Pour pouvoir calculer l'évaluation ascendante de Π_3' par la méthode décrite au chapitre 9, nous utiliserons deux techniques bien connues de transformation de programmes, à savoir :

La Stratification de Programmes : on effectue une partition du programme en des ensembles d'au plus 3 règles récursives. Ensuite, nous appliquons itérativement notre méthode à chacun de ces ensembles. A chaque étape, on prend comme cas de base l'ensemble engendré à l'étape précédente. Ce processus se termine lorsque aucun nouveau atome n'est engendré, mais cette terminaison n'est pas toujours assurée. Toutefois, dans la pratique, on arrive souvent à obtenir une partition des règles du programme telle que ce processus termine.

Elimination de Contraintes : on élimine les contraintes qui sont invariantes par rapport à toutes les règles récursives du programme et qui appartiennent à la règle de

base. Ensuite, on calcule l'évaluation ascendante d'un programme sans ces contraintes invariantes et on les réintroduit dans le résultat de l'évaluation ascendante de programme. Ceci est similaire au concept de "pushing" contraintes (voir, par exemple, [LEV 92, SRI 92b]; *cf* [MAR 93, KEM 93]).

Considérons d'abord que les règles $\mathcal{R}_0, \ldots, \mathcal{R}_5$. A fin d'éliminer les contraintes $v = w$ and $v \neq w$ de ces règles, on remplacera la règle de base $\mathcal{R}_0$ par deux règles de base $\mathcal{R}'_0$: $(p'_3(v, w, x, y, z) \ :- \ v = w,\ x = 0,\ y = 0,\ z = 0$ et $\mathcal{R}''_0$: $p'_3(v, w, x, y, z) \ :- \ v \neq w,\ x = 0,\ y = 0,\ z = 0)$.

En analysant le programme $\mathcal{R}_0 \cdots \mathcal{R}_5$, notons que:

- Les règles $\mathcal{R}_1$ et $\mathcal{R}_4$ laissent la contrainte $v = w$ invariante.
- Les règles $\mathcal{R}_2$, $\mathcal{R}_3$ et $\mathcal{R}_4$ laissent la contrainte $v \neq w$ invariante.
- La règle $\mathcal{R}_5$ transforme la contrainte $v \neq w$ en $v = w$.

Donc, le résultat de l'évaluation ascendante de $\mathcal{R}_0, \ldots, \mathcal{R}_5$ peut être obtenu par l'union du:

1. Résultat de l'évaluation ascendante de $\mathcal{R}'_0$, $\mathcal{R}_1$ et $\mathcal{R}_4$.
2. Résultat de l'évaluation ascendante de $\mathcal{R}''_0$, $\mathcal{R}_1$, $\mathcal{R}_2$, $\mathcal{R}_3$, $\mathcal{R}_4$ et $\mathcal{R}_5$. En outre, ces règles peuvent être stratifiées en:
   (a) Règles $\mathcal{R}''_0$, $\mathcal{R}_2$, $\mathcal{R}_3$ et $\mathcal{R}_4$.
   (b) Règle $\mathcal{R}_5$.
   (c) Règles $\mathcal{R}_1$, $\mathcal{R}_2$ et $\mathcal{R}_4$.

Ceci est représenté à la figure 14.1.

Par conséquent, le programme $\mathcal{R}_0, \ldots, \mathcal{R}_5$ peut être traité par notre méthode.

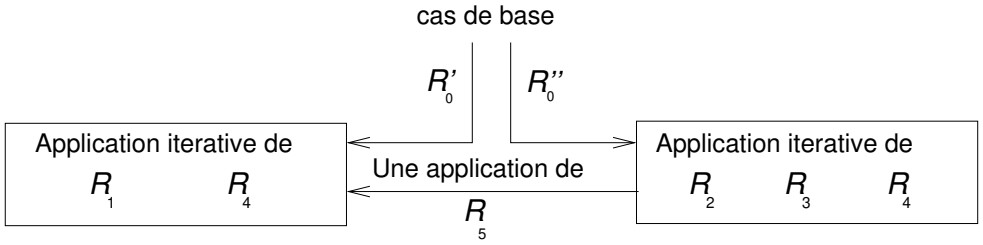


FIG. 14.1 - *Schéma de stratification des règles* $\mathcal{R}'_0, \mathcal{R}''_0, \mathcal{R}_1, \ldots, \mathcal{R}_5$

1 - Evaluation ascendante de $\mathcal{R}'_0$, $\mathcal{R}_1$ et $\mathcal{R}_4$

Considérons le programme $\mathcal{R}'_0$, $\mathcal{R}_1$ et $\mathcal{R}_4$. La contrainte $v = w$ appartient au cas de base et elle est invariante par rapport aux règles $\mathcal{R}_1$ et $\mathcal{R}_4$. Donc elle peut être supprimée et on calculera l'évaluation ascendante du programme Π''_3 :

$p'_3(v,w,x,y,z)$	:–	$x = 0,\ y = 0,\ z = 0$	règle $\mathcal{R}'_0$
$p'_3(v,w,x+1,y+1,z+1)$	:–	$p'_3(v,w,x,y,z)$	règle $\mathcal{R}_1$
$p'_3(v,w,x,y+1,z)$	:–	$2x > y,\ p'_3(v,w,x,y,z)$	règle $\mathcal{R}_4$

Au résultat de cette évaluation ascendante, on ajoutera la contrainte $v = w$.

Le programme Π''_3 contient 2 règles récursives. Il appartient à la classe $\rightarrow\uparrow\Rightarrow\Uparrow$ (entre autres), donc $\Delta'(\lambda,\mu)$, pour $\lambda,\mu \in \mathbb{N}^*$, équivaut à $\Phi_1(0,0) \wedge \Phi_2(\lambda,\mu-1)$. Dans ce cas $\Phi_1(0,0) =$ vrai et $\Phi_2(\lambda,\mu-1) = \lambda + 2x > y + \mu - 1$.

L'évaluation ascendante de Π''_3 engendre l'ensemble :

$\{p(v,w,x+\lambda,y+\lambda+\mu,z+\lambda) \ / \ \lambda \geq 0 \wedge \mu \geq 0 \wedge x = 0 \wedge y = 0 \wedge z = 0 \wedge \Delta(\lambda,\mu)\ \}$.

En remplaçant $\Delta(\lambda,\mu)$ par l'expression $\lambda + 2x > y + \mu - 1$, avec $x = 0$ et $y = 0$, on obtient l'ensemble $\{p(v,w,\lambda,\lambda+\mu,\lambda) \ / \ \lambda \geq 0 \ \wedge \ \mu \geq 0 \ \wedge \lambda > \mu - 1\}$.

Donc, l'évaluation ascendante de $\mathcal{R}'_0$, $\mathcal{R}_1$ et $\mathcal{R}_4$ engendre l'ensemble $S_1 = \{p(v,w,\lambda,\lambda+\mu,\lambda) \ / \ u = v \wedge \ \lambda \geq 0 \wedge \mu \geq 0 \wedge \lambda > \mu - 1\}$.

2 - Evaluation ascendante de $\mathcal{R}''_0$, $\mathcal{R}_1$, $\mathcal{R}_2$, $\mathcal{R}_3$, $\mathcal{R}_4$ et $\mathcal{R}_5$

2.a - Evaluation ascendante de $\mathcal{R}''_0$, $\mathcal{R}_2$, $\mathcal{R}_3$, $\mathcal{R}_4$

Considérons le programme :

$$
\begin{array}{lll}
p'_3(v,w,x,y,z) & :- \quad v \neq w,\ x=0,\ y=0,\ z=0 & \text{règle } \mathcal{R}''_0 \\
p'_3(v,w,x+1,y+1,z) & :- \quad v \neq w,\ p'_3(v,w,x,y,z) & \text{règle } \mathcal{R}_2 \\
p'_3(v,w,x,y+1,z+1) & :- \quad v \neq w, 2x > y,\ p'_3(v,w,x,y,z) & \text{règle } \mathcal{R}_3 \\
p'_3(v,w,x,y+1,z) & :- \quad 2x > y,\ p'_3(v,w,x,y,z) & \text{règle } \mathcal{R}_4
\end{array}
$$

La contrainte $v \neq w$ appartient au cas de base et elle est invariante par rapport aux règles $\mathcal{R}_2$, $\mathcal{R}_3$ et $\mathcal{R}_4$. Donc elle peut être supprimée et on calculera l'évaluation ascendante du programme Π'''_3:

$$
\begin{array}{lll}
p'_3(v,w,x,y,z) & :- \quad x=0,\ y=0,\ z=0 \\
p'_3(v,w,x+1,y+1,z) & :- \quad p'_3(v,w,x,y,z) \\
p'_3(v,w,x,y+1,z+1) & :- \quad 2x > y,\ p'_3(v,w,x,y,z) \\
p'_3(v,w,x,y+1,z) & :- \quad 2x > y, p'_3(v,w,x,y,z)
\end{array}
$$

Au résultat de cette évaluation ascendante, on ajoutera la contrainte $v \neq w$. Le programme Π'''_3 contient 3 règles récursives. Les implications de contraintes pour ce programme sont représentées à la figure 14.2. Dans la sous-partie 9.1, nous calculons la forme des chemins en $\mathcal{S}'(\lambda,\mu,\nu)$ pour des programmes avec ces implications de contraintes.

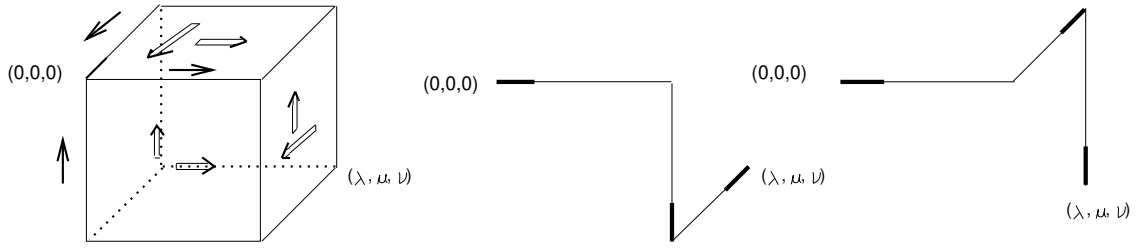


FIG. 14.2 - *Implications de contraintes et formes des chemins en* $\mathcal{S}'(\lambda,\mu,\nu)$

Les contraintes plus fortes de chaque chemin en $\mathcal{S}'(\lambda,\mu,\nu)$ sont représentées à la figure 14.2 par un trait plus épais. Il s'ensuit que, pour $\lambda > 0$, $\mu > 0$ et $\nu > 0$:

$$\Delta(\lambda,\mu,\nu) = \Phi_1(0,0,0) \wedge ((\Phi_2(\lambda,\mu-1,0) \wedge \Phi_3(\lambda,\mu,\nu-1)) \vee (\Phi_2(\lambda,\mu,\nu-1) \wedge \Phi_3(\lambda,0,\nu-1)),$$

où :

$$\begin{array}{lcl}
\Phi_1(0,0,0) & = & \text{vrai} \\
\Phi_2(\lambda,\mu-1,0) & = & \lambda+2x > y+\mu-1 \\
\Phi_3(\lambda,\mu,\nu-1) & = & \lambda+2x > y+\mu+\nu-1 \\
\Phi_2(\lambda,\mu,\nu-1) & = & \lambda+2x > y+\mu+\nu-1 \\
\Phi_3(\lambda,0,\nu-1) & = & \lambda+2x > y+\nu-1
\end{array}$$

Donc, pour $\lambda > 0$, $\mu > 0$ et $\nu > 0$, $\Delta(\lambda,\mu,\nu) \equiv \lambda + 2x > y + \mu + \nu - 1$.

L'évaluation ascendante de Π_3'' engendre l'ensemble :

$\{p(v,w,x+\lambda,y+\lambda+\mu+\nu,z+\mu) \,/\, \lambda \geq 0 \wedge \mu \geq 0 \wedge \nu \geq 0 x = 0 \wedge y = 0 \wedge z = 0 \wedge \Delta(\lambda,\mu,\nu)$.

En remplaçant $\Delta(\lambda,\mu)$ par l'expression $\lambda + 2x > y + \mu - 1$, avec $x = 0$ et $y = 0$, on obtient $\{p(v,w,\lambda,\lambda+\mu+\nu,\mu) \,/\, \lambda \geq 0 \wedge \mu \geq 0 \wedge \nu \geq 0, \lambda > \mu + \nu - 1\}$.

Par conséquent, l'évaluation ascendante de $\mathcal{R}_0''$, $\mathcal{R}_2$, $\mathcal{R}_3$ et $\mathcal{R}_4$ engendre l'ensemble $S_2 = \{p(v,w,\lambda,\lambda+\mu+\nu,\mu) \,/\, v \neq w \ \wedge \ \lambda \geq 0 \ \wedge \ \mu \geq 0 \wedge \ \nu \geq 0 \ \wedge \ \lambda > \mu + \nu - 1\}$.

2.b - Application de $\mathcal{R}_5$

Une seule application de la règle $\mathcal{R}_5$ à S_2 transforme la contrainte $v \neq w$ en $v = w$. On engendre l'ensemble :

$S_3 = \{p(v,w,\lambda+1,2\lambda+1,\mu) \,/\, v = w \wedge \ \lambda \geq 0 \wedge \nu \geq 0 \wedge \lambda \geq \mu\}$

2.c - Application itérative de $\mathcal{R}_1$ et $\mathcal{R}_4$

De façon analogue à l'étape 1, on effectue l'évaluation ascendante de programme formé par les règles $\mathcal{R}_1$ et $\mathcal{R}_4$, en considérant l'ensemble S_3 comme cas de base. L'application itérative de $\mathcal{R}_1$ et $\mathcal{R}_4$ à S_3 engendre l'ensemble :

$S_4 = \{p(v,w,\lambda+1+\lambda',2\lambda+1\lambda'+\mu',\mu+\mu') \,/\, v = w \wedge \ \lambda \geq 0 \wedge \nu \geq 0 \wedge \lambda' \geq 0 \wedge \mu' \geq 0 \wedge \lambda \geq \mu \wedge \lambda' + 2 > \mu'\}$

L'union S des ensembles S_1, S_2, S_3, S_4 donne le résultat de l'évaluation ascendante de $\mathcal{R}_0, \ldots, \mathcal{R}_5$. Ce n'est pas difficile de vérifier que :

$$\begin{array}{lcl}
S_1 & = & \{p_3'(v,w,x,y,z) \,/\, v = w,\ 2x \geq y,\ y \geq x,\ y = z,\ x \geq 0,\ y \geq 0,\ z \geq 0\} \\
S_2 & = & \{p_3'(v,w,x,y,z) \,/\, v \neq w,\ 2x \geq y,\ y \geq z + x,\ x \geq 0,\ y \geq 0,\ z \geq 0\} \\
S_3 & = & \{p_3'(v,w,x,y,z) \,/\, v = w,\ 2x = y + 1, y \geq x,\ x \geq z,\ z \geq 1\} \\
S_4 & = & \{p_3'(v,w,x,y,z) \,/\, v = w,\ 2x \geq y, y \geq x,\ x \geq z,\ z \geq 1\}
\end{array}$$

Notons que $S_3 \subset S_4$ et que $S_1 \cup S_4 = \{p'_3(v,w,x,y,z) \,/\, v = w,\ 2x \geq y, y \geq x,\ x \geq z,\ z \geq 0\}$.

Par conséquent, un atome $p'_3(v,w,x,y,z)$ appartient à S ssi:

$$(v = w \ \wedge \ 2x \geq y \ \wedge \ y \geq x \ \wedge \ x \geq z \ \wedge \ z \geq 0) \\ \vee \ (v \neq w \ \wedge \ 2x \geq y \ \wedge \ y \geq z + x \ \wedge \ y \geq 0 \ \wedge \ z \geq 0) \qquad (**)$$

Considérons maintenant la règle récursive $\mathcal{R}_6$. Comme $T_6(S) \subseteq S$, l'application de cette règle n'engendre aucun nouvel atome à S. Il en résulte que le résultat de l'évaluation ascendante de $\mathcal{R}_0, \ldots, \mathcal{R}_6$ est l'ensemble S.

La relation $v \neq w \ \Rightarrow \ z \leq y \ div \ 2$ découle directement de $(**)$.

Annexe A

Le Protocole de la Fenêtre Glissante

Dans cette annexe, nous appliquons l'opération de composition parallèle des automates en vue de spécifier le *protocole de la fenêtre glissante* . La description du protocole est reprise essentiellement de [GAR 89].

Comme pour le cas du protocole du bit alterné, le protocole de la fenêtre glissante réalise un transfert de données uni-directionnel entre deux entités, dénommées $SENDER$ et $RECEIVER$. La principale différence entre ces deux protocoles est que dans le cas de la fenêtre glissante, l'entité $SENDER$ n'attend pas la réception d'acquittement pour envoyer le message suivant.

Le transfert de données entre le $SENDER$ et le $RECEIVER$ (resp. entre le $RECEIVER$ et le $SENDER$) est effectué en passant par une entité appelée $MEDIUM_1$ (resp. $MEDIUM_2$). Les entités $MEDIUM_1$ et $MEDIUM_2$ peuvent perdre, dupliquer ou réordonner les messages.

Le tableau ci-dessous (voir [GAR 89]) donne les actions qui sont exécutées (et partagées) par ces entités.

Actions	Origine	Destination	Description
$put(N)$	environnement	$SENDER$	acquisition d'un message de l'environnement
$sdt(N)$	$SENDER$	$MEDIUM_1$	envoi d'un message
$rdt(N)$	$MEDIUM_1$	$RECEIVER$	transmission d'un message
$get(N)$	$RECEIVER$	environnement	émission d'un message vers l'environnement
$sack(N)$	$MEDIUM_2$	$SENDER$	transmission de l'acquittement
$rack(N)$	$RECEIVER$	$MEDIUM_1$	renvoi d'un acquittement

Dans la modélisation que nous présentons, le contenu de chaque message n'est pas pris en compte. Chaque message est représenté uniquement par un numéro entier N (appelé étiquette du message) entre 0 et k. Cette énumération est faite de façon croissante modulo k, c'est-à-dire : les messages sont numérotés de la façon suivante : 0, 1, ..., $(k-1)$, 0, 1, ..., $(k-1)$, ...

Chaque entité ($SENDER$, $RECEIVER$, $MEDIUM_1$ et $MEDIUM_2$) du protocole de la fenêtre glissante peut être caractérisée par un automate généralisé. Une spécification du protocole de la fenêtre glissante est obtenue par composition parallèle de ces quatres entités.

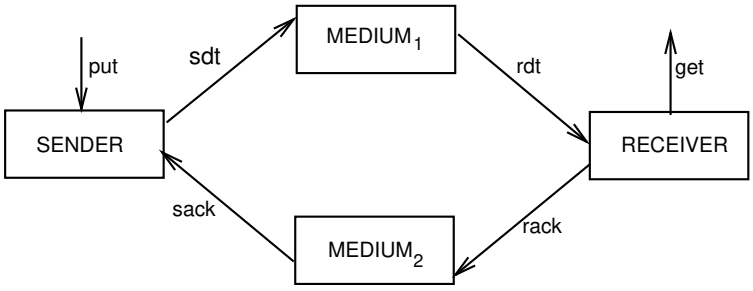


FIG. A.1 - *Protocole de la Fenêtre Glissante*

A.1 Processus $SENDER$

Le processus $SENDER$ acquiert (via put) une série de messages, numérotés en ordre croissant modulo k.

Ce processus envoie des messages (via sdt) et reçoit des acquittements (via $sack$). Il peut envoyer plusieurs messages successivement, sans attendre l'acquittement de chaque message envoyé.

Le $SENDER$ maintient une liste (appelée *fenêtre d'émission*), contenant les numéros

des messages envoyés mais non acquittés. La taille de la fenêtre d'émission ne doit pas excéder une valeur TWS fixée, avec $TWS \leq k$.

La fenêtre d'émission est identifiée par des variables : $BASE$, qui représente le numéro du message le plus ancien dans la fenêtre d'émission et $TAILLE$ qui représente la dimension de la fenêtre d'émission (soit le nombre de messages dans la fenêtre d'émission).

Un message de nombre N est dans la fenêtre d'émission ssi $BASE \leq N < BASE +_{\mathrm{mod}\, k} SIZE - 1$. Cette relation définit le prédicat $window$ (après un renommage de variables) :

$$window(N, B, S) \quad :- \quad B \leq N < B +_{\mathrm{mod}\, k} S - 1.$$

Lors de la réception d'un acquittement d'un message de nombre N, les situations suivantes peuvent se produire :

- N n'est pas dans la fenêtre d'émission. Dans ce cas, l'acquittement est ignoré.
- N est dans la fenêtre d'émission. Dans ce cas, cet acquittement valide tous les messages entre $BASE$ et N. On retire de la fenêtre tous ces messages, en attribuant à $BASE$ la valeur de $N + 1$ et en en diminuant la valeur de $TAILLE$.
- Après un certain délai (via une action τ), le $SENDER$ peut choisir un certain nombre N dans la fenêtre et réémettre tous les messages avec nombre compris entre N et $BASE +_{\mathrm{mod}\, k} (TAILLE - 1)$.

Ce processus est caractérisé par le programme logique Π_{SENDER}:

$$\begin{array}{lll}
sender([\,], wait(B, 0)) & :- & B = 0. \\
sender([sack(N)|L], wait(B, S)) & :- & sender(L, wait(B, S)), notwindow(N, B, S). \\
sender([put(N)|L], readysend(B, T)) & :- & sender(L, wait(B, T)), inf(T, TWS), \\
 & & \qquad plusmodulo(B, T, N). \\
sender([sack(N)|L], wait(N + 1, S_1)) & :- & sender(L, wait(B, T)), window(N, B, T), \\
 & & \qquad plusmodulo(T - S_1, B, N + 1). \\
sender([\tau|L], repeat(N, S_1, B, S)) & :- & sender(L, wait(B, S)), window(N, B, S), \\
 & & \qquad plusmodulo(S - S_1, B, N). \\
sender([sdt(N)|L], wait(B, S + 1)) & :- & sender(L, readysend(B, S)), \\
 & & \qquad plusmodulo(B, S, N). \\
sender([sdt(N)|L], repeat(N + 1, S_1 + 1, B, S)) & :- & sender(L, repeat(N, S_1, B, S)), inf(1, S_1). \\
sender([sdt(N)|L], wait(B, S)) & :- & sender(L, repeat(N, 1, B, S)).
\end{array}$$

Les opérations + et − dans le programme Π_{SENDER} ci-dessus sont des opérations modulo k.

L'automate AUT_{SENDER} est représenté figure A.2 .

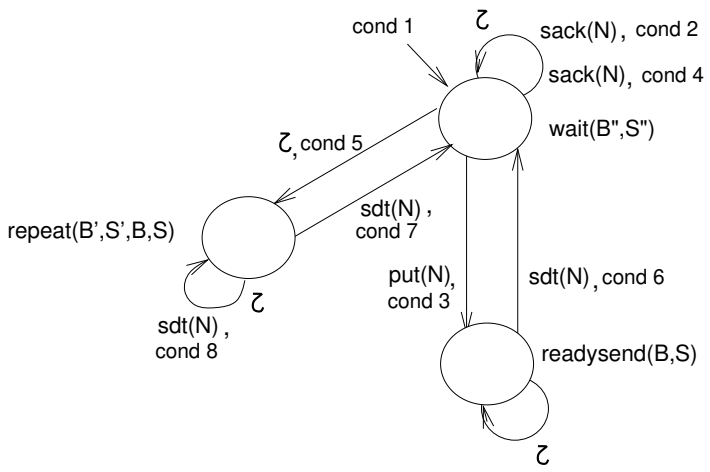


FIG. A.2 - *Processus SENDER*

A.2 Processus *RECEIVER*

Le processus *RECEIVER* renvoie une série de messages à l'environnement (via *get*). Ces messages sont numérotés dans l'ordre croissant modulo k. Le processus *RECEIVER* acquiert des messages (via *rdt*) et renvoie des acquittements (via *rack*).

Le *RECEIVER* maintient une liste (appelée *fenêtre de réception*), contenant les numéros des messages reçus mais non renvoyés. La taille de la fenêtre de réception ne doit pas excéder une valeur *RWS* fixée, avec $RWS \leq k$. Notons que le *RECEIVER* reçoit des messages dans le désordre.

Soit *BASE* le numéro du message le plus petit dans cette liste. Le message d'étiquette *BASE* doit être le prochain message à être expédié. Un message avec nombre N peut être renvoyé ssi $N = BASE$.

A l'arrivée d'un message N, les situations suivantes peuvent se produire :

- N est déjà dans la fenêtre d'émission. Dans ce cas, la fenêtre d'émission reste inchangée et un acquittement (via rdt) étiqueté avec $BASE -_{\mathrm{mod}\,k} 1$ est renvoyé.
- N n'est pas dans la fenêtre d'émission et N ne vérifie pas la relation $window(N, BASE, RWS)$. Dans ce cas, la fenêtre de réception reste aussi inchangée et un acquittement (via rdt) étiqueté avec $BASE -_{\mathrm{mod}\,k} 1$ est émis.
- N n'est pas dans la fenêtre d'émission et N vérifie la relation $window(N, BASE, RWS)$. Dans ce cas, le récepteur essaie d'expédier tous les messages disponibles à partir de la $BASE$, jusqu'à rencontrer le premier message qui manque dans la fenêtre de réception. Soit N_1 ce premier message : le $RECEIVER$ renvoie un acquittement étiqueté avec $N_1 -_{\mathrm{mod}\,k} 1$ (via $rack$) et retire de la fenêtre de réception tous les messages avec nombre compris entre $BASE$ et $N_1 -_{\mathrm{mod}\,k} 1$, inclus.

La détermination de N_1 et l'actualisation de la fenêtre de réception est faite en utilisant les prédicats set et $test$ (voir [GAR 89]).

Ce processus est caractérisé par le programme logique $\Pi_{RECEIVER}$:

$$
\begin{array}{lcl}
receiver([\,], wait(B,R)) & :- & reset(R), B = 0. \\
receiver([rdt(N)|L], deliver(B, R_1)) & :- & receiver(L, wait(B,R)), notest(R,N), \\
 & & \qquad window(N, B, RWS), set(R, N, R_1). \\
receiver([rdt(N)|L], readyrack(B,R)) & :- & receiver(L, wait(B,R)), test(R,N). \\
receiver([rdt(N)|L], readyrack(B,R)) & :- & receiver(L, wait(B,R)), notwindow(N, B, RWS). \\
receiver([get(N)|L], deliver(N+1, R_1)) & :- & receiver(L, deliver(N,R)), \\
 & & \qquad test(R,N), unset(R, N, R_1). \\
receiver([rack(N-1)|L], wait(N,R)) & :- & receiver(L, deliver(N,R)), notest(R,N). \\
receiver([rack(N-1)|L], wait(N,R)) & :- & receiver(L, readyrack(N,R)).
\end{array}
$$

Les opérations $+$ et $-$ dans le programme $\Pi_{RECEIVER}$ ci-dessus sont des opérations modulo k. L'automate $AUT_{RECEIVER}$ est représenté figure A.3 .

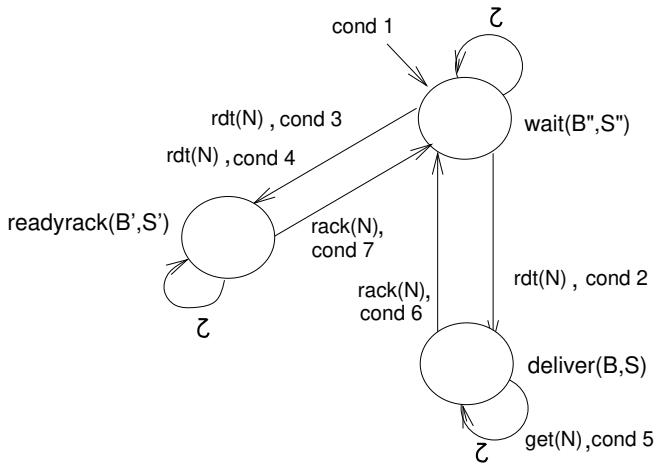


FIG. A.3 - *Processus RECEIVER*

A.3 Processus $MEDIUM_1$

Le processus $MEDIUM_1$ contient une variable d'état R, qui est un registre. Chaque cellule de ce registre peut contenir soit un numéro entre 0 et $k-1$, correspondant au numéro d'un message, soit la valeur -1, qui indique que la cellule est vide.

Lorsqu'un message de numéro N est reçu (via sdt), le $MEDIUM_1$ peut soit décaler le registre M pour y insérer N, soit décaler le registre M sans y insérer N, ce qui signifie une perte du message. Le décalage du registre M donne le registre M_1. Dans le premier cas, cette opération est implementée par un prédicat $insert(M, N, M_1)$, et dans le deuxième cas, elle l'est par un prédicat $insert(M, N, M_1)$.

Le $MEDIUM_1$ peut également choisir un numéro N, $N \neq -1$, dans le registre M et émettre le signal $rdt(N)$. Ensuite, il peut soit retirer N de M (implementé par un prédicat $delete(M, N, M_1)$), soit laisser M inchangé, ce qui rend possible une duplication du message. Le choix arbitraire de N dans le registre M est le facteur responsable de la modification de l'ordre de transmission des messages.

Une définition pour les prédicats $empty$, $value$, $insert$, $shift$ et $delete$ se trouve en [GAR 89] .

On définit le prédicat :

$$choix(M, E, N) \quad :- \quad value(M, E, N), N \neq -1.$$

Ce processus est caractérisé par le programme logique Π_{MEDIUM_1}:

$$
\begin{array}{lcl}
medium_1([\,], M) & :- & empty(M). \\
medium_1([sdt(N)|L], M_1) & :- & medium_1(L, M), insert(M, N, M_1). \\
medium_1([sdt(N)|L], M_1) & :- & medium_1(L, M), shift(M, N, M_1). \\
medium_1([rdt(N)|L], M_1) & :- & medium_1(L, M), choix(M, E, N), delete(M, E, M_1). \\
medium_1([rdt(N)|L], M) & :- & medium_1(L, M), choix(M, E, N).
\end{array}
$$

L'automate AUT_{MEDIUM_1} est représenté figure A.4 .

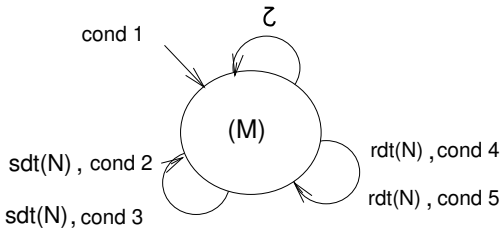


FIG. A.4 - *Processus* $MEDIUM_1$

A.4 Processus $MEDIUM_2$

Le comportement de $MEDIUM_2$ est analogue à celui de $MEDIUM_1$. La seule différence réside dans le nom des actions qui sont exécutées.

Ce processus est caractérisé par le programme Π_{MEDIUM_2}:

$$
\begin{array}{lcl}
medium_2([\,], M) & :- & empty(M). \\
medium_2([rack(N)|L], M_1) & :- & medium_2(L, M), insert(M, N, M_1). \\
medium_2([rack(N)|L], M_1) & :- & medium_2(L, M), shift(M, N, M_1). \\
medium_2([sack(N)|L], M_1) & :- & medium_2(L, M), choix(M, E, N), delete(M, E, M_1). \\
medium_2([sack(N)|L], M) & :- & medium_2(L, M), choix(M, E, N).
\end{array}
$$

L'automate AUT_{MEDIUM_2} est représenté figure A.5 .

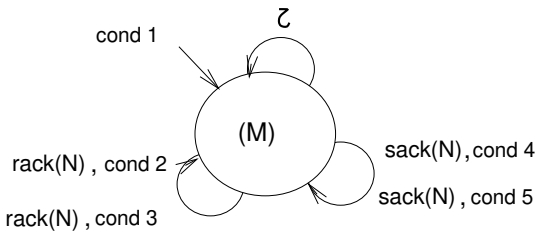


FIG. A.5 - *Processus* $MEDIUM_2$

A.5 Composition parallèle

L'automate généralisé AUT_{slide} associé au Protocole de la fenêtre glissante s'obtient en appliquant les opérations de composition parallèle aux automates associés à $SENDER$, $RECEIVER$, $MEDIUM_1$ et $MEDIUM_2$. Les actions synchronisées sont celles qui sont partagées par les différentes entités (voir figure A.1).

On engendre ainsi le programme-composition Π_{slide}:

$syst([\,], wait(B_1, 0), wait(B_2, R), M_1, M_2)$:- $empty(M_1), empty(M_2), reset(R), B_1 = 0, B_2 = 0.$

$syst([put(N)|L], readysend(B, T), U, M_1, M_2)$:- $syst(L, wait(B, T), U, M_1, M_2), inf(T, TWS), plusmodulo(B, T, N).$

$syst([sack(N)|L], wait(B, S), U, M_1, M_3)$:- $syst(L, wait(B, S), U, M_1, M_2), notwindow(N, B, S), delete(M_2, E, M_3), choix(M_2, E, N).$

$syst([sack(N)|L], wait(B, S), U, M_1, M_2)$:- $syst(L, wait(B, S), U, M_1, M_2), notwindow(N, B, S), choix(M_2, E, N).$

$syst([sack(N)|L], wait(N+1, S_1), U, M_1, M_3)$:- $syst(L, wait(B, T), U, M_1, M_2), window(N, B, T), plusmodulo(T - S_1, B, N+1), delete(M_2, E, M_3), choix(M_2, E, N).$

$syst([sack(N)|L], wait(N+1, S_1), U, M_1, M_2)$:- $syst(L, wait(B, T), U, M_1, M_2), window(N, B, T), plusmodulo(T - S_1, B, N+1), choix(M_2, E, N).$

$syst([\tau|L], repeat(N, S_1, B, S), U, M_1, M_2)$:- $syst(L, wait(B, S), U, M_1, M_2), window(N, B, S), plusmodulo(S - S_1, B, N).$

$syst([sdt(N)|L], wait(B, S+1), U, M_3, M_2)$:- $syst(L, readysend(B, S), U, M_1, M_2), plusmodulo(B, S, N), insert(M_1, N, M_3).$

$syst([sdt(N)|L], wait(B, S+1), U, M_3, M_2)$:- $syst(L, readysend(B, S), U, M_1, M_2), plusmodulo(B, S, N), shift(M_1, N, M_3).$

$syst([sdt(N)|L], repeat(N+1, S_1+1, B, S), U, M_3, M_2)$:- $syst(L, repeat(B, S, B, S), U, M_1, M_2), inf(1, S), insert(M_1, B, M_3).$

$syst([sdt(N)|L], repeat(N+1, S_1+1, B, S), U, M_3, M_2)$:- $syst(L, repeat(B, S, B, S), U, M_1, M_2), inf(1, S), shift(M_1, B, M_3).$

$syst([sdt(N)|L], wait(B, S), U, M_3, M_2)$:- $syst(L, repeat(N, 1, B, S), U, M_1, M_2), insert(M_1, B, M_3).$

$syst([sdt(N)|L], wait(B, S), U, M_3, M_2)$:- $syst(L, repeat(N, 1, B, S), U, M_1, M_2), shift(M_1, B, M_3).$

$syst([get(N)|L], U, deliver(N+1, R_1), M_1, M_2)$:- $syst(L, U, deliver(N, R), M_1, M_2), test(R, N), unset(R, N, R_1).$

$syst([rdt(N)|L], U, deliver(B, R_1), M_3, M_2)$:- $syst(L, U, wait(B, R), M_1, M_2), notest(R, N), window(N, B, RWS), set(R, N, R_1), choix(M_1, E, N), delete(M_1, E, M_3).$

$syst([rdt(N)|L], U, deliver(B, R_1), M_1, M_2)$:- $syst(L, U, wait(B, R), M_1, M_2), notest(R, N), window(N, B, RWS), set(R, N, R_1), choix(M_1, E, N).$

$syst([rdt(N)|L], U, readyrack(B, R), M3, M_2)$:- $syst(L, U, wait(B, R), M_1, M_2), test(R, N), choix(M_1, E, N), delete(M_1, E, M_3).$

$syst([rdt(N)|L], U, readyrack(B, R), M_1, M_2)$:- $syst(L, U, wait(B, R), M_1, M_2), test(R, N), choix(M_1, E, N).$

$syst([rdt(N)|L], U, readyrack(B, R), M3, M_2)$:- $syst(L, U, wait(B, R), M_1, M_2), choix(M_1, E, N), notwindow(N, B, RWS), delete(M_1, E, M_3).$

$syst([rdt(N)|L], U, readyrack(B, R), M_1, M_2)$:- $syst(L, U, wait(B, R), M_1, M_2), choix(M_1, E, N) notwindow(N, B, RWS).$

$syst([rack(N)|L], U, wait(B, R), M_1, M_3)$:- $syst(L, U, deliver(B, R), M_1, M_2), notest(R, B), plusmodulo(N, 1, B), insert(M_2, N, M_3).$

$$syst([rack(N)|L], U, wait(B,R), M_1, M_3) \quad :- \quad syst(L, U, deliver(B,R), M_1, M_2), notest(R,B), plusmodulo(N,1,B), shift(M_2, N, M_3).$$

$$syst([rack(N)|L], U, wait(B,R), M_1, M_3) \quad :- \quad syst(L, U, readyrack(B,R), M_1, M_2), plusmodulo(N,1,B), insert(M_2, N, M_3).$$

$$syst([rack(N)|L], U, wait(B,R), M_1, M_3) \quad :- \quad syst(L, U, readyrack(B,R), M_1, M_2), plusmodulo(N,1,B), shift(M_2, N, M_3).$$

Bibliographie

[APT 82] Apt K. R. et Van Emden M. H., «Contributions to the Theory of Logic Programming», *J. ACM* 29, 1982, pp. 841-862.

[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.

[BAN 86b] Bancillon F. et Ramakrishnan R., «An Amateur's Introduction to Recursive Query Processing Strategies», *Proc. ACM Conf. on Management of Data*, Washington, 1986, pp. 16-52.

[BAU 89] Baudinet M., «Temporal Logic Programming is Complete and Expressive», *Proc. 16th ACM Symp. on Principles of Programming Languages*, Austin, 1989, pp. 267-280.

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

[BID 92] Bidoit N., *Bases de Données Déductives Présentation de DATALOG*, Armand Colin Editeur, 1992.

[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.

[BOU 91] Bouajjani A. et *al.*, « Safety Branching Time Semantics », *Proc. 18th International Colloquium Automata, Langages and Programming*, Berlin, 1991, p. 82-114.

[BRO 91] Brodsky A. et Sagiv Y., «Inference of Inequality Constraints in Logic Programs», *Proc. 10th ACM Symp. on Principles of Database Systems*, Denver, 1991, pp. 227-240.

[CHO 88] Chomicki J. et Imielinski T., «Temporal Deductive Databases and Infinite Objects», *Proc. 7th ACM Symp. on Principles of Database Systems*, Austin, 1988, pp. 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.

[COR 94] Corsini M. et Rauzy A., «Symbolic Model Cheching and Constraint Logic Programming : a Cross-Fertilization», *Proc. of the European Symposium on Programming ESOP'94*, 1994.

[COU 77] Cousot P. et Cousot R., «Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints», *Conference Record of the 4th ACM Symposium on Principles of Programming Languages*, Los Angeles, 1977, pp. 238-252.

[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, pp. 84-96.

[DSS 85] Dssouli R. et Bochmmann G., «Error Detection with Multiple Observers», *Proc. of the IFIP WG 6.1 5th Intl. Workshop on Protocol Specification, Testing and Verification*, North-Holland, 1985, Toulouse-Moissac, pp.483-494.

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

[END 76] Van Emden M. H. et Kowalski R. A., «The Semantics of Predicate Logic as a Programming Language», *J. ACM* 23:4, 1976, pp. 733-742.

[FRI 92] Fribourg L., «Mixing List Recursion and Arithmetic», *Proc. 7th IEEE Symp. on Logic in Computer Science*, Santa Cruz, 1992, pp. 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, Juin 1992.

[FRI 93] Fribourg L. et Veloso Peixoto M., «Concurrent Constraint Automata», Rapport de Recherche, LIENS 93-10, May 1993.

[FRI 94] Fribourg L. et Veloso Peixoto M., «Bottom-up Evaluation of Datalog Programs with Arithmetic Constraints: The Case of 3 Recursive Rules», Rapport de Recherche, LIENS 94-09, Juillet 1994.

[FRI 94b] Fribourg L. et Veloso Peixoto M., «Bottom-up Evaluation of Datalog Programs with Arithmetic Constraints», Proc. CADE-12, LNAI 814, pp. 311-325, Juin 1994.

[FRI 94c] Fribourg L. et Olsen H., «Direct, Dual and Contrapositive Proofs by Induction», *Pre-Proc. LOPSTR 94*, Pisa, 1994.

[FRI 94d] Fribourg L. et Veloso Peixoto M., «Automates Concurrents à Contraintes », à paraître dans *Technique et Science Informatiques*.

[FRI 94e] Fribourg L. et Veloso Peixoto M., «Bottom-up Evaluation of Datalog Programs with Incremental Arguments and Linear Arithmetic Constraints», à paraître dans *Proc. Post-ILPS 94 Workshop on Constraints and Databases*, Ithaca, 1994.

[GAR 89] Garavel H., «Compilation et Vérification de Programmes LOTOS», *Thèse Université Joseph Fourier - Grenoble I*, Novembre 1989.

[GAR 89a] Garavel H., «Protocole de la Fenêtre Glissante», manuscript non publié, 1989.

[GEL 90] Van Gelder A., «Deriving Constraints among Argument Sizes in Logic Programs», *Proc. 9th ACM Symp. on Principles of Database Systems*, Nashville, 1990, pp. 47-60.

[GRO 83] Groz R., Jard C. et Pitie J., «Simulation as an Approach to Distributed Algorithms Validation », *Proc. European Workshop Distributed Computing Systems*, 1983.

[HAL 93] Halbwachs N., « Delay Analysis in Synchronous 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 L. et Lassez J. L., «Constraint Logic Programming», *Proc. 14th ACM Symp. on Principles of Programming Languages*, 1987, pp. 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.

[KUP 90] Kuper G., «On the Expressive Power of the Relational Calculus with Arithmetic Constraints», *Proc. 3rd International Conference on Database Theory*, Paris, 1990, pp. 203-313.

[KAN 90] Kanellakis P., Kuper G. et Revesz P., «Constraint Query Languages», *Proc. 9th ACM Symp. on Principles of Database Systems*, Nashville, 1990, pp. 299-313.

[KEM 93] Kemp D. B. et Stuckey P. J., «Analysis Based Constraint Query Optimization», *Proc. 10th Intl. Conf. on Logic Programming*, Budapest, The MIT Press, 1993, pp. 666-682.

[KAB 90] Kabanza F., Stevenne J. M. et Wolper P., «Handling Infinite Temporal Data», *Proc. 9th ACM Symp. on Principles of Database Systems*, Nashville, 1990, pp. 392-403.

[LAM 90] Lam S. S. et Shankar A., «A Relational Notation for State Transition Systems», *IEEE Transactions on Software Engineering*, Vol. 16, No. 7, juillet 1990, p. 755-775.

[LAS 90] Lassez J. L., «Querying Constraints», *Proc. 9th ACM Symp. on Principles of Database Systems*, Nashville, 1990, pp. 288-298.

[LAS 90b] Lassez J. L., «Parametric Queries, linear constraints and variable elimination», *Proc. Conference on Design and Implementation of Symbolic Computer*, LNCS 429, 1990, pp. 164-173.

[LEV 92] Levy A. et Sagiv Y., «Constraints and Redundancy in Datalog», *Proc. 11th ACM Symp. on Principles of Database Systems*, San Diego, 1992, pp. 67-80.

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

[MAR 93] Marriot K. et Stuckey P. J., «The 3 R's of Optimizing Constraint Logic Programs: Refinement, Removal and Reordering», *Proc. 20th ACM Symp. on Principles of Programming Languages*, Charleston, 1993, pp. 334-344.

[MEY 91] Van der Meyden R., «Reasoning with Recursive Relations: Negation, Inequality and Linear Order (Extended Abstract)», *Proc. ILPS Workshop on Deductive Databases*, San Diego, 1991.

[MIN 67] MINSKY M., *Computation: Finite and Infite Machines*, Prentece-Hall, 1967.

[MIS 82] MISRA J. et GRIES D., «Finding Repeated Elements», *Science of Computer Programming 2*, 1982, pp. 143-152.

[PLU 90] PLÜMER L., «Termination Proofs for Logic Programs based on Predicate Inequalities»,*Proc. 7th International Conference on Logic Programming*, Jerusalem, 1990, pp. 634-648.

[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 90] REVESZ P., «A Closed Form for Datalog Queries with Integer Order», *Proc. 3rd International Conference on Database Theory*, Paris, 1990, pp. 187-201.

[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).

[ROU 94] ROUTIER J.C., *Terminaison, Satisfiabilité, Pouvoir Calculatoire d'une Clause de Horn Binaire*, Thèse de Doctorat, LIFL, L'Université des Sciences et Technologies de Lille, 1994.

[ROY 90] ROY V. et SIMONE R., «Auto/Autograph » *Proc. of the 2nd Workshop on Computer-Aided Verification*, Rutgers, 1990, p. 477-491.

[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.

[SRI 92] Srivastava D., «Subsumption in Constraint Query Languages with Linear Arithmetic Constraints», *Proc. 2nd International Symp. on Artificial Intelligence and Mathematics*, Fort Lauderdale, 1992.

[SRI 92b] Srivastava D. et Ramakrishnan R., «Pushing Constraint Selections», *Proc. 11th ACM Symp. on Principles of Database Systems*, San Diego, 1992, pp. 301-315.

[ULL 88] Ullman J. D. et Van Gelder A., «Efficient Test for Top-Down Termination», *J. ACM* 35:2, 1988, pp. 345-373.

Table des matières

Remerciements

Je remercie Guy Cousineau de l'honneur qu'il me fait en présidant le jury de cette thèse.

Je remercie Gérard Ferrand et Phillipe Devienne d'avoir accepté d'être rapporteurs de ma thèse. Je les remercie pour leurs remarques, critiques et suggestions.

Je voudrais très specialement remercier mon directeur de thèse, Laurent Fribourg, tant pour la confiance qu'il m'a toujours montré, que pour son esprit critique et constructif dont mon travail a grandement profité.

Je suis aussi très reconnaissant à Irène Guessarian, avec qui j'ai travaillé pendant la première année de ma formation doctarale. Son constant appui et ses encouragements ont été un support essentiel pour l'évolution de mon travail.

Mes remerciements vont également à Pierre Deransart, Hubert Garavel et Jan Maluszyński qui m'ont fait l'honneur de participer au jury.