Accélération abstraite pour l'amélioration de la précision en Analyse des Relations Linéaires

Laure Danthony,gonnord

UNIVERSITÉ JOSEPH FOURIER - GRENOBLE I
SCIENCES ET GÉOGRAPHIE

T H È S E

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ JOSEPH FOURIER

Discipline : « INFORMATIQUE »

préparée au laboratoire VERIMAG
dans le cadre de l'**École doctorale "MATHÉMATIQUES, SCIENCES ET TECHNOLOGIES DE L'INFORMATION"**

présentée et soutenue publiquement par

Laure DANTHONY, GONNORD

le 25 octobre 2007

Titre :

Accélération abstraite pour l'amélioration de la précision en Analyse des Relations Linéaires

Directeur de thèse :

Nicolas Halbwachs

JURY

M. Yves Ledru	Président, Professeur à Université Joseph Fourier
M. François Irigoin	Rapporteur, Maître de recherche à l'École des Mines de Paris
M. Philippe Schnoebelen	Rapporteur, Directeur de recherche CNRS, LSV, ENS Cachan
M. Bertrand Jeannet	Examinateur, Chargé de Recherche INRIA, Inria Rhône Alpes
M. Thomas Reps	Examinateur, Professeur à l'Université du Wisconstin
M. Nicolas Halbwachs	Directeur de thèse, Directeur de Recherche CNRS, Verimag, Grenoble

Remerciements

J'ai bénéficié lors de la préparation de cette thèse de l'aide de nombreuses personnes. Par ces quelques lignes, j'aimerais leur exprimer ma profonde gratitude.

En premier lieu, je tiens à remercier chaleureusement les membres du jury :
Yves Ledru, professeur à l'Université Joseph Fourier, membre du laboratoire LIG, pour avoir accepté d'être président de mon jury de thèse.
François Irigoin, professeur à l'école des Mines de Paris, et *Philippe Schoebelen*, directeur de recherche CNRS/LSV, pour avoir accepté d'être rapporteurs de ce manuscrit. Leur relecture minutieuse en grandement amélioré la qualité. Je tiens à remercier plus particulièrement le premier pour les nombreux échanges notamment dans le cadre du projet APRON et le second pour m'avoir ouvert au monde merveilleux de la vérification de logiciel en me proposant un sujet de recherche en Licence (L3 maintenant).
Bertrand Jeannet, chargé de recherche à l'Inria Rhône Alpes, pour avoir accepté d'être membre de ce jury. Merci pour les nombreuses discussions tout au long de cette thèse, formelles et moins formelles, et merci pour le support technique qui m'a permis de réaliser mon outil en réutilisant une partie de son code.
Thomas Reps, pour avoir également accepté de faire partie de mon jury.
Nicolas Halbwachs, directeur de recherche CNRS/Vérimag, pour avoir accepté la lourde tâche d'encadrer ma thèse. Merci pour (toutes !) ces années de patience, de bonne humeur, de discussions et de conseils précieux. Merci pour les relectures nombreuses du manuscrit et surtout merci pour la disponibilité dont il a fait preuve tout au long de ce travail, même lorsque sa nouvelle charge de directeur du laboratoire est venu manger son temps si précieux.

Je tiens également à remercier toutes les personnes qui ont rendu possible la réalisation de ce travail :
Joseph Sifakis, directeur de recherche au CNRS, qui m'a accueilli au sein du laboratoire Vérimag lors de mon stage de maîtrise (M1). Merci à tous les membres du personnel administratif et technique, pour leur efficacité au quotidien qui permet d'avancer sereinement sans trop se préoccuper des problèmes administratifs, de machine, de réseau.
Tous les membres de l'équipe synchrone avec qui j'ai eu grand plaisir à travailler. Merci pour les remarques constructives lors des réunions et séminaires d'équipe, pour la bonne humeur au coin café, pour le désormais célèbre mot croisé de 13h qui manque terriblement lorsqu'on est plus à Vérimag. Un merci particulièrement appuyé à *Fabienne*, pour les nombreux échanges au sujet nos enseignement communs, qui m'ont permis d'avancer dans ma pratique pédagogique, et sur d'autres sujets aussi, et à *Karine*, pour le partage d'expériences.
Mes collègues du célèbre bureau 39, les anciens et les nouveaux, *Yvan*, *Ismaïl*, *Matthieu* (merci pour les conseils Emacs, le soutien en fin de thèse, la participation au pot), *Romain*, *Sophie* (mention spéciale pour son dynamisme et l'institution du Veripicnic), *Claude*, dans le rôle de mouche du coche, et les derniers arrivés, qui ont dû survivre à la dure période de rédaction, *Yussef* et *Jérôme*. Ces personnes sont plus que des collègues maintenant, et je les en remercie.
Mathias, autre thésard de Nicolas, pour les réunions de thèse enrichissantes, ses conseils, son soutien, et pour avoir fait diversion auprès « du chef » aux bons moments !

Tous mes autres collègues de Vérimag, thésards, postdocs, permanents, pour tous les échanges sérieux et moins sérieux que nous avons eu ensemble.

Enfin, je voudrais remercier mes amis proches et ma famille :
Mes parents Martine et Jacques, qui ont su m'apprendre le goût de l'effort, m'encourager tout au long de mes études. Merci pour l'amour que vous me portez.
Ma petite sœur Audrey, pour les moments de complicité notamment lors de notre collocation grenobloise.
Mes amis proches et moins proches géographiquement, mais qui savent être là aux bons moments, *Laurent*, *Xavier*, *Isabelle* (merci pour le logo !) et les autres. Merci pour votre soutien lors de la rédaction !

Merci à *Stéphane*, enfin et surtout, d'avoir accepté mes humeurs, mes doutes, d'avoir su relancer ma motivation lors de mes périodes de doute, d'avoir géré l'intendance lors des moments d'intense activité scientifique et de rédaction. Merci pour le pot, aussi. Et merci pour le reste, merci pour *Marine*, notre plus grande réalisation !

Table des matières

Table des figures

Chapitre 1

Introduction

1.1 Contexte et motivation

On s'intéresse dans cette thèse à la vérification de propriétés de programmes à états infinis, pour lesquels les problèmes de vérification sont pour la plupart indécidables. Ce problème peut être abordé principalement de deux façons :

- Certaines classes de problèmes ont été identifiées comme décidables. L'inconvénient est que les langages d'expression des propriétés et/ou de modélisation des systèmes sont peu expressifs, et on ne peut vérifier que des systèmes « simples » . On peut aussi se ramener à ces classes en utilisant des abstractions, mais ces abstractions peuvent se révéler insuffisantes.
- Face à un problème intrinsèquement indécidable (ou simplement trop coûteux à résoudre), on peut se contenter d'une vérification *conservative* : seule une réponse positive est significative, c'est le cas en particulier des méthodes d'Interprétation Abstraite ([CC77]).

1.2 Cadre théorique de la thèse, objectifs

Si l'on se restreint aux propriétés de sûreté, le problème de la vérification de systèmes se ramène presque toujours à caractériser, exactement ou approximativement, l'ensemble des états atteignables du programme, ensemble qui est la solution d'une équation de point-fixe : l'ensemble cherché est la limite $F^*(X_0)$ d'une suite $X_0, ...X_n, ...$ d'ensembles, avec $X_{n+1} = F(X_n)$, où F est une fonction croissante. Classiquement, ce calcul pose le problème de la représentation des ensembles d'états X_n, et celui de la convergence de la suite, qui est en général infinie.

Dans cette thèse, on s'intéresse plus particulièrement aux propriétés numériques. Deux approches différentes ont été étudiées : des résultats récents ont été obtenus par les méthodes d'*accélération* : selon la forme de la fonction F, on peut calculer directement F^*. Les méthodes d'interprétation abstraite s'attachent quant à elles à calculer une approximation supérieure de la limite $F^*(X_0)$ (l'application d'un opérateur d'*élargissement* permet d'extrapoler à partir des premiers termes de la séquence cette approximation supérieure).

Au laboratoire Verimag, les travaux précédents au sein de l'équipe synchrone ont principalement porté sur une Interprétation Abstraite fondée sur une approximation des ensembles

d'états numériques par des polyèdres convexes, sur lesquels un opérateur d'élargissement a été introduit (cet opérateur garantissant la terminaison de l'analyse). Cette analyse s'appelle l'Analyse des Relations Linéaires, et a été introduite dans [Hal79a]. Un des problèmes posé par l'élargissement est celui de l'amélioration de la précision, lorsque la vérification échoue : une solution classique consiste alors à retarder l'application de l'élargissement, c'est à dire à baser l'extrapolation sur un plus grand nombre de termes. Malheureusement, il arrive fréquemment que le nombre de termes nécessaires à une extrapolation satisfaisante dépende directement de constantes numériques apparaissant dans le programme. D'une part, les performances peuvent en être profondément pénalisées, et d'autre part, la méthode est inapplicable si les constantes concernées sont des paramètres symboliques du problème.

Ces remarques nous ont amené à regarder plus précisément des (classes de) programmes où (la précision de) l'élargissement peut être amélioré de manière évidente (c'est-à-dire que l'on peut converger plus vite vers un point fixe plus précis). Pour cela, les méthodes d'accélérations proposées par [BW94, WB98, CJ98, FS00b, BFLP03] sont une source d'inspiration. Ces travaux consistent à identifier certaines catégories de boucles dont l'effet peut être calculé exactement. Plus précisément, l'effet d'une boucle simple, gardée par une condition linéaire sur des variables *entières* et consistant en l'incrémentation/décrémentation de ces variables, peut être calculée de façon exacte sous la forme d'une formule de Presburger. Ces méthodes ont l'avantage d'être exactes, mais elles sont restreintes à certaines classes de programmes, les programmes « plats » (c'est-à-dire sans boucles imbriquées). De plus, ces calculs exacts ont une complexité théorique (et surtout *pratique*) très élevée (généralement triple-exponentielle). Les applications de ces méthodes sont donc limitées.

Dans cette thèse, nous proposons une approche de technique de vérification qui combine ces deux approches. Nous pensons que l'approximation est une clé pour l'obtention de performances compatibles avec le traitement de programmes de taille réelle. Nous désirons donc introduire des techniques d'accélération au sein de la technique d'Analyse des Relations Linéaires. Pour cela, nous identifions certains types de boucles pour lesquelles l'enveloppe convexe des états atteignables (ou une approximation supérieure de cet ensemble), que nous appellerons *accélération abstraite* est calculable à faible coût. Les algorithmes d'Analyse des relations linéaires seront alors modifiés pour prendre en compte ces résultats, tout en garantissant la terminaison.

1.3 Plan

La première partie de cette thèse est un état de l'art détaillé des deux approches que nous désirons combiner.

Le chapitre 2 introduit les notions générales de vérification automatique de programme. Nous présenterons notamment la théorie générale de l'interprétation abstraite et son application à la vérification de propriétés numériques de programme via l'Analyse des Relations Linéaires. Nous terminerons par un exemple introductif.

Le chapitre 3 décrit les techniques d'accélération qui s'attachent à calculer précisément l'effet des applications itérées des relations/fonctions de transitions. Nous verrons que ces méthodes ont l'inconvénient de s'appliquer à un nombre restreint de programmes.

Le chapitre 4 présente les améliorations apportées à l'Analyse des Relations Linéaires dans divers travaux. Nous verrons que ces améliorations ont chacune des avantages, mais elles ne prennent pas en compte les fonctions de transformation de boucles.

La deuxième partie de cette thèse décrit nos contributions.

Le chapitre 5 présente nos résultats théoriques d'accélération d'une boucle simple. Nous introduisons une notion nouvelle d'*accélération abstraite* pour certains types de transformations, et nous donnons des algorithmes simples pour le calcul de cette accélération.

Le chapitre 6 présente nos résultats théoriques d'accélération simultanées de plusieurs boucles de translations. Nous verrons que dans certains cas nous traitons des boucles dont la combinaison n'est calculable exactement par aucun algorithme connu.

Le chapitre 7 présente des algorithmes pour le calcul de l'effet de plusieurs boucles de remise à constante et de translations simultanées.

Le chapitre 8 décrit l'implémentation de nos techniques d'accélération dans un nouvel outil de vérification nommé ASPIC. Nous décrivons aussi comment cet outil se place dans la chaîne d'outils de Vérimag autour du langage LUSTRE.

Le chapitre 9 présente les résultats expérimentaux obtenus .

Enfin le chapitre 10 conclut le travail et présente quelques perspectives.

Première partie

Principes de l'Analyse des Relations Linéaires et des méthodes d'Accélération

Chapitre 2

Vérification de Propriétés par Analyse des Relations Linéaires

Dans ce chapitre, nous introduisons le cadre d'études retenu, l'Analyse des Relations Linéaires. Dans un premier temps, nous décrirons le modèle de programmes utilisé (les automates interprétés), dont nous préciserons une sémantique en termes de systèmes de transitions. Nous définirons ensuite les propriétés de tels systèmes, puis nous introduirons le cadre général de vérification qui est l'Analyse des Relations Linéaires. Nous terminerons par un exemple complet de vérification de recherche d'invariants par l'Analyse des Relations Linéaires, puis par l'exemple motivant notre étude, l'exemple classique de la chaudière.

2.1 Systèmes de transitions discrets, automates interprétés

2.1.1 Les systèmes de transitions discrets

Classiquement, la représentation choisie pour un programme au plus bas niveau sera un système de transitions, dont nous rappelons ici les principales définitions et notations.

Définition 1 — Système de transitions

> Un *système de transitions* est un triplet $(Q, \rightarrow, Q_{init})$ où Q est un ensemble d'états. $Q_{init} \subseteq Q$ est un sous-ensemble d'états appelés états *initiaux* et $\rightarrow \subseteq Q \times Q$ est une relation binaire sur Q appelée *relation de transition*. On note généralement $q \rightarrow q'$ le fait que $(q, q') \in \rightarrow$. On dit alors que q' est un *successeur* de q et q un *prédécesseur* de q'.

Définition 2 — Ensembles Post et Pre

> Soit $X \subseteq Q$ un sous ensemble d'états. On note $Pre(X) = \{q \in Q \mid \exists q' \in X, q \rightarrow q'\}$ l'ensemble des états prédécesseurs d'un état de X.
> De même, $Post(X) = \{q' \in Q \mid \exists q \in X, q \rightarrow q'\}$ désigne l'ensemble des (états) successeurs d'un état de X.

L'ensemble des états accessibles $Post^*(X) = \bigcup_{n \in \mathbb{N}} Post^n(X)$ est noté $acc(X)$.

De même on définit $Coacc(X) = Pre^*(X)$ l'ensemble des états *coaccessibles* à partir de X.

2.1.2 Les automates interprétés et leur sémantique en termes de systèmes de transitions discrets

Les systèmes de transitions que nous venons d'introduire sont d'assez bas niveau, et nous avons besoin de description d'un plus haut niveau. Comme nous ne voulons pas nous restreindre à un langage particulier, nous choisissons un modèle relativement général, les automates interprétés, dont nous allons donner une définition et une sémantique en termes de systèmes de transitions discrets.

Syntaxe Soit Var un ensemble fini fixé de n variables typées (par exemple des variables entières, booléennes). Une valuation est une fonction associant à chaque variable une valeur possible de son type. On note Val l'ensemble des valuations sur Var.

Définition 3 — Commande

Une *commande* est un couple (g, a) avec g une fonction de Val dans $\mathbb{B} = \{\text{Vrai}, \text{Faux}\}$ (*garde*) et a une fonction de Val dans Val (*action*).

Définition 4 — Automate interprété

Un *automate interprété* est un triplet $(K, \text{Trans}, k_{init})$ où K est un ensemble fini (de points de contrôle), $k_{init} \in K$ est le point de contrôle initial, et Trans est un ensemble fini de *transitions*, c'est à dire un ensemble de triplets (k', c, k'), avec $k, k' \in K$ et c une commande.

Exemple 2.1 *Voici un exemple d'automate interprété :*

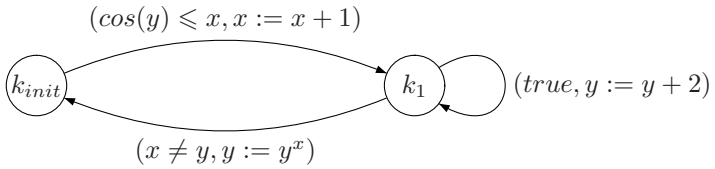


Fig. 2.1 – Un automate interprété

Sémantique Donnons maintenant une sémantique en terme de systèmes de transitions à ces automates interprétés.

Soit $(K, \text{Trans}, k_{init})$ un automate interprété. On lui associe un système de transitions $(Q, \rightarrow, Q_{init})$ de la façon suivante :

- $Q = K \times \text{Val}$: un état est un couple formé d'un point de contrôle $k \in K$ et d'une valuation.
- $Q_{init} = \{k_{init}\} \times \text{Val}$: les états initiaux ont leur première composante égale à k_{init}.
- La relation de transition : $(k, v) \rightarrow (k', v')$ si et seulement si il existe une transition $(k, c, k') \in \text{Trans}$ avec $c = (g, a)$, et vérifiant $g(v) = \text{Vrai}$ et $a(v) = v'$.

Maintenant que nous disposons d'un modèle symbolique pour nos programmes, nous allons présenter le type de propriétés que nous voulons vérifier, et quelle méthodologie nous allons utiliser. C'est l'objet de la section suivante.

2.2 Vérification de propriétés de sûreté

On distingue deux principales classes de propriétés sur les systèmes de transitions ([Lam77])

- **Les propriétés de sûreté** expriment que « quelque chose de mauvais n'arrivera jamais ». Par exemple il s'agit de prouver que le programme ne réalise jamais de dépassement arithmétique, qu'il n'y aura jamais d'accès à une case en dehors d'un tableau, ou bien encore que la ressource demandée sera disponible dans un délai fixé à l'avance.
- **Les propriétés de vivacité** spécifient que quelque chose de « bon » se produit immanquablement durant l'exécution du programme. La garantie de service ainsi que la terminaison du système sont de telles propriétés.

Ces deux classes sont distinguées parce que leur vérification fait appel à des techniques différentes : dans le premier cas, la violation d'une propriété peut se détecter à l'aide d'une exécution finie du système, alors que ce n'est pas le cas dans le second. Dans la suite, nous ne nous intéresserons qu'aux propriétés de sûreté, qui, dans le cas des systèmes de transitions, peuvent se ramener (*via* l'utilisation d'observateurs, voir le chapitre 8, section 8.2, page 105) à des propriétés d'atteignabilité de certains états dit d'erreur. Nous sommes donc ramenés à calculer des ensembles d'états atteignables. Différentes méthodes de vérification sont alors possibles.

2.2.1 Vérification par Model Checking

Dans certains cas, l'espace d'états du programme peut être fini, ou bien on peut s'y ramener par abstractions (en ignorant certains aspects) à partir d'un espace infini ([GL93], [CGL94]). Cet espace peut donc être calculé en un temps fini en appliquant itérativement les commandes de l'automate. À la fin de la procédure (qui termine puisque l'ensemble d'états est fini, à condition de ne pas traiter deux fois le même état), l'ensemble calculé est exactement l'ensemble des états atteignables. Il ne reste plus alors qu'à réaliser l'intersection avec les états dits « mauvais ».

2.2.2 Vérification par accélération

Dans certains cas, l'espace d'états est infini mais représentable et surtout calculable par des techniques exactes nommées *accélérations*. Informellement, elles consistent à calculer en un nombre fini d'itérations l'ensemble exact $acc(Q_{init})$ avec Q_{init} l'ensemble initial. Un état de l'art de ces travaux se trouve au chapitre 3. Nous verrons que le principal inconvénient de ces méthodes réside dans l'ensemble restreint de programmes qu'elles peuvent effectivement vérifier.

2.2.3 Vérification par (sur-)approximation

Une troisième solution consiste à calculer l'espace d'états accessibles de manière approchée. L'idée est de calculer un sur-ensemble Q' de $acc(Q_{init})$. Si ce sur-ensemble n'intersecte pas l'ensemble des états d'erreur (sur le dessin, Q_{err}), alors la propriété est certainement satisfaite. En revanche, si l'intersection est non vide, alors on ne peut savoir si la propriété est violée ou si le calcul était trop grossier. On parle de vérification *conservative*.

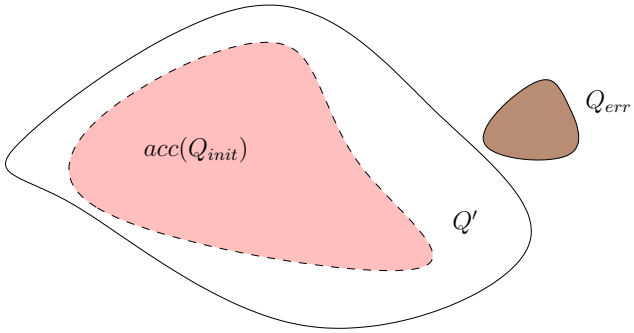


FIG. 2.2 – Vérification conservative

L'objet de la section suivante est d'introduire la méthode d'interprétation abstraite, qui va nous fournir un cadre général pour réaliser des analyses efficaces de programmes par sur-approximation des états atteignables.

2.3 Principes de l'interprétation abstraite

Pour introduire les notions d'interprétation abstraite, nous avons besoin de la définition d'un treillis :

DÉFINITION 5 — Treillis

> Un *treillis* est un ensemble muni d'une relation d'ordre, tel que pour tout couple d'éléments il existe une borne supérieure et une borne inférieure
> Un treillis est dit *complet* si tout sous-ensemble (fini ou infini) du treillis possède une borne supérieure et une borne inférieure.

2.3.1 Introduction

L'ensemble $acc(Q_{init})$ satisfait l'équation de point fixe suivante :

$$acc(Q_{init}) = Q_{init} \cup Post(acc(Q_{init}))$$

On peut tirer parti de la structure de contrôle de l'automate interprété pour décomposer le problème en un système d'équations de points fixes. Ainsi, pour tout point de contrôle $k \in K$, on va noter $\mathcal{A}_k$ l'ensemble des valuations accessibles au point k :

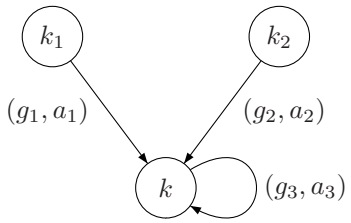
$$\mathcal{A}_k = \{v \mid (k, v) \in acc(Q_{init})\}$$

On note $\mathcal{A} \cap g$ l'ensemble des valuations de $\mathcal{A}$ satisfaisant g et $a(\mathcal{A})$ l'ensemble des images des valuations de $\mathcal{A}$ par l'action a. Ainsi, on récupère le système d'équations suivant :

$$\mathcal{A}_{k_{init}} = \text{Val} \quad , \quad \forall k \neq k_{init}, \quad \mathcal{A}_k = \bigcup_{(k',(g,a),k) \in \text{Trans}} a(\mathcal{A}_{k'} \cap g)$$

Informellement, les équations expriment que l'ensemble des valuations accessibles à un point de contrôle donné peut être déduit de l'ensemble des valuations accessibles à chacun de ses prédécesseurs, en appliquant les transitions correspondantes (en prenant en compte les gardes et les actions).

EXEMPLE 2.2 *Dans l'exemple ci-dessous :*



L'équation associée au nœud k est la suivante :

$$\mathcal{A}_k = a_1(\mathcal{A}_{k_1} \cap g_1) \cup a_2(\mathcal{A}_{k_2} \cap g_2) \cup a_3(\mathcal{A}_k \cap g_3)$$

REMARQUE 1 Si on parle des états coaccessibles, on obtient (en notant $\mathcal{C}_k$ l'ensemble des valuations coaccessibles à partir de l'état d'erreur k_{erreur} et a^{-1} l'action « inverse » de a) :

$$\mathcal{A}_{k_{err}} = \text{Val} \quad , \quad \forall k \neq k_{err}, \mathcal{A}_k = \bigcup_{(k,(g,a),k') \in \text{Trans}} a^{-1}(\mathcal{C}_{k'} \cap g)$$

On obtient donc un système d'équations mutuellement récurrentes, de la forme $\overrightarrow{X} = F(\overrightarrow{X})$ avec $X \in C$ et C un treillis complet (ici, l'ensemble des parties des vecteurs de $\mathbb{N}^n$, $\mathbb{Z}^n$ ou $\mathbb{Q}^n$, ordonné par l'inclusion, la borne supérieure étant l'union ensembliste, la borne inférieure l'intersection et le plus petit élément l'ensemble vide) et F est une fonction continue. Les théorèmes classiques de point fixe (Kleene) nous assurent alors l'existence du plus petit point fixe (lfp) donné par $\text{lfp}(F) = \bigcup_{n \geqslant 0} F^n(\emptyset)$.

Dans le cas général, la résolution de cette équation est impossible, le calcul des itérés $F(\emptyset)$, $F^2(\emptyset)$, ... posant deux types de problèmes :

- le domaine de calcul C peut être trop complexe : les ensembles d'états doivent pouvoir être représentés finiment, et les calculs (intersection, union, postconditions) doivent pouvoir être réalisés, ce qui est quelquefois impossible.
- le calcul itératif peut ne pas terminer.

L'article fondateur [CC77] propose une solution générale à ces deux problèmes, que nous allons développer dans la suite.

2.3.2 Notion de domaine abstrait

Le domaine concret des valeurs C est généralement trop complexe. Les éléments de ce domaine doivent être représentés finiment (et efficacement), et les opérations de calcul (et de décision de l'ordre) sur ce domaine doivent être effectives, ce qui n'est pas toujours le cas. On approxime alors le treillis complet $(C, \sqsubseteq, \sqcup, \sqcap, \top, \bot)$ (noté abusivement C) par un *treillis abstrait*, qui doit lui aussi être un treillis complet et que nous notons $(C^\sharp, \sqsubseteq^\sharp, \sqcup^\sharp, \sqcap^\sharp, \top^\sharp, \bot^\sharp)$ (abusivement, $C^\sharp$). La relation entre les deux treillis est formalisée par la notion suivante :

Définition 6 — Connexion de Galois

Une connexion de Galois entre C et $C^\sharp$ est un couple de fonctions $\alpha : C \to C^\sharp$ (abstraction) et $\gamma : C^\sharp \to C$ (concrétisation) vérifiant :

$$\forall x \in C, \forall y \in C^\sharp, \alpha(x) \sqsubseteq^\sharp y \iff x \sqsubseteq \gamma(y)$$

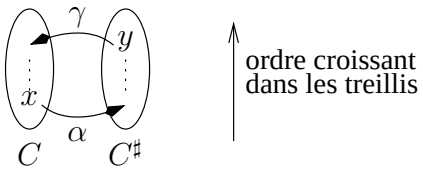


Fig. 2.3 – Connexion de Galois

L'existence d'une connexion de Galois entre les deux treillis C et $C^\sharp$ est une condition suffisante pour pouvoir appliquer le théorème suivant :

Théorème 1 *Si C et $C^\sharp$ sont reliés par une connexion de Galois (α, γ), si F est une fonction continue de C dans lui-même, si G est une fonction continue de $C^\sharp$ dans lui même, alors :*

$$\textit{si } \forall x \in C, \alpha(F(x)) \sqsubseteq^\sharp G(\alpha(x)), \textit{ alors } \mathit{lfp}(F) \sqsubseteq \gamma(\mathit{lfp}(G))$$

En résumé, pour calculer une sur-approximation du plus petit point fixe de la fonction concrète F (supposée continue, ce qui ne pose pas de problème dans nos analyses), il suffit de choisir une fonction abstraite G, vérifiant $\forall x \in C, \alpha(F(x)) \sqsubseteq^\sharp G(\alpha(x))$, de calculer le plus petit point fixe de G dans le treillis abstrait, puis d'appliquer la fonction γ à ce résultat.

2.3.3 Construction de la fonction abstraite

Le meilleur choix en terme de précision pour la fonction G, $\alpha \circ F \circ \gamma$, n'est en général pas calculable (à cause de la complexité des opérations sur le treillis concret C notamment), nous allons donc chercher à construire automatiquement G à partir de l'automate interprété à analyser (d'une manière similaire à celle de F, mais cette fois à l'intérieur du domaine abstrait $C^\sharp$). Il s'agit donc de trouver :

- une interprétation des gardes dans le treillis abstrait, c'est-à-dire un moyen effectif d'associer à g une valeur abstraite $g^\sharp$ vérifiant $g \sqsubseteq \gamma(g^\sharp)$;
- une interprétation des actions, c'est-à-dire un moyen effectif d'associer à toute action a une fonction abstraite $a^\sharp$ telle que $\forall x \in C, a(x) \sqsubseteq \gamma(a^\sharp(\alpha(x)))$.

Ensuite, on est ramené à calculer le plus petit point fixe du système d'équations récurrentes dans $C^\sharp$:

$$\mathcal{A}^\sharp_{k_{init}} = \top^\sharp \quad , \quad \forall k \neq k_{init}, \quad \mathcal{A}_k = \bigsqcup^\sharp_{(k',(g,a),k) \in \mathrm{Trans}} a^\sharp(\mathcal{A}^\sharp_{k'} \sqcap^\sharp g^\sharp)$$

2.3.4 Résolution du système abstrait

Dans le cas où le treillis abstrait considéré est de profondeur finie (*i.e.* toute chaîne strictement croissante est finie), le calcul itératif du point fixe précédent converge en un nombre fini d'itérations. Cependant, il peut s'avérer intéressant en terme de précision des analyses de se placer dans un treillis plus riche mais de profondeur infinie ([CC92a]). Dans ce cas, l'idée est d'extrapoler la limite des suites infinies à partir de ses premiers termes ([CC76]). Dans le cas d'une suite croissante, on utilise un opérateur d'élargissement (en anglais, *widening*) :

Définition 7 — Opérateur d'élargissement

Un opérateur d'élargissement est une fonction $\nabla : C^\sharp \times C^\sharp \to C^\sharp$ satisfaisant les deux propriétés suivantes :

1. $\forall y_1, y_2 \in C^\sharp, (y_1 \sqcup^\sharp y_2) \sqsubseteq^\sharp (y_1 \nabla y_2)$,
2. Pour toute suite croissante $(x_n)_{n\in\mathbb{N}}$ dans $C^\sharp$, la suite définie par $y_0 = x_0$ et $y_{n+1} = y_n \nabla x_{n+1}$ converge en un nombre fini d'itérations.

Pour résoudre le système abstrait, on s'appuie en général sur la structure de contrôle de l'automate analysé. La stratégie d'itération consiste donc à calculer les ensembles $\mathcal{A}_k^\sharp$ les uns après les autres, et aux points d'élargissements choisis (voir plus bas), au lieu de garder le nouvel ensemble $\mathcal{A}_{k'}^\sharp$ (plus grand que l'ensemble $\mathcal{A}_k^\sharp$ calculé à l'itération précédente), nous gardons en mémoire l'ensemble $\mathcal{A}_k^\sharp \nabla \mathcal{A}_{k'}^\sharp$.

Pour garantir la terminaison de la procédure, il faut que les points d'élargissement coupent tous les cycles du graphe de flot de contrôle. D'un autre côté, l'application de l'opérateur étant source d'une perte de précision, il est crucial de choisir un ensemble de points de contrôle minimal pour l'appliquer. C'est l'objet du prochain paragraphe.

2.3.5 Choix des points d'élargissement

Ce problème se ramenant au choix de la coupure minimale dans un graphe, il est NP-complet. Néanmoins, classiquement, on applique l'heuristique des sous-composantes fortement connexes proposée dans [Bou92]. L'algorithme, basé sur l'algorithme de Tarjan ([Tar72]) consiste à rechercher les sous-composantes fortement connexes du graphe (SCFC) et à prendre comme point d'élargissement toutes les têtes de sous-composantes.

L'algorithme de Tarjan effectue une recherche en profondeur, et fournit la liste des composantes fortement connexes (CFC) ainsi qu'un *point d'entrée* de chaque CFC, qui est le premier point de contrôle rencontré deux fois dans la recherche. Sur l'exemple 2.3.5, qui provient de [Mer05], le graphe se décompose en trois CFC (cercles en trais épais) :

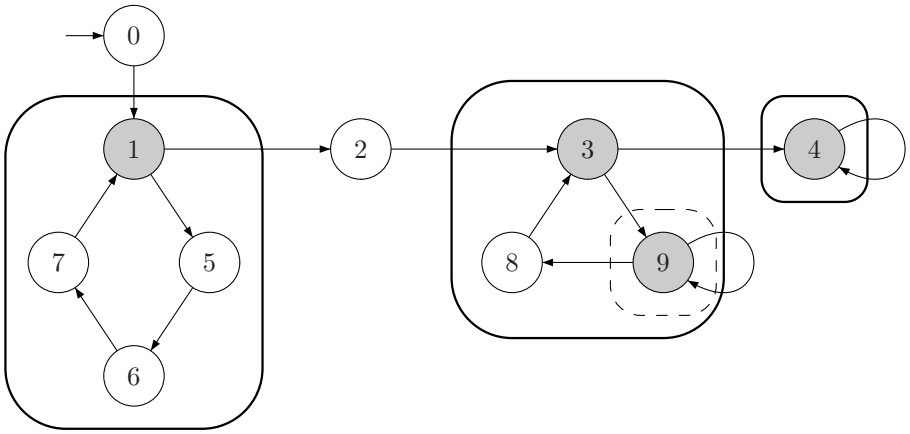


FIG. 2.4 – Un automate et ses CFC et sCFC

Les points d'entrée de ces composantes font de bons candidats pour être nœuds d'élargissement. Cependant, ils ne sont pas suffisants puisqu'en les éliminant, le graphe peut encore comporter des cycles, ce qui est le cas pour l'exemple (le nœud 9 crée un cycle à l'intérieur de la deuxième CFC). Bourdoncle propose donc de rappliquer l'algorithme de Tarjan à chacune des CFC jusqu'à obtenir un graphe sans sous-graphe fortement connexe. L'ensemble des nœuds d'élargissement comprend alors tous les nœuds retirés lors de la procédure. Sur l'exemple précédent, on trouve les (sous-)composantes suivantes : $\{1,5,6,7\}$, $\{3,8,9\}$, $\{9\}$ (cercle pointillé), et $\{4\}$, et les nœuds d'élargissement sont $\{1,3,9,4\}$ (en gris). Dans cet exemple, l'ensemble n'est pas minimal, car on aurait pu prendre l'ensemble $\{1,9,4\}$.

2.3.6 Différents treillis numériques

Dans cette thèse, nous nous restreindrons au cas numérique, c'est-à-dire que toutes les variables prendront leurs valeurs dans l'ensemble $\mathcal{N}$, qui sera $\mathbb{Z}$, $\mathbb{Q}$ ou $\mathbb{R}$. Nous verrons dans le chapitre 8 comment s'y ramener.

Comme nous nous restreignons au cas des automates interprétés numériques à n variables, il s'agit de choisir un treillis abstrait pour le treillis concret $C = (2^{\mathcal{N}^n}, \subseteq, \cup, \cap, \{\mathcal{N}^n\}, \emptyset)$. De nombreux exemples de treillis sont proposés dans la littérature, qui correspondent à différentes façons d'abstraire un ensemble de points de $\mathcal{N}^n$.

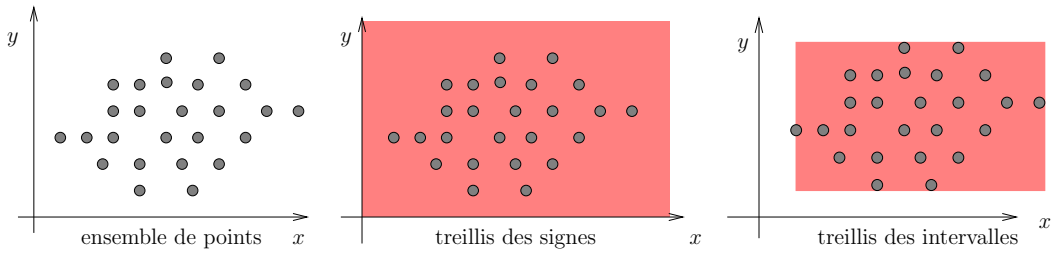


FIG. 2.5 – Différents treillis numériques

Un ensemble de points de $\mathcal{N}^n$ peut ainsi être représenté de différentes manières :
- On peut ne stocker que les signes des variables. Cette représentation est compacte et efficace, mais peu précise. Ce treillis est fini donc l'élargissement est inutile.
- On peut choisir la plus petite variété affine le contenant, c'est le treillis des équations affines, proposé par [Kar76]. Avec un tel treillis on peut découvrir des relations du type $2x - y = 5$. Ce treillis est de profondeur finie, donc il n'y a pas lieu de définir un élargissement.
- Dans la thèse [Gra91] est proposé le treillis des congruences linéaires, la représentation se fait sous formes d'équations de la forme $ax \equiv b \mod c$. Ce treillis satisfaisant la condition de chaîne ascendante, il n'y a pas non plus lieu de définir un élargissement.
- On peut représenter les intervalles de variations des variables, c'est le treillis des intervalles ([CC76]). Ici un élargissement est proposé, dont l'idée est la suivante : si la borne droite (resp. gauche) d'un intervalle a crû strictement (resp. décrû strictement), alors on peut supposer que les bornes vont croître tout le temps, et on remplace la borne de droite (resp. la borne de gauche) par $+\infty$ (resp $-\infty$).
- On peut aussi représenter un ensemble de points par le plus petit polyèdre convexe le contenant. Ce treillis est plus précis, il permet en particulier de découvrir des relations affines entre les variables, par exemple $2x - t \leqslant y + 20$. C'est ce treillis que nous allons utiliser, nous allons détailler ses caractéristiques dans la section suivante.
- Les opérations sur les polyèdres étant coûteuses (ce coût provient essentiellement du passage entre les deux représentations, comme nous allons le voir dans la section 2.4.2), on peut choisir de se restreindre à des sous-classes moins coûteuses. Par exemple, les treillis des zones (contraintes contenant au plus deux variables et de la forme $x \leqslant c$ et $x - y \geqslant c$, codées sous la forme de matrices de bornes, [AD90, HNSY92]), des octogones (contraintes avec au plus deux variables de la forme $x \leqslant c$ et $\pm x \pm y \geqslant c$, [Min01]), permettent de récupérer une complexité polynomiale sur l'ensemble des opérations. Citons enfin les octaèdres ([CC04]), dont la complexité n'est plus polynomiale.

2.4 Analyse des Relations Linéaires

Dans cette section, nous présentons le cas particulier de l'interprétation abstraite avec le treillis numérique des polyèdres. Commençons par une première remarque.

Remarque 2 Val est isomorphe à $\mathcal{N}^n$, ce qui permet de représenter une valuation v des variables par un vecteur $X_v \in \mathcal{N}^n$. La i-ème composante de i représente la valeur de la i-ème variable dans la valuation v. Nous utiliserons fréquemment la notation vectorielle, en la notant X lorsqu'il n'y a pas d'ambiguïté sur la valuation sous-jacente.

2.4.1 Le Modèle

Les automates interprétés affines sont des automates interprétés dont les commandes ont une forme particulière, ce sont des *commandes affines gardées* :

Définition 8 — Garde affine, Syntaxe

> Une garde affine est soit *true*, soit un couple (A, B) avec $A \in \mathcal{M}_{m,n}(\mathcal{N})$ une matrice de taille $m \times n$ et $B \in \mathcal{M}_{m,1}(\mathcal{N})$ un vecteur de taille m, dans ce cas on note $g = (AX \leqslant B)$.

Définition 9 — Satisfaction d'une garde affine

Soit g une garde affine. La fonction booléenne associée est :
- si $g = true$ alors $\widetilde{g}$ vérifie $\forall v \in \text{Val}, \widetilde{g}(v) = \text{Vrai}$.
- si $g = (AX \leqslant B)$, alors $\widetilde{g}$ est définie par $\begin{cases} \text{Vrai} & \text{si } AX_v \leqslant B \\ \text{Faux} & \text{sinon.} \end{cases}$. Autrement dit, la valuation v satisfait la garde g si le vecteur associé X_v appartient au polyèdre formé par l'ensemble de contraintes (A, B) (voir la sous-section 2.4.2).

Par abus de langage, on appelle garde affine, et on note g la fonction $\widetilde{g}$.

Définition 10 — Action affine

Une action affine est un couple (C, D) avec $C \in \mathcal{M}_n(\mathcal{N})$ une matrice carrée de taille n et $D \in \mathcal{M}_{n,1}(\mathcal{N})$ un vecteur dit « de décalage ». On note $a = (X := CX + D)$. La fonction affine associée $\widetilde{a}$ est la fonction $X_v \mapsto C.X_v + D$ (opérations matricielles). De la même manière que précédemment, on appelle action affine, et on note a la fonction $\widetilde{a}$.

Définition 11 — Commande affine gardée

Une *commande affine gardée* est un couple (g, a) avec g une garde affine et a une action affine. Dans la suite, on notera $\tau : g \rightarrow a$.

Une transition affine gardée est donc un quadruplet $(k, (g, a), k')$ avec $k, k' \in K$ et (g, a) une commande affine gardée. Sur les dessins (par exemple la figure 2.6) nous notons $g \rightarrow a$.

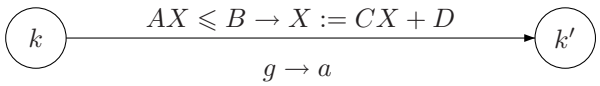


Fig. 2.6 – Une transition affine gardée

Exemple 2.3 *Voici un exemple d'automate interprété affine :*

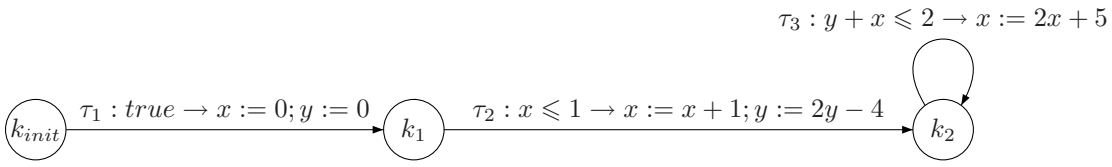


Fig. 2.7 – Un automate interprété affine

La sémantique de la transition τ_1 donne pour seule valuation possible au point de contrôle k_1 $v_1 : x \mapsto 0, y \mapsto 0$. Puisque la valuation v_1 satisfait la garde $x \leqslant 1$, on peut calculer l'ensemble des valuations accessibles au point k_1, qui est $\{v_2\}$ avec $v_2 : x \mapsto 1, y \mapsto -4$. Et ainsi de suite...

2.4.2 Le treillis des polyèdres convexes

Nous choisissons dans cette thèse de travailler à l'aide du treillis des polyèdres convexes, qui est certes coûteux mais qui expérimentalement donne de bons résultats en terme de précision

d'analyse. Dans cette section, nous allons détailler les caractéristiques de ce treillis. Dans ce cadre précis, la vérification par interprétation abstraite est nommée **Analyse des relations linéaires** (en anglais, *Linear Relation Analysis*).

[CH78] a proposé le treillis des polyèdres convexes comme domaine abstrait adapté à la vérification de programmes numériques. Ce treillis n'est cependant pas complet : en effet on peut construire une suite de polyèdres convergeant vers un disque. On contourne cette difficulté en se plaçant dans le treillis complet des convexes, dans lequel toute partie (finie ou infinie) de $\mathcal{N}^n$ est abstraite par le plus petit convexe la contenant. Ensuite, nos opérations et l'utilisation d'un opérateur d'élargissement nous garantiront que nous demeurons dans le sous-treillis des polyèdres convexes.

Un polyèdre convexe fermé (dans la suite, on utilisera le terme *polyèdre*) peut être représenté des deux manières duales suivantes ([MT53]) :

– par une conjonction d'inégalités linéaires ($a_j, b_j \in \mathcal{N}^n$) :

$$P = \{X \mid \bigwedge_{1 \leqslant j \leqslant m} a_j X \leqslant b_j\} = \{X \mid AX \leqslant B\}$$

– ou par ses éléments générateurs : S est un ensemble fini de *sommets* (éléments de $\mathcal{N}^n$) et R un ensemble de *rayons* (vecteurs de $\mathcal{N}^n$) :

$$\left\{ \sum_{s_i \in S} \lambda_i s_i + \sum_{r_j \in R} \mu_j r_j \mid \lambda_i \geqslant 0, \mu_j \geqslant 0, \sum_i \lambda_i = 1 \right\}.$$

(A, B) et (S, R) sont respectivement un système de contraintes et un système générateur du polyèdre P.

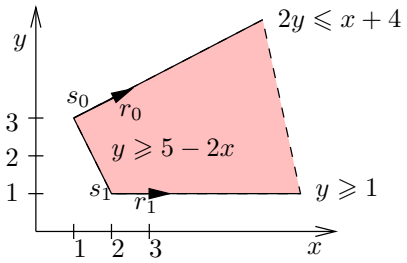


FIG. 2.8 – Les deux représentations d'un polyèdre

Une algorithmique a été développée afin de réaliser les opérations sur ce treillis abstrait le plus efficacement possible. Nous la présentons succinctement dans la section suivante.

2.4.3 Opérations sur les polyèdres

Nous allons voir dans cette section que suivant les opérations, une représentation du polyèdre peut être plus intéressante que l'autre. Dans nos analyses, il sera donc intéressant de gérer en parallèle les deux représentations. Comme le coût de passage d'une représentation à l'autre peut être exponentiel en n, un tel calcul ne sera fait qu'aux endroits nécessaires.

Dans la suite, nous allons rapidement rappeler les opérations simples à l'intérieur du treillis des polyèdres convexes.

Test du vide et du plein Un polyèdre est vide si et seulement si son ensemble de sommets est vide. Un polyèdre est égal au polyèdre univers si et seulement si son ensemble de contraintes est vide.

Test d'inclusion Le test d'inclusion s'effectue à l'aide des deux représentations et de la remarque suivante :

$$P \subseteq P' \Leftrightarrow \big(\forall s \in S, A's \leqslant B' \wedge \forall r \in R, A'r \leqslant 0\big),$$

avec (S, R) un système générateur de P et (A', B') un système de contraintes de P'.

L'intersection L'intersection est obtenue en faisant la conjonction des deux systèmes de contraintes.

Union convexe L'union convexe de deux polyèdres n'étant en général pas convexe, l'opération de borne supérieure dans le treillis des polyèdres est l'enveloppe convexe (notée $\sqcup$) qui associe à deux polyèdres le plus petit polyèdre les contenant.

Un système générateur de l'enveloppe convexe de deux polyèdres peut être obtenu en faisant l'union des rayons et l'union des sommets. On remarque que l'union convexe de deux polyèdres génère une perte d'informations puisque des points qui ne sont pas dans l'union seront dans l'union convexe (figure 2.9).

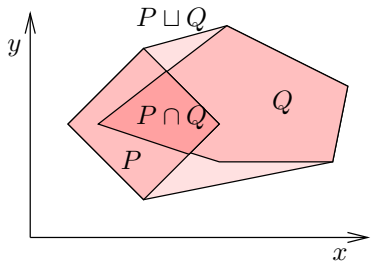


FIG. 2.9 – Intersection et union convexe de deux polyèdres

Projection d'un polyèdre sur une variable Il est souvent utile d'éliminer certaines variables, par exemple pour sur-approximer l'effet d'une affectation non affine concernant ces variables. Cette opération correspond à la quantification existentielle $\exists x.P$, et peut être réalisée sur la représentation par générateurs. Il suffit d'ajouter les deux rayons $\overrightarrow{Ox}$ et $\overrightarrow{-xO}$.

Image et Pré-image Soit a une action affine $X := CX + D$, alors $(\{Cs + D \mid s \in S\}, \{Cr \mid r \in R\})$ est un système générateur de $a(P)$. Dualement, $(AC, B - AD)$ est un système de contraintes de $a^{-1}(P)$.

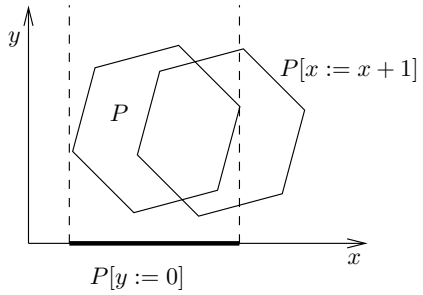


FIG. 2.10 – Image d'un polyèdre par une action affine

Élargissement L'opérateur standard d'élargissement sur les polyèdres, proposé dans [CH78], consiste à supprimer toute inégalité qui a été translatée ou a subi une rotation, en supposant que cette action peut se répéter une infinité de fois. On ne garde donc dans $P \nabla Q$ que les inégalités de P qui sont aussi vérifiées par Q (Figure 2.11). Cet opérateur satisfait alors la condition de chaîne :

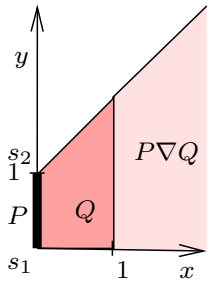


FIG. 2.11 – Élargissement du polyèdre P par Q (contenant P)

Cependant, si on utilise cette définition « naïve », on peut obtenir un résultat dépendant de l'encodage choisi, comme le montre l'exemple 2.4.

EXEMPLE 2.4 *Soient les deux polyèdres* $P = \{x = 0, 0 \leqslant y \leqslant 1\}$ *et* $Q = \{0 \leqslant x \leqslant 1, 0 \leqslant y \leqslant x + 1\}$ *(voir Figure 2.11). Si on applique l'idée de l'élargissement énoncée ci-dessus, on obtiendrait* $P \nabla Q = \{0 \leqslant x, 0 \leqslant y\}$ *(c'est-à-dire le premier quadrant). En récrivant* $P = \{x = 0, 0 \leqslant y \leqslant x+1\}$*, on obtiendrait* $P \nabla Q = \{0 \leqslant x, 0 \leqslant y \leqslant x+1\}$*, qui est le résultat souhaité (*$P \nabla Q$ *sur la figure).*

L'idée est de prendre en compte les contraintes mutuellement redondantes :

DÉFINITION 12 — Saturation

Un générateur *sature* une contrainte $aX \leqslant b$ si ce générateur est dans l'hyperplan défini par la contrainte, *i.e.* :
- si ce générateur est un sommet s : $b - as = 0$.
- si ce générateur est un rayon r : $ar = 0$.

DÉFINITION 13 — Redondance

Soit P un polyèdre. Deux contraintes sont *mutuellement redondantes* dans P si elles sont saturées par les mêmes générateurs de P.

Deux contraintes mutuellement redondantes sont interchangeables : on peut utiliser l'une ou l'autre à l'intérieur d'un système de contraintes donné. Sur l'exemple 2.4, la contrainte $y \leqslant x+1$ de Q est mutuellement redondante avec la contrainte $y \leqslant 1$ dans le polyèdre P, donc le résultat $P \nabla Q$ comprendra cette contrainte.

En pratique ([Hal79b]), l'opérateur d'élargissement utilise la notion de matrice de saturation (qui va permettre d'identifier les contraintes mutuellement redondantes) : pour chaque générateur de P, pour chaque contrainte $aX \leqslant b$ de P et de Q, on calcule $b - as$ si c'est un sommet s , et $-ar$ si c'est un rayon r. Pour l'exemple 2.4, cela donne :

	Contraintes de P			Contraintes de Q			
	$x = 0$	$0 \leqslant y$	$y \leqslant 1$	$0 \leqslant x$	$x \leqslant 1$	$0 \leqslant y$	$y \leqslant x+1$
Générateurs de P	x	y	$1-y$	x	$1-x$	y	$x+1-y$
$s_1 = (0,0)$	0	0	1	0	1	0	1
$s_2 = (0,1)$	0	1	0	0	1	1	0

On prend alors pour système générateur de $P \nabla Q$ toutes les contraintes de Q qui sont mutuellement redondantes, dans P, avec une contrainte de P, c'est-à-dire les contraintes de Q qui saturent les mêmes générateurs qu'une contrainte de P existante. Sur la matrice, étant donnée une contrainte de Q, elle est gardée dans le polyèdre $P \nabla Q$ si et seulement si il existe une contrainte de P avec une position identique des 0. Pour l'exemple, les contraintes concernées sont $0 \leqslant x$, $0 \leqslant y$, et $y \leqslant x+1$. On trouve donc $P \nabla Q = \{0 \leqslant x, 0 \leqslant y \leqslant x+1\}$.

Le résultat est donc indépendant du système de contraintes choisi pour P et Q (on n'a donc pas besoin de minimiser le premier argument). On montre ensuite que cet opérateur est bien un opérateur d'élargissement.

Remarque 3 La condition de chaîne n'est en fait vérifiée que si à chaque fois, le deuxième argument de l'opérateur d'élargissement est plus grand (au sens de l'inclusion du treillis abstrait) que le premier. Dans nos analyses, c'est toujours le cas.

Ajout de rayon Soit R un ensemble fini de rayons, alors le calcul de l'ensemble :

$$P \nearrow R = P \nearrow \{r_1, r_2, \ldots, r_k\} = \{X + \sum_{r_j \in R} \mu_j r_j \mid X \in P, \mu_j \in \mathbb{Q}^+\}$$

se fait en ajoutant les k rayons à la représentation par générateurs du polyèdre P. Cette opération est souvent utilisée lors de l'analyse de systèmes hybrides ([HPR97]).

2.5 Exemple d'analyse d'un automate interprété

On considère dans cette section l'exemple classique suivant, construit d'après [Hal79b].

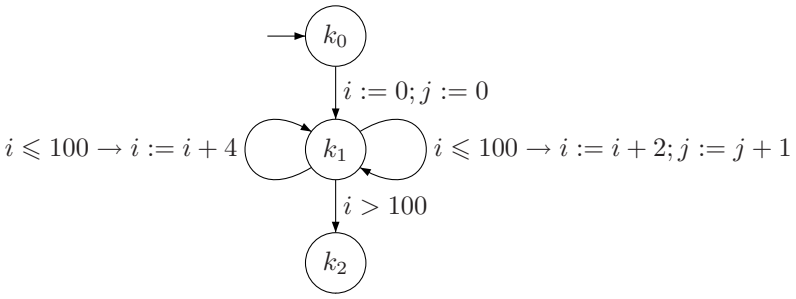


FIG. 2.12 – Automate exemple (k_0 est le point de contrôle initial)

Le système d'équations abstrait est le suivant :

$$\begin{cases} P_{k_0} = \top \\ P_{k_1} = P_{k_0}[i := 0; j := 0] \sqcup (P_{k_1} \cap i \leqslant 100)[i := i + 2; j := j + 1] \sqcup (P_{k_1} \cap i \leqslant 100)[i := i + 4] \\ P_{k_2} = P_{k_1} \cap \{i > 100\} \end{cases}$$

La recherche des points d´élargissement donne l'unique point k_1.

Initialisation On initialise les invariants de la façon suivante :
- $P^0_{k_0} = \top$ (polyèdre univers)
- $P^0_{k_1} = \emptyset$
- $P^0_{k_2} = \emptyset$

La stratégie utilisée consiste à converger à l'intérieur de la composante fortement connexe $\{k_1\}$, puis à propager vers la composante suivante, c'est-à-dire $\{k_2\}$.

Composante $\{k_1\}$ On trouve à la première itération $P^1_{k_1} = \{i = j = 0\}$ (contribution de la transition provenant du nœud k_{init}, et aucune des autres transitions ne peut être appliquée puisque le polyèdre $P^0_{k_1}$ est vide.

À la deuxième itération, les deux transitions qui bouclent sur k_1 peuvent être appliquées sur $P^1_{k_1}$, on trouve donc :

$$\begin{aligned} P^1_{k_2} &= P^1_{k_1} \nabla \Big(\{i = j = 0\} \sqcup \{i = 2, j = 1\} \sqcup \{i = 4, j = 0\} \Big) \\ &= P^1_{k_1} \nabla \{2j + i \leqslant 4, 2j \leqslant i, 0 \leqslant j\} \\ &= \{2j \leqslant i, 0 \leqslant j\} \end{aligned}$$

À la troisième itération, les trois transitions qui arrivent en k_1 peuvent toujours être appliquées, et on calcule :

$$\begin{aligned} P^3_{k_1} &= P^2_{k_1} \nabla \Big(\{i = j = 0\} \sqcup \{2j \leqslant i, 1 \leqslant j, i \leqslant 102\} \sqcup \{2j + 4 \leqslant i, 0 \leqslant j, i \leqslant 104\} \Big) \\ &= P^2_{k_1} \nabla \{2j \leqslant i, 2j + i \leqslant 204, 0 \leqslant j, i \leqslant 104\} \\ &= \{2j \leqslant i, 0 \leqslant j\} \\ &= P^2_{k_1} \end{aligned}$$

La composante k_1 est stabilisée, on peut donc propager vers la composante $\{k_2\}$.

Composante $\{k_2\}$ et résultat final Finalement, on trouve pour invariants :
- $P^{fin}_{k_0} = \top$.
- $P^{fin}_{k_1} = \{2j \leqslant i, 0 \leqslant j\}$.
- $P^{fin}_{k_2} = P^{fin}_{k_1} \cap \{i > 100\} = \{2j \leqslant i, 0 \leqslant j, i > 100\}$.

2.6 Exemple introductif : « gas burner »

Dans cette section, nous allons présenter les motivations de ce travail sur l'exemple classique d'une chaudière qui fuit (ou « gas burner »). Cet exemple provient de [CHR91a] : on veut modéliser une chaudière sous l'hypothèse qu'à chaque fois qu'une fuite de gaz apparaît, cette fuite est détectée et stoppée dans les 10 secondes, et que le temps minimal qui sépare deux périodes où la chaudière fuit est 50 secondes.

2.6.1 Le cas hybride linéaire

La modélisation standard de ce problème se fait sous la forme de l'automate hybride linéaire de la figure 2.13 ([ACH+95, HHWT97]). Les variables t et ℓ représentent respectivement le temps total écoulé depuis le début et le temps global de fuite de gaz. La variable x est une variable locale utilisée pour compter le temps passé dans chacun des points de contrôle.

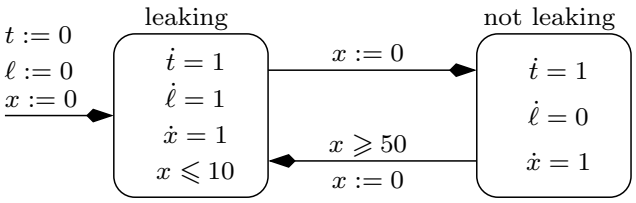


FIG. 2.13 – Modélisation de la chaudière par un automate hybride linéaire

Un automate hybride linéaire permet de modéliser des comportements continus des variables, le temps passant de manière implicite dans les points de contrôle. Ainsi, à chaque point de contrôle q sont associés un invariant $Inv(q)$ (une contrainte linéaire sur les variables) qui est toujours vrai à ce point de contrôle, et un ensemble de contraintes « dérivées » $D(q)$ ($\dot{x} \leqslant 3$ par exemple) qui donne l'évolution des variables lorsque le temps s'écoule. Deux types de transitions sont donc possibles :
- Une transition discrète (notée $\rightarrow$) peut être prise si la garde correspondante est vraie. L'action est alors appliquée à la valuation courante. La transition est instantanée.
- Une transition « passage du temps » (notée $\rightarrow^{\delta}$) peut être prise tant que les valeurs des variables satisfont l'invariant du point de contrôle.

Par exemple, sur la figure 2.13, le chemin suivant est possible (les vecteurs représentent les valeurs des variables (t, ℓ, x) dans cet ordre) :

$$(\text{leaking}, (0,0,0)) \rightarrow^{\delta} (\text{leaking}, (5,5,5)) \rightarrow (\text{not\_leaking}, (5,5,0)) \rightarrow^{\delta} (\text{not\_leaking}, (10,5,0)) \rightarrow \dots$$

Sur la figure 2.14, on représente l'analyse de cet automate selon la méthode proposée dans [HPR97]. C'est une analyse des relations linéaires adaptée aux systèmes hybrides. La prise en compte de l'écoulement du temps est effectuée par l'introduction d'une opération « passage du temps » (*time elapse operator*) : si D est un système de contraintes dérivées représentées par un polyèdre (V', R')), et P un polyèdre de générateurs (V, R), on définit :

$$P \nearrow D = \{X + td \,|X \in P, d \in D, t \in \mathbb{Q}^+\},$$

polyèdre dont un système générateur est $(V, V \cup R \cup R')$.

On obtient ensuite le système d'équations permettant de calculer $Post^*(q)$ (en reprenant des notations similaires à la section 2.3.1, page 20)

$$Post^*(q) = \Big(\big(Init_\ell \bigcup_{(q,(a,g),q') \in \text{Trans}} a(Post^*(q') \cap g_i) \cap Inv(q)) \nearrow D \big) \cap Inv(q) \Big)$$

L'analyse est donc très similaire à celle de l'Analyse des Relations Linéaires classique, sauf qu'elle prend en compte l'écoulement du temps dans chaque point de contrôle. L'analyse de l'automate de la figure 2.13 est illustrée sur la figure 2.14. Le vecteur $\overrightarrow{\delta}$ illustre le « passage du temps ». Les résultats successifs obtenus par cette analyse sont projetés sur les variables t et ℓ.

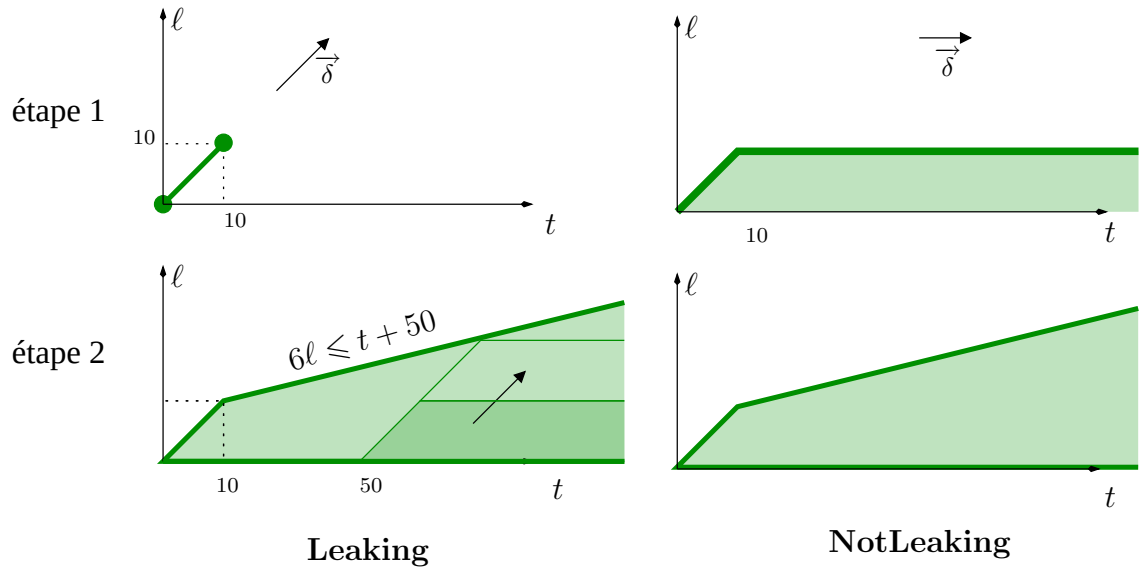


FIG. 2.14 – Analyse de la chaudière hybride

Voici les étapes de l'analyse :

– À la première itération, le point de contrôle « leaking » est atteint avec l'unique point $\{t = \ell = 0\}$ et l'application de l'opérateur de passage du temps donne le segment $\{0 \leqslant t = \ell \leqslant 10\}$. Ainsi, le point de contrôle « not leaking » est atteint avec le polyèdre $\{0 \leqslant t = \ell \leqslant 10\}$, et l'application du passage du temps permet d'obtenir le polyèdre $\{0 \leqslant \ell \leqslant 10, t \geqslant \ell\}$.

– À la deuxième itération, la contribution de la transition de retour « not leaking » vers « leaking » fournit le polyèdre $\{0 \leqslant \ell \leqslant 10, t \geqslant \ell + 50\}$ (en foncé sur la figure en bas à gauche), on applique ensuite l'opérateur de passage du temps, puis on réalise l'enveloppe convexe du polyèdre obtenu avec le polyèdre trouvé à l'étape précédente, et enfin l'application de l'opérateur d'élargissement donne le polyèdre $\{0 \leqslant \ell \leqslant t, t \geqslant 6\ell - 50\}$. En propageant on trouve le même polyèdre pour le point de contrôle « not leaking » et on termine avec le résultat optimal, c'est-à-dire que les invariants obtenus sont l'enveloppe convexe des états atteignables.

2.6.2 Version discrète de la chaudière

Maintenant considérons une version discrète de la chaudière (Figure 2.15). Le but est de trouver le même invariant que dans la version hybride, c'est à dire $6\ell \leqslant t + 50$.

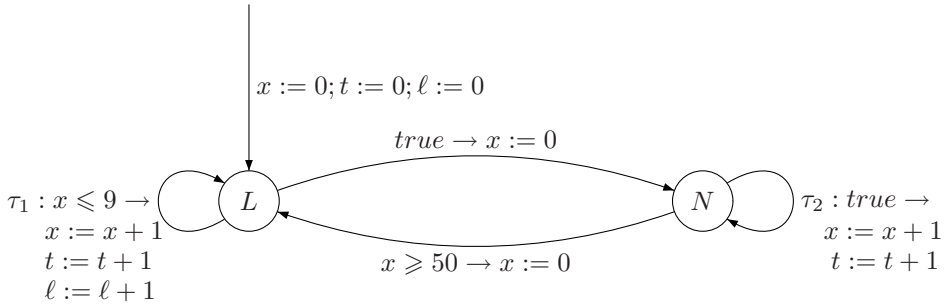


FIG. 2.15 – Modélisation de la chaudière sous la forme d'automate interprété numérique

Comme il y a une boucle autour de chaque point de contrôle, les deux points de contrôle L et N sont des points d'élargissement. Appliquons le schéma d'analyse des relations linéaires avec l'opérateur d'élargissement standard. Au point de contrôle L, initialement $t = \ell = 0$ puis $t = \ell = 1$ (aucune contribution du nœud N). L'enveloppe convexe est $\{0 \leqslant t = \ell \leqslant 1\}$, et l'élargissement donne $\{0 \leqslant t = \ell\}$. L'analyse converge sur les invariants suivants :

$$P_L = \{0 \leqslant x \leqslant \ell, \ell \leqslant t\} \qquad P_N = \{0 \leqslant x, 0 \leqslant \ell; x + \ell \leqslant t\}$$

Cet exemple montre comment l'analyse d'un automate hybride peut être beaucoup plus précise et efficace que l'analyse de l'automate discret correspondant. La raison évidente est que dans le cas hybride on dispose de l'opérateur « passage du temps », qui joue le rôle d'un opérateur d'élargissement **exact**.

L'objectif de cette thèse est de détecter les boucles de l'automate interprété numérique dont le comportement peut être calculé exactement, de façon à *résumer* leur effet par une unique *meta*-transition, c'est-à-dire exactement ce qui est fait lors de l'application de l'opérateur de passage du temps dans les automates hybrides. En d'autres termes, dans le cas de l'exemple de la Figure 2.15, on appliquera l'analyse standard à l'automate de la Figure 2.16, dans lequel $\tau_1^{\otimes}$ et $\tau_2^{\otimes}$ dénotent l'effet des deux boucles simples dans l'automate initial. Ces deux boucles seront donc « accélérées », mais il reste une boucle dans l'automate, donc l'opérateur d'élargissement sera appliqué au point de contrôle L afin de garantir l'arrêt de l'analyse.

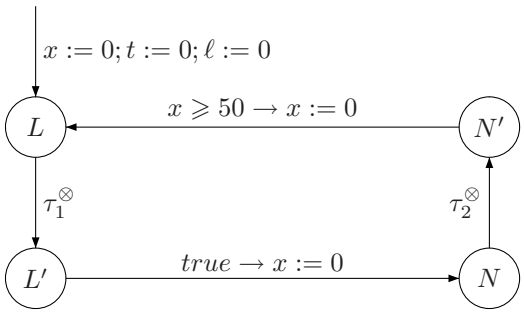


FIG. 2.16 – Automate « accéléré »

Conclusion du chapitre

Ainsi l'Analyse des Relations Linéaires propose une méthode générale d'étude d'automates interprétés numériques, qui permet la découverte d'invariants numériques polyèdriques sur chaque point de contrôle de l'automate. Cette approche nécessite l'introduction d'un opérateur d'élargissement dont le but est d'extrapoler la limite d'une suite de polyèdres. Le principal avantage de la méthode est que les analyses convergent toujours. Cependant, l'exemple de la chaudière montre que les résultats ne sont pas toujours intéressants en terme de précision, alors qu'une analyse hybride (utilisant l'opérateur « passage du temps ») fournit des invariants plus précis. Nous cherchons donc à « accélérer » le traitement de certaines boucles, *i.e.* d'en calculer l'effet sans élargissement. Le chapitre suivant fait un état de l'art des méthodes connues d'accélération, ainsi que de leur mise en œuvre.

Chapitre 3

Principes des méthodes d'accélération

Les techniques d'accélération permettent de « résumer » le calcul d'un ensemble d'états accessibles (procédure souvent infinie) par un calcul plus rapide et direct. Par exemple, on peut résumer l'effet du code `i:=0;while i<10 do i:=i+1 done` *par « $i \in [\![0, 10]\!]$ ». Dans un premier temps, nous rappelons quelques résultats de décidabilité concernant l'itération de quelques relations de transitions particulières (les transitions affines gardées, les translations convexes, ...). Ensuite, nous décrivons différentes techniques de mise en œuvre de ces accélérations.*

3.1 Une premiere notion d'accélération

L'itération quelconque de fonctions affines est étudiée dans [BW94]. Les auteurs partent du constat que non seulement l'ensemble des états accessibles est difficile à calculer, mais aussi que le résultat du calcul de ces états accessibles peut être difficile à représenter de manière compacte (nécessairement symbolique). Ils présentent donc une méthode comparable à celle des BDD pour représenter des ensembles de valeurs, les « periodic sets ». Sur l'exemple 3.1, l'espace des états contient plus de cinq cent mille états accessibles, et il est indipensable de trouver une représentation symbolique compacte de cet ensemble, ainsi qu'un moyen rapide de le calculer.

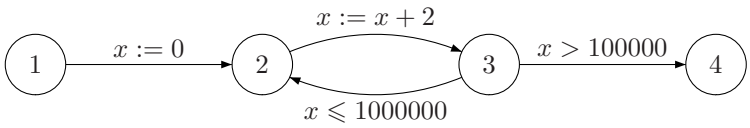


Fig. 3.1 – Exemple de l'article

Dans le cas de l'exemple, les auteurs proposent l'utilisation des *ensembles périodiques de vecteurs*[1] :

[1] Les notations ont été modifiées pour éviter les conflits avec les nôtres.

Définition 14 — Ensemble périodique de vecteurs

Un ensemble périodique de vecteurs est un ensemble de vecteurs de la forme :

$$\{\overrightarrow{x} \in \mathbb{Z}^n; \exists \overrightarrow{k} \in \mathbb{Z}^m \text{ tel que } \overrightarrow{x} = \Delta \overrightarrow{k} + \overrightarrow{\delta} \text{ et } P\overrightarrow{k} \leqslant \overrightarrow{q}\}$$

avec :
- $\Delta \in \mathcal{M}_{n,m}(\mathbb{Z})$ une matrice dite de « périodicité »
- P et $\overrightarrow{q}$ forment un système linéaire.

Pour représenter un tel ensemble périodique, on stocke $m, \Delta, \overrightarrow{\delta}$ et le polyèdre associé à P et $\overrightarrow{q}$ sous la double représentation usuelle.

Les auteurs montrent ensuite que cette représentation permet de calculer efficacement (en restant dans la classe des ensembles périodiques de vecteurs) l'application simple ou itérée d'une transition affine gardée idempotente :

Proposition 1 ([BW94]) *L'application d'une transition affine gardée quelconque sur un ensemble périodique de vecteurs est calculable.*

Preuve : Soit $S = \{\overrightarrow{x}; \exists k \in \mathbb{Z}^m, \overrightarrow{x} := \Delta \overrightarrow{k} + \overrightarrow{\delta} \wedge P\overrightarrow{k} \leqslant \overrightarrow{q}\}$. Un $\overrightarrow{x'}$ obtenu après application de $\tau : A\overrightarrow{x} \leqslant \overrightarrow{b} \rightarrow \overrightarrow{x} := C\overrightarrow{x} + \overrightarrow{d}$ vérifie :

$$\exists \overrightarrow{k} \in \mathbb{Z}^m; \overrightarrow{x'} = C\Delta\overrightarrow{k} + (C\overrightarrow{\delta} + \overrightarrow{d}) \wedge A(\Delta\overrightarrow{k} + \overrightarrow{\delta}) \leqslant \overrightarrow{b} \wedge P\overrightarrow{k} \leqslant \overrightarrow{q};$$

donc $\overrightarrow{x'}$ vérifie $\exists k \in \mathbb{Z}^m, \overrightarrow{x'} = \Delta'\overrightarrow{k} + \overrightarrow{\delta} \wedge P'\overrightarrow{k} + \overrightarrow{q'}$ avec $\Delta' = C\Delta$, $\overrightarrow{\delta'} = C\overrightarrow{\delta} + \overrightarrow{d}$ et le nouveau système $P'\overrightarrow{k} \leqslant \overrightarrow{q'}$ est la conjonction des systèmes $P\overrightarrow{k} \leqslant \overrightarrow{q}$ et $A(\Delta\overrightarrow{k} + \overrightarrow{\delta}) \leqslant \overrightarrow{b}$. ■

Proposition 2 ([BW94]) *L'application itérée (une ou plusieurs fois) d'une transition affine gardée vérifiant $C^2 = C$ sur un ensemble périodique de vecteurs est calculable.*

Preuve : On prend le même ensemble périodique de vecteurs et la même transformation affine gardée que la preuve précédente, en supposant en plus C idempotent. On vérifie assez aisément que $\bigcup_{i \in \mathbb{N}} \tau^i(S)$ est un ensemble périodique de vecteur :

$$\left\{\overrightarrow{x'}; \exists \overrightarrow{k'} = \begin{bmatrix}\overrightarrow{k} \\ k_1\end{bmatrix} \in \mathbb{Z}^{m+1}, \overrightarrow{x'} = \Delta\overrightarrow{k'} + \overrightarrow{\delta} \wedge P'\overrightarrow{k'}\right\},$$

avec $\Delta' = [C\Delta; C\overrightarrow{d}]$, $\overrightarrow{\delta'} = C\overrightarrow{\delta} + \overrightarrow{d}$ et le système $P'\overrightarrow{k'} \leqslant \overrightarrow{q'}$ formé par la conjonction des contraintes $P\overrightarrow{k} \leqslant \overrightarrow{q}$, $k_1 \geqslant 0$, $A(\overrightarrow{\Delta}\overrightarrow{k} + \overrightarrow{\delta}) \leqslant \overrightarrow{b}$, $A(C\overrightarrow{\Delta}\overrightarrow{k} + C\overrightarrow{\delta} + \overrightarrow{d}) \leqslant \overrightarrow{b}$ et $A(C\overrightarrow{\Delta}\overrightarrow{k} + (k_1 - 1)C\overrightarrow{d} + C\overrightarrow{\delta} + \overrightarrow{d}) \leqslant \overrightarrow{b}$.

■

Le calcul des contraintes est effectué à l'aide des algorithmes classiques de calcul d'intersection sur les polyèdres. Nous verrons dans la section 3.4, page 45 comment ses résultats sont utilisés pour calculer l'ensemble d'accessibilité de systèmes à compteurs.

3.2 Accélération de fonctions affines gardées

3.2.1 SMA et méta-transitions

Dans la thèse [Boi99], la notion de méta-transition est définie de façon formelle et dans le cadre général des `SMA` : « Structured Memory Automata » :

DÉFINITION 15 — Méta-transition

> Une méta transition est un triplet (c, f, c') avec c, c' des états de contrôle, et f associant à un ensemble de mémoires un ensemble de mémoires, tel que pour tout ensemble U de contenus de mémoires, on a : :
>
> $$\forall m' \in f(U), \exists m \in U, \qquad (c, m) \rightarrow_R^* (c', m')$$
>
> où R est la relation « accessible en un pas ».

Une méta-transition propage donc l'information d'accessibilité, mais son image n'est qu'une approximation inférieure de la clôture transitive R^*. Cette définition faible aura une conséquence sur l'algorithme final de calcul d'accessibilité, comme nous le verrons section 3.4.

L'auteur identifie ensuite les propriétés nécessaires à une représentation symbolique d'ensemble de valeurs pour pouvoir appliquer des accélérations (pour prouver l'accessibilité ou bien la terminaison). Divers exemples de `SMA` sont donnés ensuite, avec les représentations symboliques correspondantes.

3.2.2 Un résultat de décidabilité sur les systèmes à compteurs

Le chapitre 8 de [Boi99] traite le cas des systèmes à compteur. La représentation utilisée est celle des `NDD` (Number Decision Diagrams), qui est un encodage sous forme d'automates d'ensembles de vecteurs. Cette classe est stable par application des fonctions usuelles, en particulier par l'application d'une transition affine gardée. Le principal résultat de ce chapitre est une condition nécessaire sur la matrice de transformation C pour que l'accélération soit effectivement définissable en `NDD`, ou en logique de Presburger. Le résultat suivant donne la version « Presburger » :

THÉORÈME 2 *Soit* $\tau : (A\overrightarrow{x} \leqslant \overrightarrow{b} \rightarrow \overrightarrow{x} := C\overrightarrow{x} + \overrightarrow{d})$. *Si il existe* $p \in \mathbb{N}$ *tel que* C^p *est diagonalisable et l'ensemble des valeurs propres de* C^p *est inclus dans* $\{0, 1\}$, *alors pour tout ensemble de Presburger* $S \in \mathbb{Z}^n$, *l'ensemble* $\bigcup_{i \in \mathbb{N}} \tau^i(S)$ *est définissable en logique de Presburger.*

Remarquons que la condition sur C^p revient tout simplement à dire que C^p est une matrice de projection.

Le même chapitre fournit une procédure pour décider l'hypothèse de ce théorème (pour plus de détails, voir le chapitre 5 et l'Annexe A), et donc étant donnée une transition linéaire gardée, on sait déterminer algébriquement si elle est accélérable (au sens de la clôture transitive), puis construire le `NDD` correspondant à l'accélération. Cependant, le principal inconvénient de cette approche est que les automates `NDD` ont une grande complexité, qui est comparables à celles des `UBA` (voir le paragraphe 3.3.1).

Plus tard, dans [BW02] les mêmes auteurs introduisent une représentation un peu plus compacte que les `NDD`, ce sont les `RVA` (Real Vector Automata). Cette représentation a été

utilisée pour le calcul de la relation d'accessibilité pour les systèmes hybrides dans [BJH03] et l'outil [LAS].

Résultat de Finkel/Leroux Dans [FL02], les auteurs obtiennent un résultat similaire à celui du théorème précédent, avec une caractérisation différente (mais équivalente) et plus simple à énoncer (ϕ est la garde de la transition codée sous la forme d'une formule de Presburger, et la transformation est $\overrightarrow{x} := C\overrightarrow{x} + \overrightarrow{d}$) :

THÉORÈME 3 *Soit $f = (C, \overrightarrow{d}, \phi)$ une fonction de Presburger avec C de monoïde $<C>= \{Id, C, C^2, \ldots\}$ fini, alors la relation f^* est Presburger-définissable avec une procédure effective.*

Décider si un monoïde est fini est un problème dont la complexité est inconnue. Nous réaborderons cette question dans le chapitre 5 et l'annexe A.

3.2.3 Une sous-classe des fonctions affines gardées

Dans [CJ98] et la thèse [Jur99], sont considérés des systèmes à compteur avec les « gardes » qui sont des conjonctions de termes de la forme $y_j \# y_i + c$ ($\# \in \{=, <, \leqslant, \}$) avec les y_i qui sont soit des x_i (variables avant la transition), soit des x'_i (variables après les transitions). Une garde constitue donc une relation entre les variables avant et après la flèche. Dans la suite, nous appelons ces relations des *transitions à relations affines*.

Le papier montre le résultat suivant (les $\mathcal{C}_i$ et $\mathcal{C}'_i$ désignent respectivement les noms des compteurs et les compteurs « primés », les fonctions v sont les valuations des variables) :

THÉORÈME 4 *Pour toute transition à relation affine, il existe une formule de Presburger $\phi(\mathcal{C}, \mathcal{C}')$ effectivement calculable telle que $(v, v') \models \phi(\mathcal{C}, \mathcal{C}')$ ssi il existe un entier n tel que $(v, v') \models \exists \mathcal{C}_1, \mathcal{C}_2, \ldots g(\mathcal{C}, \mathcal{C}_1) \wedge \ldots \wedge g(\mathcal{C}_n, \mathcal{C}')$.*

Ce résultat est obtenu par des considérations sur le graphe de contraintes codant la transition. Le résultat théorique est intéressant, mais la taille de la formule et la complexité des calculs à mettre en jeu font que cette accélération semble difficilement implémentable. Notons aussi que la classe des transitions à relations affines est une sous-classe de la classe des transitions affines gardées par une formule de Presburger ou par un polyèdre convexe.

L'article [BIL06] simplifie la preuve et étend le résultat au cas des gardes paramétrées. Néanmoins, si les boucles avec gardes paramétrées sont accélérables, le problème d'accessibilité avec de telles gardes n'est décidable que dans le cas des automates paramétrés avec une seule boucle de contrôle, dans tous les autres cas l'article montre que le problème est de nouveau indécidable.

3.2.4 Les translations convexes

Dans [BFL04], on considère la classe des translations convexes, *i.e.* les fonctions linéaires de Presburger de la forme $(I, \overrightarrow{d}, \varphi)$, avec la garde φ qui est un polyèdre convexe, et la transformation est une translation de vecteur $\overrightarrow{d}$. Le résultat obtenu est le suivant (les `UBA` sont des structures d'automates similaires aux NDD, mais qui sont plus compactes ([Ler03])) :

Théorème 5 *Étant donnée f une translation convexe, on peut calculer un* `UBA` *représentant la relation de transition accélérée en temps et espace bornés par le carré de la taille de l'*`UBA` *représentant* Φ*.*

Le résultat provient du fait que la convexité permet de considérer la garde uniquement au début et à la fin de l'itération de la fonction, puisqu'automatiquement les points intermédiaires vérifient la garde.

En pratique dans l'outil [FAS], cette accélération convexe est utilisée pour diminuer la complexité de l'accélération qui est plus coûteuse dans le cas général (ici elle est juste exponentielle en le nombre de variables).

Remarque 4 La sous-classe des *translations positives*, *i.e.* des translations avec des gardes closes par le haut, est aussi considérée dans [Bar05].

3.3 L'accélération plate - Outil FastER

Pour prendre en compte les itérations imbriquées, il est utile de définir différents types d'accélération ([Bar05] et [BALS05]) :

Définition 16 — Accélérations

Soit τ une transition affine gardée.

- Une *accélération de boucle* est une procédure qui calcule la fonction $(\tau, X) \mapsto \tau^*(X)$ pour toute transition τ et tout ensemble X.
- Une *accélération plate* est une procédure qui calcule la fonction $(w, X) \mapsto w^*(X)$ quel que soit X et quel que soit w un mot sur l'alphabet T des transitions. Par exemple, $(\tau_1.\tau_2.\tau_3)^*(X)$.
- Une *accélération globale* est une procédure qui calcule la fonction $(e, X) \mapsto e^*(X)$ quel que soit X et quelle que soit e une expression régulière sur les transitions. Par exemple, $\big((\tau_1 + \tau_2^* + \tau_3)^*.\tau_4\big)(X)$.

Ces notions sont introduites dans le cas général des systèmes L-définissables, c'est-à-dire des systèmes qui offrent des représentations symboliques qui ont des propriétés intéressantes (la représentation par polyèdre fait partie de ces représentations, dans le cas dit « faible » car l'union n'est pas une union exacte).

Le résultat d'accélération de [FL02] et [Boi99] (voir la section 3.2.2, page 39) peut donc se reformuler de la manière suivante :

Proposition 3 *Les systèmes à compteurs affines à monoïdes finis admettent une accélération plate.*

La procédure `POST_STAR` est donnée dans l'article et construit une formule de Presburger codant la relation f^* à partir de la donnée de la fonction de Presburger $f = (C, \overrightarrow{d}, \phi)$. Cependant, cette procédure ne suffit pas à calculer l'ensemble d'accessibilité d'un automate à compteur, car en particulier les boucles peuvent être imbriquées. Les automates plats sont alors introduits comme objets d'étude intermédiaire intéressants, car le calcul de $Post^*$ y devient décidable :

Définition 17 — Système plat ([CJ98],...)

> Un système est plat si pour tout sommet q de son graphe de contrôle, il existe au plus un cycle élémentaire (*i.e.* chaque sommet est visitée au plus une fois, sauf l'entrée) contenant q.

Le problème est qu'en général, les systèmes ne sont pas plats. On définit alors ce qu'est un aplatissement :

Définition 18 — Aplatissement ([Bar05])

> Un *aplatissement* d'un système (d'ensemble d'états Q) est un système plat S' (d'ensemble d'états Q') tel qu'il existe une fonction de repliage $z : Q' \to Q$ telle que si (q'_1, τ, q'_2) est une transition du système plat, alors $(z(q'_1), \tau, z(q'_2))$ est une transition du système initial.

Un exemple de système et un aplatissement peut être trouvé à la figure 3.2. Sur cette figure, la fonction $\begin{cases} q'_i & \mapsto q_i \\ q"_i & \mapsto q_i \end{cases}, i \in \{1,2\}$ est un aplatissement du système de gauche car elle préserve les sources et destinations des transitions.

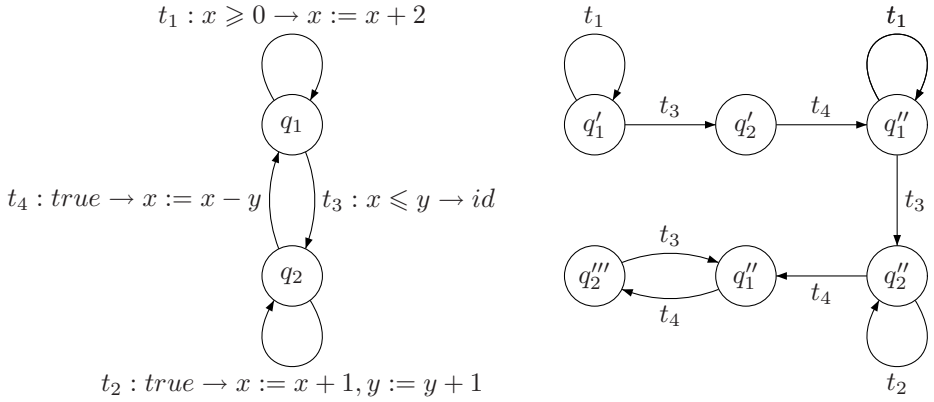


Fig. 3.2 – Exemple de système aplati ([Bar05])

Il peut être intéressant d'obtenir un L-aplatissement d'un système, c'est-à-dire un aplatissement à partir duquel on peut calculer l'ensemble d'accessibilité du système non plat (c'est-à-dire que l'ensemble des chemins du système plat permet d'obtenir tous les comportements du système non plat, et ceci, sans boucles imbriquées, voir la définition plus précise dans la thèse [Bar05]). Ce qui est intéressant est qu'on a en plus la réciproque :

Proposition 4 *Un système est L-aplatissable ssi on peut calculer l'ensemble d'accessibilité en utilisant seulement l'accélération plate.*

(L'aplatissement de la figure 3.2 n'est pas un L-aplatissement car il ne permet pas de calculer l'ensemble d'accessibilité du système : il existe une séquence de transitions partant de q_1 non réalisable par l'automate plat de droite, et qui fournit une valuation des variables x, y non atteignable par une autre séquence de transitions).

Le problème de savoir si un automate possède un L-aplatissement est connu pour être indécidable. Cependant, quelques systèmes pour lequels il existe une procédure de décision ont été étudiés. C'est par exemple le cas des VASS (Vector Addition Systems with States) à deux compteurs ([FS00a]), des automates temporisés ([CJ99]), etc...

3.3.1 Mise en œuvre dans l'outil FastER ([FAS])

Dans [BALS05] et [Bar05], le théorème suivant fournit un semi-algorithme de calcul des ensembles d'états accessibles dans un système (une RLRE ou « restricted linear regular expression » sur un alphabet A est une expression régulière de la forme $u_1^* \dots u_n^*$ avec les u_i mots sur A) :

Théorème 6 *Un système est L-aplatissable ssi pour tout ensemble de valeurs X il existe une RLRE ρ sur l'alphabet des transitions telle que $Post^*([\![X]\!]) = Post(\rho, [\![X]\!])$.*

Le semi-algorithme proposé pour le calcul de $Post^*(X_0)$ est alors le suivant (on suppose que l'on dispose des fonctions `POST` et `POST_STAR` qui implémentent respectivement la fonction Post et l'accélération plate) :
- Initialisation $X \leftarrow X_0$
- Tant que $\texttt{POST}(X) \subsetneq X$, choisir une nouvelle RLRE w et faire $X \leftarrow \texttt{POST\_STAR}(w, x)$.
- Retourner X.

Si cette procédure termine, alors la sortie est $Post^*(X_0)$. Cette procédure termine pour toute entrée X_0 ssi le système est L-aplatissable (voir la proposition 4). On ne peut donc pas savoir a-priori si cette procédure termine.

Dans le cas de FastER et des systèmes à compteurs, la procédure est mise en œuvre de la façon suivante :
- Les formules de Presburger sont codées à l'aide d'automates, les `UBA`.
- On énumère les RLRE à taille fixée. Un compteur permet de passer à la taille supérieure si l'algorithme n'a pas terminé.
- Des *réductions* ([Bar05]) génériques (valables quels que soient les cadres symboliques) sont utilisées pour diminuer le nombre de RLRE de taille k fixée à considérer : la réduction par conjugaison (si $\tau_1\tau_2\tau_3$ a été considérée, ne pas considérer $\tau_2\tau_3\tau_1$) et la réduction par commutation (si τ_1 et τ_2 commutent, alors on enlève $\tau_1\tau_2$ et $\tau_2\tau_1$ de l'énumération, car considérer séparément τ_1^* et τ_2^* suffit).
- Une réduction spécfique aux systèmes à compteurs est mise en œuvre : la réduction par union ([FL02]). Dans le cas favorable, c'est-à-dire dans le cas où le système considéré est à monoïde fini, cette réduction permet de réduire exponentiellement le nombre de transitions à considérer. Si deux transitions ont la même action, on prend comme représentant l'union des deux, c'est à dire la transition de garde $g_1 \cup g_2$ et d'action commune. On obtient ainsi une partition de l'ensemble des transitions avec un représentant par classe. On ne considère ensuite qu'un seul représentant par classe dans l'énumération des RLRE.

Accélération de boucles imbriquées dans FastER La *réduction par union* permet de calculer l'accélération de certaines boucles imbriquées, et dans [Bar05] on peut trouver un exemple d'automate non Presburger-aplatissable et qui est calculable en utilisant les stratégies décrites plus haut, ce qui montre que l'on sort strictement du cadre des automates plats.

Cependant, ces cas semblent en petit nombre et le traitement des boucles imbriquées est en fait réalisé dans l'énumération des RLRE. Sur la Figure 3.3, l'énumération des RLRE donne sans plus d'hypothèses sur les fonctions τ_i : τ_1, puis τ_2, puis $\tau_1.\tau_2$, $\tau_2.\tau_1$, puis τ_1^2, *etc.* L'imbrication quelconque des deux fonctions est donc traitée si l'énumération se stabilise, cependant cette façon de procéder ne garantit aucun résultat de terminaison.

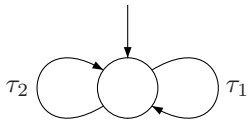


Fig. 3.3 – Traitement de deux boucles imbriquées

Inconvénients de FastER Cette approche présente des inconvénients :

- Elle ne fournit qu'un semi-algorithme. Aucun résultat intermédiaire ne peut être utilisé (il n'y a pas de résultat de sur-approximation par exemple).
- Dans le cas de FastER, ce sont les `UBA` qui sont choisis pour coder les ensembles Presburger-définissables. Cependant, les opérations sur les `UBA` sont coûteuses. En particulier, l'algorithme de [BFL04] fournissant un `UBA` représentant f^* à partir de la fonction f a une complexité 5-EXP en m le nombre de variables, et 3-EXP en la taille de l'automate binaire représentant la garde. D'autres résultats intéressants sur les complexités sont résumés dans [BB03] et [Bar05]. Ces complexités proviennent essentiellement du fait du codage *en nombre entier*.
- Le calcul de la procédure `POST_STAR` n'est donné que dans le cas des systèmes à monoïde fini ou dans certaines sous-classes accélérables comme les translations convexes, mais que se passe-t-il si la fonction `POST_STAR` n'est pas définie pour tout couple RLRE/ensemble de valeurs ?
- Que se passe-t-il si le programme n'est pas affine/déterministe ? Dans le cas de l'étude du protocole TTP ([Bar05] et [BFL04]), une boucle englobante est supprimée afin de rendre l'analyse du système possible. Il semble que rien n'est dit dans le cas général, cette technique semblant difficilement se généraliser à des exemples significatifs.
- L'énumération des RLRE ne prend pas en compte la structure du graphe à analyser. La stratégie qui consiste à d'abord accélérer les boucles les plus internes doit être codée à la main (voir par exemple l'étude du protocole TTP dans [Bar05]).

L'article [BCALS06] utilise lui-aussi un déroulement des boucles imbriquées, mais avec une stratégie un peu différente. Ne sont déroulées que les boucles dont toutes les boucles internes ont un nombre d'exécution borné. Pour détecter de telles boucles, on calcule une matrice de « différence » entre deux exécutions successives de la boucle. Des conditions algébriques sur cette matrice de différence permettent de savoir si cette matrice est itérable ou non. L'effet de l'itération d'une boucle n'est pas effectivement calculé car on cherche à générer des conditions d'équité sur les variables du programme. Cependant, les classes de transitions traitées par cet article sont intéressantes car elles permettent des gardes de la forme $x + 2 \leqslant d.d'$ avec d et d' deux paramètres. Ces gardes étant non convexes, elles sont exclues de notre cadre d'étude. En revanche, la matrice de transformation doit contenir au plus une valeur non-nulle par ligne (un 1), ce qui est plus restrictif que les transformations acceptées dans FastER.

3.4 Mise en œuvre chez Boigelot-Wolper

Contrairement à la mise en œuvre précédente, Boigelot et Wolper prennent en compte la structure du graphe de contrôle dans l'analyse de leurs systèmes.

3.4.1 Un premier prototype

La mise en pratique des résultats dans [BW94] se fait de la façon suivante :
- Certains cycles du graphe sont détectés (restrictions aux cycles correspondant aux têtes de composantes connexes).
- Si possible, une *méta-transition* est associée à chaque cycle.
- Ensuite, l'algorithme classique d'exploration des états accessibles par parcours (en profondeur ou en largeur) est « accéléré » lorsqu'une méta-transition a été calculée pour un cycle.

Remarque 5 Lors de l'ajout d'une méta-transition calculée pour un cycle, la définition faible d'une méta transition (voir définition 3.2.1) impose de garder les transitions qui composent le cycle, sinon on ne calcule qu'une approximation inférieure des états accessibles.

Un premier prototype a été réalisé et permettait de calculer l'ensemble d'accessibilité d'un exemple paramétré d'ascenseur (avec plus de 200000 états atteignables pour la valeur 50 du paramètre), le temps d'analyse étant indépendant de la valeur du paramètre. Dans la suite, ce prototype semble ne pas avoir été mis à jour.

Remarque 6 Le cas des instructions imbriquées est traité dans le cas simple où la boucle la plus interne est « déterministe », c'est-à-dire que le nombre de ses itérations est complètement déterminé par les valeurs initiales des variables. Les auteurs calculent alors l'instruction équivalente à la boucle et ensuite traitent la boucle supérieure comme une boucle normale.

3.4.2 L'outil LasH

LasH utilise les `NDD` (ou d'autres représentations similaires) pour représenter les automates de Presburger, et utilise aussi des techniques d'accélération pour vérifier les systèmes à compteurs, restreints aux gardes convexes. L'utilisateur doit néanmoins spécifier les cycles à accélérer. En cours de développement, un package permettra de vérifier des systèmes temporisés ou hybrides avec le même type de techniques (voir [BLW03]).

3.5 Autres techniques d'accélération

TReX L'article [DFV05] effectue une comparaison intéressante entre l'outil FastER et l'outil **TReX** ([TRE]). Les gardes traitées dans **TReX** sont des PDBM (« Parametric Difference bound Matrices », une extension des DBM permettant de prendre en compte les contraintes de la forme $x - y \leqslant t$, avec t un paramètre). Ces gardes sont donc strictement moins expressives que les Presburgers. Les fonctions permises dans l'outil sont des « fonctions simples affines » : $c_i := c_j + t$ avec t un terme de Presburger sur les paramètres (il n'y a pas de restriction sur le monoïde). La représentation interne (EPDBM) ([AAB00]) dans **TReX** est plus compacte que celle utilisée dans FastER puisque ce sont des matrices couplées à des formules arithmétiques sans quantificateur.

L'accélération dans TreX consiste à regarder deux itérations de suite pour calculer une différence dans les formules décrivants les états, et si cette différence (dérivée) est la même sous certaines conditions, on itère cette différence (en rajoutant une variable d'itération, mais il n'est pas dit comment on la quantifie).

L'inconvénient de la méthode est que tester l'égalité des dérivées peut être indécidable.

Conclusion du chapitre

Les méthodes d'accélération introduites dans ce chapitre offrent des résultats théoriques intéressants, cependant elles ne répondent pas telles quelles à notre objectif :

- Les algorithmes ne s'adressent qu'à une classe réduite de programmes, pour FastER les automates plats dont les fonctions de transition sont « à monoïde fini ».
- Les algorithmes font appel à un encodage d'ensembles d'entiers sous la forme d'automates, et les algorithmes associés sont coûteux.
- Enfin, les algorithmes d'accélération ne peuvent s'appliquer aux polyèdres, il s'agit en effet d'accélérer l'effet des boucles sur des ensembles de nombres *entiers*, tandis que les ensembles que nous manipulons sont *denses*.

Dans le chapitre suivant, nous faisons un état de l'art des méthodes qui ont été développées jusqu'à présent pour améliorer la précision d'une Analyse des Relations Linéaires.

Chapitre 4

Amélioration de la précision lors d'une Analyse de Relation Linéaires

L'Analyse des Relations Linéaires introduite au chapitre 2 a pour principal avantage le fait que les calculs terminent quels que soient les programmes analysés. En revanche, le polyèdre calculé peut n'être qu'une sur-approximation très peu précise du plus petit point fixe cherché. Ce chapitre présente donc diverses méthodes qui ont été proposées dans le cadre de cette analyse afin d'améliorer la précision des calculs. Nous verrons les inconvénients de ces méthodes, puis présenterons deux méthodes qui utilisent une notion que nous appellerons accélération abstraite*, et nous verrons en quoi ces méthodes sont intéressantes.*

4.1 La séquence décroissante

À la fin de l'analyse, si le point-fixe obtenu n'est pas assez précis, on peut réaliser *une séquence décroissante* ([CC76]) : on améliore la précision du résultat en continuant le calcul de la séquence, mais cette fois sans rappliquer l'opérateur d'élargissement (voir la figure 4.1). La séquence avec élargissement converge vers un post-point fixe $\hat{x}$, alors comme $\text{lfp}(F) \sqsubseteq \hat{x}$, par croissance de F on obtient $F(\text{lfp}(F)) = \text{lfp}(F) \sqsubseteq F(\hat{x})$ et donc $F(\hat{x})$ est encore une solution correcte. Si la solution $\hat{x}$ n'est pas un point fixe, on a $F(\hat{x}) \sqsubset \hat{x}$, c'est-à-dire que $F(\hat{x})$ est une *meilleure* solution que $\hat{x}$. Ainsi, contrairement à la séquence croissante, tous les termes de cette séquence décroissante sont corrects, *i.e.* sont des sur-approximations supérieures du plus petit point fixe, et sont de plus en plus précises.

Pour éviter une itération infinie, [CC76] propose un opérateur similaire à l'élargissement (« rétrécissement »). Cependant, la plupart du temps, on préfère appliquer la séquence décroissante un nombre de fois k fixé à l'avance. La séquence décroissante s'arrête alors soit lorsque l'on a trouvé un point-fixe (qui n'est pas forcément le plus petit), soit lorsque l'on a fait k pas de calcul.

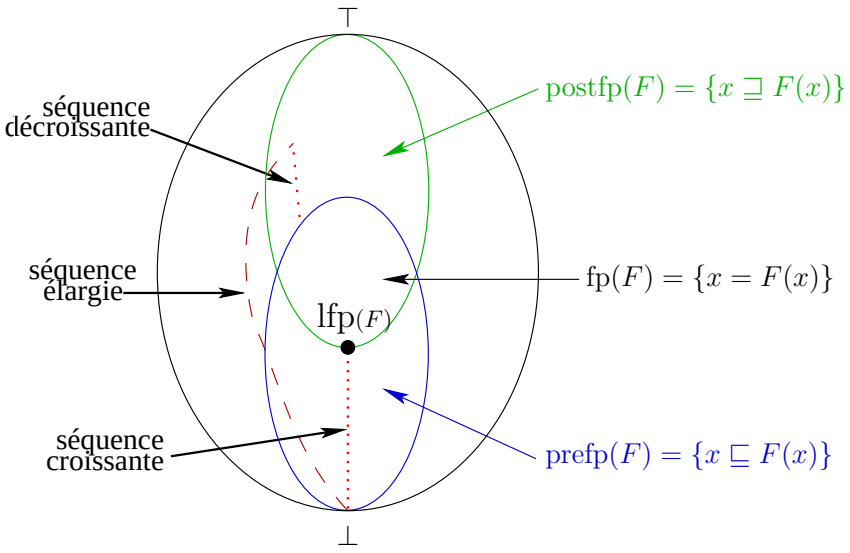


FIG. 4.1 – Séquence décroissante

EXEMPLE 4.1 *Nous reprenons ici l'exemple de l'article [CC76]. La figure ci-dessous montre le système obtenu si on se place dans le treillis des intervalles :*

$X_0 = [-\infty, +\infty]$

x := 0

$X_1 = X_0[x := 0]$

while

$X_2 = X_1 \sqcup X_4$

x<100 do

$X_3 = X_2 \sqcap [-\infty, 99]$

x := x+1

$X_4 = X_3[x := x + 1]$

end

$X_5 = X_2 \sqcap [100, +\infty]$

Nous nous intéressons au calcul d'un invariant pour le point de contrôle 2 *(qui est un point d'élargissement), cet invariant* X_2 *satisfaisant l'équation de point-fixe suivante :*

$$X_2 = [0,0] \sqcup \Big(\big(X_2 \cap [-\infty, 99] \big)[x := x + 1] \Big)$$

On trouve successivement :

- $X_2^0 = \bot$;
- $X_2^1 = \bot \nabla [0,0] = [0,0]$;
- $X_2^3 = [0,0] \nabla \big([0,0] \sqcup [1,1] \big) = [0, +\infty]$;
- $X_2^4 = X_2^3$.

En propageant on trouve $X_5^4 = [100, +\infty]$. *Appliquons maintenant la séquence décroissante : on rapplique l'équation sans appliquer l'opérateur d'élargissement :* $X_2^4 = [0,0] \sqcup \Big(\big(X_2^3 \cap$

$[-\infty, 99])[x := x + 1]\Big) = [0, 100]$, *ensuite la séquence décroissante a convergé vers* $X_5 = [100, 100]$, *ce qui est le plus petit point fixe.*

4.2 Retarder l'application de l'opérateur d'élargissement

Quelquefois le premier terme de la séquence d'élargissement n'est pas assez « représentatif » de la séquence toute entière. [Hal93] propose de calculer k termes de la séquence croissante, puis d'appliquer l'opérateur d'élargissement à partir de la $k+1$-ième itération afin de gagner en précision.

Plus formellement, au lieu de calculer la séquence $X_0 = \bot$, $X_1 = X_0 \nabla F(X_0)$, $X_2 = X_1 \nabla F(X_1)$, on fixe $k > 0$ et on calcule la séquence :

$$X_n = \begin{cases} \bot & \text{si } n = 0 \\ F(X_{n-1}) & \text{si } n \leqslant k \\ X_{n-1} \nabla F(X_{n-1}) & \text{si } n > k \end{cases}$$

En pratique, le nombre k est souvent un paramètre de l'analyseur. Si une analyse est trop grossière, l'utilisateur peut alors augmenter le paramètre k afin d'essayer d'augmenter la précision de cette analyse ([Hal79b],[BCC+03], [CGG+05]). Cependant, le principal inconvénient de la méthode est que les valeurs abstraites calculées lors de la phase « séquence exacte » ($n \leqslant k$) sont de plus en plus complexes. Dans le cas des polyèdres, le nombre de sommets ou de rayons peut augmenter fortement et causer une saturation de la mémoire.

Remarque 7 Le fait de retarder l'application de l'opérateur d'élargissement peut être vu comme une certaine forme de « déroulement des boucles » ([Gou01]). En effet, sur la figure 4.2 (à gauche), retarder l'application avec $k = 2$ revient à faire la transformation vers le graphe de droite.

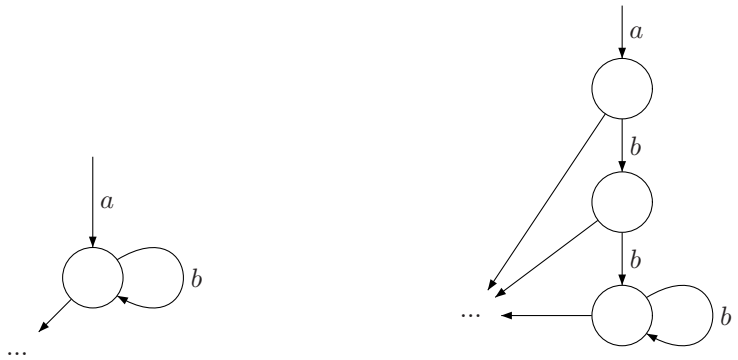


Fig. 4.2 – Déroulement des boucles

Exemple 4.2 *Considérons l'automate de gauche sur la figure 4.3.*

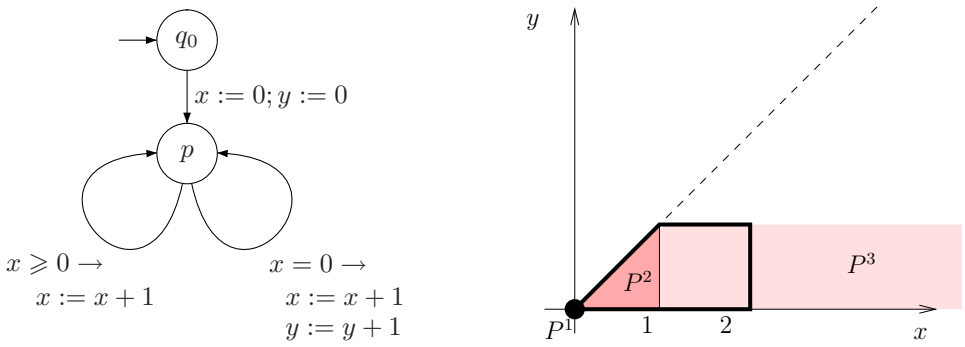


FIG. 4.3 – Automate et invariants calculés

À la première itération, le polyèdre associé au point de contrôle p est $P^1 = \{x = y = 0\}$. *À la deuxième, on trouve* $P^2 = \{x = y = 0\} \nabla \{0 \leqslant y \leqslant x \leqslant 1\} = \{0 \leqslant y \leqslant x\}$ *(la moitié inférieure du premier quadrant), et à l'étape suivante l'analyse a convergé vers* P^2. *En revanche, si l'on garde (délai* $k = 1$*)* $P^2 = \{0 \leqslant y \leqslant x \leqslant 1\}$ *(en foncé sur la figure), on trouve à l'itération suivante :* $P^3 = P^2 \nabla \{0 \leqslant x \leqslant 2; y \leqslant x; y \leqslant 1\}$ *(le deuxième argument de l'élargissement est le polyèdre surligné), soit* $P^3 = \{0 \leqslant x; 0 \leqslant y \leqslant x; y \leqslant 1\}$ *(polyèdre plus clair infini sur l'axe des x), puis* $P^4 = P^3$. *Ainsi l'invariant trouvé est plus précis.*

De manière générale, retarder l'application de l'opérateur d'élargissement permet de prendre en compte les transitions qui ne sont pas permises avant un certain nombre d'itérations, et qui induisent donc un comportement « non régulier ».

4.3 Amélioration de l'opérateur d'élargissement

L'article [BHRZ03] propose un cadre formel pour le développement de nouveaux opérateurs d'élargissements à partir d'opérateurs préexistants. La notion de « croissance limitée », est introduite :

DÉFINITION 19 — Relation $\curvearrowright$ ([BHRZ03])

Soient P_1 et P_2 deux polyèdres vérifiant $P_1 \subseteq P_2$, les systèmes de contraintes $cons(P_i)$ étant sous forme minimale et les systèmes de générateurs (V_i, R_i) sous forme orthogonale (obtenue par exemple en utilisant la méthode de Gram-Schmidt), alors on définit la relation $P_1 \curvearrowright P_2$ de la façon suivante :

$$P_1 \curvearrowright P_2 \stackrel{def}{=} \begin{cases} \text{si} & \dim(P_2) > \dim(P_1) \\ \text{ou} & \text{codim}(P_2) > \text{codim}(P_1) \\ \text{ou} & \sharp cons(P_1) > \sharp cons(P_2) \\ \text{ou} & \sharp cons(P_1) = \sharp cons(P_2) \text{ et } \sharp V_1 > \sharp V_2 \\ \text{ou} & \sharp cons(P_1) = \sharp cons(P_2) \text{ et } \sharp V_1 = \sharp V_2 \text{ et } \kappa(R_1) >> \kappa(R_2) \end{cases}$$

Autrement dit $P_1 \curvearrowright P_2$ dans les cas suivants :
- la dimension du polyèdre P_2 est strictement supérieure à celle du polyèdre P_1 ;
- la codimension du polyèdre P_2 est strictement supérieure à celle du polyèdre P_1 (la codimension d'un polyèdre est la dimension du plus grand sous-espace inclus dans ce polyèdre) ;
- le nombre de contraintes de P_1 est strictement supérieur au nombre de contraintes de P_2.

- le nombre de contraintes de P_1 est égal au nombre de contraintes de P_2 et le nombre de sommets de P_1 est strictement supérieur au nombre de sommets de P_2.
- les contraintes et les sommets sont en nombre identiques et les multiensembles $\kappa(R_i)$ formés des nombres de coordonnées non nulles des vecteurs de R_i vérifient : $\kappa(P_1) >> \kappa(P_2)$ (au sens de l'ordre multiensemble).

Les contraintes de minimalité et de forme orthogonale assurent que cette relation est non ambiguë. De plus, la relation est construite afin d'assurer le résultat suivant :

PROPOSITION 5 ([BHRZ03],TH. 1 PAGE 7) *Une suite de polyèdres vérifiant* $\forall i \geqslant 0$, $P_i \curvearrowright P_{i+1}$ *est stationnaire, i.e.* $\curvearrowright$ *est un ordre bien fondé.*

La notion de croissance limitée, variante de celle de [CF99], généralise ainsi la notion d'élargissement et garantit que le nombre d'itérations nécessaire à la convergence est fini.

L'article propose ensuite une manière systématique de construire un nouvel élargissement à l'aide de l'opérateur d'élargissement standard et d'heuristiques h_i qui doivent être des « opérateurs majorants » (c'est-à-dire que $P_1 \sqcup P_2 \subseteq h_i(P_1, P_2)$), dont l'opérateur ∇_B final est une instance :

$$P_1 \nabla_B P_2 \stackrel{def}{=} \begin{cases} P_2 & \text{si } P_1 \curvearrowright P_2 \\ h_c(P_1, P_2) & \text{si } P_1 \curvearrowright h_c(P_1, P_2) \\ h_p(P_1, P_2) & \text{si } P_1 \curvearrowright h_p(P_1, P_2) \\ h_r(P_1, P_2) & \text{si } P_1 \curvearrowright h_r(P_1, P_2) \\ P_1 \nabla P_2 & \text{sinon.} \end{cases}$$

Les heuristiques mises en œuvre sont les suivantes :

- Si $P_1 \curvearrowright P_2$, l'analyse décrit une chaîne finie, il n'y a donc pas lieu d'élargir à ce stade. Cette heuristique est suggérée dans [CC92b].
- L'opérateur h_c consiste à ajouter aux contraintes de $P_1 \nabla P_2$ des contraintes qui sont des combinaisons convexes des contraintes de Q qui vérifient la propriété $\mathcal{P}$ suivante : $c \in \mathcal{P}$ ssi il existe un point $p \in P_1$ qui ne sature aucune des contraintes de $P_1 \nabla P_2$ mais qui sature au moins une contrainte de P_2. Le choix de la combinaison est laissée libre. L'article [CF99] propose par exemple une sorte de « moyenne normée » des contraintes vérifiant $\mathcal{P}$.
- L'opérateur h_p propose de voir tout nouveau point de P_2 comme l'évolution d'un point de P_1, et donc d'ajouter le nouveau rayon $v_2 - v_1$. Le nouveau polyèdre obtenu est intersecté avec le polyèdre $P_1 \nabla P_2$. Cette heuristique est illustrée à la figure 4.4, partie de gauche.
- L'opérateur h_r propose de voir tout nouveau rayon $r_2 \in R_2$ comme une évolution d'un rayon de R_1. Cette idée est très similaire à l'idée de l'élargissement standard, sauf que la rotation du rayon est faite jusqu'à ce qu'une des composantes non nulles du vecteur devienne 0. Cette idée est illustrée à la figure 4.4, partie de droite. Le principal inconvénient de cette heuristique est de favoriser la base canonique par rapport aux autres bases. Le résultat de l'élargissement sera donc différent si on change la base de calcul.

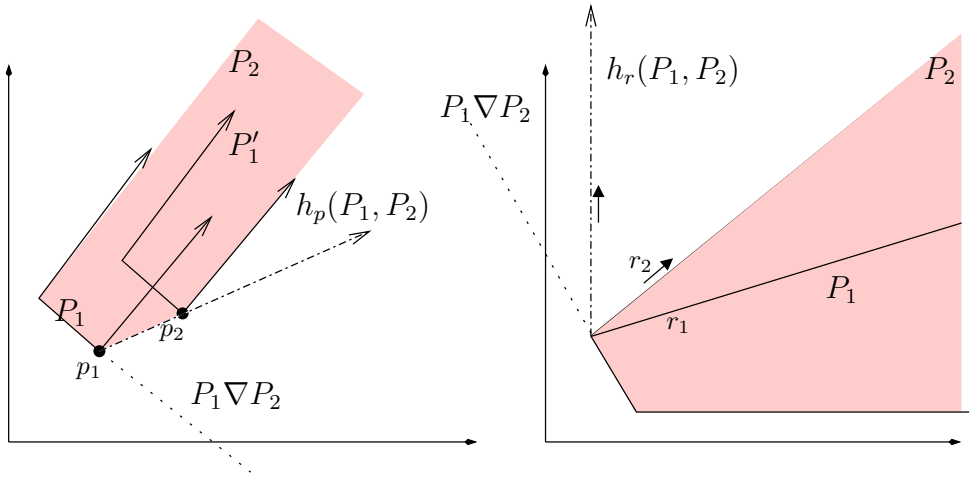


FIG. 4.4 – Heuristiques h_p et h_r

PROPOSITION 6 ([BHRZ03]) *L'opérateur d'élargissement ∇_B est au moins aussi précis que l'opérateur ∇.*

Cependant, le fait que le résultat soit plus petit en un coup ne garantit pas que le point fixe obtenu soit meilleur. En effet, l'opérateur d'élargissement standard n'est pas monotone. L'expérimentation fournie dans l'article montre que les pertes de précision par rapport à l'élargissement standard sont assez rares en pratique. Cependant, l'étude [SSM04] montre une augmentation non négligeable du temps de calcul total.

4.4 Prise en compte du programme dans l'analyse

4.4.1 Élargissement limité par un ensemble de contraintes

Dans [HPR97], l'élargissement proposé consiste à identifier *une fois pour toutes* dans le programme un ensemble fini $\mathcal{U}$ de contraintes susceptibles d'être invariantes dans un état d'élargissement, et à ajouter aux contraintes de $P \nabla Q$ les contraintes de $\mathcal{U}$ satisfaites à la fois par P et par Q.

Il s'agit ensuite de choisir convenablement cet ensemble de contraintes. Classiquement ([Hal93]), on choisit comme ensemble de contraintes pour un point de contrôle donné l'ensemble des contraintes qui permettent de rester dans cet état de contrôle. Nous allons illustrer ceci dans l'exemple suivant.

EXEMPLE 4.3 *Dans [Hal93], on modélise le compteur de vitesse d'une voiture par un automate interprété avec deux entrées booléennes,* `metre` *et* `seconde`*. L'automate est représenté sur la figure 4.5 : au point de contrôle k_1, la voiture reçoit les signaux* `metre` *et* `seconde` *de façon non simultanée. Selon le signal reçu, les variables d, t et v sont incrémentées ou remises à zéro :*

- *la distance parcourue : la variable d compte le nombre de signaux* `metre` *reçus depuis le début ;*
- *le temps écoulé : la variable t stocke le nombre de signaux* `seconde` *reçus depuis le début ;*

– *la vitesse instantanée : la variable v est un compteur de signaux* `metre` *et est réinitialisée à chaque signal* `seconde`.

L'automate détermine aussi si la voiture roule trop vite, s'arrête, ou percute le mur.

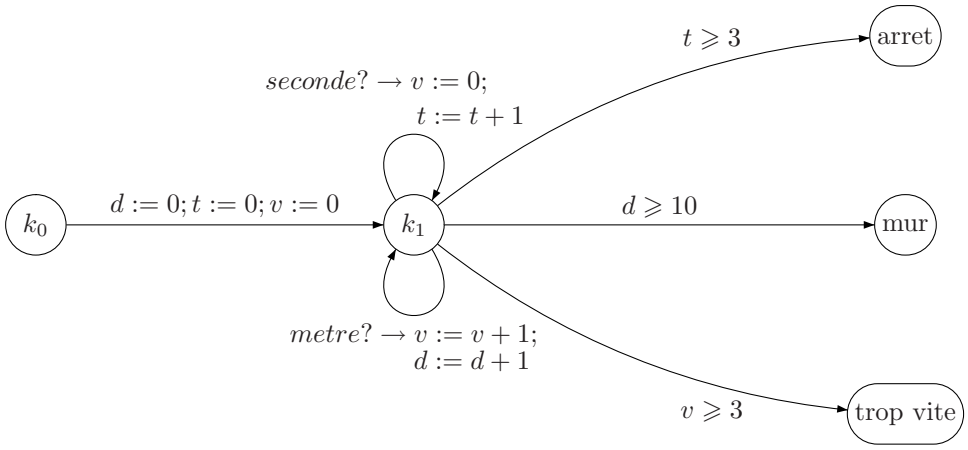


FIG. 4.5 – Automate interprété avec entrées booléennes modélisant une voiture

Comme dans [Mer05], l'automate est légèrement modifié pour ne prendre en compte que les aspects numériques (figure 4.6). Le but est ici de vérifier que le point de contrôle P_{mur} n'est jamais atteint.

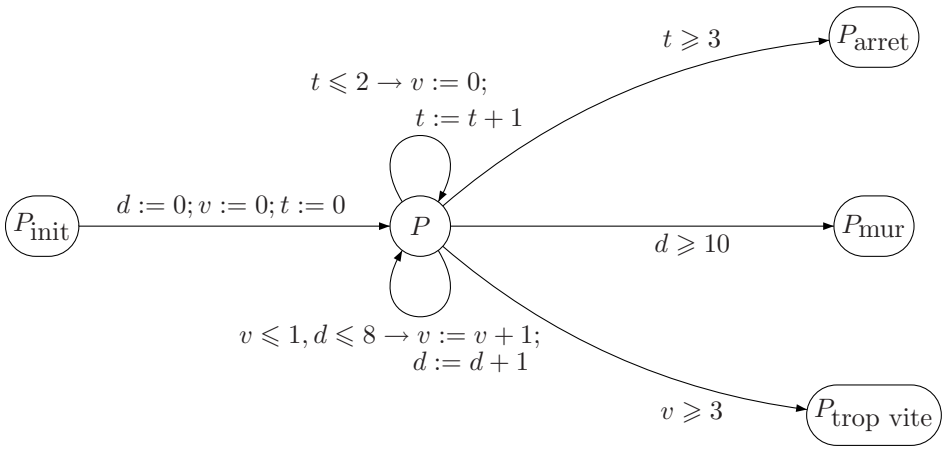


FIG. 4.6 – Exemple de la voiture

On choisit comme ensemble de contraintes au point de contrôle P (seul point où l'on applique l'élargissement) l'ensemble $\mathcal{C} = \{v \leqslant 2, d \leqslant 9, t \leqslant 3\}$. En effet, ces contraintes ont de bonnes chances d'être des invariants de ce point de contrôle car ce point est quitté lorsque ces conditions sont violées. Voici les différentes étapes de l'analyse :

– *Tous les polyèdres sont initialisés à $\bot$, la première itération fournit donc :*

$$P^1 = \{d = 0; t = 0; v = 0\}$$

– *Deuxième itération :*

$$\begin{aligned} P^2 &= P^1 \nabla (\{d = t = v = 0\} \sqcup \{d = v = 1; t = 0\} \sqcup \{d = v = 0; t = 1\}) \\ &= P^1 \nabla \{0 \leqslant d = v \leqslant 1; 0 \leqslant t \leqslant 1; t + d \leqslant 1\} \end{aligned}$$

À ce stade, l'élargissement standard fournirait $P^2 = \{0 \leqslant v = d; 0 \leqslant t;\}$, mais toutes les contraintes de l'ensemble $\mathcal{C}$ sont satisfaites par chacun des deux opérandes de l'élargissement, donc on obtient : $P^2 = \{0 \leqslant v = d \leqslant 2; 0 \leqslant t \leqslant 3\}$.

– *Troisième itération (en gras on voit que l'on a trouvé la « relation vitesse » :*

$$\begin{aligned} P^3 &= P^2 \nabla \{0 \leqslant v \leqslant d \leqslant 2t + v; t \leqslant 3; d \leqslant 2\} \\ &= \{0 \leqslant v \leqslant \boldsymbol{d \leqslant 2t + v}; t \leqslant 3; d \leqslant 2\} \end{aligned}$$

– *Quatrième itération :*

$$P^4 = \{0 \leqslant v \leqslant d \leqslant 2t + v; t \leqslant 3; v \leqslant 2\}$$

– *Stabilisation à la cinquième itération. On peut maintenant calculer les différents invariants des autres états, en particulier on trouve $P_{mur} = P \cap \{d \geqslant 9\} = \bot$.*

L'analyse a bien prouvé que l'état P_{mur} n'est jamais atteint. L'élargissement limité a permis d'améliorer la précision de l'invariant associé à l'état de contrôle P. On aurait pu aussi retarder l'application de l'opérateur d'élargissement de 4 itérations.

Le principal inconvénient de la méthode est que propager les conditions de sortie des boucles sur le chemin a un coup exponentiel en la taille de l'automate. En général, on se contente d'un sous-ensemble raisonnable de ces conditions « upto ».

4.4.2 Élargissement avec bornes limites

L'idée de limiter l´élargissement par un ensemble de contraintes est reprise dans [SK06]. L'article propose de calculer des inégalités qui ne sont pas satisfaisables à l'intérieur d'une boucle. Contrairement à l'élargissement limité, les « bornes limites » sont calculées dynamiquement au cours de l'analyse. Le calcul d'un invariant associé à un point de contrôle peut dépendre de plusieurs transitions entrantes, et ces transitions ne portent pas toutes forcément une contribution non vide. L'idée est de prendre en compte ces gardes pour « deviner » quand elles vont être permises.

En pratique, au point d'élargissement considéré, un ensemble de « bornes limites » (landmarks) caractérise les inégalités non satisfaisables qui sont collectées à l'intérieur de la sous-composante fortement connexe associée : ces bornes sont de la forme $< c, d_1, d_2 >$ avec c une contrainte, d_1 la distance du polyèdre courant à la contrainte c (au sens de la distance euclidienne), et d_2 stocke la même distance calculée avec le polyèdre obtenu au même point lors d'une itération précédente, la mise à jour n'étant effectuée que lorsque les polyèdres grossissent strictement. Ensuite, l'application de l'opérateur d'élargissement est modifiée pour prendre en compte les informations des bornes limites : on suppose que les distances suivent une suite

arithmétique pour estimer le nombre d'itérations à faire pour satisfaire la contrainte c, et ensuite on prend le minimum de ces estimations. Si ce minimum n'existe pas, l'algorithme revient à appliquer l'élargissement standard.

Le principal intérêt de la méthode est qu'elle permet de traiter des gardes non convexes du type $i \neq 0$, l'inconvénient est que collecter les bornes limites dynamiquement coûte cher, en particulier les calculs de distances utilisent de la programmation linéaire à chaque itération (minimisation d'une expression linéaire sur un polyèdre). De plus, la convergence d'une analyse n'est pas garantie car le résultat de $P \nabla_L Q$ est calculé à l'aide des contraintes de P et le nombre de contraintes n'est pas strictement décroissant.

4.4.3 Heuristique du « nouveau chemin »

L'opérateur d'élargissement est construit de façon à prendre en compte les régularités du programme. Si on obtient le polyèdre $\{x = y = 0\}$ à la première itération, et $\{0 \leqslant y \leqslant x \leqslant 1\}$ à la seconde, alors appliquer l'opérateur d'élargissement revient à supposer que dans la suite les variables x et y vont vérifier $\{0 \leqslant y \leqslant x \leqslant 2\}$, et donc à extrapoler vers la limite $\{0 \leqslant y \leqslant x\}$. Cependant, cette hypothèse de régularité est évidemment fausse dans le cas où un chemin de boucle devient possible à l'itération n, l'effet de ce chemin n'étant pas pris en compte avant cette itération. En conséquence, si le polyèdre associé à un point de contrôle d'élargissement dépend d'un polyèdre qui devient non vide seulement à l'étape n, le résultat obtenu peut-être amélioré :

- [Hal93] propose d'extrapoler le résultat à partir du premier polyèdre non vide : autrement dit, à l'étape n, si une transition de la composante fortement connexe (associée au nœud d'élargissement) considérée est activée (c'est-à-dire que la garde associée devient vraie), alors le premier argument de l'opérateur d'élargissement est pris comme étant le premier polyèdre non vide au point d'élargissement considéré.
- [BCC+03] propose de ne pas élargir lorsque l'on est dans un tel cas.

Exemple 4.4 *Considérons l'automate suivant :*

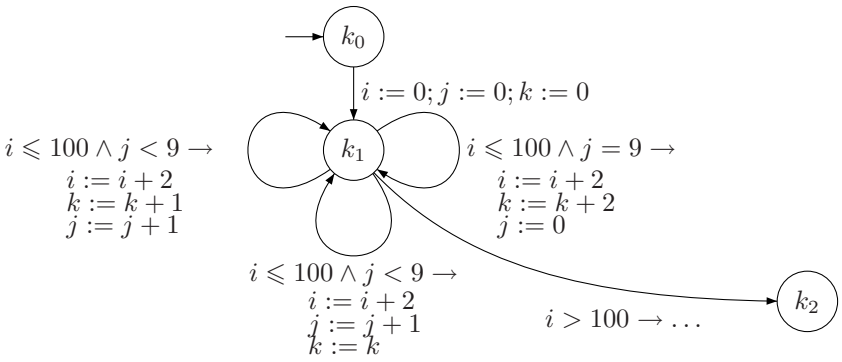


Fig. 4.7 – Exemple d'application de l'heuristique du nouveau chemin

Le dessin 4.8 représente la projection des invariants sur le plan (i, k).

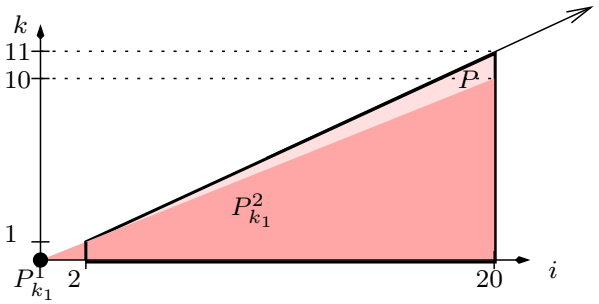


FIG. 4.8 – L'évolution des variables k et i de l'exemple

À la première itération, on obtient $P_{k_1}^1 = \{i = j = k = 0\}$. *La transition de garde* $i \leqslant 100 \wedge j = 9$ *n'est pas prise en compte à la deuxième itération, le polyèdre associé au point* k_1 *est donc (en foncé sur la figure) :*

$$\begin{aligned} P_{k_1}^2 &= \{i = 0, j = 0, k = 0\} \nabla \big(\{i = j = k = 0\} \sqcup \{i = 2, j = 1, k = 0\} \sqcup \{i = 2, j = 1, k = 1\}\big) \\ &= \{i = j = k = 0\} \nabla \{2j = i, 2k \leqslant i, i \leqslant 2, 0 \leqslant k\} \\ &= \{2j = i, k \leqslant j, 0 \leqslant k\} \end{aligned}$$

À la troisième itération, la transition de garde $i \leqslant 100 \wedge j = 9$ *est rendue possible et la contribution des trois transitions autour de* k_1 *est* $P = \{2j + 19i < 400, i + 18j \geqslant 20, i \leqslant 10k + 2j, 20k + 2j \leqslant 11i, 2j \leqslant i, i \leqslant 20\}$ *(en clair entouré par une ligne épaisse sur le dessin). Si on ne prend pas en compte l'information « nouveau chemin » on calcule :*

$$P_{k_1}^3 = P_{k_2}^2 \nabla (P \sqcup P_{k_2}^2) = \{i \geqslant 2j, 11i \geqslant 2j + 20k, i \leqslant 2j + 10k\}$$

et l'on a perdu l'information $j \geqslant 0$ *(sur la projection* i, k*, l'effet de cet élargissement donne le premier quadrant), alors que si on calcule :*

$$P_{k_1}^3 = P_{k_2}^1 \nabla (P \sqcup P_{k_2}^2) = \{i \geqslant 2j, 11i \geqslant 2j + 20k, i \leqslant 2j + 10k, j \geqslant 0\}$$

on récupère $j \geqslant 0$*, et donc l'invariant plus précis* $20k \leqslant 11i$ *(direction de la flèche). Sur cet exemple, si on décide de ne pas élargir à ce moment là, on ne trouve pas non plus l'invariant voulu.*

4.4.4 Lookahead Widening

Les nouveaux chemins sont aussi pris en compte dans [GR06], mais d'une autre manière. L'idée générale est de faire une analyse complète (croissante et décroissante) par *phase* de boucle (une phase étant la période durant laquelle aucun nouveau chemin ne devient actif).

Cette idée est mise en œuvre en utilisant deux valeurs abstraites :
- La « valeur principale » : cette valeur permet de décider quelle branche des tests prendre. Lors d'une « phase », on fait croître cette valeur *exactement*, c'est-à-dire sans appliquer d'élargissement. Cette valeur n'est élargie que lorsque la valeur pilote a convergé.
- La « valeur pilote » : cette valeur est utilisée pour calculer le polyèdre solution de la phase considérée. Lorsque cette valeur a convergé (après élargissement et séquence décroissante), elle est copiée vers la valeur principale et l'analyse de la phase suivante peut commencer.

L'article propose un schéma général d'implémentation de ce nouvel élargissement appelé « lookahead widening » à partir d'un opérateur d'élargissement existant, ce qui permet de réaliser facilement des analyses à l'aide d'un analyseur déjà existant[1]. La consistance ainsi que la convergence de la méthode sont prouvées. En pratique, l'opérateur d'élargissement sur les couples de valeurs abstraites est défini de la façon suivante :

$$< c_m, c_p > \nabla_{LA} < d_m, d_p >= \begin{cases} < c_m, c_p > & \text{si } < d_m, d_p > \sqsubseteq_{LA} < c_m, c_p > \\ < d_p, d_p > & \text{si } d_p \sqsubseteq c_p \\ < c_m \sqcup d_m, c_p \nabla d_p > & \text{sinon.} \end{cases}$$

Autrement dit, si la valeur abstraite par laquelle on doit élargir est plus petite (au sens de l'ordre lexicographique) que la précédente, on garde la valeur précédente. Le second cas traite le cas où la valeur pilote est stabilisée (cela signifie qu'une phase d'analyse est terminée) et donc on peut copier la valeur pilote vers la valeur principale. Le dernier cas réalise la séquence croissante : union sur la valeur principale, élargissement sur la valeur pilote.

Les expérimentations montrent que les invariants obtenus avec cette méthode sont quelquefois plus précis que la méthode classique avec prise en compte des nouveaux chemins, et quelquefois moins :

Exemple 4.5 *Sur l'exemple suivant :*

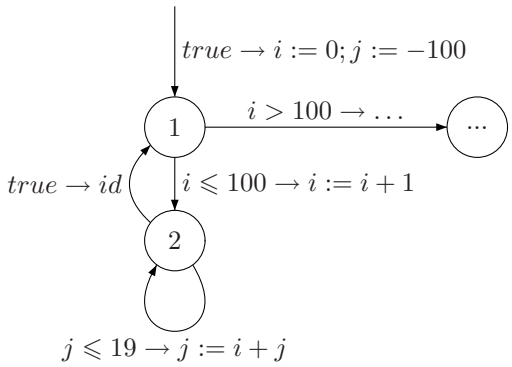


Fig. 4.9 – Exemple d'application du « lookahead widening »

La méthode classique (sans élargissement limité mais avec prise en compte des nouveaux chemins) donne l'invariant pour l'état 1 *:* $\{0 \leqslant i; 0 \leqslant j + 100\}$ *alors que l'application du « lookahead widening » donne l'invariant* $\{0 \leqslant i; 0 \leqslant j + 100; i \leqslant j + 201\}$*. Sur cet exemple on obtient donc un résultat plus précis.*

Exemple 4.6 *Sur l'exemple 4.4, l'utilisation du « lookahead widening » donne l'invariant* $\{20k + 2j \leqslant 11i, 2j \leqslant i, i \leqslant 102, 0 \leqslant 102j + 449i, i \leqslant 10k + 2j\}$*, ce qui ne permet pas de prouver* $20k \leqslant 11i$*. La perte de précision vient du fait que l'application de l'élargissement est beaucoup plus précise lorsque l'on part d'une dimension inférieure, ce qui est le cas lorsque*

[1] Un opérateur d'élargissement stable doit être implémenté, et l'analyseur doit suivre une stratégie itérative ([Bou93]), *i.e.*, converger à l'intérieur des composantes fortement connexes les plus internes en premier.

l'on part du point $\{i = j = k = 0\}$*. Pour augmenter la précision de la méthode, on pourrait par exemple décider de retarder l'élargissement d'une itération lorsque l'on a fini une séquence descendante.*

Le principal inconvénient de cette méthode est que tout comme la méthode de retarder l'élargissement elle génère des contraintes complexes, qui augmentent considérablement la taille des polyèdres en mémoire. Cependant, cette méthode est intéressante car elle s'attache à améliorer l'itération, et elle pourra se combiner à nos méthodes d'accélération.

4.5 Retour sur l'exemple de la chaudière

Sur l'exemple introductif (section 2.6, page 32), les méthodes présentées ci-dessus fournissent les résultats suivants :

1. Séquence décroissante : cette technique ne pourra jamais découvrir un invariant qui n'apparaît pas textuellement dans l'automate.
2. Avec élargissement limité : pas d'amélioration de la précision de l'invariant. L'élargissement avec bornes limites donne le même invariant.
3. En retardant l'application de l'élargissement de 62, on peut trouver l'invariant cherché. Cependant, ce délai est dépendant des constantes du programme, et le coût deviendrait prohibitif si on multipliait les constantes par 10 par exemple. En plus, une telle méthode ne fonctionne pas si les constantes étaient remplacées par des paramètres symboliques.
4. Sur cet exemple, l'élargissement proposé par [BHRZ03] n'apporte aucune amélioration.
5. L'heuristique du nouveau chemin mais sans élargissement limité donne le même invariant, en revanche, en combinant élargissement limité et heuristique du nouveau chemin, on obtient l'invariant voulu. Le principal inconvénient de la méthode est l'utilisation de l'élargissement limité qui impose de bien choisir les contraintes « pertinentes ».
6. L'analyse avec « look ahead » widening fournit les deux invariants suivants :

$$P_L = \{0 \leqslant x \leqslant 10, x \leqslant l \leqslant t\} \qquad P_N = \{0 \leqslant x + l \leqslant t, 0 \leqslant l\}$$

Ces invariants sont plus précis, mais sont quand même des approximations assez grossières de l'invariant voulu.

4.6 Amélioration de la stratégie de calcul de point fixe

Certains travaux cherchent à améliorer le calcul de points fixes en général. L'idée est de simplifier le calcul du plus petit point fixe en utilisant plusieurs calculs de points fixes sur des fonctions plus « simples ».

Principes de la méthode

Définition 20 — Sélection

Soit C un treillis complet et $\mathcal{G}$ un ensemble de fonctions monotones de C dans lui-même, appelées « politiques ». On dit qu'une fonction $F : C \to C$ satisfait la *propriété de sélection* pour l'ensemble $\mathcal{G}$ si les deux propriétés suivantes sont satisfaites (cl désigne l'opération de normalisation de C, c'est-à-dire qu'à chaque point $x \in C$, on peut calculer une *forme normale* $cl(x)$ qui est « minimale » en un certain sens, cette fonction est étendue aux fonctions et aux sous ensembles de C :

1. $F = cl(F) = \text{Inf}\{g \mid g \in \mathcal{G}\}$.
2. pour tout $X \in C$, il existe une fonction $h \in \mathcal{G}$ (une « politique ») tel que $F(X) = h(X)$.

Remarque 8 L'opération « forme normale » des DBM (Difference Bound Matrices, [HNSY92]) est une telle clôture. Cette forme normale peut être calculée à l'aide d'un algorithme de plus court chemin.

Si F satisfait une telle propriété, alors son plus petit point fixe est le plus petit point fixe d'une certaine politique :

Théorème 7 ([GGTZ07]) *Soit F une fonction monotone de C dans C qui satisfait la propriété de sélection pour un ensemble $\mathcal{G}$. Alors :*

$$lfp(F) = \text{Inf}\{cl(lfp(g)), g \in \mathcal{G}\}$$

Si on suppose que l'on dispose d'une application F qui satisfait la propriété de sélection pour un certain ensemble $\mathcal{G}$, alors, on peut appliquer l'algorithme suivant :

- on part d'une politique initiale $g_1 \in \mathcal{G}$.
- à la k-ième itération, on calcule x_k la clôture du plus petit point fixe de g_k, si c'est un point fixe de F, alors l'algorithme termine et on retourne x_k, sinon on sélectionne g_{k+1} tel que $cl(g)(x_k) = F(x_k)$ (c'est possible grâce à la propriété de sélection) et on réalise une autre itération.

Cet algorithme peut ne pas terminer, mais si il termine, alors le résultat est un point fixe de F (pour certaines classes de fonctions F, c'est le plus petit, mais pas toujours). Si $\mathcal{G}$ est fini de cardinal p, alors cet algorithme termine en au plus p itérations.

Pour un treillis donné, il reste donc à trouver un moyen d'exprimer les fonctions de transfert comme des bornes inférieures de fonctions plus simples. Dans [CGG+05], ce travail est effectué dans le cas particulier du treillis des intervalles. Il est plus facile à réaliser car le treillis est non relationnel.

Cas des intervalles Dans le cas des intervalles la simplification utilisée consiste à « séparer » les cas lors de l'intersection, en récrivant l'intersection de deux intervalles $I_1 = [a, b]$ et $I_2 = [c, d]$ de la façon suivante :

$$I_1 \cap I_2 = l(I_1, I_2) \cap r(I_1, I_2) \cap lr(I_1, I_2) \cap rl(I_1, I_2)$$

avec :
- $l(I_1, I_2) = I_1$, $r(I_1, I_2) = I_2$ (« left », « right ») ;
- $lr(I_1, I_2) = [a, d]$, $rl(I_1, I_2) = [c, b]$ (« leftright », « rightleft »).

Étant donnée une fonction F sur le treillis des intervalles (étendu avec des opérations de min et de max), on construit $\mathcal{G}(F)$ l'ensemble des politiques pour F en remplaçant chacune des occurrences de l'opération $\cap$ par l'une des fonctions précédentes. Il y a donc un nombre exponentiel de politiques en le nombre d'occurences de $\cap$. F a alors la propriété de sélection pour cet ensemble de politiques (puisque l'intersection de deux ensembles I_1 et I_2 est clairement la borne inférieure des $l(I_1, I_2)$, $r(I_1, I_2)$, $lr(I_1, I_2)$ et $rl(I_1, I_2)$.

Le calcul du plus petit point fixe de f est donc ramené à des calculs de plus petits points fixes sur des fonctions ne contenant aucune opération d'intersection (on pourrait se servir de cette caractéristique pour « spécialiser » les calculs « plus simples », mais ce n'est pas fait ici.)

Exemple 4.7 *Sur l'exemple 4.1 page 48, les heuristiques donnent pour politique initiale en x_3 (resp. x_5) rl (resp. lr), on modifie donc la définition de x_3 (qui était $(x_1 \cup x_4) \cap [-\infty, 99]$) en $[\mathrm{Inf}(x_1 \cup x_4), 99]$. Une première itération de point fixe est réalisée sur le système modifié pour x_3 et x_5 :*

$$x_3 = [\mathrm{Inf}(x_1 \cup x_4), 99] \text{ et } x_5 = [100, \mathrm{Sup}(x_1 \cup x_4)]$$

La résolution de ce système donne un point fixe du système initial, les théorèmes assurent que c'est donc le plus petit point fixe et donc l'algorithme termine.

Remarque 9 Dans [GGTZ07] est abordé le cas des treillis relationnels des octogones et des « Template Contraints Matrix » ([SISG06]), mais toutes les opérations ne sont pas traitées.

Résultats obtenus Les deux articles montrent des résultats préliminaires encourageants en terme de précision, sans pour autant dégrader le temps de calcul. Cependant, la complexité et la précision de l'analyse dépendent fortement :
- du choix de la « politique » initiale (pour l'instant, ce choix est basé sur des heuristiques) ;
- de la complexité et de la précision des algorithmes utilisés pour le calcul du point fixe à politique fixé. Dans [CGG+05], les auteurs utilisent l'algorithme de Kleene, dans [GGTZ07], ils se ramènent à des problèmes de programmation linéaire.

Remarque 10 Ces méthodes ne tiennent pas compte de la structure du programme à analyser, elles pourraient donc être combinées avec d'autres approches.

4.7 Vers la notion d'accélération abstraite

4.7.1 L'outil PIPS

Le logiciel Pips ([IJT91, Iri05]), initialement conçu pour faire de la parallélisation automatique de programmes Fortran ou C, utilise quelques techniques assez similaires afin de calculer des invariants de boucles. Les caractéristiques de l'outil sont les suivantes :

1. Les calculs se font en terme de relation de transition (et pas de fonction). L'avantage de cette méthode est que l'on peut précalculer pour chaque (sous-)bloc du programme une relation de transition (ou une sur-approximation), puis composer en s'affranchissant du polyèdre d'entrée de ce bloc.
2. Il n'y a pas de calcul de point fixe. Un appel de l'outil réalise un calcul de « transformateurs » associés aux points de contrôles (entrée, sortie de boucle, branches de tests) et qui sont des (sur-)approximations de la clôture de la relation de transition R associée aux parties de code associées (le bloc suivant le point de contrôle considéré). La projection sur les variables d'entrée fournit ensuite des sur-approximations des « préconditions » (invariants sur les états de contrôle). Comme il n'y a pas d'itération, il n'y a pas besoin d'opérateur d'élargissement.
3. Le calcul de la (sur-approximation) de la clôture R^* de la relation de transition d'une boucle est effectué en supposant que le corps de la boucle est régulier. L'effet après k exécutions de la boucle est ensuite calculé, puis la variable k est abstraite. Plus précisément, à partir de la relation exacte R du corps de la boucle, on calcule successivement :
   - une surapproximation affine $\widetilde{R}(dx, x)$ du corps de la boucle, dx représentant les différences $x' - x$ (x' est obtenue à partir de la variable x après un tour de boucle). Cette relation est valide quel que soit le nombre de tours de boucles effectués auparavant.
   - une surapproximation affine de la précédente en faisant abstraction des variables x, on obtient alors une relation $\widetilde{R}(dx)$, qui s'exprime sous la forme d'un système de contraintes $\{dx \mid Adx = b \wedge A'dx \leqslant b'\}$.
   - un calcul exact de $\widetilde{R}^k(x, x_0, k)$ où k représente le nombre de tours de boucles, obtenu à l'aide de la relation précédente en écrivant $x_k = x_0 + \sum_{i=1}^{k} dx^i$ (avec $dx^i = x_{i+1} - x_i$, vérifiant $\forall i, \widetilde{R}(dx^i)$). On obtient donc :
   $$(x, x_0) \in \widetilde{R}^k \quad \Leftrightarrow \quad Ax = Ax_0 + kb \wedge A'x \leqslant A'x_0 + kb',$$
   qui est un polyèdre en x, x_0 et k.
   - une sur-approximation de $\left(\widetilde{R}\right)^*(x, x_0)$ (donc de $R^*(x, x_0)$) en faisant abstraction de la variable k dans le polyèdre précédent.
4. Une analyse Pips calcule donc les relations associées à chaque bloc de programme. Un raffinement peut ensuite être réalisé en prenant en compte les préconditions de boucle (dans l'exemple qui suit, les valeurs initiales des variables). On obtient alors des relations plus spécialisées, dont on peut calculer plus précisément la clôture par exemple.

Remarque 11 Le fait de faire abstraction de la variable k dans la définition l'ensemble $\left(\widetilde{R}\right)^*(x, x_0)$ permet d'obtenir un polyèdre convexe. Cependant, l'abstraction est réalisée avec une opération de projection, *sans tenir compte du fait que k est un entier*. Nous verrons plus tard (chapitre 5), que cette « abstraction dense » est source de pertes de précision, mais que nous ferons de même.

Exemple 4.8 *Pour illustrer l'algorithme de* PIPS*, nous allons reprendre l'analyse de la voiture de l'exemple 4.3, page 52 (cette analyse est adaptée de celle de [Iri05]). On rappelle la partie du graphe de flot de contrôle qui nous intéresse :*

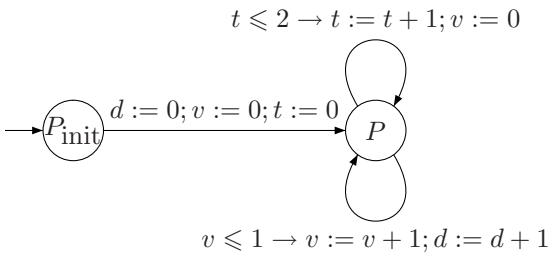


FIG. 4.10 – Rappel de l'exemple de la voiture

Les relations calculées pour les boucles sont les suivantes :
- *Pour la première boucle :* $T_{boucle1} = \{v' = 0, t' = t+1, d' = d\}$.
- *Pour la deuxième boucle :* $T_{boucle2} = \{d' = d+1, v' = v+1, t = t', \}$.

Une union convexe est réalisée ensuite sur les deux relations dans lesquelles on a rajouté $v \leqslant 2$ *et* $t \leqslant 3$ *(cela revient à faire un calcul « upto ») :*

$$T = \{t + d + 1 = t' + d', d \leqslant d' \leqslant d + 1, 3d + v' \leqslant 3d', t \leqslant 3, v + 3d' \leqslant 3d + v' + 2\}$$

qui peut se récrire :

$$T_{if} = \{(d' - d) + (t' - t) = 1, 1 \leqslant (v' - v) + 3(d' - d), 0 \leqslant (t' - t) \leqslant 1, t \leqslant 3, v' + 3t' \leqslant 3t + 3\}$$

Une surapproximation de la relation T^* *est de la boucle entière est ensuite obtenue en prenant une surapproximation de* T *selon les variables de différence* $dt = t' - t$, $ds = s' - s$, $dd = d' - d$, *puis en supposant que cette relation est itérée* k *fois avec* $k \geqslant 1$ *:*

$$\widetilde{T} = \{(d' - d) + (t' - t) = 1, 1 \leqslant (v' - v) + 3(d' - d), 0 \leqslant (t' - t) \leqslant 1\}$$

$$\widetilde{T^*(x, x_0, k)} = \{1 \leqslant k, d + t = d_0 + t_0 + k, v_0 + t_0 + k \leqslant v + 2t, t_0 \leqslant t\}$$

et la projection selon la variable k *fournit finalement :*

$$\widetilde{T^*(x, x_0)} = \{d_0 \leqslant d, t_0 \leqslant t, v_0 + 2t_0 + 1 \leqslant v + 2t, d + v_0 + 2t_0 \leqslant d_0 + v + 2t\}$$

On obtient donc, finalement, en utilisant $d_0 = s_0 = t_0 = 0$, *l'invariant* $d \leqslant 2t + s$ *qui est celui que l'on cherche (voir l'exemple 4.3).*

Signalons enfin que diverses extensions de cet algorithme ont été étudiées, elle permettent par exemple de traiter dans certains cas les relations non affines (modulo, flip-flop, exponentielles).

4.7.2 Résolution exacte « abstraite » dans le treillis des intervalles

L'article [SW04] se place dans le cadre d'analyses de programmes avec le treillis des intervalles. L'article identifie une classe de systèmes de contraintes d'intervalles pour laquelle la résolution du système abstrait est *exacte*.

Cet article utilise un algorithme de résolution de systèmes de contraintes qui ne fait pas intervenir d'élargissement. Cet algorithme est applicable à une large classe de programmes, y compris ceux comprenant des boucles imbriquées. Nous décrivons ces algorithmes dans le cas d'une variable unique positive. L'article décrit comment réduire le cas général (fonctions affines à d variables non forcément positives, les seules intersections autorisées étant les intersections avec un intervalle *constant*) à ce cas particulier. Dans le cas général, la complexité de l'algorithme de résolution est polynomiale.

On peut aussi supposer que l'on dispose toujours d'un système avec unique condition initiale :

LEMME 4.1 *Dans un graphe de contraintes, une composante fortement connexe avec un nombre quelconque de « contraintes initiales » (de la forme $[l, u] \sqsubseteq X$) peut être transformé en temps et en espace linéaires à une composante fortement connexe avec une unique contrainte initiale.*

Résolution dans le cas d'une boucle simple Une boucle simple est une unique boucle combinatoire (de taille 1), de la forme de la figure 4.11. X représente l'intervalle de variation de la variable x au point de contrôle considéré.

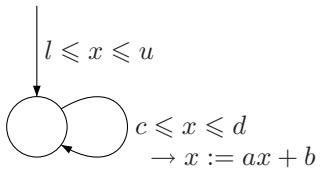


FIG. 4.11 – Boucle simple

Le système de contraintes associé à cette boucle est donc :

$$\begin{cases} [l, u] \sqsubseteq X \\ (aX + b) \sqcap [c, d] \sqsubseteq X \end{cases}$$

L'article propose l'algorithme de résolution suivant :
- Soit $\rho = [l, u]$
- On calcule $\rho' \leftarrow \tau([l, u]) = (a[l, u] + b) \sqcap [c, d] = [l', u']$ (opérations dans le treillis abstrait). Alors :
  - si $\rho' \sqsubseteq \rho$, ne rien faire.
  - sinon, si $l' < l$ et $u' > u$, alors $\rho \leftarrow [c, d]$.
  - sinon, si $l < l$, alors $\rho \leftarrow [c, u]$.
  - sinon, si $u > u$ alors $\rho \leftarrow [l, d]$.
- Retourner ρ.

REMARQUE 12 Cet algorithme est similaire à l'opérateur d'élargissement limité sur les intervalles. Dans le cas d'une boucle simple, on montre que cet algorithme calcule *sans élargissement* la plus petite solution du système de contraintes *abstrait*.

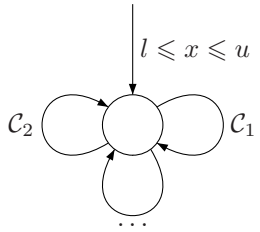


FIG. 4.12 – Boucles multiples

Résolution dans le cas de boucles multiples Dans le cas de boucles multiples, c'est-à-dire d'un ensemble de boucles simples associées au même point de contrôle (figure 4.12), avec $\mathcal{C}_i = c_i \leqslant x \leqslant d_i \rightarrow x := a_i x + b$. l'algorithme proposé consiste à réaliser une itération chaotique, c'est-à-dire d'utiliser l'algorithme précédent jusqu'à ce que toutes les contraintes soient satisfaites :

- Soit $\rho \leftarrow [l, u]$.
- Tant que l'ensemble des contraintes ne sont pas satisfaites :
  - Prendre une contrainte $\mathcal{C}_i = (a_i X + b) \sqcap [c_i, d_i] \sqsubseteq X$ non satisfaite par $\rho = [l', u']$.
  - Calculer (à l'aide de l'algorithme précédent) la plus petite solution du système formé de la contrainte $\mathcal{C}_i$ et de la condition initiale $[l', u'] \sqsubseteq X$, la stocker dans ρ.

La structure particulière du treillis des intervalles (une borne ne peut être saturée plus d'une fois) permet de prouver que cet algorithme termine en au plus $2n$ itérations (n étant le nombre de contraintes) et calcule la plus petite solution du système abstrait.

Résolution dans le cas d'une composante fortement connexe L'algorithme présenté est le cœur de la résolution. Le principal résultat est que l'on peut résoudre en temps cubique (en le nombre de contraintes) un système de contraintes associé à une composante fortement connexe.

Pour résoudre un système de contraintes associé à une composante fortement connexe, l'algorithme proposé est le suivant (on suppose que l'on n'a qu'une seule contrainte initiale pour l'entrée de la composante fortement connexe) :

1. Dérouler la composante fortement connexe : à l'aide d'un parcours *en profondeur d'abord*, construire un DAG correspondant à l'arbre du parcours, mais en considérant les « arcs de retour ». Dans l'opération, un certain nombre de points de contrôle ont été dédoublés, et on note X_{i0} (resp. X_{i1}) le sommet correspondant à la première (resp. deuxième) occurence de la variable X_i dans le parcours.
2. Calculer la plus petite solution ρ pour le sous-graphe composé des sommets X_0^*, X_{10}, $X_{20} \dots X_{n0}$ (ce sous graphe, par construction, est acyclique).
3. Si cette solution ρ satisfait toutes les contraintes des arcs de retour, (*i.e.* si pour tout arc de retour $X_{i0} \xrightarrow{\tau} X_{j1}$, on a $\tau(\rho(X_{i0})) \sqsubseteq X_{j1}$), alors terminer avec cette solution.
4. Sinon, pour tout arc de retour ne satisfaisant pas $f(\rho(X_{i0})) \sqsubseteq X_{j1}$:
   - Calculer la plus petite solution du système « multi-boucles » sur le sommet, X_j de valeur initiale $\rho(X_{j0})$, les fonctions considérées étant associées aux chemins $X_{j0} \rightarrow X_{j1}$ (on calcule aisément la composition des fonctions associées à chaque arc du chemin).
   - Mettre à jour la valeur de ρ.

5. Si toutes les contraintes sont satisfaites, retourner la valeur courante de ρ. Sinon, recommencer à l'étape 2 avec $\rho(X_{i0})$ comme contrainte initiale.

Des arguments similaires à précédemment induisent la correction et la terminaison de l'algorithme en un temps cubique.

4.7.3 La notion d'accélération abstraite dense

Les deux articles précédents utilisent des idées que nous trouvons intéressantes pour notre objectif d'amélioration de précision de l'Analyse des Relations linéaires :

- Dans [Iri05], l'opération « étoile » d'une relation est effectuée, en supposant que l'effet de la boucle est « capturé » par la différence $x' - x$ (valeur des variables après/avant l'application de la relation), et qu'il peut être répété. Dans le cas favorable (voir la remarque 11, page 61) le polyèdre obtenu est exactement l'enveloppe convexe de la relation de boucle, mais dans le cas général ce n'est qu'une sur-approximation. Nous verrons dans le chapitre 5 que nous effectuerons la même « approximation dense » de l'enveloppe convexe, en relaxion la condition $k \in \mathbb{N}$ sur le nombre de tours de boucles.
- Un certain nombre de résultats de l'article [SW04] proviennent du fait de l'utilisation du treillis des intervalles. Cependant, on peut noter :
  - l'utilisation d'algorithmes de graphes qui permettent une simplification et l'éclatement de la structure des boucles du programme.
  - l'utilisation des algorithmes de boucles simples dans des structures de graphes plus compliquées, en utilisant en particulier la composition des contraintes sur les chemins.
  - l'utilisation d'un algorithme que l'on pourrait qualifier d'« accélération abstraite » dans le cas de boucles simples. En effet, le point fixe calculé est le plus petit point fixe du système *abstrait*.
  - Les aspects arithmétiques du problème sont ici aussi complètement oubliés dans le calcul.

Conclusion du chapitre

Dans ce chapitre, nous avons étudié les différentes approches proposées jusqu'à présent afin d'améliorer la précision en Analyse des Relations Linéaires. Les approches proposées sont principalement de deux sortes :

- Celles qui s'attachent à améliorer la précision localement, c'est-à-dire à améliorer la précision de l'opérateur d'élargissement.
- Celles qui essaient d'améliorer la stratégie d'application de l'opérateur d'élargissement.

Ces méthodes sont intéressantes car elles permettent d'améliorer un certain nombre d'analyses, cependant elles ne prennent pas en compte la forme des transitions du programme, en particulier les fonctions « accélérables » au sens du chapitre précédent sont traitées comme les autres fonctions, en prenant en compte le résultat d'une application sur un polyèdre.

Deux travaux vont dans le sens de notre objectif : en effet, ils montrent que la notion d'*accélération abstraite* a un sens en Analyse des Relations linéaires : il s'agit de définir un opérateur d'accélération sur les éléments du domaine abstrait, qui calcule une surapproximation de l'effet d'une boucle sur un élément du domaine. Dans le cas des polyèdres, il s'agit entre autres d'oublier les aspects arithmétiques du calcul, comme nous le verrons au chapitre suivant.

Deuxième partie

Contributions, aspects théoriques et pratiques

Chapitre 5

Le cas d'une boucle unique

Dans ce chapitre, on s'intéresse au cas d'une unique boucle simple, c'est à dire d'une seule boucle élémentaire sur le point de contrôle considéré. Le but de ce chapitre est de calculer l'enveloppe convexe de l'effet itéré d'une fonction affine gardée sur un polyèdre convexe de $\mathbb{Q}^n$. Nous verrons que nous identifions une classe de fonctions qui se ramène à la classe des translations/remises à constante, et nous fournissons un algorithme de calcul d'une surapproximation de $\tau^*(P_0)$ pour cette classe de fonctions.

5.1 Quelques définitions et premières remarques

Définition 21 — Boucle complexe

Une *boucle complexe* (de taille p) autour du point de contrôle q est un circuit $(q, (g_1, a_1), q_1) \to (q_1, (g_2, a_2), q_2) \to \ldots \to (q_{p-1}, (g_p, a_p), q)$.

Définition 22 — Boucle simple

Une *boucle simple* est une boucle complexe de taille 1.

Dans un premier temps, nous allons étudier les boucles simples uniques (nous verrons dans le chapitre 8 comment nous pouvons traiter le cas des boucles complexes). Soit donc $\tau : AX \leqslant B \to X := CX + D$ une transition affine gardée, et soit P_0 un polyèdre d'entrée. On se propose d'étudier l'effet de l'application itérée de τ sur P_0, autrement dit $\tau^*(P_0) = \bigcup_{i \in \mathbb{N}} \tau^i(P_0)$. Dans la suite, on s'intéressera souvent au calcul de $\tau^+(P_0)$, résultat de l'application de τ un nombre de fois quelconque *strictement positif*.

L'ensemble $\tau^*(P_0)$ n'est pas convexe dans le cas général. Comme nous souhaitons rester à l'intérieur du cadre classique des polyèdres, notre objectif premier est de trouver l'enveloppe convexe de cet ensemble. Cependant, l'enveloppe convexe de $\tau^*(P_0)$ peut aussi ne pas être un polyèdre (on peut par exemple décrire un cercle).

Nous allons voir dans ce chapitre que dans certains cas nous sommes en mesure de calculer une sur-approximation convexe de $\tau^*(P_0)$. C'est en particulier le cas des translations, qui font l'objet de la section suivante.

5.2 Un premier cas simple : les translations

Soit τ une translation, c'est-à-dire une fonction affine gardée avec $C = I_n$ (identité sur les n variables) , ce que l'on note $\tau : AX \leqslant B \rightarrow X := X + D$. On va montrer que l'on peut calculer une sur-approximation convexe de $\tau^+(P_0)$. A chaque itération, il faut vérifier que le point obtenu vérifie la garde de τ, ce qui fournit le calcul suivant :

$$X \in \bigcup_{i>0} \tau^i(P_0) \Leftrightarrow \exists i \in \mathbb{N}, i > 0, \exists X_0 \in P_0, X = X_0 + iD \wedge \forall 0 \leqslant j \leqslant i-1, A(X_0 + jD) \leqslant B$$

Comme la garde est convexe et que la fonction est une translation, il suffit de tester la garde pour $j = 0$ et $j = i - 1$:

$$X \in \bigcup_{i>0} \tau^i(P_0) \Leftrightarrow \exists i \in \mathbb{N}, i > 0, \exists X_0 \in P_0, X = X_0 + iD \wedge AX_0 \leqslant B \wedge A(X_0 + (i-1))D) \leqslant B$$

La section suivante montre comment nous allons calculer une sur-approximation convexe de $\tau^+(P_0)$.

5.2.1 Une sur-approximation à faible coût

Remarquons tout d'abord que l'expression $A(X_0 + (i-1))D) \leqslant B$ est équivalente à l'expression $A(X - D) \leqslant B$, par définition de X. Ensuite, le point X est obtenu à partir d'un point de P_0 en ajoutant i fois le vecteur D ($+$ désigne temporairement l'opération « ajout d'un nombre entier de vecteur ») :

$$\begin{aligned} X \in \bigcup_{i>0} \tau^i(P_0) &\Leftrightarrow \exists i \in \mathbb{N}, i > 0, \exists X_0 \in \big(P_0 \cap \{AX \leqslant B\}\big), X = X_0 + iD \wedge A(X - D) \leqslant B \\ &\Leftrightarrow \exists i \in \mathbb{N}, i > 0, X \in \Big(\big(P_0 \cap \{AX \leqslant B\}\big) + iD\Big) \cap A(X - D) \leqslant B \end{aligned}$$

Notre objectif est de calculer l'enveloppe convexe de l'effet exact de la boucle sur le polyèdre P_0, mais nous sommes confrontés à un problème d'arithmétique. En effet, l'effet exact d'une boucle étant définie à l'aide de l'entier k, il ne peut être calculé uniquement à l'aide d'opérations simples sur les polyèdres (qui sont des ensembles de points denses). Afin d'éviter les problèmes induits par l'arithmétique exacte, nous décidons donc d'effectuer une *approximation dense* en relaxant la condition $i \in \mathbb{N}$. On obtient ainsi une notion d'*accélération abstraite dense*, dont la définition est donnée ci-dessous :

Définition 23 — Accélération abstraite dense ([GH06])

> Soit τ une translation. On appelle *accélération abstraite dense* (abstraite parce que l'on parle de valeurs dans le treillis abstrait, dense parce que l'on « oublie » que k est un entier) de τ la fonction suivante :
>
> $$\tau^{\otimes} : P_0 \mapsto P_0 \sqcup \tau^{\oplus}(P_0)$$
>
> avec :
>
> $$\tau^{\oplus} : P_0 \mapsto \big\{X \mid \exists i \in \mathbb{Q}^+, \exists X_0 \in P_0,\ g(X_0) \wedge g(X - D),\ X = X_0 + iD\big\}$$

REMARQUE 13 L'ensemble $\{X \mid \exists i \in \mathbb{Q}^+, \exists X_0 \in P_0,\ g(X_0) \wedge g(X - D),\ X = X_0 + iD\}$ est un polyèdre convexe, et donc $\tau^{\oplus}(P_0)$ (puis $\tau^{\otimes}(P_0)$) aussi.

$\tau^{\oplus}(P_0)$ se calcule donc en ajoutant le rayon D, jusqu'à la garde $g(X - D)$ (*i.e.* la postcondition de la garde par l'action τ), voir la figure 5.1 :

PROPOSITION 7 *Soit* $\tau : AX \leqslant B \to X := X + D$. *Alors*

$$\tau^{\oplus}(P_0) = \Big((P_0 \cap \{AX \leqslant B\}) \nearrow \{D\}\Big) \cap \{A(X - D) \leqslant B\}$$

PREUVE : Immédiate d'après la définition de $\tau^{\oplus}$. ■

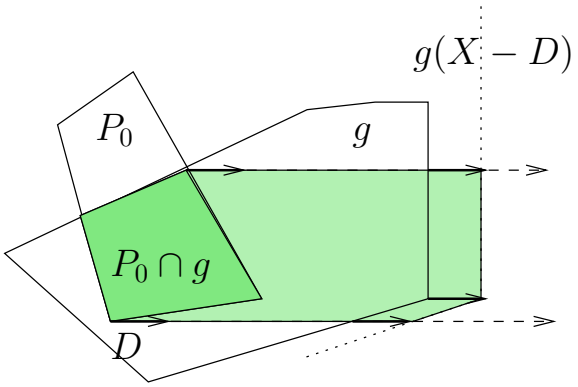


FIG. 5.1 – Effet de l'ajout de rayons

On obtient donc finalement :

PROPOSITION 8 *Soit* $\tau : g \to a$ *une translation gardée. Si* P *est un polyèdre convexe (sous la forme du systèmes de contraintes), alors on peut calculer une sur-approximation convexe de* $\tau^*(P_0)$.

PREUVE : $\tau^{\otimes}(P_0)$ est une sur-approximation convexe de $\tau^*(P_0)$: $\tau^{\otimes}(P_0)$ est un polyèdre convexe par construction. Ensuite, tous les points de $\tau^*(P_0)$ vérifient les conditions de $\tau^{\otimes}(P_0)$. ■

REMARQUE 14 L'expression utilisée pour le calcul de $\tau^{\otimes}(P_0)$ fournit un algorithme performant. En effet, l'ajout de rayon est en temps constant si on dispose du système de générateurs. En pratique, le temps de calcul de $\tau^{\otimes}(P_0)$ est comparable à celui de $\tau(P_0)$.

Notons aussi que le calcul de $g(X - D) = \{A(X - D) \leqslant B\}$ est indépendant du polyèdre d'entrée P_0. Ce polyèdre peut donc être stocké et réutilisé.

L'inconvénient est que l'on perd en précision, comme le montre l'exemple suivant :

EXEMPLE 5.1 *Soit* $\tau : \{x \leqslant 11\} \to x := x + 2$ *avec* $P_0 = \{x = 0\}$. *Le polyèdre « exact »* *est* $\tau^*(P_0) = \{0 \leqslant x \leqslant 12\}$ *alors que l'ajout de rayon donne* $\tau^{\otimes}(P_0) = P_0 \sqcup \Big(\{x = 0\} \nearrow (1) \cap \{x \leqslant 13\}\Big) = \{0 \leqslant x \leqslant 13\}$.

En conclusion, nous avons obtenu une expression simple pour une sur-approximation convexe de $\tau^*(P_0)$, mais cette sur-approximation n'est pas le plus petit polyèdre convexe contenant $\tau^*(P_0)$ (si il existe). Cependant, l'exemple suivant montre que le comportement des variables peut être complexe à caractériser :

EXEMPLE 5.2 *Soit* $P_0 = \{0 \leqslant x \leqslant 4, y = \frac{x}{2}\}$ *et* $\tau : y \leqslant 4 \rightarrow y := y+2, x := x+1$. *Le dessin 5.2 illustre le comportement pour des applications successives de* τ. *Sur cette figure, on peut voir que le polyèdre* $\tau^*(P_0)$ *est très complexe, son calcul ferait intervenir des considérations arithmétiques. Remarquons que dans cet exemple la surapproximation calculée est exactement l'enveloppe convexe du résultat.*

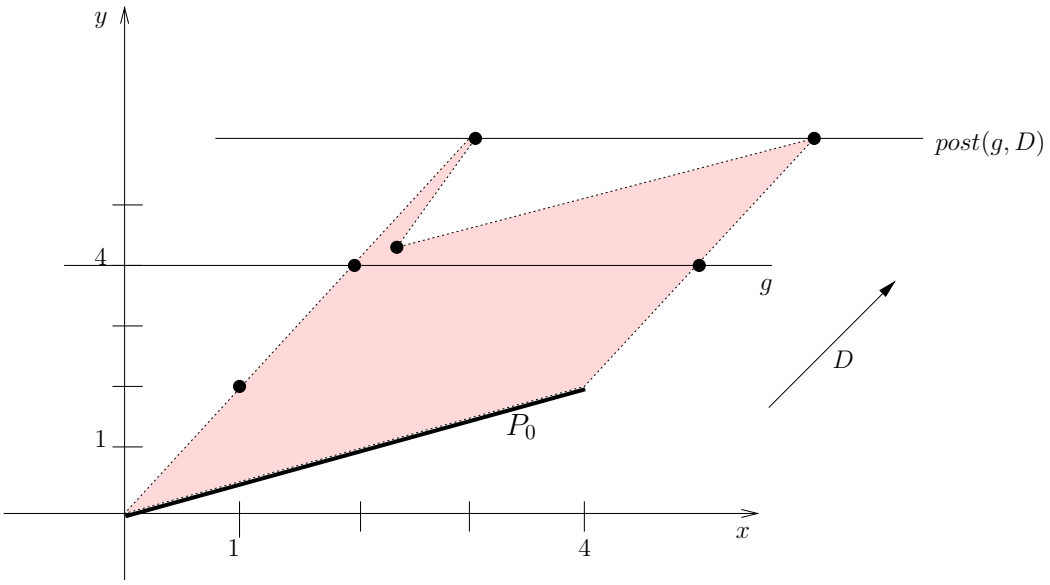


FIG. 5.2 – Comportement « en dents de scie »

Dans le cas particulier de l'exemple 5.1, l'algorithme Omega ([Pug91]), qui utilise la programmation linéaire en nombre entiers, fournirait l'exemple exact, et donc son enveloppe convexe. Dans le cas général, les algorithmes de programmation linéaire en nombre entiers nous donnent des résultats plus précis *dans le cas particulier où les programmes ne traitent que de variables entières*. L'algorithme de projection du test Omega réalise en particulier la projection selon une variable entière de l'intersection d'un polyèdre avec un « réseau » ($\mathbb{Z}^n$ par exemple), et peut donc être utilisé.

5.3 Un deuxième cas simple, les translations/remise à constante

Si la matrice C est une translation/remise à constante, c'est à dire une transformation τ avec C diagonale avec uniquement des 0 et des 1 sur la diagonale.

$$\tau \text{ s'écrit } \tau : AX \leqslant B \rightarrow \begin{cases} Y := Y + D_y & \text{(composantes translatées)} \\ Z := D_z & \text{(composantes mises à constante)} \end{cases}$$

L'analyse est très similaire à la précédente, et donne les résultats suivants :

DÉFINITION 24 — Accélération abstraite dense dans le cas translation/remise à constante

Soit τ une translation/remise à constante. On appelle *accélération abstraite dense* de τ la fonction suivante :

$$\tau^{\otimes} : P_0 \mapsto P_0 \sqcup \tau^{\oplus}(\tau(P_0))$$

avec :

$$\tau^{\oplus} : P_1 \mapsto \left\{ X \mid \exists i \in \mathbb{Q}^+, \exists X_1 \in P_1,\ g(X_1) \wedge g\Big(X - \begin{bmatrix} D_y \\ 0 \end{bmatrix}\Big),\ X = X_1 + i \begin{bmatrix} D_y \\ 0 \end{bmatrix} \right\}$$

PROPOSITION 9 *Soit $\tau : AX \leqslant B \to X := CX + D$ avec C diagonale avec des 0 et des 1 sur la diagonale. Alors*

$$\tau^{\oplus}(P_1) = \Big(\big(P_1 \cap \{AX \leqslant B\}\big) \nearrow \left\{ \begin{bmatrix} D_y \\ 0 \end{bmatrix} \right\} \Big) \cap g\Big(X - \begin{bmatrix} D_y \\ 0 \end{bmatrix}\Big)$$

PREUVE : L'expression vient du fait qu'une translation/remise à constante se comporte comme une translation de vecteur $\begin{bmatrix} D_y \\ 0 \end{bmatrix}$ si le nombre i de tours de boucles est strictement supérieur à 1. De plus, $X \in \tau(P_0) \Leftrightarrow \exists X_0, AX_0 \leqslant B \wedge X = \tau(X_0)$.

■

De la même façon qu'à la section précédente, on a le résultat :

PROPOSITION 10 *Si τ est une translation/remise à constante, alors $\tau^{\otimes}(P_0)$ est une sur-approximation convexe de $\tau^*(P_0)$.*

5.4 Une première réduction intéressante

Dans cette section, considérons $\tau : AX \leqslant B \to X := CX + D$ une transition affine gardée vérifiant $\exists p \geqslant 1, C^{2p} = C^p$. En effet, dans le chapitre 3, section 3.2.2 (page 39), on a vu que [Boi99] puis [FL02] obtiennent tous les deux des résultats intéressants pour cette classe de transformations.

Nous allons voir dans cette section que nous sommes en mesure de traiter cette famille de fonctions puisqu'elle se réduit au cas des translations/remises à constante. Nous verrons ensuite dans la section 5.5 comment décider si un tel p existe.

LEMME 5.1 *Soit $\tau : AX \leqslant B \to X := CX + D$ une transition affine gardée vérifiant $\exists p \geqslant 1, C^{2p} = C^p$. Alors le calcul de $\tau^*(P_0)$ peut se ramener en temps polynômial en p et n (taille de C) au calcul de l'image itérée de p polyèdres par une transition affine gardée τ' vérifiant $C'^2 = C'$, c'est à dire avec C une matrice de projection.*

PREUVE :
Soit $P' = \tau^*(P_0) = P_0 \sqcup P_1 \ldots \sqcup P_{p-1} \sqcup P_p \sqcup \ldots$ alors on peut écrire :

$$\begin{aligned} P' \quad = \quad & \big(P_0 \sqcup \tau^p(P_0) \sqcup \tau^{2p}(P_0) \sqcup \ldots\big) \sqcup \big(P_1 \sqcup \tau^p(P_1) \sqcup \tau^{2p}(P_1) \sqcup \ldots\big) \\ & \sqcup \ldots \sqcup \big(P_{p-1} \sqcup \tau^p(P_{p-1}) \sqcup \ldots\big) \end{aligned}$$

On remarque alors que X a une image par τ^p si et seulement si la condition : $AX \leqslant B \wedge A(CX+D) \leqslant B \wedge \ldots \wedge A\big(C^{p-1}X + \big(C^{p-2}+C^{p-3}+\ldots+I\big)D\big) \leqslant B$ est vérifiée.

Finalement, si on note $C' = C^p$, $B' = \begin{bmatrix} B \\ B - AD \\ B - A\big((C+I)D\big) \\ \ldots \\ B - A\big((C^{p-1}+\ldots+I)D\big) \end{bmatrix}$, $A' = \begin{bmatrix} A \\ AC \\ AC^2 \\ \ldots \\ AC^{p-1} \end{bmatrix}$, $D' = (C^{p-1}+C^{p-1}+\ldots I)D$, et enfin $\tau' : A'X \leqslant B' \to C'X + D'$, alors on a :

$$P' = \bigsqcup_{0 \leqslant i \leqslant p-1} \widetilde{P_p} \text{ avec } \widetilde{P_p} = \tau'^*(P_p).$$

On est donc ramené au calcul de p images itérées de polyèdres par une fonction affine gardée τ', qui vérifie $C'^2 = C'$. Le calcul de la nouvelle transformation est indépendant de P_0 et coûte $4p$ multiplications de matrices. Il faut quand même remarquer que la taille de la garde de τ' est p fois la taille de la garde de τ. ■

Exemple 5.3 *Soit τ la fonction affine gardée suivante :* [1]

$$\{t+v \leqslant 5\} \to \begin{bmatrix} x \\ y \\ z \\ t \\ u \end{bmatrix} := \begin{bmatrix} 83 & -109 & -114 & -91 & 27 \\ 185 & -172 & -152 & -124 & 38 \\ -109 & -31 & -102 & -77 & 20 \\ -347 & 1116 & 1433 & 1131 & -325 \\ -1138 & 3269 & 4133 & 3264 & -940 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ t \\ u \end{bmatrix} + \begin{bmatrix} 7 \\ 2 \\ 1 \\ 0 \\ 2 \end{bmatrix}$$

On a alors $C^4 = C^8$. Notons donc $C' = C^4 = \begin{bmatrix} 49 & -32 & -32 & -48 & 16 \\ 39 & -25 & -26 & -39 & 13 \\ 105 & -70 & -69 & -105 & 35 \\ -843 & 562 & 562 & 844 & -281 \\ -2388 & 1592 & 1592 & 2388 & -795 \end{bmatrix}$ et calculons $D' = (C^3+C^2+C+I)D = \begin{bmatrix} -32 \\ -77 \\ 77 \\ -23 \\ 44 \end{bmatrix}$. La nouvelle garde est alors après calcul :

$$A' = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ -1485 & 4385 & 5566 & 4395 & -1265 \\ -4219 & 4634 & 4828 & 4598 & -1420 \\ 1315 & 2543 & 4216 & 3015 & -811 \end{bmatrix} \text{ et } B' = \begin{bmatrix} 5 \\ 3 \\ -1408 \\ 16869 \end{bmatrix}.$$

Lemme 5.2 *Le cas d'une fonction affine gardée avec $C^2 = C$ se ramène polynomialement (en n) au cas d'une matrice diagonale avec des 0 et des 1 sur la diagonale.*

Preuve : Si $C^2 = C$ alors C est une matrice de projection et donc il s'agit de se placer dans une base adaptée à cette projection. On calcule donc les points fixes de C (noyau de $C - I$)

[1] Tous les calculs ont été faits à l'aide d'un logiciel de calcul formel

et les points d'image nulle (noyau de C). Ces vecteurs constituent une matrice de passage Q telle que $Q^{-1}CQ$ est une matrice diagonale avec 0 et 1. Le coût de ce calcul (ainsi que celui de l'inverse de Q) est $O(n^3)$. Ensuite, il s'agit de traduire la garde dans la nouvelle base, ainsi que le vecteur D et le polyèdre P_0. Chacune de ces opérations est en $O(n^3)$. ■

EXEMPLE 5.4 *Reprenons l'exemple précédent, la matrice C' vérifie $C'^2 = C$. Le calcul des noyaux de C' et de $C' - I$ (à l'aide d'un logiciel de calcul formel) donne pour matrice de changement de base (et son inverse) :*

$$Q = \begin{bmatrix} 0 & 0 & 1 & 0 & \frac{16}{13} \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & \frac{35}{13} \\ 0 & 0 & 0 & 1 & \frac{-281}{13} \\ 2 & 2 & -3 & 3 & \frac{-796}{13} \end{bmatrix} \quad et \quad Q^{-1} = \begin{bmatrix} 39 & -25 & -26 & -39 & 13 \\ 105 & -70 & -69 & -105 & 35 \\ 49 & -32 & -32 & -48 & 16 \\ -843 & 562 & 562 & 844 & -281 \\ -39 & 26 & 26 & 39 & -13 \end{bmatrix}$$

Finalement $C'' = Q^{-1}C'Q = diag(1, 1, 1, 1, 0)$ (matrice diagonale). Il faut aussi calculer le nouveau vecteur $D'' = Q^{-1}D$ et la nouvelle garde en appliquant toujours $X = QX'$, nous ne détaillons pas ces calculs ici. Dans la nouvelle base, la transformation est une translation/remise à constante (matrice diagonale avec 0 et 1).

Finalement, les deux lemmes combinés aux résultats des deux premières sections fournissent le résultat suivant :

PROPOSITION 11 *Soit $\tau : AX \leqslant B \to X := CX + D$ une transition affine gardée vérifiant $\exists p, C^{2p} = C^p$ et P_0 un polyèdre d'entrée. Alors on peut se ramener en temps polynomial en p et et la taille de τ et n au calcul d'une surapproximation du polyèdre $\tau^*(P_0)$.*

PREUVE : Immédiate à l'aide des lemmes 5.1 et 5.2. ■

REMARQUE 15 La définition de $\tau^{\otimes}(P_0)$ pour τ vérifiant $C^{2p} = C^p$ en découle immédiatement.

Comparaison avec l'accélération « exacte » La complexité obtenue précédemment est à comparer à celle obtenue dans [BFL04] : la taille obtenue pour l'`UBA` représentant f^* (étoile « exacte ») est 5-EXP en n et 3-EXP en la taille de l'`UBA` représentant la garde (un `UBA` représentant m contraintes sur n variables peut être calculé en temps et en espace n^m). Cependant, les ensembles mis en jeu ne sont pas comparables : l'accélération exacte traite d'ensemble de Presburger, c'est à dire des ensembles de vecteurs *entiers* (ou dans $\mathbb{Z}^n$), alors que nos polyèdres convexes sont des ensembles de vecteurs de $\mathbb{R}^n$. De plus, les formules de Presburger permettent la quantification existentielle sur i *entier*, alors que notre structure ne le permet pas. D'un autre côté, la représentation sous forme de polyèdre convexe est plus compacte que celle des `UBA` et permet d'être plus efficace algorithmiquement, même si cette efficacité se fait au détriment de la précision.

5.5 Le problème du monoïde fini

Dans la section précédente, on a supposé que p était donné, mais on ne dit pas comment calculer un tel p, ni décider si il existe. Tout d'abord, rappelons les résultats simples suivants :

LEMME 5.3 *Soit $C \in \mathcal{M}_n(\mathbb{Q})$ une matrice carrée de taille n à coefficients dans $\mathbb{Q}$. Il y a équivalence entre les problèmes suivants :*

1. *Est ce qu'il existe une puissance de C qui est diagonalisable et de valeurs propres dans $\{0, 1\}$?*
2. *Est-ce qu'il existe $p \in \mathbb{N}$, tel que $C^{2p} = C^p$?*
3. *Est que que le monoïde engendré par les puissances de C, c'est-à-dire $\{I, C, C^2, \ldots\}$ est fini ?*

PREUVE : Voir annexe A. ■

Ainsi les résultats d'accélération exacte de [Boi99] puis [FL02] sont équivalents. Cependant, alors que dans **Fast** aucun algorithme de décision n'est implémenté, la thèse de Boigelot fournit une procédure de décision qui retourne, si il existe, un p tel que $C^{2p} = C^p$:

PROPOSITION 12 *On peut décider en temps $O(n^4)$ si le monoïde engendré par C est fini, et retourner p tel que $C^{2p} = C^p$ si un tel p existe.*

PREUVE : Voir l'annexe A. ■

Cependant, si la complexité de cette procédure n'est pas très élevée, on a le résultat suivant :

PROPOSITION 13 ([MIL87]) *Un éventuel p tel que $C^{2p} = C^p$ peut être très grand. Plus précisément, le maximum vérifie $\ln(p_{max}) \sim \sqrt{n \ln(n)}$.*

Ce dernier résultat montre qu'implémenter une procédure de décision pour trouver un tel p n'est pas une bonne solution, d'autant plus que la réduction de la section précédente impose de calculer toutes les puissances de C de 2 jusqu'à p. Pour p trop grand, la complexité du calcul de $\tau^{\otimes}(P_0)$ devient ainsi trop grande, et les polyèdres obtenus seront trop complexes. Il semble raisonnable de nous limiter à l'algorithme force-brute pour la recherche d'un $p \leqslant 4$ par exemple.

5.6 Retour sur l'exemple de la chaudière

On rappelle à la figure 5.3 l'automate interprété discret modélisant la chaudière (section 2.6, page 32).

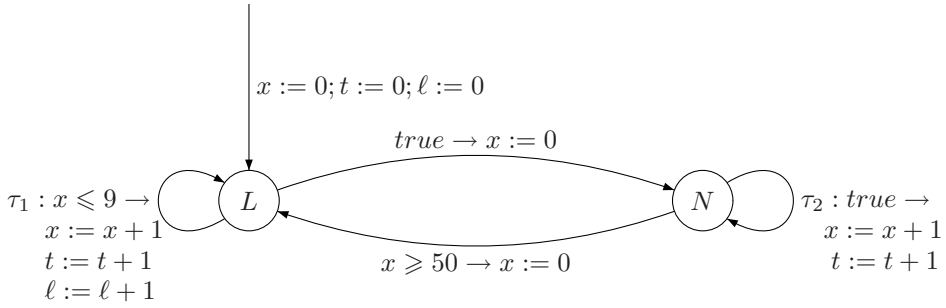


FIG. 5.3 – Modélisation de la chaudière sous la forme d'automate interprété numérique

Les deux transformations τ_1 et τ_2 sont accélérables à l'aide les expressions suivantes (dans les vecteurs, les variables x, t, ℓ sont considérés dans cet ordre) :

- $\tau_1^{\otimes}(P) = P \nearrow \begin{pmatrix}1\\1\\1\end{pmatrix} \cap \{x \leqslant 10\}$
- $\tau_2^{\otimes}(P) = P \nearrow \begin{pmatrix}1\\1\\0\end{pmatrix}$

Dans les deux cas, l'expression donnée calcule exactement l'enveloppe convexe de $\tau_i^*(P)$. Cela provient de l'incrémentation de 1 pour les variables qui fait que $post(g_1, D_1)$ est effectivement atteint.

On obtient donc l'analyse suivante :

- Étape 1. $P_L^1 = \{x = t = \ell = 0\} \nearrow \begin{pmatrix}1\\1\\1\end{pmatrix} \cap \{x \leqslant 10\} = \{0 \leqslant x = t = \ell \leqslant 10\}$, puis $P_N^1 = \tau_2^{\otimes}\left(P_{L|x=0}^1\right) = \{\ell + x = t, \ell \leqslant 10, 0 \leqslant \ell \leqslant t\}$.
- Étape 2.
  - On calcule la contribution $\left(P_N^1 \cap \{x \geqslant 50\}\right)_{|x=0}$, puis l'enveloppe convexe avec P_L^1, on obtient $P_L^{2aux} = \{x = 0, 0 \leqslant 6\ell \leqslant t, \ell \leqslant 10\}$. Ensuite, on réalise l'élargissement $P_L^1 \nabla \tau_1^{\otimes}(P_L^{2aux})$, et on obtient $P_L^2 = \{\mathbf{6}\ell \leqslant \mathbf{t} + \mathbf{5x}, 0 \leqslant x \leqslant 10, x \leqslant \ell\}$.
  - Des calculs similaires (sans élargissement) donnent pour $P_N^2 = \{\mathbf{6}\ell + \mathbf{x} \leqslant \mathbf{t} + \mathbf{50}, \ell + x \leqslant t, 0 \leqslant l, 0 \leqslant x\}$.
- Étape 3. Convergence.

Nous avons donc obtenu les mêmes invariants que dans le cas de l'analyse hybride, en ne retardant pas l'application de l'élargissement.

Conclusion du chapitre

Dans ce chapitre, nous avons étudié le cas de l'itération d'une boucle unique sur un point de contrôle particulier. Nous avons défini la notion d'accélération abstraite qui calcule une sur-approximation du polyèdre image d'un polyèdre par l'application d'un nombre quelconque

de translations gardées. Nous avons vu comment calculer cette surapproximation dans le cas d'une transformation « à monoïde fini », c'est-à-dire d'une transformation $\tau : g \to X := CX + D$ avec C vérifiant $\exists p, C^{2p} = C^p$, qui est l'une des classes de transformations identifiées comme accélérables dans les travaux d'accélération.

Dans le chapitre suivant, nous allons voir que le cas de boucles multiples est beaucoup plus complexe.

Chapitre 6

Le cas des translations multiples

Dans ce chapitre, on s'intéresse au cas des boucles simples multiples, c'est-à-dire des transformations appliquées simultanément sur un même point de contrôle. Nous fournissons des résultats théoriques pour le cas de plusieurs boucles translations, et donnons un algorithme afin de calculer de façon efficace et précise une surapproximation de l'enveloppe convexe des états atteignables.

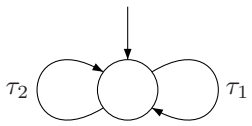


FIG. 6.1 – Deux transitions sur le même point de contrôle

6.1 Premières remarques

Dans le cas de plusieurs transitions situées sur le même point de contrôle, les polyèdres obtenus deviennent plus complexes. Même dans le cas des translations, l'enchaînement des transitions peut introduire des non convexités, ou alors des oscillations, comme le montrent les figures 6.2 et 6.3.

Sur ces figures, on prend $P_0 = \{x_0\}$, et on applique à ce point une succession de translations de vecteur D_1 ou D_2, si l'application est possible (les gardes sont les demi-espaces délimités par lignes épaisses). La figure 6.2 montre que l'ensemble des points obtenus est non convexe, la figure 6.3 montre qu'il peut y avoir des oscillations complexes autour des gardes.

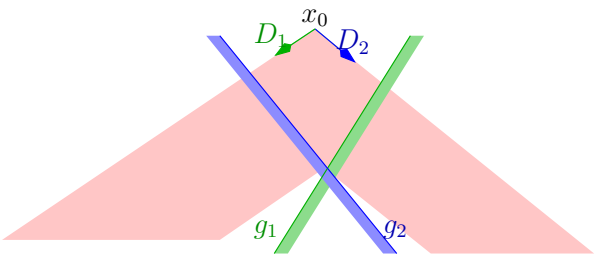


FIG. 6.2 – L'ensemble « exact » est non convexe

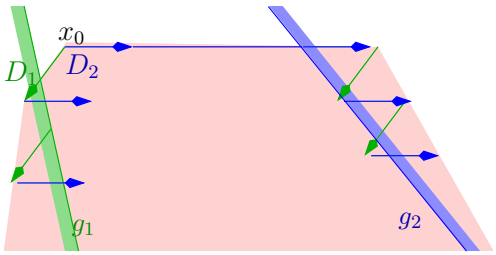


FIG. 6.3 – Trajectoire oscillante d'un point

Le problème de calculer $(\tau_1 + \tau_2 + \ldots)^*(P_0)$ est connu pour être difficile, comme le montrent les différents travaux concernant l'accélération et le résultat suivant, qui illustre le cas le plus simple, c'est-à dire le cas où les actions sont des translations. En effet, le comportement d'un point de P_0 peut s'apparenter à la trajectoire d'un ensemble dans un PCD (Piecewise-Constant Derivatives, [AMP95]). L'espace est divisé en secteurs à l'intérieur desquels un point évolue selon une droite :

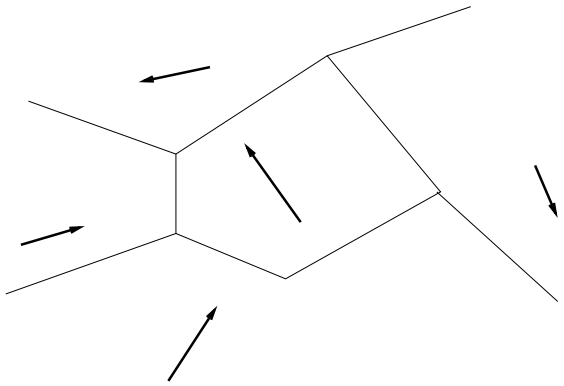


FIG. 6.4 – Un PCD en dimension 2

THÉORÈME 8 ([AMP95]) *Le problème d'accessibilié pour les PCD en trois dimensions est indécidable.*

Les résultats d'accélération du chapitre 3 ne peuvent s'appliquer parce que le graphe de

contrôle n'est pas plat dans le cas général.

Dans ce chapitre et le suivant, nous présentons quelques résultats d'accélération exacte (c'est-à-dire des expressions qui calculent exactement l'enveloppe convexe du polyèdre $(\tau_1 + \tau_2 + \ldots + \ldots)^*(P_0)$), et des résultats de surapproximation de cette enveloppe convexe. Nous décidons de nous restreindre aux cas de translations simples et de translations/remises à constante, pour lesquelles il est facile de calculer individuellement une surapproximation de l'enveloppe convexe de $\tau^*(P_0)$ (chapitre 5).

Dans ce chapitre on étudie le cas de deux translations $\tau_i : g_1 \to x := x + D_i$. Dans ce cas, l'algorithme qui consisterait à appliquer successivement τ_1^* puis τ_2^* puis ..., peut ne pas converger, comme le montre la figure 6.5. Dans ce cas, les techniques d'accélérations ne terminent pas.

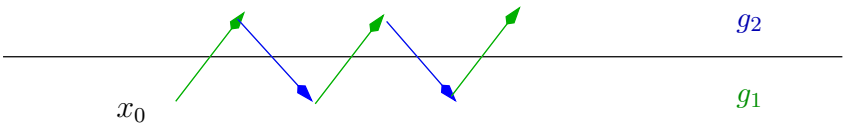


FIG. 6.5 – Trajectoire oscillante d'un point

6.2 Première proposition, le partitionnement

Dans cette section, nous proposons une première solution, le partitionnement du point de contrôle considéré selon les gardes g_1 et g_2. Cette méthode possède l'avantage de se ramener aux cas des boucles uniques du chapitre précédent, et au cas d'accélération combinée *les deux gardes étant simultanément vérifiées*. Dans un premier temps, nous exposons nos résultats obtenus pour ce sous-cas, dans un deuxième temps nous exposerons la technique de partionnement, et enfin nous discuterons des inconvénients de cette approche.

6.2.1 Accélération pour les gardes simultanément vérifiées

Dans un premier temps, nous donnons un résultat qui permet d'obtenir l'accélération de deux boucles (ou plus) *tant que les deux gardes* g_1 *et* g_2 *sont satisfaites*.

Pour cela, nous allons tout d'abord étudier l'ensemble $P' = \tau^+(P_0) \cap g$ des images d'éléments de P_0 par une succession d'applications de la transformation τ *tout en restant dans* g. Cet ensemble est (sur) approximé par $\tau^{\oplus}(P_0) \cap g = \{X = X_0 + kD \mid X_0 \in P_0, k \in \mathbb{Q}^+, X \in g\}$, comme nous l'avons vu dans le chapitre précédent. Dans cette expression, tout se passe comme si le point X était obtenu comme limite de suite de points de $P \cap g$, les points de la suite s'écrivant $x_{i+1} = \tau^{k_i}(x_i)$, $k_i \in \mathbb{Q}$ (applications *rationnelles*).

Maintenant, considérons l'ensemble $(\tau_1 + \tau_2)^+(P_0 \cap g_1 \cap g_2) \cap g_1 \cap g_2$ l'ensemble des images d'éléments de P_0 qui vérifient initialement (et à la fin) g_1 et g_2. Cet ensemble contient en partie des points résultat d'une suite d'applications de τ_1 et τ_2 en restant dans $g_1 \cap g_2$. Nous allons voir qu'une surapproximation de ces derniers est calculable. Pour cela, on note $\tau_{1,2}^{\circ}(P_0)$ l'image d'un polyèdre initial P_0 par deux boucles de translation $\tau_i :\ g_i \to x := x + D_i,\ (i = 1, 2)$ *tant que les deux gardes sont satisfaites* :

DÉFINITION 25 — $\tau^{\circ}_{1,2}$ ([GH06])

$\tau^{\circ}_{1,2}(P_0)$ est composé de tous les points x qui peuvent être atteints de $P_0 \cap g_1 \cap g_2$ en appliquant « de manière rationnelle » les translations τ_1 and τ_2 tout en restant dans $g_1 \cap g_2$:

$$\begin{aligned} x \in \tau^{\circ}_{1,2}(P_0) \text{ ssi} \quad & \exists x_0 \in P_0 \cap g_1 \cap g_2, \\ & \exists x_1, x_2 \ldots, x_\ell \in g_1 \cap g_2, \exists x'_1, x'_2 \ldots, x'_\ell \in g_1 \cap g_2, \\ & \exists i_1, i_2, \ldots, i_\ell, i'_1, i'_2, \ldots, i'_\ell \in \mathbb{Q}^+, \\ & \text{tels que } x = x'_\ell, \text{ et } x_j = \tau_1^{i_j}(x'_{j-1}), x'_j = \tau_2^{i'_j}(x_j), j = 1..\ell \end{aligned}$$

REMARQUE 16 Comme nous nous plaçons dans le cas de polyèdres convexes fermés, nous désirons en fait calculer la clôture de l'ensemble précédent, c'est à dire l'ensemble des limites des suites de points restant dans $g_1 \cap g_2$ et résultant d'une suite d'applications rationnelles de τ_1 et de τ_2. Nous notons cette clôture de la même façon dans la suite.

REMARQUE 17 Il est clair que $\tau^{\circ}_{1,2}(P_0)$ est une surapproximation de l'enveloppe convexe de $\big((\tau_1+\tau_2)^*(P_0)\cap g_1 \cap g_2\big)$. La figure 6.6 montre que la sur-approximation (ensemble grisé clair) peut être bien plus grande que $\big((\tau_1 + \tau_2)^*(P_0) \cap g_1 \cap g_2\big)$ (grisé plus foncé).

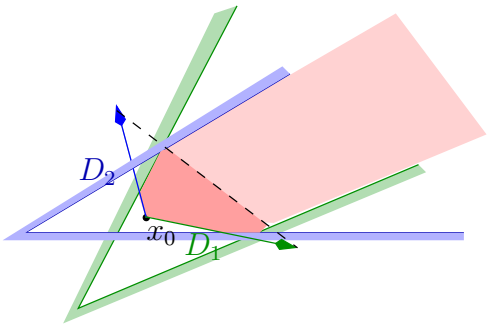


FIG. 6.6 – $\tau^{\circ}_{1,2}(P_0)$ est une surapproximation grossière de l'ensemble voulu

La proposition suivante donne un algorithme pour calculer $\tau^{\circ}_{1,2}(P_0)$:

PROPOSITION 14 *Soit $\tau_i : g_i \to x := x + D_i$, $(i = 1, 2)$, alors :*

- *Si il existe $\widetilde{x} \in P_0 \cap g_1 \cap g_2$, $\exists \varepsilon > 0$ tel que soit $\widetilde{x} + \varepsilon D_1 \in g_1 \cap g_2$, soit $\widetilde{x} + \varepsilon D_2 \in g_1 \cap g_2$ (i.e., il y a au moins un point de P_0 à partir duquel au moins une des deux transitions peut être « rationnellement » appliquée tout en restant dans $g_1 \wedge g_2$), alors*

$$\tau^{\circ}_{1,2}(P_0) = ((P_0 \cap g_1 \cap g_2) \nearrow \{D_1, D_2\}) \cap g_1 \cap g_2$$

- *Sinon $\tau^{\circ}_{1,2}(P_0) = P_0 \cap g_1 \cap g_2$*

PREUVE : Soit $P_l = ((\tau_1 + \tau_2)^{\circ}(P_0))$ et $P_r = ((P_0 \cap g_1 \cap g_2) \nearrow \{D_1, D_2\}) \cap g_1 \cap g_2$.

- $P_r \subseteq P_l$. Cette inclusion est évidente.

– $P_l \supseteq P_r$.

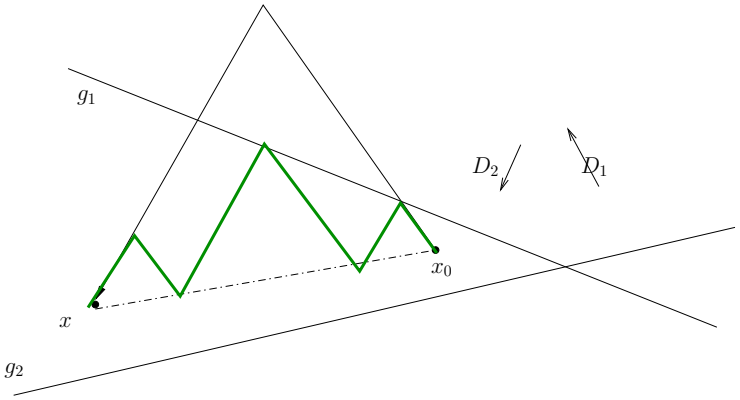


FIG. 6.7 – Démonstration de la proposition 14

Soit $x \in P_r$. Alors il existe $k_1, k_2 \in \mathbb{Q}^+$, et $x_0 \in P_0 \cap g_1 \cap g_2$ tels que $x = x_0 + k_1 D_1 + k_2 D_2$. L'hypothèse de l´énoncé induit que x_0 (ou un point de son voisinage) satisfait $\exists i_0, x_0 + i_0 D_1 \in P_0 \cap g_1 \cap g_2$. Soit alors $x_1 = x_0 + \varepsilon D_1 + \frac{\varepsilon}{k_1} k_2 D_2$. Par contruction et convexité x_1 est dans $P_0 \cap g_1 \cap g_2$ (sur le segment $[x_0, x_1]$), et $\|x - x_1\| < \|x - x_0\|$. En itérant le processus, on obtient une suite de x_i qui converge vers x (trait en gras sur la figure 6.7). x étant limite de points de $\tau^\circ_{1,2}(P_0)$, il appartient au polyèdre fermé clôture de $\tau^\circ_{1,2}(P_0)$. ■

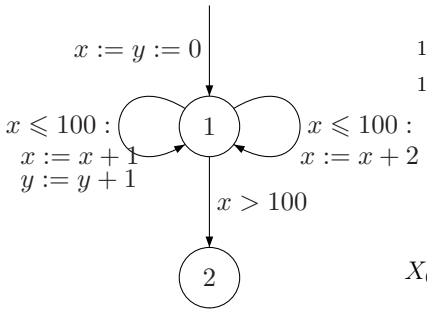
REMARQUE 18 La première condition sur $P_0 \cap g_1 \cap g_2$ provient du fait que l'application « rationnelle » de τ_1 ou τ_2 doit être initialisée. Cette condition est facile à tester (comme dans l'exemple 6.1) et peut être réduite à un problème de Programmation linéaire.

REMARQUE 19 Sur la figure 6.7, on a représenté g_1 comme une garde simple (hyperplan), mais le résultat est valable dans le cas général.

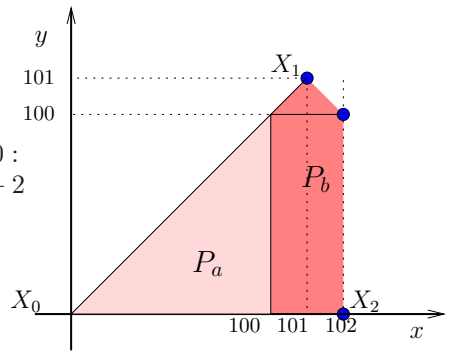
EXEMPLE 6.1 *Considérons le programme suivant tiré de [Hal79a] :*

```
x:=y:=0
tq x<=100
{ --1
  if ??
      x:=x+2
  else
      x:=x+1
      y:=y+1
} --2
```

(a)



(b)



(c)

FIG. 6.8 – Le programme, son CFG associé, le comportement des variables

Sur le « programme » (a) de la Figure, on obtient le graphe de flot de contrôle (b) en abstrayant l'instruction if-then-else *par un choix non déterministe autour du point de contrôle* 1. *Les deux gardes sont identiques, on peut donc calculer donc* $\tau^{\circ}_{1,2}(\{x = y = 0\}) = \{(0,0)\} \nearrow \{(1,1),(2,0)\} \cap \{x \leqslant 100\} = \{0 \leqslant y \leqslant x \leqslant 100\}$ *(polyèdre* P_a *sur la figure (c)).*

6.2.2 Partitionnement du GFC

Nous allons voir dans cette section comment utiliser le résultat précédent pour calculer l'effet des boucles multiples. Nous décidons dans un premier temps de partitionner le point de contrôle q (lieu des boucles multiples) selon les gardes g_1 et g_2.

Cette opération illustrée à la figure 6.9 consiste à :

- Créer les nouveaux points de contrôle q_1, q_2, q_{12} dont la sémantique est la suivante : lorsque le point de contrôle courant est q_1 (resp. q_2), la valuation courante satisfait $g_1 \wedge \neg g_2$ (resp. $g_2 \wedge \neg g_1$) ; lorsque le point de contrôle courant est q_{12}, cela signifie que le point de contrôle courant satisfait $g_1 \wedge g_2$.
- Créer les transitions correspondant à l'initialisation de ces différents points de contrôle : par exemple, pour le point de contrôle q_1, on crée la transition $(q_0, (g_1 \wedge \neg g_2, \varepsilon), q_1)$ (ε désigne l'action vide).
- Créer les transitions de retour des points de contrôle q_1, q_2 et q_{12} vers q', ces transitions sont de la forme $true \ \rightarrow \varepsilon$ (notées ε sur la figure).
- Créer les boucles simples autour des points de contrôle q_1, q_2 et q_{12}, en modifiant la garde de façon en prendre en compte l'invariant de l'état : par exemple, pour q_1, on crée la transition $\tau'_1 = (q_1, (pre((g_1 \wedge \neg g_2, a_1)), q_1)$, pour q_{12} on crée la transition $q_{1,2}$ avec la définition de la section précédente.
- Créer les transitions qui permettent de passer d'un point de contrôle à un autre. Par exemple, la transition $\mu_{1\rightarrow 2}$ de la figure représente la transition $(q_1, (pre(g_2 \wedge \neg g_1, a_1), \varepsilon), q_2)$.

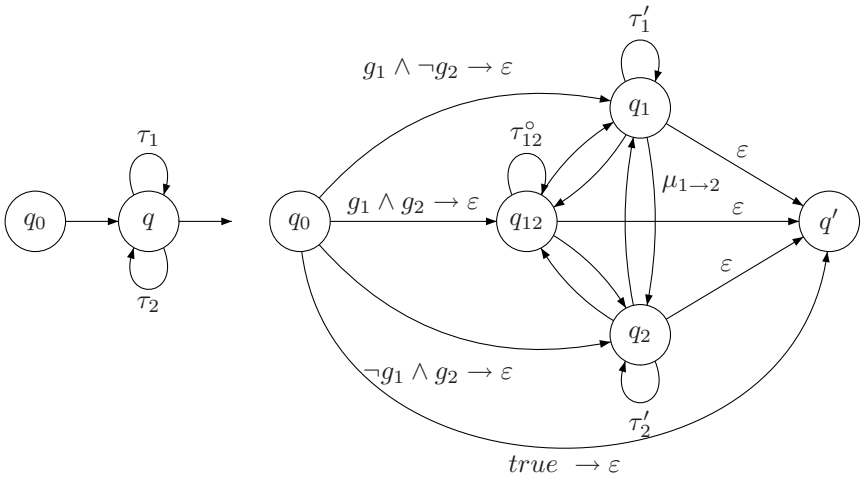


FIG. 6.9 – Partitionnement du point de contrôle selon les gardes

REMARQUE 20 La négation $\neg g_2$ d'un polyèdre g_2 n'est pas forcément un polyèdre mais une

union de polyèdres, on peut alors créer une nouvelle transition pour chaque polyèdre de cette union.

Il est clair que cette nouvelle configuration du sous graphe fortement connexe permet de calculer l'invariant associé au point de contrôle q comme étant l'union des invariants des points de contrôle q_1, q_2, q_{12} et q'.

6.2.3 Utilisation du partionnement

La stratégie utilisée lorsqu'on partionne est celle de l'Analyse des Relations Linéaire classique ; en effet ce partitionnement induit de nouvelles boucles, donc de nouvelles composantes fortement connexes, en particulier chaque nœud q_i est donc un nœud d'élargissement. Cependant, on remplace chaque application de τ_i par son accélération, *i.e.* on calcule $\tau_i^{\otimes}$ (ou τ_{12}°) du polyèdre considéré, ce qui réduit considérablement le nombre d'itérations.
Cependant, ce partionnement du graphe de flot de contrôle n'est en fait pas réalisé car il impliquerait une explosion combinatoire de la taille du graphe dans le cas de plusieurs boucles.

Une première heuristique proposée est d'utiliser l'expression calculée dans la Proposition 14 page 82, puis de calculer la solution approchée du système $P = P_0 \sqcup \tau_{1,2}^{\circ}(P) \sqcup \tau_1^{\otimes}(P) \sqcup \tau_2^{\otimes}(P)$, en utilisant l'opérateur d'élargissement si cela est nécessaire.
Expérimentalement, il arrive souvent que $P_0 \sqcup \tau_{1,2}^{\circ}(P_0) \sqcup \tau_1^{\otimes}(\tau_{1,2}^{\circ}(P_0)) \sqcup \tau_2^{\otimes}(\tau_{1,2}^{\circ}(P_0))$ est un post-point fixe, et alors l'élargissement n'a pas à être utilisé. Bien sûr, ce n'est qu'une stratégie parmi d'autres, mais elle donne expérimentalement de bons résultats. On aurait pu utiliser par exemple : $P_0 \sqcup \tau_{1,2}^{\circ}(P_0) \sqcup \tau_2^{\otimes}\big(\tau_1^{\otimes}(P_0)\big) \sqcup \tau_1^{\otimes}\big(\tau_2^{\otimes}(P_0)\big)$, ou d'autres combinaisons (comme cela est fait, par exemple, dans l'outil Fast [BFLP03]).

Exemple 6.2 *Revenons sur l'exemple 6.1. Une fois le polyèdre $\tau_{1,2}^{\circ}(P_0)$ obtenu, on effectue les calculs suivants :*
- $\tau_1^{\otimes}(\tau_{1,2}^{\circ}(P_0)) = \{0 \leqslant y \leqslant 100, y \leqslant x, x \leqslant 102\}$
- $\tau_2^{\otimes}(\tau_{1,2}^{\circ}(P_0)) = \{0 \leqslant y \leqslant x \leqslant 101, x - 100 \leqslant y\}$
- *L'enveloppe convexe des trois polyèdres précédents est* $\{0 \leqslant y \leqslant x \leqslant 102, y + x \leqslant 202\}$.

Ce dernier polyèdre est stable par l'application de τ_1 ou τ_2, donc on a atteint un post point fixe et donc on peut propager les informations obtenues vers le point de contrôle 2, pour lequel nous obtenons le polyèdre $\{x + y \leqslant 202, x \leqslant 102, 0 \leqslant y \leqslant x, x > 100\}$ (sur la figure 6.1 (c), c'est l'union des polyèdres P_a et P_b). Ce résultat obtenu en une seule itération est identique à celui obtenu par l'Analyse des Relations Linéaires avec élargissement « utpto », en faisant un pas de rétrécissement après l'analyse. On voit donc sur cet exemple que l'accélération permet dans certains cas les itérations descendantes.

Une deuxième possibilité est d'identifier les cas où l'on peut calculer une surapproximation plus précise de $\bigsqcup(\tau_1 + \tau_2)^*$, avec ou sans élargissement. C'est l'objet de la section suivante.

6.3 Vers une augmentation de la précision

Dans cette section, notre objectif est de calculer une sur-approximation la plus précise possible de $\bigsqcup(\tau_1 + \tau_2)^*$. Nous allons voir que nous sommes en mesure de caractériser dans certains cas cette enveloppe convexe, et dans d'autres cas nous pouvons donner la meilleure

sur-approximation convexe sans utiliser l'arithmétique entière. Dans un troisième temps, nous fournissons un algorithme qui permet de traiter le cas général.

6.3.1 Quelques résultats

Le premier résultat est une expression exact du polyèdre voulu :

PROPOSITION 15 *Si D_1 est un rayon de g_1, D_2 un rayon de g_2, alors*

$$\bigsqcup(\tau_1+\tau_2)^*(P_0) = \begin{cases} P_0 & \text{si } P_0 \cap g_1 = \emptyset \text{ et } P_0 \cap g_2 = \emptyset \\ P_0 \nearrow \{D_1\} & \text{si } P_0 \cap g_2 = \emptyset \text{ et } \big(P_0 \nearrow \{D_1\}\big) \cap g_2 = \emptyset \\ P_0 \nearrow \{D_2\} & \text{si } P_0 \cap g_1 = \emptyset \text{ et } \big(P_0 \nearrow \{D_2\}\big) \cap g_1 = \emptyset \\ P_0 \nearrow \{D_1, D_2\} & \text{sinon} \end{cases}$$

PREUVE :
Notons P le polyèdre $\bigsqcup(\tau_1+\tau_2)^*(P_0)$ Si $P_0 \cap g_1 = \emptyset$ et $P_0 \cap g_2 = \emptyset$, alors il est clair que $P = P_0$. Supposons maintenant $P_0 \cap g_2 = \emptyset$ et $P_0 \cap g_1 \neq \emptyset$. Dans ce cas, chacun des points de $P_0 \cap g_1 \nearrow \{D_1\}$ appartient à P (D_1 étant un rayon de g_1). D_1 étant un rayon, cela implique donc que chacun des points $X_0 + kD_1$, $k \in \mathbb{N}$, $X_0 \in P_0$ est dans P, et donc finalement $P_0 \nearrow \{D_1\} \subseteq P$. Maintenant, si la condition $P_0 \cap g_2 = \emptyset$ et $\big(P_0 \nearrow \{D_1\}\big) \cap g_2 = \emptyset$ est vérifiée, celà signifie que $P = \tau_1^*(P_0) = P_0 \cap \{D_1\}$, on a donc montré la deuxième égalité. Le reste de la démonstration est similaire

■

La proposition suivante montre que dans certains cas, les applications de τ_2 peuvent se faire avant les applications de τ_1. :

PROPOSITION 16 *Si D_2 est un rayon de g_1 et de g_2, alors $(\tau_1+\tau_2)^*(P_0) = \tau_1^*(\tau_2^*(P_0))$.*

PREUVE : Soit $X = \tau_2^2(\tau_1^3(\tau_2^4(X_0)))$. Montrons que $X = \tau_1^3(\tau_2^6(X_0))$. Pour cela, il suffit de montrer :

- $X_0 + 4D_2 \models g_2$ et $X_0 + 5D_2 \models g_2$. Cela vient du fait que $X_0 + 3D_2$ satifait g_2 (définition de X) et que D_2 est un rayon de g_2.
- $X_0 + 6D_2 \models g_1$. Cela provient du fait que $X_0 + 4D_2 \models g_1$ et D_2 est un rayon de g_1.
- $X_0 + 6D_2 + D_1 \models g_1$ et $X_0 + 6D_2 + 2D_1 \models g_1$. Cela provient de $X_0 + 4D_2 + D_1 \models g_1$ et D_2 rayon de g_1.

On obtient un résultat symétrique avec D_1. ■

Cette proposition fournit donc un algorithme simple calculant une sur-approximation de $(\tau_1+\tau_2)^*(P_0)$ en utilisant les résultats de sur-approximation du chapitre précédent (cas d'une unique boucle). Nous allons utiliser ces résultats dans l'algorithme de la section suivante.

6.3.2 Algorithme proposé

Précalcul : Tester si D_1 (resp. D_2) est un rayon de g_1 et/ou de g_2.

1. Si D_1 est un rayon de g_1 et D_2 un rayon de g_2 alors (cf proposition 15) :
   - Si $P_0 \cap g_1 = \emptyset$ et $P_0 \cap g_2 = \emptyset$ retourner P_0.
   - Sinon, si $P_0 \cap g_2 = \emptyset$, calculer $P_1 = P_0 \nearrow \{D_1\}$, puis :
     - si $P_1 \cap g_1 = \emptyset$, retourner $P_0 \nearrow \{D_1\}$,

  - sinon retourner $P_0 \nearrow \{D_1, D_2\}$.
- Sinon, si $P_0 \cap g_2 = \emptyset$, calculer $P_2 = \nearrow \{D_2\}$, puis :
  - si $P_2 \cap g_1 = \emptyset$, retourner $P_0 \nearrow \{D_2\}$,
  - sinon retourner $P_0 \nearrow \{D_1, D_2\}$.
- Sinon retourner $P_0 \nearrow \{D_1, D_2\}$.

2. Si D_2 est un rayon de g_1 ET de g_2, alors :
   - Calculer $P_2 = (P_0 \cap g_2) \nearrow \{D_2\}$ (applications de τ_2 en premier).
   - Calculer $P_{21} = (P_2 \cap g_1) \nearrow \{D_1\} \cap post(g_1, D_1)$
   - Retourner $P_0 \sqcup P_2 \sqcup P_{21}$
3. Si D_1 est un rayon de g_1 ET de g_2, alors algorithme symétrique du précédent (applications de τ_2 en premier)
4. Dans les autres cas :
   - Calculer $P_1 = (P_0 \cap g_1) \nearrow \{D_1\} \cap post(g_1, D_1)$.
   - Calculer $P_2 = (P_0 \cap g_2) \nearrow \{D_2\} \cap post(g_2, D_2)$.
   - Si $P_1 \subsetneq P_0$ alors calculer $P_{12} = (P_1 \cap g_2) \nearrow \{D_2\} \cap post(g_2, D_2)$.
   - Si $P_2 \subsetneq P_0$ alors calculer $P_{21} = (P_2 \cap g_1) \nearrow \{D_1\} \cap post(g_1, D_1)$.
   - Calculer $P' = P_0 \sqcup P_1 \sqcup P_2 \sqcup P_{21} \sqcup P_{12}$. Si ce polyèdre est stable par τ_1 et τ_2, retourner P' sinon, retourner $P_0 \nabla_{\mathcal{C}} \big(P' \sqcup \tau_1(P') \sqcup \tau_2(P')\big)$, avec $\mathcal{C} = \{post(g_1, D_1), post(g_2, D_2)\}$).

PROPOSITION 17 *Cet algorithme calcule une surapproximation de* $\bigsqcup(\tau_1 + \tau_2)^*(P_0)$. *Dans le premier cas, le résultat est l'enveloppe convexe exacte.*

PREUVE : Pour le premier cas, voir la proposition 15.

Pour l'expression suivante, la proposition 16 garantit que l'on peut faire toutes les applications de τ_2 en premier. L'algorithme utilisé (ajout de rayons et intersection par la post condition des gardes) garantit que l'on obtient une sur-approximation de l'ensemble voulu.

La démontration de la troisième expression est identique.

La dernière expression provient du fait que tous les points de $P_0 \sqcup P_1 \sqcup P_2 \sqcup P_{21} \sqcup P_{12}$ appartiennent au polyèdre cherché, mais ce ne sont pas les seuls. L'application de l'élargissement limité par les postconditions des gardes permet d'assurer que le polyèdre obtenu est bien une surapproximation du polyèdre voulu.

■

REMARQUE 21 Les expressions des cas 2 et 3 sont incluses dans la partie 4 de l'algorithme. Si le test 1 renvoie `faux`, alors on utilise le sous-algorithme 4 pour calculer l'expression voulue pour la sur-approximation de $\bigsqcup(\tau_1 + \tau_2)^*(P_0)$.

6.3.3 Quelques exemples

Pour clore le chapitre, voici quelques exemples de calculs :

EXEMPLE 6.3 *Sur l'exemple de la figure 6.5, on obtient successivement les polyèdres* P_1 *et* P_{12}, *l'enveloppe convexe permet d'obtenir la contrainte* g, *et finalement l'application de l'opérateur d'élargissement donne le polyèdre* P.

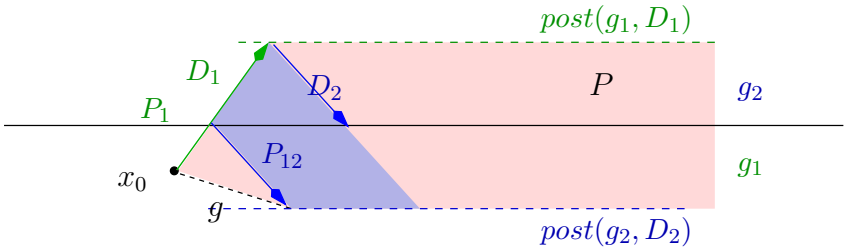


FIG. 6.10 – Trajectoire oscillante d'un point, le polyèdre calculé

EXEMPLE 6.4 *Reprenons l'exemple 6.1. Les deux vecteurs* $D_2 = \begin{pmatrix}2\\0\end{pmatrix}$ *et* $D_1 = \begin{pmatrix}1\\1\end{pmatrix}$ *ne sont pas des rayons de la garde* $g_1 = g_2 = \{x \leqslant 100\}$*. On calcule donc successivement :*

- $P_1 = \{0 \leqslant x \leqslant y \leqslant 101\}$.
- $P_2 = \{y = 0, 0 \leqslant x \leqslant 102\}$.
- $P_{12} = \{0 \leqslant x \leqslant 102, x \leqslant y, 0 \leqslant y \leqslant 100\}$
- $P_{21} = \{y = 0, 0 \leqslant x \leqslant 101\}$
- *L'enveloppe convexe donne* $\{0 \leqslant y \leqslant x \leqslant 102, x + y \leqslant 202\}$*. Cet ensemble est stable par l'application de* τ_1 *ou de* τ_2*, on ne fait pas d'élargissement.*

L'algorithme et les preuves précédentes peuvent facilement être adaptées aux cas de p boucles translations simples, nous donnons l'algorithme dans la section suivante.

6.4 Le cas p boucles

Dans le cas de p boucles simples autour du même point de contrôle, nous proposons l'algorithme suivant :

1. Pour tout $i \in [1, p]$, on calcule $P_{1p} = (P_0 \cap g_i) \nearrow \{D_i\} \cap post(g_i, D_i)$.
2. Pour tout i tel que $P_i \subsetneq P_0$, on calcule tous les $P_{2ji} = (P_i \cap g_j) \nearrow \{D_j\} \cap post(g_j, D_j)$
3. On calcule $P' = P_0 \sqcup \bigsqcup_i P_{1i} \sqcup \bigsqcup_{i,j \neq i} P_{2ij}$.
4. Si P' est stable par chacun des τ_i, retourner P', sinon, retourner $P' \nabla_{\mathcal{C}} \left(\bigsqcup_{i \in [1,p]} \tau_i(P') \right)$ avec $\mathcal{C} = \{post(g_1, D_1), post(g_2, D_2), \ldots, post(g_p, D_p)\}$.

Conclusion du chapitre

Dans ce chapitre, nous étudions le cas de p boucles de translation sur le même point de contrôle. Nous définissons une notion d'accélération abstraite pour des boucles dont les gardes sont simultanément vérifiées. Deux traitements sont alors proposés :

- Une première solution consiste à partitionner le contrôle selon les gardes des translations considérées, cette solutionayant le principal inconvénient étant que ce partitionnement induit une explosion du contrôle.

– Une deuxième solution identifie une sous-classe des translations précédentes dont nous pouvons donner une accélération simultanée. Dans les autres cas, une sur-approximation de l'ensemble voulu est donné en utilisant un élargissement limité.

Expérimentalement, la deuxième solution donne des résultats plus précis, et c'est donc cette solution qui sera implémentée dans notre outil (voir le chapitre 8).

Chapitre 7

Le cas des boucles multiples composées de translations et translations remises à constantes

Dans ce chapitre, on s'intéresse au cas des combinaisons de boucles translations et remises à constantes. Nous commençons par quelques cas faciles de combinaisons pour lesquels nous fournissons des algorithmes simples pour une sur-approximation, nous identifions ensuite une classe intéressante de boucles combinées pour lesquelles une surapproximation précise est calculable, et nous terminons par une proposition d'algorithme pour le calcul d'une sur-approximation convexe dans tous les cas.

7.1 Combinaison d'une boucle translation et d'une boucle remise à constante, un premier cas simple

Dans cette section et la suivante, nous nous intéressons à la combinaison d'une unique boucle de translation, notée τ_1 et d'une unique boucle de translation/remise à constante, τ_2. La figure 7.1 résume les notations.

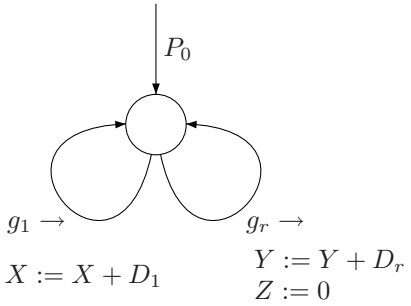


FIG. 7.1 – Une boucle translation et une boucle translation/remise à constante

On décompose $X = (Y, Z)$. Dans la première boucle, toutes les variables sont translatées,

tandis que dans la seconde, seules les variables Y sont translatées et les variables Z sont remises à 0 (le cas de remise à constante c est similaire). ici explication sur ces notations!!!

REMARQUE 22 Il est très utile de supposer que P_0 satisfait $Z = C$, c'est-à-dire que chaque variable Z vaut initialement sa valeur de remise à constante. On peut se ramener à ce cas en appliquant une première fois τ_r à P_0 (ou à $\tau_1^*(P_0)$).

7.1.1 Un premier cas simple : la deuxième boucle ne réalise que des remises à constante

Un sous-cas facile du problème est celui dans lequel la seconde boucle remet à 0 toutes ses variables, *i.e.* $Y = \emptyset$. Si l'on note alors $d_1 = D_1 \downarrow [z = 0]$ la projection de D_1 sur le sous-espace $z = 0$, on s'aperçoit alors que la trajectoire des variables sur le plan (D_1, d_1) (figure 7.2) est simple et son enveloppe convexe peut aisément être calculée en ajoutant les rayons d_1 et D_1 au polyèdre initial.

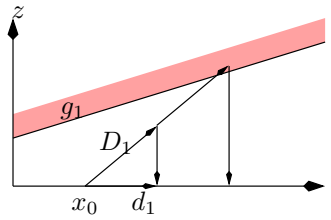


FIG. 7.2 – Remise totale à constante

Plus formellement, nous obtenons la proposition :

PROPOSITION 18 *Si* $\tau_1 : g_1 \to x := x + D_1$ *et* $\tau_2 : true \to z := 0$ *et* $P_0 \subseteq g_1 \cap \{z = 0\}$, *alors :* $P_0 \nearrow \{D_1, d_1\} \cap g_1$ *est une surapproximation de* $(\tau_1 + \tau_r)^*(P_0)$.

PREUVE : Tout d'abord, remarquons que $\tau_r^+ = \tau_r$ (τ_r est une projection). Si $x \in (\tau_1 + \tau_r)^*(x_0)$, alors on peut écrire la succession des applications de τ_1 et τ_r de la façon suivante (Les i_j sont dans $\mathbb{Q}^+$) :

$$x_0 \begin{pmatrix} y_0 \\ 0 \end{pmatrix} \to^{\tau_1^{i_1}} x_0 + i_1 D_1 \quad \to^{\tau_r} \quad \begin{pmatrix} y_0 + i_1 d_1 \\ 0 \end{pmatrix} \to^{\tau_1^{i_2}} x_0 + (i_1 + i_2) D_1$$
$$\to^{\tau_r} \quad \begin{pmatrix} y_0 + (i_1 + i_2) d_1 \\ 0 \end{pmatrix} \to \dots$$

Ainsi, si la chaîne se termine avec une application de τ_r, alors il existe $I_1 \in \mathbb{N}$ tel que $x = x_0 + I_1 d_1$ et $x \in g_1$. Si la chaîne termine avec une (ou plusieurs) applications de τ_1, alors $x = x_0 + I_1 d_1 + I_2 D_1$, avec $x \in g_1$. ■

REMARQUE 23 Sur le dessin, on a dessiné g_1 comme une garde simple, mais il n'y a pas de restriction sur le type de la garde g_1.

Ce cas simple se généralise facilement au cas $g_r \neq true$ et fournit ainsi un algorithme de surapproximation de l'enveloppe convexe :

PROPOSITION 19 *Si* $\tau_1 : g_1 \to x := x + D_1$ *et* $\tau_2 : g_r \to z := 0$ *et* $P_0 \subseteq g_1 \cap \{z = 0\}$, *alors :*
- *Si* $P_0 \nearrow \{D_1\} \cap post(g_1, D_1) \cap g_r = \emptyset$, *alors* $P_0 \nearrow \{D_1\} \cap post(g_1, D_1)$ *est une surapproximation convexe de* $(\tau_1 + \tau_r)^*(P_0)$.
- *Sinon,* $P_0 \nearrow \{D_1, d_1\} \cap post(g_1, D_1)$ *est une surapproximation de* $(\tau_1 + \tau_r)^*(P_0)$.

EXEMPLE 7.1 *Pour le programme de la figure 7.3, on obtient exactement l'enveloppe convexe des états accessibles.*

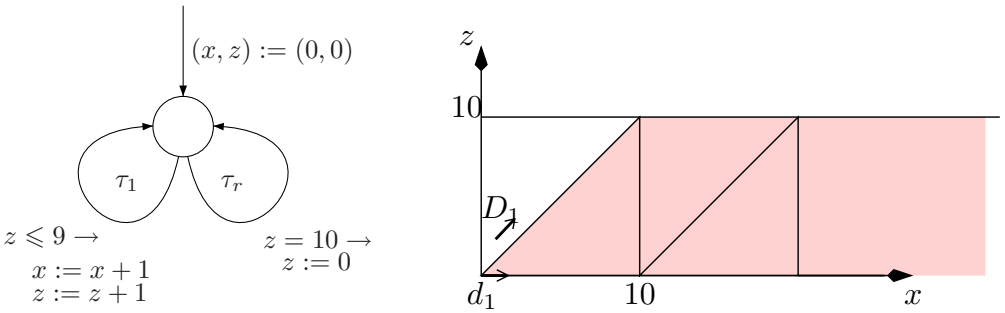


FIG. 7.3 – Deux boucles combinées et l'enveloppe convexe des états atteints

Avec les notations précédentes (x *est la variable* 1, z *la variable 2), on a* $D_1 = \begin{pmatrix}1\\1\end{pmatrix}$ *et* $d_1 = \begin{pmatrix}1\\0\end{pmatrix}$. *On calcule* $P_0 \nearrow \{D_1\} \cap \{x \leqslant 10\}$, *ce polyèdre intersecte* $g_r = \{x = 10\}$. *Finalement, on trouve le polyèdre* $P_0 \nearrow \{D_1, d_1\} \cap \{x \leqslant 10\}$, *ce qui est bien le résultat voulu.*

REMARQUE 24 On peut remarquer que de telles combinaisons de boucles peuvent produire des domaines d'accessibilité qui ne peuvent être décrits par des formules de Presburger, comme on peut le voir sur la figure 7.4. L'ensemble exact est en effet $\exists k \geqslant 0, \left(2^{k-1} - 1 \leqslant x \leqslant 2^k - 1 \wedge z \geqslant 0 \wedge x \geqslant 2z - 1\right)$ (en foncé sur la figure). Dans ce cas, les méthodes d'accélérations ne peuvent donc pas converger. Ici notre accélération combinée fournit l'ensemble : $\{x \geqslant z \geqslant 0, x \geqslant 2z - 1\}$ (en clair sur la figure).

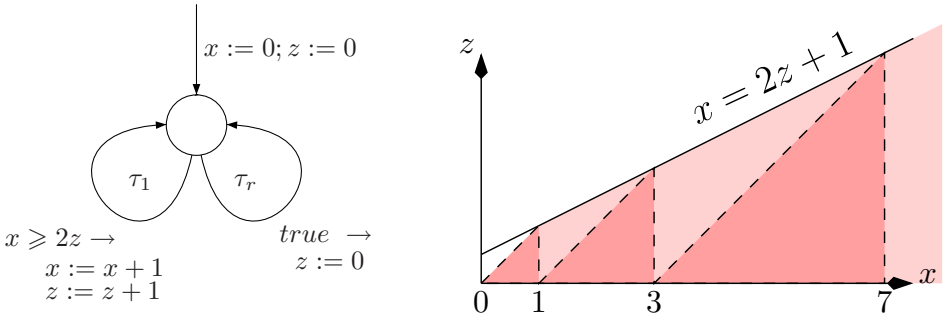


FIG. 7.4 – Deux boucles combinées avec comportement non Presburger-définissable

7.1.2 Vers une généralisation au cas $Y \neq \emptyset$

On peut remarquer que si le vecteur D_r appartient au plan D_1, d_1, alors le résultat de la proposition 18 se généralise facilement, (on ajoute alors le rayon D_r en plus de d_1 si $P_0 \nearrow \{D_1\}$ intersecte la garde). Les autres cas sont détaillés dans la section suivante.

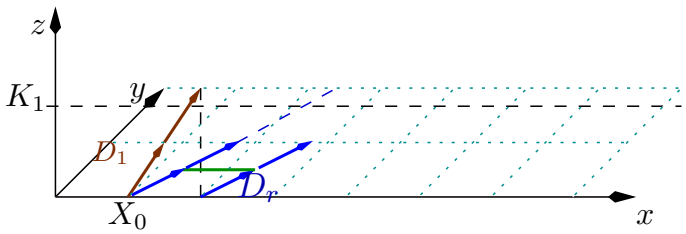
7.2 Combinaison boucle de translation et boucle remise à constante : le cas $Y \neq \emptyset$

7.2.1 Un premier résultat

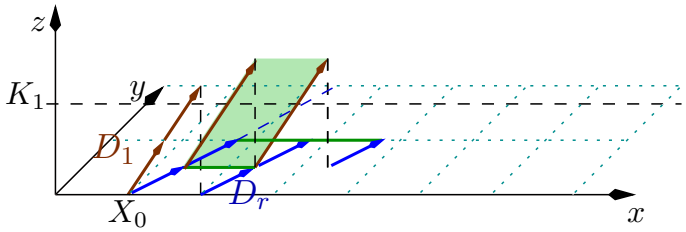
Maintenant considérons le cas où $D_r \neq 0$ n'appartient pas au plan (D_1, d_1). Cela signifie que certaines variables sont translatées dans la seconde boucle, mais pas dans le plan (D_1, d_1). On suppose dans un premier temps que $g_r = true$, mais aussi que g_1 est une garde simple de la forme $z \leqslant K$, c'est-à-dire est parallèle à l'hyperplan $z = 0$ (voir à ce sujet la remarque 26 ci-dessous). De plus, on considère aussi que $P_0 \subseteq \{z = 0\} \cap g_1$. Les variables évoluent donc comme schématisé ci-dessous :

À partir de x_0 satisfaisant g_1 et faisant partie du plan $z = 0$, on peut appliquer τ_1 ou τ_r. Du point x_0, la translation τ_1 peut être appliquée au plus k_{max} fois, où k_{max} est le minimum des quantités $\lfloor K_{z_i}/D_{1z_i} + 1 \rfloor$, pour toutes les variables z_i qui sont remises à 0 (c'est le nombre maximum exact de tours de boucles pouvant être effectuées à partir d'un point satisfaisant $z = 0$ et $z \in g_1$). Nous notons cette quantité $k_{max} = \lfloor K/D_{1z} + 1 \rfloor$ (ici $k_{max} = 2$).

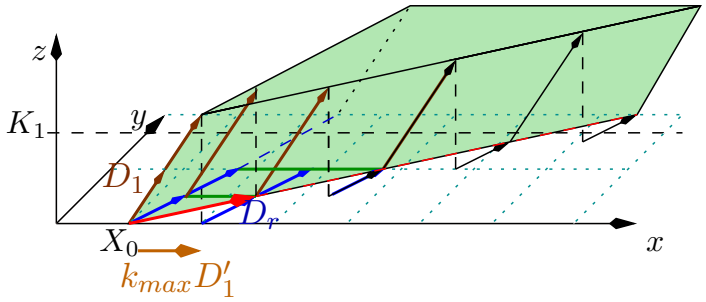
Entre chaque application de τ_1 peut être intercalée une application de τ_r (puisque la garde est toujours vérifiée). L'effet de cette application remet à 0 la variable z et effectue une translation de vecteur D_r dans le plan $\{z = 0\}$.



Ainsi, après quelques applications de τ_1 suivies d'une (unique) application de τ_r, on se retrouve avec un point $x = x_0 + kd_1 + k'D_r$ avec $d_1 = D_1 \downarrow [z = 0]$ défini comme auparavant, et $0 \leqslant k \leqslant k_{max}$. Et on peut ainsi répéter ce procédé.



Sur la figure, le vecteur $k_{max}D'_1 + D_r$ est un rayon du « segment de cylindre » :



On obtient donc la proposition :

PROPOSITION 20 *Soit τ_1 de la forme $(z \leqslant K) \rightarrow x := x + D_1$ et τ_r : true $\rightarrow y := y + D_{ry}; z := 0$. Supposons $P_0 \subset \{z = 0\}$. Soit aussi $d_1 = D_1 \downarrow [z = 0]$ et $D_r = \begin{pmatrix} D_{ry} \\ 0 \end{pmatrix}$. Alors*

- *Si $P_0 \nearrow \{D_1\}$ n'intersecte pas g_1, alors $P_0 \nearrow \{D_1, d_1, D_r\}$ est une surapproximation convexe de $(\tau_1 + \tau_r)^*(P_0)$*
- *Sinon, soit $k_{max} = \lfloor K/D_{1z} + 1 \rfloor$, alors $P_0 \nearrow \{D_1, D_r, k_{max}d_1 + D_r\} \cap post(g_1, D_1)$ est une est une surapproximation convexe de $(\tau_1 + \tau_r)^*(P_0)$*

REMARQUE 25 Si g_1 est une garde simple, alors on peut savoir si $P_0 \nearrow \{D_1\}$ intersecte g_1 en calculant le produit scalaire $D_1 \cdot u_z < 0$, avec $u_z = (0, \ldots, 0, 1, \ldots 1)$ un vecteur de taille n (nombre de variables) qui possède des 1 pour les composantes z, et des 0 ailleurs.

PREUVE : Sans perte de généralité, on peut supposer qu'à chaque fois que l'on applique τ_r^i ou τ_1^i, on a $i > 0$.

- Dans le premier cas, $P_0 \nearrow \{D_1\}$ n'intersecte pas g_1. Cela signifie que g_1 est toujours satisfait :

$$x_0 \rightarrow^{\tau_1^{i_1}} \rightarrow^{\tau_r^{i'_1}} x_0 + i_1 d_1 + i'_1 D_r \rightarrow^{\tau_1^{i_2}} \rightarrow^{\tau_r^{i'_2}} x_0 + (i_1 + i_2)d_1 + (i'_1 + i'_2)D_r \rightarrow \ldots$$

  Ainsi, si cette chaîne de transformations termine par une (ou plusieurs) applications de τ_r, alors $x = x_0 + Id_1 + I'D_r$ (en particulier, $z = 0$). Si la chaîne termine par une ou plusieurs applications de τ_1, on obtient $x = x_0 + Id_1 + I'D_r + I"D_1$, avec $I, I', I"$ non bornés.
- Dans le deuxième cas, le nombre d'itérations de τ_1 suivant une ou plusieurs applications de τ_r est au plus k_{max}. On écrit donc une chaîne similaire, sauf que l'on a la propriété $\forall j, 0 \leqslant i_j \leqslant k^{max}$. Si la chaîne termine avec τ_r^+, alors $x = x_0 + (i_1 + i_2 + \ldots + i_n)d_1 + (i'_1 + \ldots + i'_n)D_r$. Soit alors $I = i_1 + i_2 + \ldots + i_n$. En faisant la division euclidienne de I par k_{max}, on obtient $I = qk_{max} + r$ avec $r \leqslant k_{max}$ et $q \leqslant n$. Alors $x = x_0 + (q.k_max + r_1)d_1 + (q + r_2)D_r$ avec $r_1, r_2 \geqslant 0$, et donc $x = x_0 + q(k_{max}d_1 + D_r) + r_1 d_1 + r_2 D_r$, qui est de la forme voulue. x satisfait aussi g_1. Si la chaîne termine avec τ_1^*, on ajoute $i_{n+1}D_1$ à la formulation précédente.

■

EXEMPLE 7.2 *On reconsidère une version plus simple de la voiture, dont la modélisation est détaillée dans l'exemple 4.3 (Chapitre 4). Cette modélisation provient de [HPR97].*

```
t := d := s := 0 ;
while true do
  if second then
     t := t+1 ; s:= 0 ;
  else if meter and s<=3 then
     d := d+1 ; s := s+1 ;
end
```

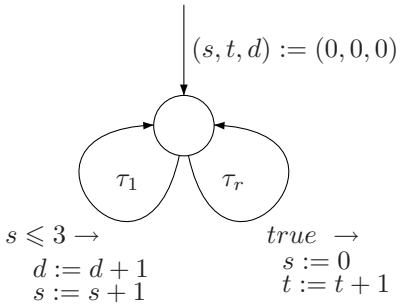


FIG. 7.5 – Simplification de l'exemple de la voiture

Avec les notations de la figure 7.1, on a $x = (t,d,s), y = (t,d), z = (s)$, $u_z = (0,0,1)$, $g_1 = (s \leqslant 3)$, $D_1 = (0,1,1)$, $D_r = (1,0,0)$, *et* $x_0 = (0,0,0)$. *De plus,* $u_z \cdot D_1 = 1 \geqslant 0$ *donc on calcule* $k_{max} = 4$ *et* $d_1 = (0,1,0)$, *puis* $k_{max}d_1 + D_r = (1,4,0)$, *et finalement on obtient comme polyèdre associé au corps de la boucle* `while` *:*

$$\begin{aligned} \tau_{1r}^{\circ}(x_0) &= (0,0,0) \nearrow \{(0,1,1),(1,0,0),(1,4,0)\} \cap \{s \leqslant 3\} \\ &= \{t \geqslant 0, 0 \leqslant s \leqslant 3, 0 \leqslant d \leqslant 4t+s)\} \end{aligned}$$

On calcule ensuite τ_1^* *du résultat, puis* τ_r, *le résultat obtenu est stable. On obtient alors le résultat le plus précis que l'on peut obtenir. En utilisant l'analyse des relations linéaires classique, il faut faire trois itérations, et utiliser l'élargissement limité.*

REMARQUE 26 Si la garde n'est pas de la forme $z \leqslant K$, le comportement des variables peut être non linéaire, par exemple, dans la figure 26, une frontière de l'ensemble des états atteignables est une parabole.

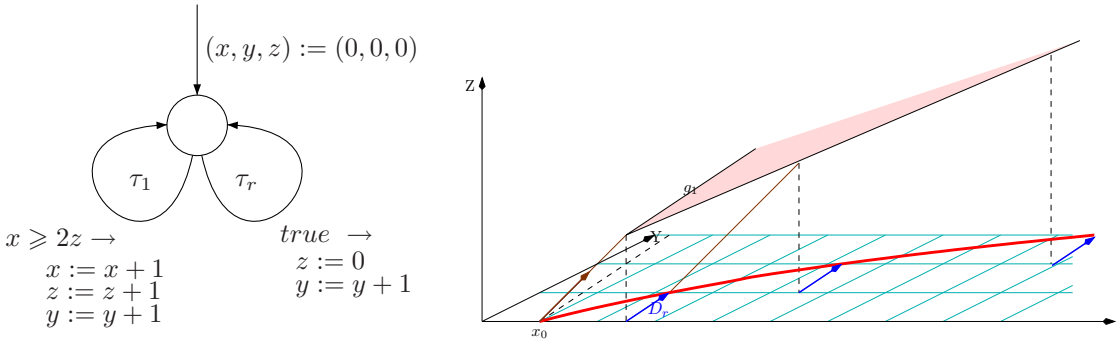


FIG. 7.6 – Un comportement parabolique

7.2.2 Résultats plus généraux

Les cas précédents peuvent être étendus à d'autres cas un peu plus généraux, par exemple :
- Si P_0 n'est pas inclus dans $\{z = 0\}$, on calcule d'abord $P_0' = \tau_r(\tau_1^{\otimes}(P_0))$, qui est inclus dans $\{z = 0\}$.
- Pour les gardes, nous allons voir que l'on peut traiter d'autres cas pour g_r. Si la garde g_r ne prend pas en compte les variables reset, alors les résultats précédents peuvent s'appliquer avec une modification mineure, l'intersection par g_r.

PROPOSITION 21 *[GH06] Soit τ_1, τ_r de la forme :*

$$\tau_1 : (z \leqslant K_1) \rightarrow x := x + D_1 \text{ et } \tau_r : (z \bowtie K_r) \rightarrow y := y + D_r; z := 0$$

où $\bowtie \in \{\leqslant, =, \geqslant\}$. Supposons $K_1 > 0$ et $D_1 \cdot u_z > 0$ (i.e. D_1 n'est pas un rayon de g_1). Notons $k_{max1} = \lfloor K_1/D_{1z} + 1 \rfloor$, $d_1 = D_1 \downarrow [z = 0]$, et $k_{maxr} = \lfloor K_r/D_{1z} + 1 \rfloor$. On suppose encore ici que $P_0 \subseteq g_1 \cap \{z = 0\}$. Alors P' calculé de la façon suivante :

- *si $\bowtie$ est "$\leqslant$" alors*
  - *si $K_r < 0$ alors (τ_r ne s'applique jamais)*

$$P' = P_0 \nearrow \{D_1\} \cap post(g_1, D_1)$$

  - *si $K_r > K_1$ alors*

$$P' = P_0 \nearrow \{D_1, D_r, D_r + k_{max1} d_1\} \cap post(g_1, D_1)$$

  - *si $K_1 \geqslant K_r > 0$ alors*

$$P' = P_0 \nearrow \{D_1, D_r, D_r + k_{maxr} d_1\} \cap post(g_1, D_1)$$

- *si $\bowtie$ est "$=$" alors*
  - *si $K_1 \geqslant K_r > 0$ et $\exists k, K_r = kD_1 z$ alors*

$$P' = P_0 \nearrow \{D_1, D_r + kd_1\} \cap post(g_1, D_1)$$

  - *sinon (τ_r ne peut jamais s'appliquer)*

$$P' = P_0 \nearrow \{D_1\} \cap post(g_1, D_1)$$

- *si $\bowtie$ est "$\geqslant$" alors*
  - *si $K_r > K_1$ et $K_r > 0$ alors (τ_r ne s'applique jamais)*

$$P' = P_0 \nearrow \{D_1\} \cap post(g_1, D_1)$$

  - *si $K_1 \geqslant K_r \geqslant 0$, alors*

$$P' = P_0 \nearrow \{D_1, D_r + k_{max1} d_1, D_r + k_{maxr} d_1\} \cap post(g_1, D_1)$$

  - *si $K_r < 0$ alors*

$$P' = P_0 \nearrow \{D_1, D_r, D_r + k_{max1} d_1\} \cap post(g_1, D_1)$$

est une surapproximation précise de $(\tau_1 + \tau_r)^*(P_0)$.

PREUVE : La démonstration est très similaire à la précédente, sauf que l'on prend en compte la garde g_r. En particulier, si la seconde garde est de la forme $z = K_2$, alors on doit vérifier si oui ou non l'ensemble *réel* $\{x + iD_1, i \in \mathbb{N}\}$ intersecte g_2. On peut aussi décider de ne pas vérifier cette condition, dans ce cas on se retrouve avec une sur-approximation moins précise. ■

REMARQUE 27 Si $D_1 \cdot u_z < 0$ avec les notations de la Proposition 21, g_1 est toujours vérifiée, la surapproximation devient $P_0 \nearrow \{D_1, d_1, D_r\}$

7.2.3 Proposition d'algorithme pour le cas d'une translation et d'une translation-reset

Les résultats partiels précédents vont être combinés dans cette section afin de fournir un algorithme pour toute combinaison de deux boucles de translation et de translation-reset. Dans le cas favorable, les expressions seront les plus précises possibles, dans d'autre cas, le polyèdre calculé sera une sur-approximation grossière de l'ensemble voulu.

1. Si $Y = \emptyset$, on calcule d'abord $d_1 = D_1 \downarrow [z = 0]$. Ensuite :
   (a) Si $P_0 \subseteq g_1 \cap \{z = 0\}$, alors
   - Si $P_0 \nearrow \{D_1\} \cap post(g_1, D_1) \cap g_r = \emptyset$, alors retourner $P_0 \nearrow \{D_1\} \cap post(g_1, D_1)$
   - Sinon, retourner $P_0 \nearrow \{D_1, d_1\} \cap post(g_1, D_1)$.

   (b) Sinon, soit $P_1 = \tau_1^{\otimes}(P_0) = (P_0 \cap g_1) \nearrow \{D_1\} \cap post(g_1, D_1)$, puis $P_{1r} = \tau_r(P_1 \cap g_r)$. $(P_{1r} \subseteq \{z = 0\})$. Soit P_2 le résultat de l'application de (a) sur $P_{1r} \cap g_1$. Soit $P' = P_0 \sqcup P_1 \sqcup P_2 \sqcup P_{1r}$. Si ce polyèdre est stable par τ_1 et τ_2, retourner P' sinon retourner $P_0 \nabla_{\mathcal{C}} P'$ avec $\mathcal{C} = \{post(g_1, D_1)\}$.
2. Si $Y \neq \emptyset$ et $D_r \in Vect(D_1, d_1)$ (plan engendré par les rayons D_1 et d_1), alors on applique le même algorithme que le 1. en ajoutant le rayon D_r si $P_0 \nearrow \{D_1\} \cap post(g_1, D_1) \cap g_r \neq \emptyset$.
3. Sinon
   (a) Si D_r appartient au plan (D_1, d_1), alors appliquer le cas 1.

   (b) Sinon, si $P_0 \subsetneq g_1 \cap \{z = 0\}$:
   - Si D_1 est un rayon de g_1 ou les contraintes sur z dans g_1 ne sont pas de la forme $z \leqslant K_1$, alors retourner $P_0 \nearrow \{D_1, d_1, D_r\} \cap post(g_1, D_1)$.
   - Sinon, on applique les résultats de la proposition 21.
4. Si $P_0 \subsetneq g_1 \cap \{z = 0\}$, on se ramène au cas précédent en calculant $P_1 = \tau_1^{\otimes}(P_0) = (P_0 \cap g_1) \nearrow \{D_1\} \cap post(g_1, D_1)$, puis $P_{1r} = \tau_r(P_1 \cap g_r)$. On calcule le résultat P_2, puis $P' = P_0 \sqcup P_1 \sqcup P_2 \sqcup P_{1r}$. Si ce polyèdre est stable par τ_1 et τ_2, retourner P' sinon retourner $P_0 \nabla_{\mathcal{C}} P'$ avec $\mathcal{C} = \{post(g_1, D_1)\}$.

PROPOSITION 22 *Cet algorithme calcule une sur-approximation convexe de* $(\tau_1 + \tau_r)^*(P_0)$.

Remarque sur le calcul des nouveaux rayons Comme $g_1 = x \leqslant K_1$ et $P_0 \subset \{z = 0\}$, les images successives d'un point x_0 de P_0 sont $\{x_0 + kD_1 \mid 0 \leqslant k \leqslant k_{max}\}$ avec k_{max} *indépendant* de x_0. Maintenant considérons le rayon $D_r + k_{max}d_1$ de la Proposition 20.
Imaginons que l'on ait un algorithme pour calculer $\sqcup\tau_1^*(x_0)$ (le polyèdre représentant l'enveloppe convexe de tous les $x_0 + kD_1$, avec $0 \leqslant k \leqslant k_{max1}$), la Proposition 20 assure alors que l'on peut utiliser l'algorithme suivant pour calculer $D_r + k_{max}d_1$:

1. Sélectionner un point $x_0 \in P_0$.
2. Calculer le segment $S = [x_0, x_0^M] = \sqcup\tau_1^*(x_0)$ de façon exacte.
3. Calculer $D_r + k_{max}d_1 = \tau_r(x_0^M) - x_0$.

Maintenant, dans la Proposition 21, si nous voulons calculer les rayons $D_r + k_{max1}d_1$ et $D_r + k_{maxr}d_1$, on doit obtenir (si ils existent) les points « réels » de l'ensemble $S \cap g_r$, *i.e.* , les points qui sont atteignables par τ_1 avec $i \in \mathbb{N}$ itérations. Remarquons que mêmes si ces calculs sont des calculs sur des entiers, nous choisissons de les réaliser exactement. Cependant, si les variables sont incrémentées de 1, nous savons que l'ajout de rayon et l'intersection par la postcondition de la garde donne le résultat exact.

7.3 Cas de plusieurs boucles de translations combinées avec une unique boucle remise à constante

Considérons maintenant le cas de plusieurs boucles de translations avec une unique remise à constante. Cette section donne des pistes de réflexion vers un traitement de telles combinaisons. Il est à noter qu'un partitionnement de telles boucles est toujours possible afin de se ramener au cas de la section précédente : par exemple, on pourrait adapter l'algorithme de la section 6.2.2 en faisant en sorte d'avoir au plus une boucle de translation et une boucle de remise à constante sur le même point de contrôle.

Nous proposons les heuristiques suivantes :
- Calculer en premier l'accélération simultanée des boucles de translation.
- Si les boucles de translation ont un comportement régulier après une application de la boucle reset, c'est-à-dire que le nombre de leurs applications maximal est constant, alors ajouter les rayons $D_r + k^i_{max} d_i$ pour chacune des boucles.
- Si la garde de la boucle de remise à constante ne concerne pas les variables translatées dans les autres boucles, alors on peut intersecter par $post(g_r, D_r)$. (cf. l'exemple 4.3) page 52.

Expérimentalement, lorsque les contraintes g_i sont de la forme $z \leqslant K_i$, les résultats obtenus sont très bon car les premières applications combinées des τ_i puis la première application de τ_r permettent de capturer la régularité du comportement des variables.

Exemple 7.3 *Considérons encore l'exemple de la chaudière, où l'on suppose seulement que dans chaque intervalle de* 60 *secondes consécutives, le temps cumulé de fuite de gaz est au plus* 10 *secondes. Une nouvelle variable est introduite pour calculer le temps de fuite depuis la dernière fois où la variable* u *est remise à* 0 *:*

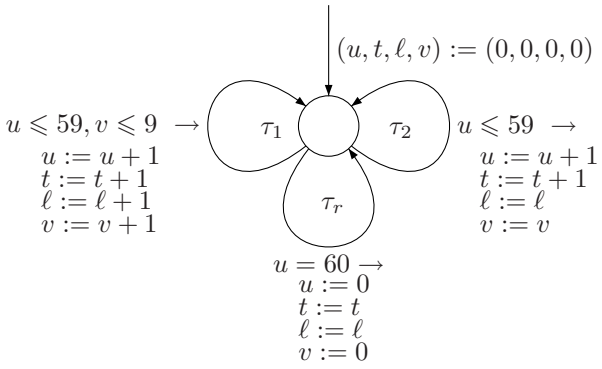


Fig. 7.7 – La chaudière deuxième version

On calcule d'abord :

$$\begin{aligned} P_1 = (\tau_1 + \tau_2)^{\otimes}(P^{(0)}) &= (0,0,0,0) \nearrow \{(1,1,1,1);(1,1,0,0)\} \cap \{u \leqslant 60, v \leqslant 10\} \\ &= \{0 \leqslant \ell = v \leqslant 10, \ell \leqslant u = t \leqslant 60\} \end{aligned}$$

Les résultats sont exacts car les actions sont uniquement des incrémentations de 1*. On intersecte ensuite avec g_r, ce qui donne le polyèdre*

$$P_2 = \{0 \leqslant l = v \leqslant 10, u = t = 60\}$$

en réappliquant l'ajout des rayons D_1 et D_2 on obtient :

$$P_3 = \{u + 60 = t, \ell \leqslant v + 10, 0 \leqslant v, v \leqslant l, v \leqslant u, u \leqslant 60, v \leqslant 10\}$$

puis en faisant l'enveloppe convexe avec P_1, et enfin en élargissant :

$$P_{fin} = \{u + 6l \leqslant t + 6v, v \leqslant l, u \leqslant 60, v \leqslant 10, 0 \leqslant, v \leqslant u\}$$

La projection sur les variables (ℓ, t) de ce dernier polyèdre est bien

$$\{0 \leqslant \ell \leqslant t, 6\ell \leqslant t + 50\}$$

Conclusion du chapitre

Dans ce chapitre nous avons vu que l'accélération des boucles simultanées de translation et de translation avec remise à constante est encore plus complexe que le cas du chapitre précédent. Nous avons identifié une classe intéressante de transformations situées sur le même point de contrôle pour laquelle nous savons donner une surapproximation précise du polyèdre image.

Chapitre 8

Aspic : un analyseur d'automates interprétés numériques avec accélération

Dans ce chapitre, nous décrivons l'outil Aspic, son langage d'entrée et ses principales options. Nous verrons aussi comment étudier les aspects numériques d'un programme Lustre*, via la compilation vers le langage intermédiaire* Oc*.*

8.1 Description de l'outil Aspic

Dans cette section, nous décrivons les techniques mises en œuvre dans notre outil Aspic (Accelerated Symbolic Polyhedral Invariant Computation).

8.1.1 Le format d'entrée

Le langage d'entrée de Aspic est un format d'automates textuel nommé Fast. Ce langage est le langage d'entrée de l'analyseur d'automates à compteur FastER ([FAS]), dont nous avons parlé au chapitre 3, section 3.3 (page 41). Un fichier Fast est formé de deux composantes (le manuel de description du langage peut être trouvé sur la page web [FAS]) :

- Un « modèle » qui comporte la description textuelle *d'un seul* automate à compteurs : déclaration des variables numériques et des noms des points de contrôle, et description des transitions : source, destination, garde numérique (convexe ou non) et action sur les variables numériques.
- Une « stratégie » qui contient un certain nombre de définitions de « régions » (des formules parlant des variables et des points de contrôle) et des objectifs de calcul. Lors du traitement du fichier Fast, un certain nombre de ces instructions de stratégies sont ignorées, comme par exemple les stratégies de parcours de l'automate. Les informations prises en compte sont les suivantes :
  - La région initiale, obligatoire : elle spécifie un ou plusieurs points de contrôle d'entrée ainsi qu'éventuellement une formule caractérisant la valuation initiale des variables.
  - La région finale, facultative : l'éventuel but de preuve est spécifié sous forme d'une formule caractérisant les états « mauvais » du programme (point de contrôle et/ou formule qui ne doit jamais être vraie). En l'absence de but de preuve, Aspic calcule

des invariants associés à chacun des points de contrôle du programme. En présence de but de preuve, ASPIC réalise une transformation du graphe afin de ramener le problème à une non atteignabilité d'un ensemble de points de contrôle.

Un exemple de fichier FAST analysé avec ASPIC peut être trouvé à l'annexe B.

8.1.2 L'analyseur

Bibliothèques utilisées Nous avons choisi d'utiliser le moteur de calcul de point fixe de Bertrand Jeannet, ANALYSEUR ([ANA]). Cette bibliothèque fournit un cadre général d'implémentation d'un analyseur utilisant l'interprétation abstraite. En particulier, la structure interne de graphe permet de représenter facilement le programme à analyser. Ce moteur est écrit en OCAML.

Nous avons ensuite choisi la bibliothèque de polyèdres NEWPOLKA ([New]) écrite en `C` mais qui possède une interface OCAML. La bibliothèque fournit les opérations classiques sur les polyèdres, permet d'utiliser les deux représentations, et fournit de plus des outils d'impressions performants.

La taille cumulée des fichiers OCAML constituant l'outil ASPIC est de 20000 lignes de code (dont 5000 lignes pour le moteur de point fixe, et sans compter la librairie NEWPOLKA).

Caractéristiques de ASPIC En interne, notre analyseur possède les caractéristiques suivantes :

- une structure complexe de type graphe encode la structure de l'automate à analyser.
- diverses tables encodent les caractéristiques des fonctions de transitions gardées, des sommets du graphe, les stratégies à utiliser.

Le moteur de calcul de point fixe utilise ces diverses informations ainsi que la bibliothèque de polyèdres afin de calculer des invariants associés à chaque point de contrôle. Nous avons modifié cet analyseur afin de prendre en compte nos résultats et heuristiques concernant l'accélération.

8.1.3 Techniques mises en œuvre

Dans cette section, nous allons détailler plus précisément les techniques mises en œuvre dans l'outil.

Analyse des relations linéaires L'outil ASPIC réalise une analyse d'accessibilité *en avant*, l'objectif étant de calculer des invariants polyédriques pour chacun des points de contrôle du programme. Si une région (formule sur les variables numériques et les états de contrôle) `bad` est définie dans le fichier d'entrée, alors un/des états de contrôle `__bad_i` est/sont créés et à la fin l'objectif de preuve est déclaré prouvé si les polyèdres associés aux points de contrôle `__bad_i` sont *tous* vides. Dans le cas contraire, on ne peut pas conclure et l'outil écrit *don't know*.

La stratégie de parcours du graphe est donnée par la structure en sous-composantes fortement connexes. La décomposition est pré-calculée au début de l'analyse par l'algorithme de Bourdoncle (cf. chapitre 2, section 2.3.5, page 23), et la stratégie choisie consiste à parcourir chacune des composantes fortement connexes en profondeur jusqu'à stabilisation.

REMARQUE 28 Nos expérimentations avec la librairie NEWPOLKA a permi de montrer que l'implémentation de l'union convexe multiple est très importante : en effet, cette union peut être effectuée à l'aide d'une suite d'unions convexes binaires, mais cela n'est pas efficace. Le coût en mémoire (et en temps) peut être fortement amélioré en modifiant l'algorithme d'union convexe multiple de la façon suivante : les rayons des polyèdres sont tous considérés en premier, et ensuite les sommets.

Détection et traitement des boucles accélérables Lors de la première phase de calcul dans laquelle sont mises en place les diverses tables concernant l'automate à analyser, les fonctions de transitions sont pré-traitées, la structure `transition` stocke alors le type de l'action (identité, translation, translation reset, transformation idempotente, ...), la garde (garde simple, garde composée, les variables impliquées)..., et le cas échéant, les informations nécessaires au calcul de l'accélération (garde post, rayon, ...).

Nous choisissons de modifier la structure de contrôle du graphe afin de traiter les boucles accélérables :

– **Cas des boucles simples**. Le cas des boucles simples (un seul circuit passant par la tête de la composante fortement connexe) est traité de la façon suivante : si la boucle est accélérable, alors le point de contrôle est découpé en deux points de contrôle reliés par la *meta-transition* correspondante. (figure 8.1).

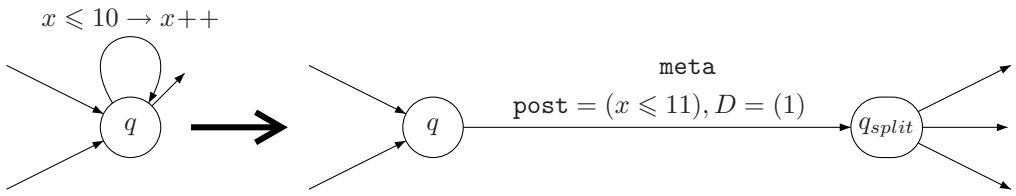


FIG. 8.1 – Cas d'une boucle simple

Ce partitionnement des boucles simples permet de supprimer l'application de l'élargissement sur le nœud q. En q_{split}, le polyèdre calculé est $\tau^{\otimes}(P_0)$. Nos résultats permettent l'amélioration de la précision puisque l'élargissement est évité.

– **Cas des boucles multiples**. Le cas des boucles multiples est plus complexe, comme nous l'avons montré dans les chapitres précédents. Nous décidons comme pour les boucles simples de décomposer le point de contrôle considéré en deux points de contrôle, comme le montre la figure 8.2.

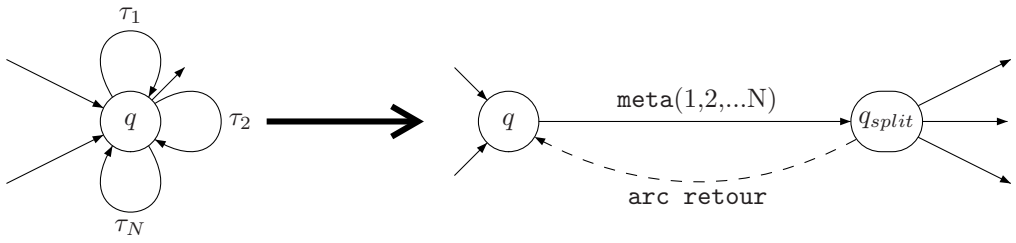


FIG. 8.2 – Cas de boucles multiples

Il y a deux possibilités de traitement des boucles multiples :
- Soit on utilise des résultats partiels d'accélération, et un arc de retour (en pointillé sur la figure, cet arc porte la fonction de transition $true \rightarrow \varepsilon$, *i.e.* la fonction identité avec garde $true$) est créé afin de récupérer les valuations manquantes ;
- Soit on utilise des résultats totaux (c'est-à-dire qui traitent une classe de boucles multiples entièrement) et alors l'arc de retour n'est pas nécessaire. Ce cas est similaire au cas des boucles simples.

La solution utilisée est en général la première.

- **Cas des boucles complexes**. Ce cas est traité de la façon suivante : lors du calcul des composantes fortement connexes, les circuits détectés sont stockés et on calcule la méta transition associée en arrière, *i.e.* on compose les actions (produit des matrices associées) et on calcule la garde en « remontant » les préconditions. Par exemple, considérons le circuit $(q, \tau_1, q_1)(q_1, \tau_2, q)$ avec $q_1 : x \leqslant 7 \rightarrow x := x+1$ et $q_2 : x \leqslant 4 \rightarrow x := x+3$. Alors on calcule la transition $q_2 \circ q_1 : x \leqslant 4 \rightarrow x := x + 4$ et puisqu'elle est accélérable, on rajoute une meta-transition autour du point de contrôle q. En revanche, les deux transitions du circuit sont conservées, pour conserver la sémantique globale du GFC. Le principal problème de cette approche est que tous les circuits du graphe ne sont pas détectés, en particulier dans le cas où deux circuits « parallèles » existent sur le même point de contrôle (figure 8.3). Cependant, la détection de tous les cycles du graphe ne nous paraissait pas envisageable pour des raisons de complexité. Le traitement des boucles complexes n'étant à ce jour pas terminé, les résultats expérimentaux ne prendront pas en compte celles-ci.

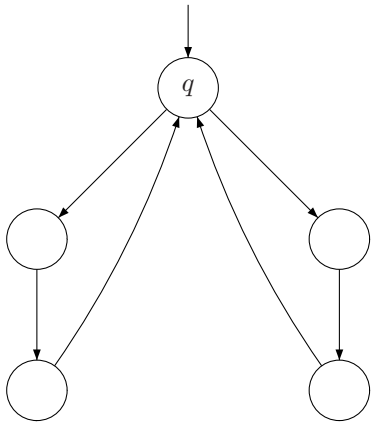


FIG. 8.3 – Circuits « parallèles »

Remarques sur la position des points d'élargissement Le calcul des points d'élargissement est effectué après la phase de partitionnement des points de contrôle comportant des boucles accélérables. En effet, la structure du graphe de contrôle se trouve modifiée (certaines boucles n'existent plus, en particulier celle du point de contrôle qui a été doublé). Pour le placement des points d'élargissement, nous avons modifié l'algorithme de Tarjan (section 2.3.5, page 23) de façon à favoriser les points de contrôle « q_{split} ». Lors du parcours du graphe, si un point de contrôle « split » q se retrouve tête de composante fortement connexe (c'est forcément le cas lorsqu'un arc de retour a été créé), alors nous choisissons de mettre q_{split} dans la liste des points d'élargissement. En effet, il est important d'élargir aux endroits où le

plus d'informations possibles sur le point-fixe ont été collectées. Expérimentalement, élargir *après* accélération est une bonne heuristique. Nous décidons aussi de retarder l'élargissement en ces points de façon à ce qu'il soit appliqué uniquement entre deux polyèdres « accélérés ».

8.1.4 Options de ASPIC

Dans cette section, nous décrivons comment utiliser ASPIC et quelles sont les options implémentées :

- `-delay n` : Première application de l'opérateur d'élargissement à la n-ième itération. Par défaut, $n = 1$. Cette option permet de jouer sur la précision.
- `-noaccel` : Analyse des relations linéaires sans accélération mais élargissement limité. Avec `-nouptos`, l'opérateur d'élargissement utilisé est celui de [CH78] (élargissement standard, sans « uptos », décrit dans le chapitre 2, section 2.4.3, page 29).
- `-simu` : Simulation de l'automate FAST avec choix interactif des valeurs des variables.
- `-reps` : Implémentation du « lookahead widening » ([GR06], voir le chapitre 4, section 4.4.4, page 47).
- `-nologs` : Impression du résultat uniquement (*don't know* ou *all the bad locations are unreachable*).

L'appel sans option sur un fichier d'entrée sous format FAST réalise une analyse des relations linéaires avec élargissement standard et accélération, avec les techniques décrites dans la section 8.1.3(page 102). Un fichier `log` est créé et permet de garder trace des différentes étapes du calcul. À la fin de l'analyse, les polyèdres correspondants à chaque point de contrôle sont affichés. Un exemple d'analyse peut être trouvé à l'Annexe B.

8.2 Vérification de programmes LUSTRE

Les automates interprétés que nous vérifions sont des automates interprétés numériques, cependant notre objectif est de pouvoir vérifier les aspects numériques de programmes plus généraux. Ce travail s'inscrivant dans une recherche à long terme sur la vérifications de programmes réactifs écrits dans le langage déclaratif synchrone LUSTRE ([HCRP91]), nous décidons d'utiliser ce langage pour décrire nos systèmes à vérifier. Le principal avantage de ce langage est que nous pouvons décrire dans le même langage système à vérifier et propriétés.

8.2.1 LUSTRE et les outils existants

LUSTRE est un langage déclaratif synchrone, pour la programmation des systèmes réactifs. LUSTRE est le socle de l'environnement de programmation SCADE, utilisé dans l'industrie pour la conception de logiciels critiques, par exemple dans l'avionique, l'automobile, l'énergie. La figure 8.4 donne un exemple de nœud LUSTRE.

```
node accumulateur(val : int ; reset : bool) returns(acc : int) ;
let
    acc = val -> if(reset) then 0 else pre(acc) + val ;
tel
```

Fig. 8.4 – Exemple de nœud Lustre

Ce nœud calcule à chaque tic d'horloge, à partir des deux entrées `val` (un entier) et `reset` (un booléen) (qui sont des flux de valeurs), une sortie entière `acc`, dont la valeur est définie par :

- la valeur initiale est la valeur initiale du flot `val` ;
- au tic $t > 0$, si la valeur du flot booléen `reset` est `true`, alors la valeur du flot `acc` sera `0`, sinon, ce sera la valeur au tic précédent (`pre(acc)`) augmenté de `val`.

Outils existants Plusieurs outils ont été développés pour effectuer la vérification de programmes Lustre (ces outils supposent que les programmes vérifiés possèdent un *observateur* ([HLR93]), *i.e.* un nœud, prenant en entrée les variables influençant la propriété, et calculant une unique sortie booléenne, dont la valeur est mise à « faux » dès que la propriété est violée) :

- Lesar ([HLR92]) est un « model-checker » dédié à Lustre, capable de vérifier des programmes purement booléens, ou des abstractions booléennes de programmes généraux, dont les aspects non booléens (en particulier, les aspects numériques) sont ignorés.
- Pour prendre en compte les aspects numériques, l'outil Nbac met en œuvre une analyse de relations linéaires avec partitionnement dynamique ([Jea00]). L'idée est de partir d'une structure de contrôle minimale, d'effectuer une analyse avant/arrière et si la propriété n'est pas vérifiée, un partitionnement de la structure de contrôle est effectuée en prenant en compte l'analyse précédente.

Placement de Aspic dans la chaîne de vérification de Lustre Nous aurions pu modifier Nbac pour prendre en compte nos algorithmes d'accélération, mais nous voulions d'une part vérifier des automates dont la structure de contrôle est déjà fixée et ne change pas, et d'autre part les aspects booléens de Nbac compliquaient notre étude. Nous avons donc développé un nouvel analyseur, mais les résultats obtenus pourraient être utilisés dans Nbac dans le cas numérique uniquement. Pour une utilisation complète il faudrait étudier une notion d'accélération de transition mixte booléenne/numérique. Cet analyseur est rattaché à l'ensemble des outils Lustre via le format intermédiaire Oc, qui fait l'objet de la section suivante.

8.2.2 Le format intermédiaire Oc

Le format Oc [PS87] (pour *object code*) a été défini pour représenter les automates interprétés générés par les premiers compilateurs des langages synchrones Lustre et Esterel. Un avantage de baser notre analyseur sur ce format est d'ailleurs que l'analyseur devrait pouvoir être utilisé tant pour des programmes Lustre que pour des programmes Esterel.

Un fichier OC est organisé de la façon suivante :
- Une série de tables, définissant, pour ce qui nous intéresse, des signaux, des variables, et des actions.
- Un automate.

Les *signaux* peuvent être d'un type prédéfini (booléens, entiers, chaîne de caractères, ...) ou d'un type défini par l'utilisateur. La définition d'un nouveau type et des fonctions qui le manipulent se fait au moyen de tables (des types ou des fonctions). Pour les types prédéfinis, on dispose déjà d'un certain nombre de fonctions prédéfinies, opérateurs booléens ou arithmétiques standards,... Les signaux correspondent aux entrées et sorties du programme, chaque entrée de la table comporte le nom, la nature (entrée ou sortie), le type et l'indice du signal. L'indice fait référence à la table des variables.

La *table des variables* comporte toutes les variables du programme : ce sont les entrées/sorties, les variables internes ou des variables spécifiques (comme une mémoire associée à un `pre`). Cette table donne le type de chaque variable.

La *table des actions* spécifie toutes les actions élémentaires que l'automate peut effectuer lors de ses transitions. Une action élémentaire est une suite de fonctions (prédéfinies ou non) à exécuter en séquence. Les actions qui vont nous intéresser sont les actions conditionnelles (if then else) et les transformations de variables (affectations). D'autres actions (émission ou test de présence d'un signal, ...) sont spécifiques à ESTEREL.

L'automate est décrit comme une suite d'états : à chaque état est associé une « transition », en fait un *dag* (directed acyclic graph) composé d'actions (faisant référence à la table d'actions). La structure de dag est due à l'ouverture, et à la fermeture éventuelle, d'actions conditionnelles. Chaque branche du dag se termine par une action de branchement à l'état suivant. Ainsi, le dag de la figure 8.5 représente les transitions issues de l'état « 1 » : en supposant que b et y sont des variables d'entrée, celles ci sont implicitement lues sur toutes les transitions. Alors,
- si b est faux, et si $y < 0$, une transition met x à 1, et conduit à l'état 2 ;
- si b est faux, et si $y \geq 0$, une transition met x à 1, et conduit à l'état 3 ;
- si b est vrai, une transition met x à 1, incrémente z, et conduit à l'état 3.

Comme il a été dit plus haut, chaque état de l'automate correspond à un nouvel instant de l'exécution du programme LUSTRE ou ESTEREL, et par conséquent, en début de chaque transition, toutes les variables d'entrée sont lues et prennent une nouvelle valeur.

8.2.3 Compilation de LUSTRE en OC

La compilation de LUSTRE en automates interprétés a été décrite dans [HRR91]. Le principe est le suivant :
- Les variables d'état (mémoire du programme) sont les valeurs retournées par les opérateurs « `pre` » du programme, ainsi que celle d'une variable auxiliaire utilisée pour distinguer le premier cycle de tous les autres (vraie au premier cycle, fausse ensuite).
- Parmi ces variables d'états, certaines sont d'états finis (en particulier, les booléens) et peuvent être codées dans la structure de contrôle : connaissant la valeur v_x d'une variable

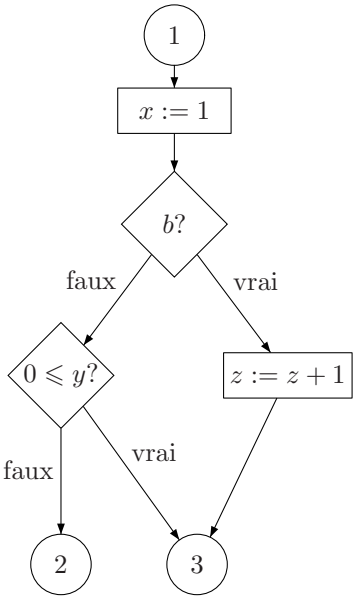


Fig. 8.5 – Une « transition » Oc

x au cycle n, on peut spécialiser le code à effectuer au cycle $n+1$, en remplaçant toutes les occurrences de `pre`(x) par v_x. Ce sont ces fragments de code spécialisés qui constituent les états de l'automate.

Voici un exemple de compilation de Lustre vers Oc.

Exemple 8.1 *Le « programme » suivant incrémente la variable x sur chaque front montant de l'entrée booléenne b :*

```
node compte_fronts (b : bool) returns (x : int) ;
var front : bool ;
let
    x = 0 -> if front then pre(x)+1 else pre(x) ;
    front = false -> (b and not pre(b)) ;
tel
```

La génération de l'automate procède comme suit (voir la figure 8.6)

- *Au cycle initial (état 0), les opérateurs « -> » s'évaluent comme leurs premiers arguments. On a donc :* x=0 ; front=false. *Si* b *est vrai, « pre(b) » sera vrai à l'instant suivant (état 1), et inversement (état 2) si* b *est faux.*
- *Dans l'état 1, on sait que le cycle n'est pas initial, et que « pre(b) » vaut vrai. Donc, «* front *» est faux, et la valeur de* x *ne change pas. Comme dans l'état 0, l'état suivant (1, ou 2) est choisi selon la valeur de* b.
- *Dans l'état 2, on sait que le cycle n'est pas initial, et que « pre(b) » vaut faux. Donc,*
  - *si* b *est vrai,* front *est vrai aussi,* x *est incrémenté, et le prochain état est l'état 1*
  - *sinon,* front *est faux,* x *est inchangé, et le prochain état est l'état 2.*

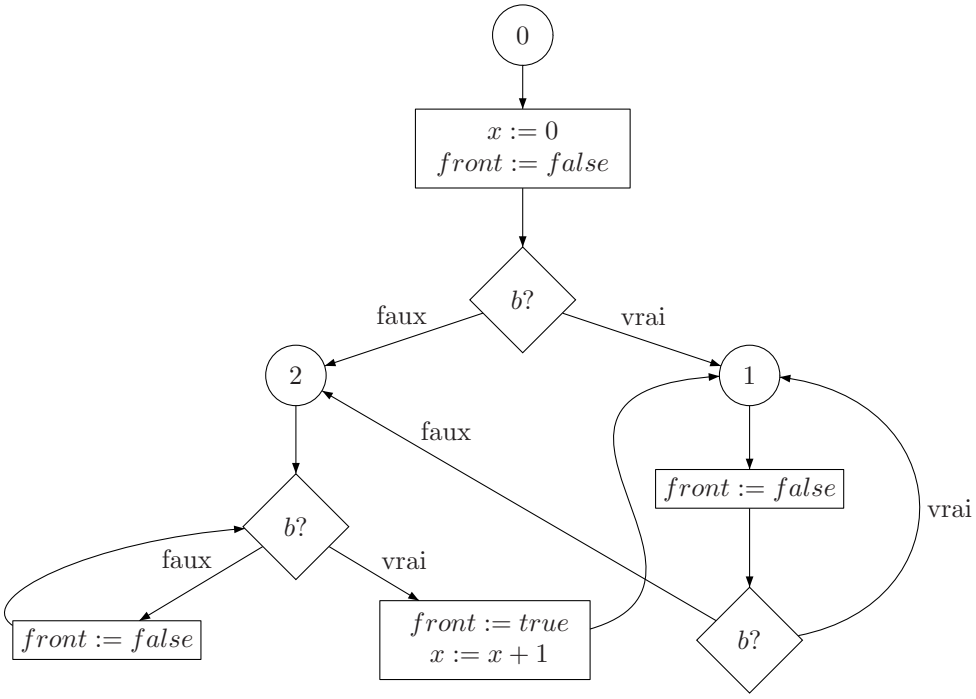


FIG. 8.6 – Automate interprété du compteur de fronts

La traduction de LUSTRE en OC est effectuée par le compilateur LUSTRE-V4. Un certain nombre d'options de compilation permettent de jouer sur la structure de contrôle obtenue. Ces options permettent de :

- choisir les variables booléennes qui sont traduites dans la structure de contrôle, les autres restant traitées comme des données ;
- choisir la stratégie d'ouverture et de fermeture des test dans les transitions, entre une version minimale, où seuls les branchements au prochain état font l'objet d'actions de test (les autres restant des expressions conditionnelles), et une version maximale, où tous les test sont ouverts, et jamais refermés (le dag est alors un arbre de transitions).

Ces options conduisent à une structure de contrôle plus ou moins complexe, tant en nombre d'états qu'en complexité des transitions OC, ce qui, en ce qui nous concerne, permettra d'ajuster le compromis entre précision et complexité de l'analyse.

Par ailleurs, l'automate obtenu peut être minimisé par bisimulation (en confondant les états qui correspondent au même code), soit à la compilation, soit au moyen de l'outil OCMIN. Cette minimisation, en général peu coûteuse, est évidemment souhaitable avant toute traduction de code OC vers un autre format.

8.2.4 Traduction de OC vers FAST

Nous avons réalisé un traducteur de OC vers le format FAST (OC2FST). Nous supposons pour cela que le fichier OC d'entrée a été obtenu à partir d'un fichier LUSTRE (avec ou sans observateur) et des options $-2-min$ (toutes les variables booléennes sont traduites dans la

structure de contrôle, et l'automate est minimal). Le code (6500 lignes de C++) utilise le l'analyseur syntaxique de OC de la distribution LUSTRE, et une partie de la structure interne d'automates de l'outil de la thèse [Mer05].

L'outil réalise essentiellement :
- le découpage de la structure de contrôle OC en introduisant des points de contrôle intermédiaire au niveau des tests ;
- l'abstraction des signaux booléens d'entrée en rajoutant des transitions non déterministes ;
- si l'automate LUSTRE initial comportait un observateur, le OC contient un produit et un unique signal booléen de sortie. OC2FST réalise alors la transformation du problème en un problème d'accessibilité : un état `bad` est créé qui peut être atteint si on écrit `false` sur cet unique signal booléen.

REMARQUE 29 Le compilateur ARGOS ([ARG]) réalise un travail similaire à partir du langage d'entrée ARGOS/LARISSA, un langage d'automates synchrone qui autorise la composition parallèle et l'encapsulation, ainsi que la programmation par aspects ([AMS06]). L'option `-FAST` réalise l'aplatissement du programme en un unique automate FAST.

EXEMPLE 8.2 *Sur l'exemple précédent, après abstraction des variables booléennes on trouve l'automate de la Figure 8.7.*

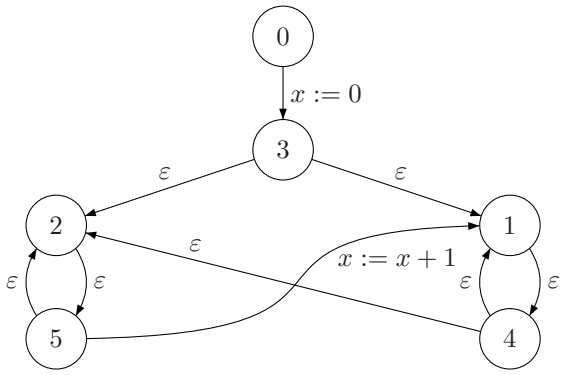


FIG. 8.7 – Automate interprété numérique du compteur de fronts

On remarquera que cet automate n'est pas déterministe et qu'il n'est pas minimal du point de vue du comportement numérique. Cependant il conserve la structure du OC *initial.*

REMARQUE 30 Dans [DG03] et [GHR04], on montre comment une formule de calcul des durées ([CHR91b],[Pan01]) peut être automatiquement transformée en un automate observateur LUSTRE. Cette transformation a été codée dans l'outil DCPROP. Les nœuds LUSTRE générés font souvent intervenir des compteurs de temps, et donc sont particulièrement adaptés à une vérification ultérieure par ASPIC. Le principal inconvénient de la méthode est que l'abstraction des valeurs booléennes peut se révéler trop grossière lors de la transformation de OC vers FAST.

Le schéma 8.2.4 résume les différents enchaînements d'outils (les outils réalisés sont en gras).

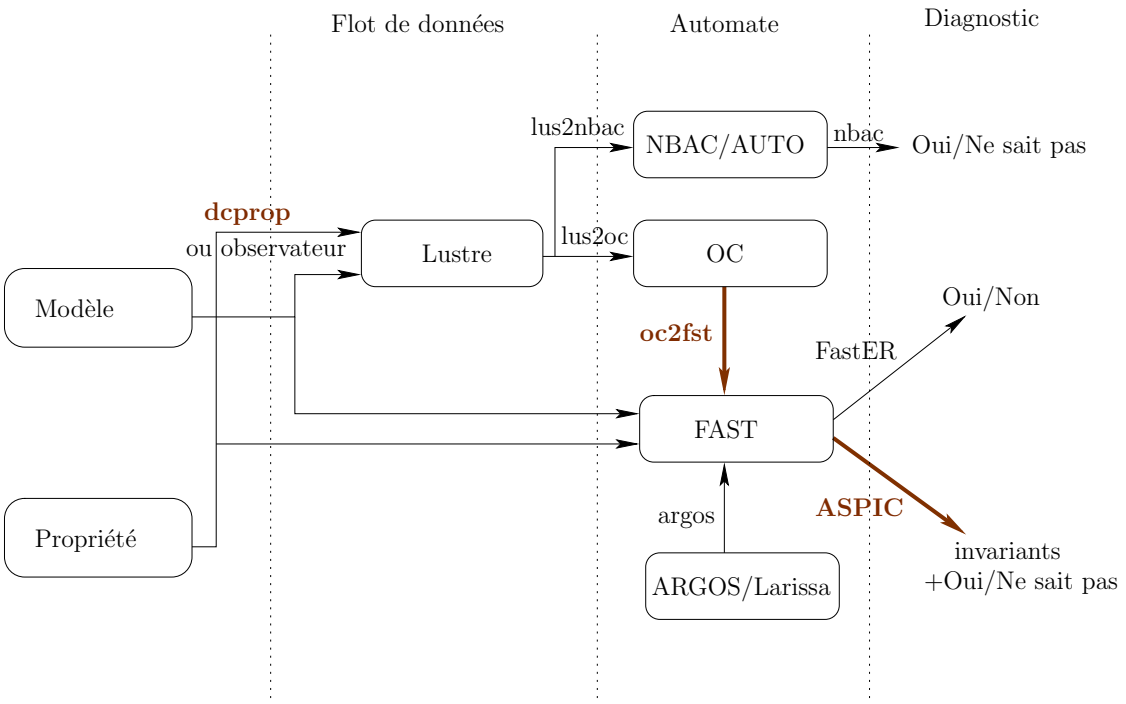


FIG. 8.8 – Placement d'ASPIC dans la chaîne d'outils LUSTRE

Conclusion du chapitre

Dans ce chapitre, nous avons présenté notre outil ASPIC, qui implémente nos algorithmes d'accélération combinés avec une Analyse des Relations Linéaires. Un certain nombre de choix ont été faits pour cette implémentation, parmi lesquels les plus importants sont ceux concernant les modifications de structure des automates à analyser.

L'outil ASPIC permet d'analyser des automates interprétés numériques, avec ou sans but de preuve, et nous avons fourni le traducteur OC2FST afin de pouvoir analyser des automates LUSTRE.

Chapitre 9

Résultats expérimentaux

Dans ce chapitre, nous présentons quelques résultats expérimentaux obtenus avec Aspic. *Ces résultats montrent que la méthode proposée dans cette thèse offre des résultats probants tant en terme de précision que de rapidité d'exécution.*

9.1 Exemples de petite taille

Pour les exemples de petite taille, nous avons comparé les résultats de notre outil avec :

- l'Analyse des Relations Linéaires classique, sans élargissement limité, sans heuristique du nouveau chemin, avec deux élargissements : l'élargissement standard ([Hal79a]) et l'élargissement proposé dans [BHRZ03]. Cette analyse est implémentée dans notre outil ainsi que dans l'outil StInG ([StI]).
- l'Analyse des Relations linéaires classique, avec ou sans élargissement limité, avec heuristique du nouveau chemin (section 4.4.3), qui est implémentée dans Aspic.
- l'Analyse des Relations Linéaires avec « lookahead widening » (section 4.4.4), que nous avons aussi implémentée dans Aspic.
- l'outil FastER, qui implémente les techniques d'accélération exactes décrites dans [Ler03] et [Bar05] (voir le chapitre 3). La version de FastER utilisée ne donne pas le point fixe, elle donne uniquement les informations sur les boucles accélérées. Les essais fournissent donc uniquement une estimation du temps de calcul.

La difficulté induite par les formats d'entrée des différents outils a été réduite par le choix du format d'entrée Fast pour notre outil. Les outils Aspic et StInG pouvant réaliser la même analyse, nous avons pu vérifier la cohérence entre deux versions du même automate numérique, l'un codé en Fast et l'autre avec le format d'entrée de StInG.

9.1.1 Description des différents exemples

Les différents exemples utilisés sont les suivants :

- La figure 9.1 représente les exemples classiques *Hal79a* et *Hal79b*, qui proviennent de la thèse [Hal79a].

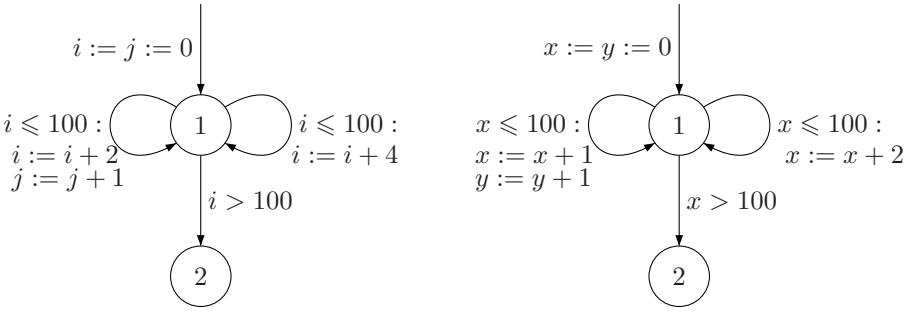


FIG. 9.1 – Les exemples Hal79a et Hal79b

– L'exemple *SP* est l'exemple « swimming pool » décrit dans [FO97] sous forme de réseau de pétri (modélisant le comprtement de différents « composants » d'une piscine) puis de « programme de Presburger ». Cet exemple est repris dans [Bar05].
– L'exemple *Chaudière* est l'exemple de la chaudière décrit à la section 2.6.2.
– L'exemple *VSimple* est l'exemple 7.2 (chapitre 7).
– L'exemple *Voiture* est l'exemple de la voiture décrit à la l'exemple 4.3 (chapitre 4).

9.1.2 Résultats

Sur le tableau de la figure 9.2, on donne les invariants trouvés par les différentes méthodes associés à des points de contrôle pertinents (la colonne CH79-V2 a été obtenue avec l'heuristique « nouveau chemin »). Aucun temps de calcul n'est précisé car les analyses sont instantanées.

Nom	BHRZ03	ssm04	CH79 v2	Lookahead	**ASPIC**
Exemples sans remise à constante dans les boucles					
Hal79a	$\{0 \leqslant j, 2j \leqslant i\}$	idem BHRZ03	$\{0 \leqslant j, 2j \leqslant i \leqslant 104\}$	idem Aspic	$\{\boldsymbol{i + 2j \leqslant 204}, i \leqslant 104, 0 \leqslant j, 2j \leqslant i\}$
Hal79b	$\{0 \leqslant y \leqslant x\}$	idem BHRZ03	$\{0 \leqslant y \leqslant x \leqslant 102\}$	idem Aspic	$\{0 \leqslant y \leqslant x \leqslant 102, \boldsymbol{x + y <= 202}\}$
SP	$\{x2 + x3 + x4 + x7 = p2, x1 + x2 + x4 + x5 + x6 = p1, \ldots\}$				idem
Chaudière	$\{0 \leqslant x \leqslant \ell \leqslant t\}$	idem ch79	idem Aspic	$\{0 \leqslant x \leqslant \ell \leqslant t\}$	$\{\boldsymbol{6\ell \leqslant t + 5x}, 0 \leqslant x \leqslant 10, x \leqslant \ell, 0 \leqslant \ell\}$
Train1	$\{d = 9, 20 \leqslant b, b \leqslant s + 20\}$	$\{\boldsymbol{11 \leqslant b}, 1 \leqslant b - s \leqslant 20 \ldots\}$	idem BHRZ03	idem BHRZ03	idem BHRZ03
Exemples avec remises à constante					
VSimple	$\{0 \leqslant s \leqslant d, 0 \leqslant t\}$	idem BHRZ03	idem Aspic	$\{0 \leqslant s \leqslant d, \boldsymbol{s \leqslant 4}, 0 \leqslant t\}$	$\{0 \leqslant s \leqslant 4, s \leqslant d, \boldsymbol{d \leqslant 4t + s}, \}$
Voiture	$\{0 \leqslant s \leqslant d, 0 \leqslant t\}$	idem BHRZ03	idem Aspic	$\{0 \leqslant s \leqslant d, \boldsymbol{s \leqslant 2}, 0 \leqslant t\}$	$\{0 \leqslant s \leqslant 2, s \leqslant d, \boldsymbol{d \leqslant 2t + s}, t \leqslant 3\}$

FIG. 9.2 – Comparaison d'invariants obtenus par différents outils

Sur ces exemples, on peut remarquer que l'outil ASPIC arrive à détecter des invariants complexes, plus précis que les autres outils. Lorsque l'Analyse des Relations Linéaires classique (colonne CH79-v2) donne le même résultat, elle utilise l'élargissement limité ainsi que l'heuristique du nouveau chemin, et converge en plus d'itérations (1 à 2 itérations de moins sur 5 sur les petits exemples).

Nous avons également lancé l'outil FASTER sur ces mêmes exemples. Les analyses sont instantanées, à l'exception de la chaudière et de la voiture simple : nous avons arrêté les analyses à 15 min, les automates de Presburger intermédiaires devenant trop gros (plus de 8000 états dans chacun des deux cas).

REMARQUE 31 Les encodages FAST des différents automates cités sont disponibles sur la page [ASP].

9.2 L'exemple du métro

Dans cette section, nous décrivons les résultats obtenus pour la vérification d'un système de régulation de métro, dont la modélisation a été proposée dans [HPR97]. Chaque train détecte des balises qui sont placés le long d'un rail, et reçoit un signal « seconde » d'une horloge centralisée. Chaque train i ajuste sa vitesse selon la valeur de $b_i - s$, la différence entre le nombre de balises vues et le nombre de secondes. Pour améliorer le confort des passagers, on applique une hystérésis et le train est considéré en avance si $b_i \geqslant s + 10$, le train met alors son frein en marche tant que $b > s$. Le train s'arrête alors si le frein est en marche depuis 10 secondes. De manière symétrique, le train accélère si $b_i \leqslant s - 10$ et ceci, tant que $b_i < s$. L'horloge centrale n'émet pas le signal seconde tant qu'il existe un train en retard, ce qui permet au train en retard de récupérer son retard en rencontrant de nouvelles balises.

Le système complet est codé en LUSTRE (cf Annexe C), la compilation vers le langage OC nous permettant d'automatiser le produit. Nous utilisons alors notre traducteur OC2FST pour créer un automate numérique en format FAST contenant uniquement les aspects numériques du programme. Le traducteur génère beaucoup de transitions, cela vient du fait que toutes les combinaisons de gardes numériques sont prises en compte, y compris celles dont la garde est formée de contraintes qui ne sont pas compatibles. Pour cet exemple, le fichier FAST pour un métro comporte 5 points de contrôle et 88 transitions, pour deux métros 17 points de contrôle et 1627 transitions. L'expérience montre que certaines transitions sont presque identiques, et donc l'outil OC2FST pourrait être amélioré.

Cas de 1 métro La figure 9.3 donne le résultat obtenu dans [HPR97] pour un train unique, obtenue en utilisant l'analyse des Relations Linéaires directement sur le produit synchrone des automates. Notre traduction générant beaucoup de transitions, le résultat obtenu avec l'Analyse des Relations linéaires classique est moins précis :

- **OnTime** : $\{d = 0, -10 \leqslant s - b < 10, 0 \leqslant b, 0 \leqslant s\}$.
- **Late** : $\{d = 0, 0 < s - b < 11, 10 \leqslant s\}$
- **OnBrake** : $\{s - b < 0, 0 \leqslant s, 0 \leqslant d < 11\}$
- **Stopped** : $\{d = 10, s - b < 10, 0 \leqslant s\}$.

Considérons les propriétés suivantes :

- p1 : `state = stopped => 19 <= 9s + b`

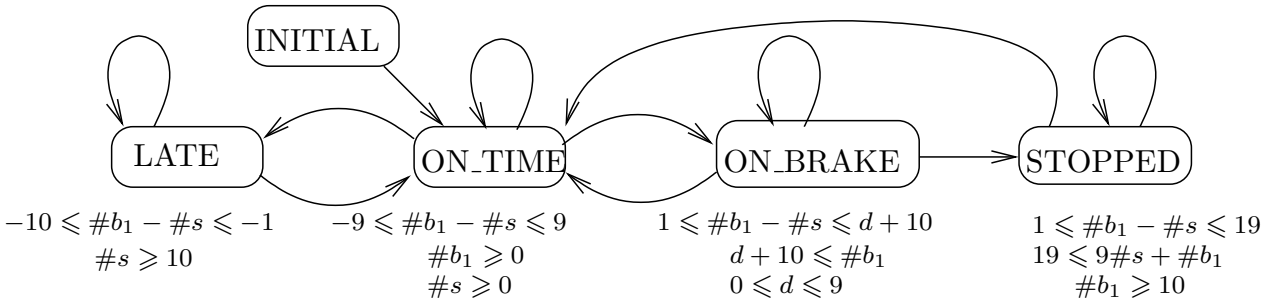


FIG. 9.3 – Le résultat théorique du métro à un train

– p2 : `state = stopped => b>=10`

L'analyse des relations linéaires classique (ainsi que celle avec *lookahead widening*) prouve uniquement la première propriété, alors que l'analyse accélérée est capable de prouver les deux.

Au niveau performances les résultats sont quasiment instantanés dans les 3 cas, l'analyse accélérée converge en 5 itérations contre 7 avec l'analyse classique et 9 avec « lookahead widening ».

Cas 2 métros Pour le cas de deux métros, les temps d'analyse sont sensiblement plus grands : 7 minutes avec accélération (6 itérations), 1 minute pour l'analyse avec « lookahead widening », 1 minute (8 itérations) pour l'analyse des relations linéaires classique. Le surcoût provient de la complexité mémoire des polyèdres obtenus après chaque accélération de plusieurs boucles. En effet, si l'on se trouve dans un cas avec plusieurs boucles accélérables en parallèles, mais que l'on ne sait pas globalement accélérer, on choisit d'effectuer l'enveloppe convexe des résultats obtenus par chacune des boucles. Les contraintes ainsi générées peuvent être en très grand nombre et faire intervenir de grands coefficients. Cependant, les invariants obtenus par l'analyse accélérée sont plus petits.

Nous pensons que la prise en compte des boucles complexes ainsi qu'une meilleure traduction de OC vers FAST pourrait sensiblement améliorer la précision et la performance de cette analyse.

9.3 Applications à d'autres sujets d'étude

9.3.1 Analyse d'atteignabilité sur des automates MicMac

Dans [CMMC07], les auteurs proposent un formalisme d'automates permettant d'encoder des processus SystemC. SystemC est une librairie C++ utilisée pour décrire des systèmes sur puce, sous forme de composants asynchrones qui peuvent exécuter des processus et des fonctions externes ([Ope05]). Pour chaque processus (ou fonction), un automate Micmac est généré (figure 9.4, l'automate de gauche représente la fonction principale `SC_THREAD`, l'automate de droite représente la fonction appelée `g`), dans lesquels deux types d'états sont distingués :

– Les micro-états représentent les états dans lesquels il est possible de rendre la main à l'ordonnanceur.

– Les macro-états représentent les états dans lesquels le processus ne peut rendre la main.

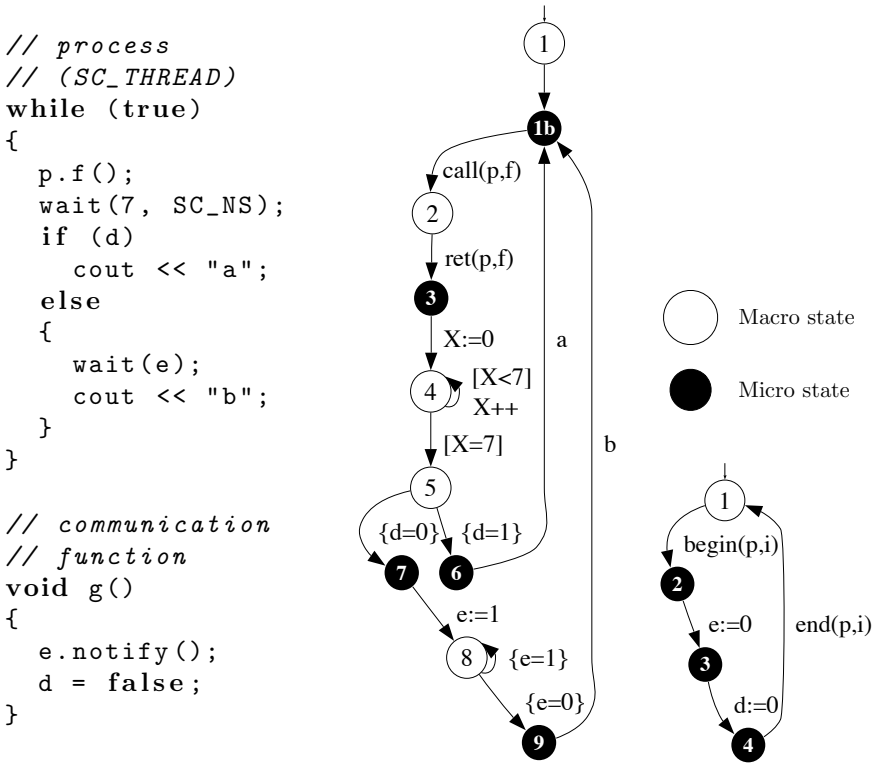


FIG. 9.4 – Code SystemC et automate **MicMac** correspondant ([CMMC07])

La communication entre ces automates s'effectue par variables partagées. Un produit *ad-hoc* (qui prend en compte les différents ordonnancements possibles) est ensuite réalisé afin de générer un automate interprété encodant les différents comportements, le but étant ensuite de pouvoir tester l'inclusion de traces entre deux tels automates. La figure 9.5 montre le produit réalisé sur un exemple « jouet » représentatif des programmes SystemC étudiés. Sur cette figure, on remarque que les automates n'ont pas de boucles imbriquées, les seules boucles apparaissant sont des boucles simples qui proviennent du « passage du temps ».

Le principal inconvénient de la méthode est que le produit généré est assez gros. Une première approche choisie consiste à utiliser des outils de vérification afin de supprimer de l'automate produit les points de contrôle et transitions non atteignables. Notre outil permet alors de calculer un sous-ensemble des points de contrôle/transitions non atteignables, et un réimport permet de supprimer les branches correspondantes dans l'automate produit.

Une traduction de l'automate **MicMac** produit vers des automates FAST a été effectuée par J. Cornet au laboratoire Verimag. Une fois le produit effectué, les automates deviennent des automates FAST, la distinction entre micro et macro état n'étant plus faite. Sur ce produit, on lance ASPIC afin de déterminer un sur-ensemble des états atteignables. Les premières expérimentations sur des exemples « jouets » montrent que l'outil ASPIC permet d'effectuer les analyses voulues en un temps raisonnable et avec des résultats non triviaux, comme le

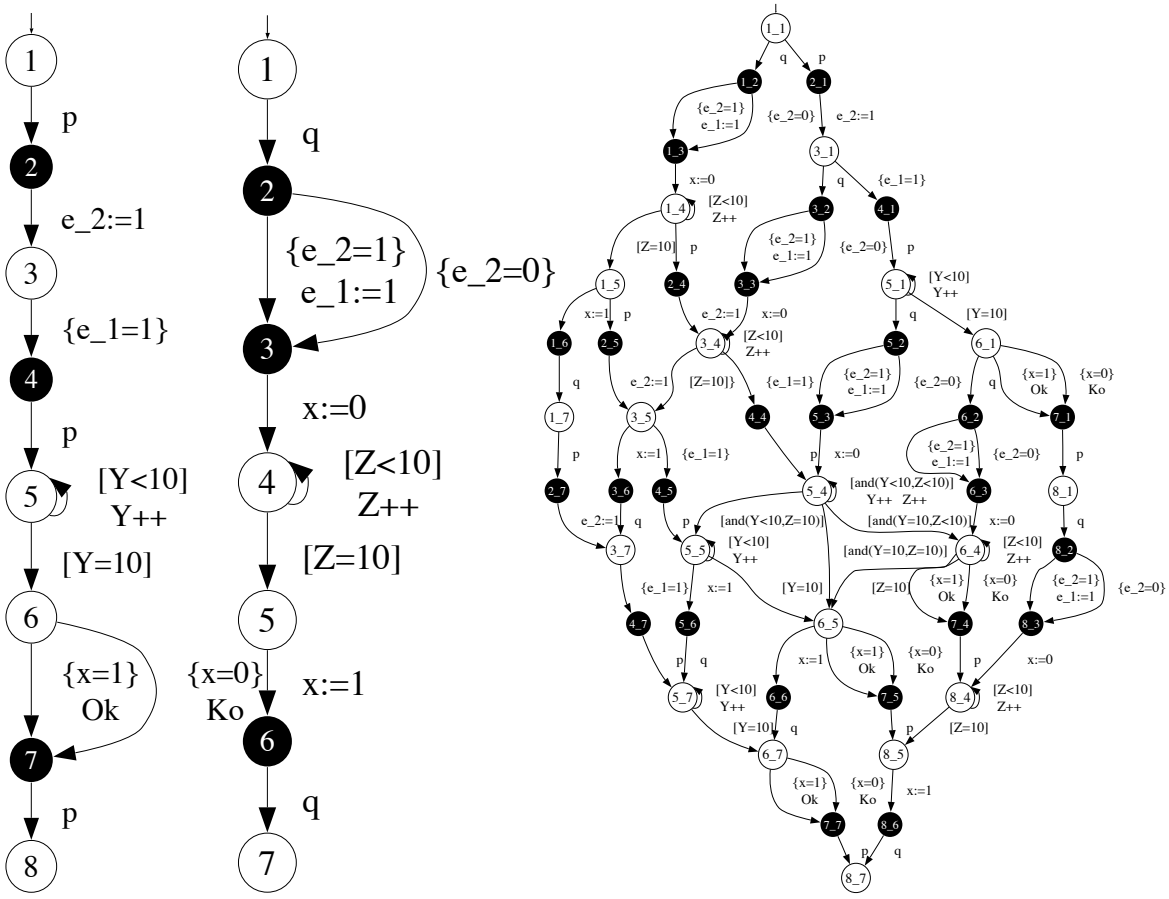


FIG. 9.5 – Deux automates **MicMac** et leur produit

montre le tableau de la figure 9.3.1.

Exemple (nb vars)	Produit (#loc,#trans)	ASPIC #boucles acc	ASPIC (#loc/#trans)	ASPIC temps	If Tool temps
Ex1 (3)	(47,93)	5	(30,36)	0.036s	qq ms
Ex2 (3)	(42,104)	13	(22,35)	0.040s	qq ms
Ex3 (4)	(144,653)	1	(144,350)	0.060s	>12min
Ex4 (4)	(180,508)	13	(50,87)	0.044s	>12min

FIG. 9.6 – Utilisation de ASPIC pour l'analyse d'accessibilité dans **MicMac**

Pour chacun des exemples, on donne le nombre de points de contrôle et de transitions de l'automate FAST généré, puis le nombre de boucles accélérées, le nombre de points de contrôle/transitions *potentiellement* atteignables donné par ASPIC, ainsi que le temps d'analyse. Les résultats donnés par **IF** sont quant à eux exacts (analyse d'automates temporisés), mais le temps d'analyse peut être très long (cf. les deux derniers exemples), et l'analyse éclate beaucoup le contrôle, rendant le retour d'information vers l'automate initial difficile.

On peut voir sur ces résultats que le nombre de transitions accélérables est assez grand (sauf pour l'exemple 3, voir la remarque plus bas), cela provient du fait que les automates font souvent intervenir le « passage du temps » afin de pouvoir traiter les entrelacements entre processus. Le fait que l'on puisse traiter des exemples de taille (relativement élevée s'explique par le nombre peu élevé de variables et de relations entre celles-ci, et donc les polyèdres manipulés sont « petits ».

Remarque 32 L'exemple 3 est un cas où l'accélération n'est effectuée que sur un point de contrôle. En fait, dans cet exemple la majeure partie des transitions non atteignables le sont en dehors des considérations numériques (gardes `false`).

9.3.2 Automates modélisant des consommations d'énergie

Dans [SMMM06], les auteurs proposent une modélisation des réseaux de capteurs *ad-hoc* et de leur environnement par du code Lustre. Afin de vérifier des propriétés quantitatives sur la consommation de tels réseaux, ils proposent une modélisation de la consommation d'énergie sous forme d'automates interprétés avec les caractéristiques suivantes :

- À chaque point de contrôle est associée une puissance p. Si l'on passe t secondes dans ce point de contrôle, alors l'énergie consommée est tp.
- À chaque transition est associé un temps t et une puissance p. La consommation associée à la transition est tp.

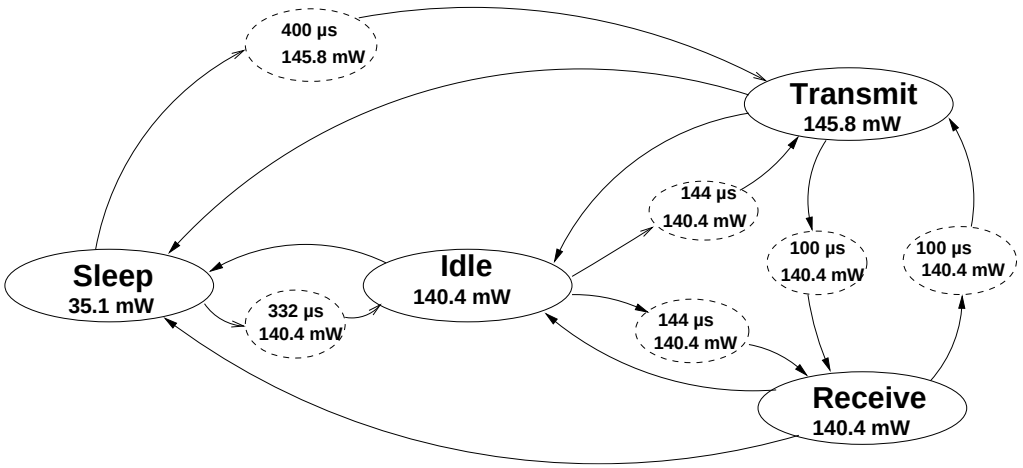


Fig. 9.7 – Automate modélisant la consommation d'une radio ([SMMM06])

Le but d'une telle modélisation est à la fois la simulation et la vérification de propriétés de comparaison du type « l'automate $\mathcal{A}_1$ consomme toujours moins que l'automate $\mathcal{A}_2$ » (dans un certain contexte, pour un environnement donné). Les automates n'ayant pas les mêmes états, l'analyse est difficile et une notion de simulation doit être définie.

Ce type d'automates peut aisément se traduire par des systèmes à compteurs faisant intervenir une variable de temps et un ou plusieurs compteurs d'énergie. Le décompte de l'énergie sur un état se fait par l'incrémentation de la variable correspondante : `e:=e+p`. Nous pouvons donc ensuite utiliser l'outil Aspic afin de réaliser deux types d'analyses :

- Un automate FAST seul peut être analysé par ASPIC et ensuite on peut comparer « à la main » les invariants obtenus pour chaque point de contrôle.
- On peut coder (cf Annexe C) les deux automates à comparer en LUSTRE en ajoutant un observateur synchrone qui compare les deux compteurs d'énergies (un par automate). Ensuite, l'outil NBAC a été utilisé pour retrouver la structure de contrôle du produit, et générer un état « mauvais » : une unique analyse en avant « guidée » par des instructions pour détecter les points de contrôle est réalisée, et ensuite une impression de ces résultats sous la forme d'un automate à compteur (en faisant abstraction des variables et signaux booléens) est fournie par l'outil NBAC. Notre outil ASPIC est ensuite en mesure de réaliser l'analyse de cet automate à compteur.

La première approche effectuée sur des petits exemples montre qu'effectivement des invariants de type « vitesse » peuvent être découverts (ils sont assez similaires à ceux du gaz burner). La deuxième approche donne des résultats assez probants :

- Si l'automate codé est déterministe, NBAC est en général suffisant pour prouver la propriété.
- Si le passage d'un état à un un autre est non déterministe (codé par des signaux booléens), la propriété « la consommation de l'automate A est toujours inférieure à celle de l'automate B » est en général fausse et celle que l'on cherche à prouver est de la forme « si l'on n'a pas changé d'état depuis 20 tics d'horloge, alors l'automate A a moins consommé que l'automate B ».
  Pour ce genre de propriétés, NBAC n'arrive en général pas à prouver ces propriétés, par contre l'analyse avec ASPIC fournit des invariants plus précis. Cependant, des analyses ont été réalisées avec des automates générés assez gros, et la précision n'est pas suffisante pour prouver les propriétés. De plus, cette analyse a été réalisée alors que certains algorithmes d'accélération n'avaient pas été encore implémentés.

Ainsi, les premières expérimentations montrent qu'ASPIC semble être un outil adapté à ce genre d'analyses puisque les algorithmes d'accélération sont capables de découvrir certaines relations de « vitesse » (cf. l'exemple de la voiture). Nous pensons que le passage par le OC (et une optimisation de la traduction de OC vers FAST), ainsi qu'un meilleur traitement des boucles complexes pourrait permettre à la fois de gagner du temps dans l'analyse et de limiter le nombre de transitions.

9.3.3 Analyse de programmes de listes

Dans [BBH+06], les auteurs proposent une traduction de certains programmes manipulant des listes vers des automates à compteur. Cette traduction fait intervenir une abstraction des listes sous forme de « segments de listes », toutes les cases de ce segment de listes ayant le même comportement. Cette modélisation étant trop grossière, des compteurs sont générés afin de pouvoir prendre en compte la taille des sous-listes. L'outil [L2C] implémente cette traduction de programmes à listes vers un automate à compteur décrit dans le langage FAST. Notre outil permet ainsi de vérifier certaines propriétés de sûreté sur les programmes manipulant des listes (par exemple, la validité des accès aux pointeurs, sur des programmes de renversement de liste, de parcours de listes, ...). Ces exemples peuvent être trouvés sur la page web de l'outil [L2C]. Pour l'instant, la traduction fait apparaître beaucoup de points de contrôles intermédiaires, et l'outil FAST ne termine pas en l'absence de stratégie. Notre outil

permet de vérifier la non atteignabilité des états mauvais en un temps raisonnable pour les exemples fournis. Toutefois, les automates générés par l'algorithme de traduction ne font pas pour l'instant intervenir des propriétés numériques complexes.

Conclusion du chapitre

Dans ce chapitre, nous avons exposé une partie de nos expérimentations réalisés avec l'outil Aspic. Afin de pouvoir comparer aux méthodes précédentes, nous avons implémenté quelques-unes ces diverses méthodes dans l'outil.

Sur des exemples « jouets » et des exemples de taille moyenne, nous avons montré que les invariants que nous trouvons sont en général plus précis.

Nous avons aussi collaboré avec d'autres personnes au laboratoire Verimag afin d'utiliser Aspic à d'autres sujets d'étude.

Les résultats obtenus sont encourageants, ils montrent une réelle augmentation de la précision sans trop de surcoût dû aux calculs préliminaires.

Chapitre 10

Conclusion

10.1 Bilan

Dans cette thèse, nous avons étudié comment combiner les résultats récents d'accélération et l'analyse des relations linéaires. Les contributions de cette thèse sont les suivantes :

1. Nous avons étudié les techniques existantes d'amélioration de l'analyse des Relations Linéaires et montré que ces différentes techniques ne sont que partielles et ne résolvent pas complètement le problème de manque de précision.
2. Nous avons étudié les techniques existantes d'accélération afin d'identifier des classes de transitions intéressantes en terme d'amélioration de la précision.
3. Nous avons défini la notion d'*accélération abstraite* de transitions affines gardées et dans divers cas particuliers (translations, ...), nous fournissons des algorithmes de calcul de surapproximation de l'enveloppe convexe de l'image d'un polyèdre par une ou plusieurs boucles. Dans certains cas, ces algorithmes donnent des résultats exacts, dans d'autres les résultats sont approchés mais plus précis que les invariants produits par les techniques d'élargissement.
4. Ces techniques d'accélération approchées de boucles sont combinées à l'approche générale d'Analyse des Relations Linéaires, qui analyse des systèmes dont les fonctions de transfert ainsi que les graphes de contrôle sont quelconques. Nous introduisons des heuristiques portant sur le choix des boucles à accélérer, l'alternance des techniques d'accélération et d'élargissement ; dans certains cas nous proposons de modifier la struture du système afin de simplifier l'analyse.
5. L'intérêt de cette approche est que théoriquement le gain de précision se fait sans perte d'efficacité. Pour valider l'approche, nous avons en parallèle de nos travaux théoriques développé un outil de vérification, Aspic. Les premières expérimentations montrent une baisse du nombre d'itérations de l'analyse (donc une analyse plus efficace, même si certains point peuvent être améliorés) et les résultats des analyses sont plus précis.

10.2 Perspectives

10.2.1 Du point de vue théorique

Comportement contractant de certaines fonctions Certaines boucles simples ont un comportement « contractant » (voir l'exemple 10.1) et donc la trajectoire d'un point est inclus

dans une certaine région du plan. Il pourrait être intéressant d'étudier certaines de ces classes.

EXEMPLE 10.1 *Considérons la transformation* $\tau : true \rightarrow x := -0.8y, y := 0.8x$*, dont l'effet sur* $P_0 = \{y = 0; x = 1\}$ *est illustré dans la figure 10.1. L'enveloppe convexe des valuations accessibles est le polyèdre grisé.*

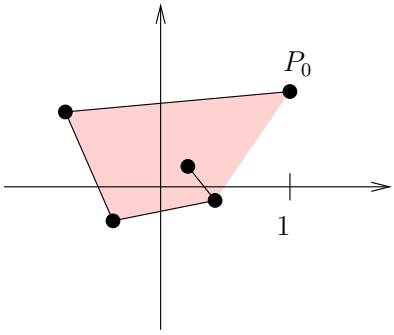


FIG. 10.1 – Trajectoire contractante d'un point et son enveloppe convexe

Analyse arrière Nous n'avons étudié que l'accélération « en avant », il serait intéressant de voir comment adapter les algorithmes à une analyse en arrière, notamment dans le cas où les fonctions de transition ne sont pas inversibles.

Si la matrice a de la transformation $\tau = (g, a)$ est inversible, nous pensons que le calcul d'une surapproximation de $pre^+_\tau(P)$ (on suppose que l'on a appliqué au moins une fois la transformation τ) peut être obtenu à partir du « renversement » des boucles. Concrètement, si la transformation est une translation, alors la translation inverse est obtenue en prenant $D' = -D$. Le calcul d'une approximation de $pre^*(P)$ se fait alors en ajoutant le rayon $-D$ au polyèdre $P \cap post(g, a)$, tant que $pre(g, a)$ est satisfaite. Sur la transformation $\tau : x \leqslant 4 \rightarrow x := x + 1$, $pre^{\oplus}_\tau(P)$ se calcule donc avec l'expression $\big((P \cap \{x \leqslant 5\}) \nearrow (-1)\big)$ (x n'est pas borné par le bas).

Le cas des transformations non inversibles est un peu plus complexe. Considérons par exemple la transformation $y \leqslant 7 \wedge x \leqslant 4 \rightarrow x := 0, y := y + 1$. Sur cet exemple, on peut calculer une surapproximation de $pre^+_\tau(P)$ de la façon suivante :

- On réalise l'intersection de P avec les contraintes $x = c$ pour toute variable x remise à constante dans la transformation τ , puis avec les postconditions des gardes concernant les variables non remises à constante. Sur l'exemple, cela donne $P \cap \{x = 0\} \cap \{y \leqslant 8\}$.
- La transformation inverse est obtenue en ajoutant le rayon obtenu en mettant 0 pour les variables remises à constante, et en inversant la composante des autres. Ici on ajoute le vecteur $\begin{pmatrix} -1 \\ 0 \end{pmatrix}$ au polyèdre précédent.
- On n'a pas besoin de réaliser une intersection à posteriori puisque si $y \leqslant 8$, alors $y' = y-1$ verifie $y' \leqslant 7$, et donc la garde de la transformation τ.

La combinaison des différentes boucles « en arrière » est à notre avis beaucoup plus complexe et mériterait d'être approfondie.

10.2.2 Améliorations de Aspic

Du point de vue de l'implémentation, notre analyseur pourrait être amélioré de la façon suivante :

- Le précalcul et le traitement des boucles complexes reste à tester et améliorer. Nous pensons que l'ajout de l'accélération de boucles complexes nous permettra de traiter des exemples plus gros.
- Enfin, le traducteur oc2fst génère trop de transitions. Nous pensons que ce traducteur peut être amélioré dans ce sens.
- De manière plus générale, nous pensons que la complexité de nos résultats expérimentaux provient du passage par le langage Lustre et les produits synchrones, alors que notre objectif est plutôt la découverte de propriétés numériques à l'intérieur de programmes séquentiels. Pour cela, il faudrait réaliser une compilation d'un langage séquentiel en un automate interprété avec variables numériques. Ces techniques sont classiques et peuvent être mises en place en peu de temps.

10.2.3 Vers une intégration dans l'outil Nbac

À plus long terme, nous pensons qu'il serait intéressant d'intégrer nos algorithmes d'accélération dans Nbac. En effet, nos expérimentations ont montré l'importance d'avoir une structure de contrôle adaptée à nos analyses, et le partitionnement dynamique de Nbac nous semble être la solution pour réaliser des analyses sur des exemples de taille plus grande. Cependant, pour réaliser cette implémentation, les adaptations suivantes sont nécessaires :

- De nouveaux algorithmes pour l'analyse en arrière devront être conçus.
- Nos algorithmes devront gérer la mixité des aspects numériques et booléens dans les fonctions de transitions (gardes et actions). En effet, les fonctions de transition de nos automates interprétés ne font intervenir que des aspects numériques, alors que l'outil Nbac gère des fonctions de transitions booléennes et numériques.
- Notre analyse fait intervenir de nombreux prétraitements. Pour des raisons d'efficacité, il est important que ces calculs ne soient pas refaits à chaque partitionnement dynamique. Il faudrait donc étudier en détail comment le partitionnement dynamique modifie les différentes structures internes d'Aspic.

Troisième partie

Annexes

Annexe A

Sur le monoïde engendré par une matrice

A.1 Preuve d'équivalence entre les différentes caractérisations

Dans cette section, nous prouvons l'équivalence entre les différentes caractérisations du problème du monoïde fini.

LEMME 5.3 *Soit $C \in \mathcal{M}_n(\mathbb{Q})$ une matrice carrée de taille n à coefficients dans $\mathbb{Q}$. Il y a équivalence entre les problèmes suivants :*

1. *Est ce qu'il existe une puissance de C qui est diagonalisable et de valeurs propres dans $\{0,1\}$?*
2. *Est-ce qu'il existe $p \in \mathbb{N}$, tel que $C^{2p} = C^p$?*
3. *Est que que le monoïde engendré par les puissances de C, c'est-à-dire $\{I, C, C^2, \ldots\}$ est fini ?*

PREUVE : La preuve est immédiate pour l'équivalence $1 \leftrightarrow 2$, puisqu'alors C^p est une projection. Le sens $2 \rightarrow 3$ est immédiat aussi.
Pour démontrer $3 \rightarrow 2$, on suppose donc l'existence de p_1, p_2 tels que $C^{p_1} = C^{p_2}$. La suite des C^k comporte donc une collision. Notons c le plus petit entier tel qu'il existe $d > 0$ tel que $C^c = C^{c+d}$, puis prenons d minimal :

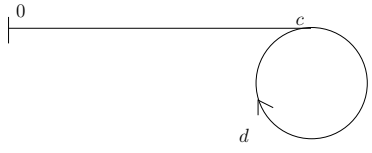


FIG. A.1 – $C^c = C^{c+d}$

On a alors $C^{c+k} = (C^{c+k})^2$ si et seulement si $C^{c+k} = C^{c+2k+c}$: il suffit pour cela d'avoir $k = 2k + c \mod d$, soit encore $k = -c \mod d$. En prenant $k > 0$ ainsi puis $p = c + k$, on a bien $C^{2p} = C^p$. ∎

Soit maintenant $C \in \mathcal{M}_n(\mathbb{Q})$. On cherche un p tel que $C^{2p} = C^p$. La condition $(C^p)^2 = C^p$ signifie que C^p est la matrice d'une projection : l'espace se décompose en somme directe de $\mathrm{Ker}(\mathrm{C^p})$ et $\mathrm{Ker}(\mathrm{C^p} - \mathrm{I_n})$. Il y a donc une partie de l'espace sur lequel C est nilpotente, et un supplémentaire sur lequel C est inversible d'ordre fini.

A.2 Un algorithme polynômial pour la question du monoïde fini

$C^p = I_n$ est équivalent au fait que $X^p - 1$ est un polynôme annulateur de C, c'est-à-dire que le polynôme minimal de C, disons μ_C, divise[1] $X^p - 1$. La factorisation en produit d'irréductibles de $X^p - 1$ dans $\mathbb{Q}[X]$ est connue : $X^p - 1 = \prod_{d|p} \Phi_d$, avec Φ_d le d-ème polynôme cyclotomique, défini par $\Phi_d = \prod_{z \in \mathbb{U}_n^*} (X - z)$ avec $\mathbb{U}_n^*$ l'ensemble des racines primitives n-nièmes de l'unité.

On montre de façon classique que Φ_d est à coefficients entiers, irréductible sur $\mathbb{Q}$, et de degré $\phi(d)$, le nombre d'entiers de $[\![1, d-1]\!]$ premiers avec d.

Ainsi, C est d'ordre fini dans $GL_n(\mathbb{Q})$ si et seulement si son polynôme minimal est produit de polynômes cyclotomiques. En particulier :

C engendre un monoïde fini si et seulement si μ_C est de la forme $X^c \prod_{i=1}^{r} \Phi_{k_i}$, ce qu'on peut décider facilement en calculant μ_C, et en regardant s'il est de la forme précédente.

Algorithme De cette caractérisation on peut déduire un algorithme simple pour savoir si la matrice C engendre un monoïde fini :

1. On calcule le polynôme minimal μ_C de C (par exemple naïvement, en pivotant sur n vecteurs de longueur n^2). Cela se fait en $O(n^4)$.
2. On initialise P à μ_C ; on calcule les Φ_k, tant que $\varphi(k) \leqslant n$ ($k \leqslant 2n^2$ naïvement, mais en fait $k \leqslant O(n)$). Pour chacun des Φ_k, s'il divise P, alors on remplace P par $\dfrac{P}{\Phi_k}$.
3. Le monoïde engendré par C est fini si et seulement si $P = X^\alpha$ à la fin de la boucle précédente.

On aura noté qu'il n'est pas utile de factoriser μ_C sur $\mathbb{Q}$. Par ailleurs, le coût de la boucle est en $O(n^4)$: $O(n)$ calculs de Φ_k, chacun étant obtenu par $O(n)$ divisions euclidiennes de polynômes de degrés $O(n)$.

[1] Dans $\mathbb{C}[X]$, $\mathbb{R}[X]$, $\mathbb{Q}[X]$ ou $\mathbb{Z}[X]$: on prouve classiquement que c'est équivalent.

Annexe B

Exemple d'analyse de fichier Fast

B.1 Le fichier d'entrée

```
model gb {

var l, t, x;

states ell,hen;

transition t1 := {
 from   := ell;
 to     := ell;
 guard  := (x<=9) ;
 action := l'=l+1,x'=x+1,t'=t+1;
};

transition t2 := {
 from   := ell;
 to     := hen;
 guard  := true;
 action := x'=0;
};

transition t3 := {
 from   := hen;
 to     := hen;
 guard  := true;
 action := t'=t+1, x'=x+1;
};

transition t4 := {
 from   := hen;
 to     := ell;
 guard  := x>=50;
 action := x'=0;
};

}
```

```
strategy s1 {

Region init := {state=ell && t=0 && l=0 && x=0};

Region bad := {6l>t+50};

}
```

B.2 Impressions de l'outil Aspic

Sur le fichier précédent, l'outil invoqué avec la ligne de commande `./aspic gb.fst` donne le résultat suivant (parallèlement, un fichier gb.log est créé, avec le détail des calculs) :

```
Initialising Polka
 ---- Configuration of Aspic
   * name of file : gb.fst
   * inputfile of type (real) Fast format
   * ANALYSIS
   * with acceleration
   * delay of widening = 1
The Fast file has been sucessfully parsed

 ---- Beginning of translation to the internal language
   * Name of model = gb, Name of file = gb.fst
   * 3 variable(s)
   * Searching objective in fast strategy
       -> we suppose that we want to compute post*(init) with init a convex set
       or post*(init) and bad if the state bad exists and bad is convex
       -> Dealing with the bad state(s).  done.
       -> Finding initial polyhedron. : initial state = ell done.
   * 2 location(s) and 4 (possibly non convex) transitions
       -> vertices/transitions for the automaton itself, done
       -> now dealing with the bad locations, we have 1 bad region
          ## we have no bad node, just a formula
       -> vertices/transitions for the safety property (bad locations), done
   * The initial state has label ell and the associated polyhedron is {
x=0,t=0,l=0,1>0,$>=0}
   * There is/are 1 bad node(s)
   * The module is done
 ---- End of translation to the internal language

 ---- Now analysing with aspic engine
   * Initialisation of the graph structure  : The encoded graph has 3 vertex and
 6 edges
   * Now splitting or calculating uptos : After splitting simpleloops, the graph
 has 5 vertex and 6 edges : 2 split nodes
   * Determinating strategy with Tarjan's modified algorithm
```

```
   * Creation of the FP module, linking with the polyhedral lattice
   * FP iteration ..... done
 ---- Finished

 ---- Now printing the results (also in the .log file)
output=[
        { v = ell, attrvertex = {x=0,6l<=t,$>=0,1>0,l>=0}  }
        { v = hen, attrvertex = {x=0,6l<=t+50,l<=t,$>=0,1>0,l>=0}  }
        { v = __bad_0, attrvertex = empty(3)  }
        { v = ell__split, attrvertex = {6l<=t+5x,x<=10,$>=0,l>=x,1>0,x>=0}  }
        { v = hen__split, attrvertex = {6l+x<=t+50,l+x<=t,$>=0,l>=0,1>0,x>=0}  }
        info = { time=0.010000; iterations=5; descendings=0; }
        ]
  * Result : All the bad locations are unreachable
```

Accelerated Symbolic Polyhedral Invariant Computation

Annexe C

Codes LUSTRE des exemples

C.1 Contrôleur de métro

```
const nbtrains = 1;

node train (s,b: bool)
returns (ontime, onbrake, late,  stopped: bool;  nbrake,
nbeacons: int);
-- un train recoit les signaux "seconde" et "balise" (supposes exclusifs)
-- retourne son etat.
var delta: int;

let
        assert #(s,b);
        assert (true -> (pre(stopped) => not b));

        -- delta = nb secondes - nb balises
        nbeacons = 0 -> if b then pre(nbeacons)+1 else pre(nbeacons);
        delta = 0 -> if s then pre(delta)+1
                         else if b then pre(delta)-1
                                   else pre(delta);

        ontime = true -> not (late or onbrake or stopped);

        late = false -> if delta>=10 then true
                                     else
                                        if delta<=0 then false
                                                    else pre(late);

        -- comptage du nombre de balises depuis freinage
        nbrake = 0 -> if onbrake or stopped then
                                            if b then pre(nbrake) +1
                                                 else pre(nbrake)
                                            else 0;

        onbrake = false -> if delta<=-10 and pre(nbrake)<10
                                 then true
```

```
                                    else
                                      if delta>=0 or pre(nbrake)>= 10
                                      then false
                                      else pre onbrake;

        stopped = false -> if pre(onbrake) and pre(nbrake)= 10
                                    then true
                                    else
                                       if delta>=0 then false
                                                   else pre(stopped);
tel;

-- l'horloge globale
node clock (late: bool^nbtrains; ss: bool) returns (s: bool);
var onelatetrain: bool; acc: bool^(nbtrains+1);
let
        acc = false^1 | (acc[0..nbtrains-1] or pre(late));
        onelatetrain = acc[nbtrains];
        s = if onelatetrain then false else ss;
tel

node metro1 (sec: bool; beacons: bool^nbtrains)
returns (ontime, onbrake, late,  stopped: bool^nbtrains;  nbrake,
nbeacons: int^nbtrains;  second: bool);

let
        second = clock(late, sec);
        (ontime, onbrake, late,  stopped, nbrake,  nbeacons) =
                        train(second^nbtrains, beacons);
tel
```

C.2 Automates consommateurs d'énergie

```
node A (a, b, tm : bool) returns (e : int) ;
var
        X, Y : bool ;
        ep, ec : int ;
let
        -- Encodage de l'automate
        assert #(X, Y);
        X = true ->  pre (X and not a or Y and b) ;
        Y = false -> pre (Y and not b or X and a) ;

        -- puissance associée aux états
        ec = (if X then 3 else 5) ;

        ep = ec -> if tm then ec
                         else max (pre ep, ec) ;

        -- énergie consommée
        e = 0 -> if tm then pre (e) + pre (ep)
             else pre (e) ;
tel.
```

```
node redge (a : bool) returns (r : bool) ;
let
        r = false -> a and not pre(a) ;
tel.

node B (a, b, tm : bool) returns (e : int) ;
var
        X1, Y1, Y2 : bool ;
        dix : int ;
        ec, ep : int ;
let
        assert #(X1, Y1, Y2) ;
        X1 = true -> pre (X1 and not a or
                          Y1 and b or
                          Y2 and b) ;
        Y1 = false -> pre (Y1 and not (dix=10) and not b or
                           X1 and a or
                           Y2 and not b) ;
        Y2 = false -> pre (Y2 and not b and not true or
                           Y1 and (dix=10) and not b) ;
        dix = 0 -> if redge(Y1) then 0 else pre(dix) + 1 ;

        ec =  (if X1 then 2 else if Y1 then 7 else 7) ;

        ep = ec -> if tm then ec
                             else max (pre ep, ec) ;

        e = 0 -> if tm then pre (e) + pre (ep)
             else pre (e) ;
tel.

-- observateur
node Compar (a, b : bool) returns (ok : bool ) ;
var
        e, ee : int ;
        tm : bool ;
        cpt : int ;
let
        assert #(a, b) ;
        -- tm = échantillonnage du temps.
        cpt = 0 -> if pre(cpt)+1> 20 then 0 else pre(cpt)+1;
        tm = cpt=20;

        e  = A (a, b, tm) ;
        ee = B (a, b, tm) ;

        ok = true -> pre(ee <= e)  ;
tel.
```

Bibliographie

[AAB00] A. Annichini, E. Asarin, and A. Bouajjani. Symbolic techniques for parametric reasoning about counter and clock systems. In *Computer Aided Verification*, pages 419–434, 2000. 45

[ACH+95] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science B*, 138 :3–34, January 1995. 32

[AD90] R. Alur and D. Dill. Automata for modeling real-time systems. In *ICALP'90*, 1990. 25

[AMP95] E. Asarin, O. Maler, and A. Pnueli. Reachability analysis of dynamical systems having piecewise-constant derivatives. *Theoretical Computer Science*, 138(1) :35–65, 1995. 80

[AMS06] K. Altisen, F. Maraninchi, and D. Stauch. Aspect-oriented programming for reactive systems : Larissa, a proposal in the synchronous framework. *Science of Computer Programming, Special Issue on Foundations of Aspect-Oriented Programming*, 63(3) :297–320, 2006. 110

[ANA] ANALYSER. http://pop-art.inrialpes.fr/people/bjeannet/analyzer/index.html. 102

[ARG] ARGOS. http://www-verimag.imag.fr/~stauch/ArgosCompiler/introduction.html. 110

[ASP] ASPIC. http://www-verimag.imag.fr/~gonnord/aspic/aspic.html. 116

[BALS05] S. Bardin, A.Finkel, J. Leroux, and P. Schnoebelen. Flat acceleration in symbolic model checking. In Doron A. Peled and Yih-Kuen Tsay, editors, *Proceedings of the 3rd International Symposium on Automated Technology for Verification and Analysis (ATVA'05)*, volume 3707 of *Lecture Notes in Computer Science*, pages 474–488, Taipei, Taiwan, ROC, October 2005. Springer. 41, 43

[Bar05] S. Bardin. *Vers un model checking avec accélération plate de systèmes hétérogènes*. Thèse, Laboratoire Spécification et Vérification, ENS Cachan, France, October 2005. 9, 41, 42, 43, 44, 113, 114

[BB03] C. Bartzis and T. Bultan. Efficient symbolic representations for arithmetic constraints in verification. In *International Journal of Foundations of Computer Science (IJFCS), special issue on Verification and Analysis of Infinite State Systems*, volume 14.4, pages 605–624, August 2003. 44

[BBH+06] A. Bouajjani, M. Bozga, P. Habermehl, R. Iosif, P. Moro, and T.Vojnar. Programs with lists are counter automata. In Thomas Ball and Robert B. Jones, editors,

Proceedings of the 18th International Conference on Computer Aided Verification (CAV'06), volume 4144 of *Lecture Notes in Computer Science*, pages 517–531, Seattle, Washington, USA, August 2006. Springer. 121

[BCALS06] A. Bouajjani, A. Collomb-Annichini, Y. Lakhnech, and M. Sighireanu. Analysing fair parametric extended automata. In *ICALP (2)*, pages 577–588, 2006. 44

[BCC+03] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. In *PLDI 2003, ACM SIGPLAN SIGSOFT Conference on Programming Language Design and Implementation*, pages 196–207, San Diego (Ca.), June 2003. 49, 55

[BFL04] S. Bardin, A. Finkel, and J. Leroux. FASTer acceleration of counter automata in practice. In Kurt Jensen and Andreas Podelski, editors, *Proceedings of the 10th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'04)*, volume 2988 of *Lecture Notes in Computer Science*, pages 576–590, Barcelona, Spain, March 2004. Springer. 40, 44, 75

[BFLP03] S. Bardin, A. Finkel, J. Leroux, and L. Petrucci. Fast : Fast acceleration of symbolic transition systems. In *CAV'03*, pages 118–121, Boulder (Colorado), July 2003. LNCS 2725, Springer-Verlag. 12, 85

[BHRZ03] R. Bagnara, P. M. Hill, E. Ricci, and E. Zaffanella. Precise widening operators for convex polyhedra. In R. Cousot, editor, *Static Analysis : Proceedings of the 10th International Symposium*, volume 2694 of *Lecture Notes in Computer Science*, pages 337–354, San Diego, California, USA, 2003. Springer-Verlag, Berlin. 50, 51, 52, 58, 113

[BIL06] M. Bozga, R. Iosif, and Y. Lakhnech. Flat parametric counter automata. In *ICALP (2)*, pages 577–588, 2006. 40

[BJH03] B. Boigelot, S. Jodogne, and F. Herbreteau. Hybrid acceleration using real vector automata. In *Proc. of the 15th International Conference on Computer Aided Verification*, volume 2725 of *Lecture Notes in Computer Science*, pages 193–205, Boulder (CO, USA), July 2003. Springer-Verlag. 40

[BLW03] B. Boigelot, A. Legay, and P. Wolper. Iterating transducers in the large. In *Proc. 15th Int. Conf. on Computer Aided Verification*, volume 2725 of *Lecture Notes in Computer Science*, pages 223–235, Boulder, July 2003. Springer-Verlag. 45

[Boi99] B. Boigelot. Symbolic methods for exploring infinite state spaces. Phd thesis, Université de Liège, 1999. 39, 41, 73, 76

[Bou92] F. Bourdoncle. Sémantique des langages impératifs d'ordre supérieur et interprétation abstraite. Thèse Ecole Polytechnique, Paris, 1992. 23

[Bou93] F. Bourdoncle. Efficient chaotic iteration strategies with widenings. *Lecture Notes in Computer Science*, 735 :128–141, 1993. 57

[BW94] B. Boigelot and P. Wolper. Symbolic verification with periodic sets. In *CAV'94*, Stanford (Ca.), 1994. LNCS 818, Springer Verlag. 12, 37, 38, 45

[BW02] B. Boigelot and P. Wolper. Representing arithmetic constraints with finite automata : An overview. In *Proc. International Conference on Logic Programming (ICLP)*, volume 2401 of *Lecture Notes in Computer Science*, pages 1–19, Copenhagen, July 2002. Springer-Verlag. 39

[CC76] P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *2nd Int. Symp. on Programming*. Dunod, Paris, 1976. 23, 25, 47, 48

[CC77] P. Cousot and R. Cousot. Abstract interpretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th ACM Symposium on Principles of Programming Languages, POPL'77*, Los Angeles, January 1977. 11, 21

[CC92a] P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *Journal of Logic Programming*, 13(1–4) :103–179, 1992. (Also, Rapport de Recherche LIX/RR/92/08, Ecole Polytechnique). 23

[CC92b] P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation, invited paper. In M. Bruynooghe and M. Wirsing, editors, *Proceedings of the International Workshop Programming Language Implementation and Logic Programming, PLILP'92,*, Leuven, Belgium, 13–17 August 1992, Lecture Notes in Computer Science 631, pages 269–295. Springer-Verlag, Berlin, Germany, 1992. 51

[CC04] R. C. Clarisó and J. Cortadella. Verification of parametric timed circuits using octahedra. In *Designing correct circuits, DCC'04*, Barcelona, March 2004. 25

[CF99] A. Cortesi and G. Filé, editors. *Static Analysis, 6th International Symposium, SAS '99, Venice, Italy, September 22-24, 1999, Proceedings*, volume 1694 of *Lecture Notes in Computer Science*. Springer, 1999. 51

[CGG+05] A. Costan, S. Gaubert, E. Goubault, M. Martel, and S. Putot. A policy iteration algorithm for computing fixed points in static analysis of programs. In *Proc. of the 17th International Conference on Computer Aided Verification*. Springer-Verlag, 2005. 49, 59, 60

[CGL94] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. *ACM TOPLAS*, 16(5) :1512–1542, 1994. 19

[CH78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *5th ACM Symposium on Principles of Programming Languages, POPL'78*, Tucson (Arizona), January 1978. 27, 29, 105

[CHR91a] Z. Chaochen, C.A.R. Hoare, and A.P. Ravn. A calculus of durations. *Information Processing Letters*, 40(5), 1991. 32

[CHR91b] Z. Chaochen, C.A.R. Hoare, and A.P. Ravn. A calculus of durations. *Information Processing Letters*, 40 :269–276, 1991. 110

[CJ98] H. Comon and Y. Jurski. Multiple counters automata, safety analysis and Presburger arithmetic. In *CAV'98*, Vancouver (B.C.), 1998. LNCS 1427, Springer Verlag. 12, 40, 42

[CJ99] H. Comon and Y. Jurski. Timed automata and the theory of real numbers. In *Proc. Conf. on Concurrency theory*, volume 1664 of *lncs*, pages 242–257, Eindhoven, 1999. Springer Verlag. 43

[CMMC07] J. Cornet, F. Maraninchi, and L. Maillet-Contoz. Formalizing systemc transaction-level models of systems-on-chip : a component-based approach. Unpublished, 2007. 10, 117, 118

[DFV05] C. Darlot, A. Finkel, and L. VanBegin. About Fast and TReX accelerations. In Michael R. A. Huth, editor, *Proceedings of the 4th International Workshop on Automated Verification of Critical Systems (AVoCS'04)*, volume 128 of *Electronic Notes in Theoretical Computer Science*, pages 87–103, London, UK, May 2005. Elsevier Science Publishers. 45

[DG03] L. Danthony-Gonnord. Du calcul des durées aux automates symboliques. Master thesis, Universités Paris VI/VII, June 2003. www-verimag.imag.fr/~gonnord/papers/rapport_dea.ps. 110

[FAS] FAST. http://www.lsv.ens-cachan.fr/fast/. 6, 41, 43, 101

[FL02] A. Finkel and J. Leroux. How to compose Presburger-accelerations : Applications to broadcast protocols. In *Proceedings of the 22nd Conf. Found. of Software Technology and Theor. Comp. Sci. (FSTTCS'2002)*, volume 2556 of *Lecture Notes in Computer Science*, pages 145–156, Kanpur, India, December 2002. Springer. 40, 41, 43, 73, 76

[FO97] L. Fribourg and H. Olsén. Proving safety properties of infinite state systems by compilation into Presburger arithmetic. In Antoni W. Mazurkiewicz and Józef Winkowski, editors, *Proceedings of the 8th International Conference on Concurrency Theory (CONCUR'97)*, volume 1243 of *Lecture Notes in Computer Science*, pages 213–227, Warsaw, Poland, July 1997. Springer. 114

[FS00a] A. Finkel and G. Sutre. An algorithm constructing the semilinear Post* for 2-dim Reset/Transfer VASS. In Mogens Nielsen and Branislav Rovan, editors, *Proceedings of the 25th International Symposium on Mathematical Fundations of Computer Science (MFCS 2000)*, volume 1893 of *Lecture Notes in Computer Science*, pages 353–362, Bratislava, Slovakia, August 2000. Springer. 43

[FS00b] A. Finkel and G. Sutre. An algorithm constructing the semilinear post* for 2-dim reset/transfer vass. In *25th Int. Symp. Math. Found. Comp. Sci. (MFCS'2000)*, Bratislava, Slovakia, August 2000. LNCS 1893, Springer Verlag. 12

[GGTZ07] S. Gaubert, E. Goubault, A. Taly, and S. Zennou. Static analysis by policy iteration on relational domains. In *European Symposium On Programming*, Braga (Portugal), April 2007. Springer Verlag. 59, 60

[GH06] L. Gonnord and N. Halbwachs. Combining widening and acceleration in linear relation analysis. In *13th International Static Analysis Symposium, SAS'06*, Seoul, Korea, August 2006. 70, 82, 97

[GHR04] L. Gonnord, N. Halbwachs, and P. Raymond. From discrete duration calculus to symbolic automata. In *3rd International Workshop on Synchronous Languages, Applications, and Programs, SLAP'04*, Barcelona, Spain, March 2004. 110

[GL93] S. Graf and C. Loiseaux. A tool for symbolic program verification and abstraction. In *Fifth Conference on Computer-Aided Verification, CAV'93*, Elounda (Greece), July 1993. LNCS 697, Springer Verlag. 19

[Gou01] E. Goubault. Static analyses of the precision of floating-point operations. In *SAS*, pages 234–259, 2001. 49

[GR06] D. Gopan and T. Reps. Lookahead widening. In *CAV'06*, Seattle, 2006. 56, 105

[Gra91] Ph. Granger. Analyses sémantiques de congruence. Phd thesis, Ecole Polytechnique, July 1991. 25

[Hal79a] N. Halbwachs. Détermination automatique de relations linéaires vérifiées par les variables d'un programme. Thèse de troisième cycle, Université de Grenoble, March 1979. 12, 83, 113

[Hal79b] N. Halbwachs. Détermination automatique de relations linéaires vérifiées par les variables d'un programme. Thèse de 3e cycle, Université de Grenoble, March 1979. 30, 49

[Hal93] N. Halbwachs. Delay analysis in synchronous programs. In *Fifth Conference on Computer-Aided Verification, CAV'93*, Elounda (Greece), July 1993. LNCS 697, Springer Verlag. 49, 52, 55

[HCRP91] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous dataflow programming language LUSTRE. *Proceedings of the IEEE*, 79(9) :1305–1320, September 1991. 105

[HHWT97] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. Hytech : A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1 :110–122, 1997. 32

[HLR92] N. Halbwachs, F. Lagnier, and C. Ratel. Programming and verifying real-time systems by means of the synchronous data-flow programming language LUSTRE. *IEEE Transactions on Software Engineering, Special Issue on the Specification and Analysis of Real-Time Systems*, pages 785–793, September 1992. 106

[HLR93] N. Halbwachs, F. Lagnier, and P. Raymond. Synchronous observers and the verification of reactive systems. In M. Nivat, C. Rattray, T. Rus, and G. Scollo, editors, *Third Int. Conf. on Algebraic Methodology and Software Technology, AMAST'93*, Twente, June 1993. Workshops in Computing, Springer Verlag. 106

[HNSY92] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model-checking for real-time systems. In *LICS'92*, June 1992. 25, 59

[HPR97] N. Halbwachs, Y.E. Proy, and P. Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2) :157–185, August 1997. 30, 33, 52, 95, 116

[HRR91] N. Halbwachs, P. Raymond, and C. Ratel. Generating efficient code from data-flow programs. In *Third International Symposium on Programming Language Implementation and Logic Programming*, Passau (Germany), August 1991. LNCS 528, Springer Verlag. 107

[IJT91] F. Irigoin, P. Jouvelot, and R. Triolet. Semantical interprocedural parallelization : An overview of the PIPS project. In *ACM Int. Conf. on Supercomputing, ICS'91, Köln*, 1991. 60

[Iri05] F. Irigoin. Detecting affine loop invariants using modular static analysis. Technical Report A/367/CRI, Centre de Recherche en Informatique, Ecole des Mines de Paris, July 2005. 60, 61, 65

[Jea00] B. Jeannet. Partitionnement dynamique dans l'analyse de relations linéaires et application à la vérification de programmes synchrones. Thèse, Institut National Polytechnique, Grenoble, September 2000. 106

[Jur99] Y. Jurski. *Expression de la relation binaire d'accessibilité pour les automates à compteurs plats et les automates temporisés.* Thèse de doctorat, Laboratoire Spécification et Vérification, ENS Cachan, France, October 1999. 40

[Kar76] M. Karr. Affine relationships among variables of a program. *Acta Informatica*, 6 :133–151, 1976. 25

[L2C] L2CA. http://www-verimag.imag.fr/~async/L2CA/l2ca.html. 121

[Lam77] L. Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, SE-3(2) :125–143, 1977. 19

[LAS] LASH. http://www.montefiore.ulg.ac.be/~boigelot/research/lash/. 40

[Ler03] J. Leroux. Algorithmique de la vérification des systèmes à compteurs – approximation et accélération – implémentation dans l'outil Fast. Phd thesis, Ecole Normale Supérieure de Cachan, December 2003. 40, 113

[Mer05] D. Merchat. Réduction du nombre de variables en analyse de relations linéaires. These, Université Joseph Fourier, May 2005. 23, 53, 110

[Mil87] W. Miller. The maximum order of an element of a finite symmetric group. *Am. Math. Monthly*, 94(6) :497–506, 1987. 76

[Min01] A. Miné. The octagon abstract domain. In *AST 2001 in WCRE 2001*, IEEE, pages 310–319. IEEE CS Press, October 2001. 25

[MT53] Thompson Motzkin, Raiffa and Thrall. The double description method. *Theodore S. Motzkin : Selected Papers*, 1953. 27

[New] NewPolka. http://pop-art.inrialpes.fr/people/bjeannet/newpolka/index.html. 102

[Ope05] Open SystemC Initiative. *IEEE 1666 : SystemC Language Reference Manual*, 2005. www.systemc.org. 117

[Pan01] P.K. Pandya. Specifying and deciding quantified discrete-time duration calculus formulae using dcvalid. In *Real-Time Tools, RTTOOLS'2001*, Aalborg, August 2001. 110

[PS87] J. A. Plaice and J-B. Saint. The LUSTRE-ESTEREL portable format. Rapport non publié, INRIA, Sophia Antipolis, 1987. 106

[Pug91] William Pugh. The omega test : a fast and practical integer programming algorithm for dependence analysis. In *Supercomputing '91 : Proceedings of the 1991 ACM/IEEE conference on Supercomputing*, pages 4–13, New York, NY, USA, 1991. ACM Press. 72

[SISG06] S. Sankaranarayanan, F. Ivancic, I. Shlyakhter, and A. Gupta. Constraint-based linear relations analysis. In *International Symposium on Static Analysis, SAS'2006*. LNCS 4134, Springer Verlag, 2006. 60

[SK06] A. Simon and A. King. Widening Polyhedra with Landmarks. In *Fourth Asian Symposium on Programming Languages and Systems*, volume 4279 of *Lecture Notes in Computer Science*, pages 166–182. Springer Verlag, November 2006. 54

[SMMM06] L. Samper, F. Maraninchi, L. Mounier, and L. Mandel. GLONEMO : Global and accurate formal models for the analysis of ad-hoc sensor networks. In *Proceedings of the First International Conference on Integrated Internet Ad hoc and Sensor Networks (InterSense'06)*, Nice, France, May 2006. 10, 120

[SSM04] S. Sankaranarayanan, H. Sipma, and Z. Manna. Constraint-based linear relations analysis. In *International Symposium on Static Analysis, SAS'2004*, pages 53–68. LNCS 3148, Springer Verlag, 2004. 52

[StI] StInG. http://theory.stanford.edu/~srirams/Software/sting.html. 113

[SW04] Z. Su and D. Wagner. A class of polynomially solvable range constraints for interval analysis without widenings and narrowings. In *TACAS'04*, pages 280–295, Barcelona, 2004. 62, 65

[Tar72] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1 :146–160, 1972. 23

[TRE] TREX. http://www.liafa.jussieu.fr/~sighira/trex/. 45

[WB98] P. Wolper and B. Boigelot. Verifying systems with infinite but regular state spaces. In *CAV'98*, pages 88–97, Vancouver, June 1998. LNCS 1427, Springer-Verlag. 12

Résumé

Le travail décrit dans cette thèse s'inscrit dans le contexte de la validation de propriétés de sûreté de programmes, et plus particulièrement des propriétés numériques. L'utilisation de la technique d'Analyse des Relations Linéaires, une interprétation abstraite fondée sur une approximation des états numériques par des polyèdres convexes, a fait ses preuves dans le domaine. Il s'agit de générer des surapproximations polyédriques de l'ensemble des valuations associées à chaque point de contrôle, l'introduction d'un opérateur *d'élargissement* assurant la convergence des analyses. Cependant dans certains cas les invariants générés ne sont pas assez précis, et l'amélioration de la précision via le retardement du moment d'application de l'élargissement est trop coûteux. Nous nous sommes donc intéressés aux méthodes dites d'*accélération*, qui consistent à calculer exactement l'effet d'une ou plusieurs boucles (sous la forme de formules de Presburger), le principal inconvénient de ces méthodes étant qu'elles ne s'appliquent qu'à une classe restreinte de programmes.

Dans cette thèse nous proposons une approche combinant l'Analyse des Relations Linéaires classique (avec élargissement) et la notion d'*Accélération abstraite* utile pour calculer une surapproximation précise de l'application itérée de certains types de boucles, dans le but d'améliorer la précision des analyses tout en garantissant toujours la terminaison. Les premiers résultats expérimentaux obtenus grâce à l'implémentation de l'analyseur Aspic ont permis de valider la méthode, et montrent un réel gain de précision ainsi qu'un gain en performance.

Mots-clés : Vérification de propriétés numériques, Génération d'invariants polynomiaux, Analyse des Relations Linéaires, Élargissement, Accélération Abstraite, Aspic

Abstract

This work deals with verification of safety properties of programs, and more specifically with numerical properties. Linear Relation Analysis, which is an abstract interpretation based on a approximation of numerical states by convex polyhedra, has proved its efficiency. It consists in generating polyhedral overapproximations of the set of valuations associated to each control point. The introduction of a widening operator ensures the convergence of the analyses. However in some cases the invariants generated are not precise enough and improving the precision by delaying the widening is too costly. That is why we have considered so-called *acceleration methods*, which consist in computing the exact effect of one or several loops (as Presbuger formulae). The main drawback of these methods is that they apply only to a restricted class of programs.

In this thesis, we propose an approach combining the classical Linear Relation Analysis (with widening) and the notion of *abstract acceleration* which is useful in order to compute a precise overapproximation of the iterate application of certain types of loops. We aim at improving the precision of the analyses while always guaranteeing termination. The first experimental results obtained thanks to our tool Aspic have allowed to validate the method. They show a gain both of precision and performance.

Keywords : Numerical Properties Verification, Generation of polynomial Invariants, Linear Relation Analysis, Widening, Abstract Acceleration, Aspic.