HAL open science

Dioïdes et idéaux de polynômes en analyse statique

Arnaud Jobin

▶ To cite this version:

Arnaud Jobin. Dioïdes et idéaux de polynômes en analyse statique. Autre [cs.OH]. École normale supérieure de Cachan - ENS Cachan, 2012. Français. NNT : 2012DENS0004 . tel-00881301

HAL Id: tel-00881301

https://theses.hal.science/tel-00881301v1

Submitted on 8 Nov 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE / ENS CACHAN - BRETAGNE

sous le sceau de l'Université européenne de Bretagne

pour obtenir le titre de

DOCTEUR DE L'ÉCOLE NORMALE SUPÉRIEURE DE CACHAN

Mention : Informatique

École doctorale MATISSE

présentée par

Arnaud Jobin

Préparée à l'Unité Mixte de Recherche 6074
Institut de recherche en informatique
et systèmes aléatoires

Dioïdes et idéaux de polynômes en analyse statique

Thèse soutenue le 16 janvier 2012
devant le jury composé de :

Xavier RIVAL,
Chargé de recherche à l'INRIA Rocquencourt / *rapporteur*
Éric GOUBAULT,
Directeur de recherches au CEA / *rapporteur*

Claude JARD,
Professeur des universités à l'ENS Cachan - Bretagne / *examinateur*
Alessandra DI PIERRO,
Professeur des universités à l'université de Vérone / *examinateur*

Thomas JENSEN,
Directeur de recherche à l'INRIA / *directeur de thèse*
David CACHERA,
Maître de conférences à l'ENS Cachan - Bretagne / *directeur de thèse*

Table des matières

Table des matières **1**

1 Introduction **5**
1.1 Motivations 5
1.2 Structure de ce manuscrit 9

2 L'interprétation abstraite : un cadre générique pour l'analyse statique **11**
2.1 Autour de la structure d'ordre 16
2.1.1 Les treillis 16
2.1.2 Calculer dans les treillis 20
2.1.2.1 Les morphismes d'ordre et d'union 20
2.1.2.2 Points fixes dans les treillis 21
2.2 Syntaxe et sémantique des programmes 27
2.2.1 Syntaxe des programmes 28
2.2.2 Sémantique concrète des programmes 29
2.2.2.1 Le domaine concret 29
2.2.2.2 Sémantique concrète des commandes 30
2.2.2.3 Sémantique concrète : approche MOP et approche par plus petit point fixe 32
2.2.2.4 Une autre (?) définition de la sémantique 36
2.3 Analyse approchée des programmes 37
2.3.1 Structure du domaine abstrait et relation avec le domaine concret : un premier aperçu 38
2.3.1.1 Structure de l'ensemble des assertions abstraites . 38
2.3.1.2 Relation entre assertions concrètes et abstraites [CC92c] 38
2.3.2 Déroulement d'une analyse par interprétation abstraite dans un cadre idéal 39
2.3.2.1 Connexions de Galois sur treillis complets 39
2.3.2.2 Calculs corrects dans le domaine abstrait 43
2.3.2.3 Notion de meilleure fonction abstraite 45

2.3.3 Analyse par interprétation abstraite dans un cadre plus général . . . 46

3 Analyse de programmes à coûts 51
3.1 Les programmes quantitatifs . . . 53
3.1.1 Syntaxe des programmes quantitatifs . . . 53
3.1.2 Sémantique des programmes quantitatifs . . . 54
3.2 Une structure mathématique adaptée : les dioïdes de coûts . . . 54
3.2.1 Les opérateurs $\oplus$ et $\otimes$ . . . 55
3.2.2 Notion de dioïde et ordre associé . . . 55
3.2.3 Les dioïdes de coûts . . . 58
3.2.3.1 L'opérateur racine n-ième . . . 58
3.2.3.2 Les dioïdes de coûts : définition . . . 59
3.2.4 Exemples de dioïdes de coûts . . . 61
3.3 Sémantique linéaire et coût long-run . . . 65
3.3.1 Matrice de transition et moduloïde . . . 65
3.3.2 Coût long-run . . . 67
3.3.2.1 Quand les traces rejoignent les cycles . . . 71
3.4 Abstraire un programme à coûts . . . 75
3.4.1 Connexions de Galois dans les moduloïdes . . . 75
3.4.2 Relèvement des abstractions . . . 77
3.4.3 Sémantique abstraite induite . . . 78
3.4.4 Analyse de coût long-run . . . 79
3.5 Intégration des abstractions classiques dans le modèle linéaire . . . 82
3.5.1 Critique du relèvement linéaire . . . 83
3.5.2 Relèvement parfait des connexions de Galois . . . 84
3.5.2.1 Relèvement parfait dans le cas des parties d'ensemble . . . 84
3.5.2.2 Relèvement parfait dans le cas général . . . 87
3.5.3 Relèvement semi-parfait de connexions de Galois . . . 89
3.6 Implémentation de la méthode sur un exemple . . . 90
3.6.1 Syntaxe . . . 90
3.6.2 Sémantique opérationnelle quantitative . . . 92
3.6.3 Abstraction . . . 94
3.6.4 Résultats expérimentaux . . . 94
3.7 Comparaison avec les approches existantes . . . 95

4 Génération rapide d'invariants inductifs sous forme d'égalités polynomiales 99
4.1 Une structure mathématique adaptée : les idéaux . . . 102
4.2 Les programmes polynomiaux . . . 105

4.2.1 Syntaxe des programmes polynomiaux 105
4.2.2 Sémantique concrète des programmes polynomiaux 106
4.2.2.1 Sémantique opérationnelle petit pas et accessibilité 106
4.2.2.2 Sémantique concrète arrière 109
4.2.2.3 Comparaison de la sémantique concrète arrière avec la `SOPP` 112
4.3 Analyse approchée : génération d'invariants polynomiaux par procédure itérative 119
4.3.1 Sémantique abstraite 119
4.3.1.1 Abstraction des programmes polynomiaux 119
4.3.1.2 Correction de l'abstraction 121
4.3.2 Génération d'invariants par procédure itérative 125
4.3.3 Des preuves *Coq* 129
4.4 Le cas particulier des invariants inductifs 132
4.4.1 L'hypothèse d'inductivité 132
4.4.2 L'analyse `fastind` 134
4.4.2.1 Les contraintes d'idéaux 134
4.4.2.2 Sémantique abstraite 135
4.4.2.3 Correction de l'analyse `fastind` 136
4.4.3 Étude d'un exemple 137
4.5 Résultats expérimentaux 138
4.5.1 Comparaison avec le travail de Müller-Olm & Seidl 140
4.5.1.1 Description de la méthode 140
4.5.1.2 Comparaison sémantique 141
4.5.1.3 Comparaison des implémentations 142
4.5.2 Comparaison avec le travail de Sankaranarayanan *et al.* . . 143
4.5.2.1 Description de la méthode 143
4.5.2.2 Comparaison sémantique 143
4.5.2.3 Comparaison des implémentations 144
4.5.3 Comparaison avec le travail de Rodríguez-Carbonell et Kapur sur les boucles simples 144
4.5.3.1 Description de la méthode 144
4.5.3.2 Comparaison sémantique 145
4.5.3.3 Comparaison des implémentations 146
4.5.4 Comparaison avec la seconde approche Rodríguez-Carbonell et Kapur 147
4.5.4.1 Description de la méthode 147
4.5.4.2 Comparaison sémantique 148
4.5.4.3 Comparaison des implémentations 148
4.5.5 Intégration de notre analyse dans l'analyseur statique `sawja` 148

5 Conclusion **151**
5.1 Bilan sur nos travaux 151
5.1.1 Dans le domaine des analyses quantitatives 151
5.1.2 Dans le domaine de la génération d'invariants 153
5.2 Perspectives 154
5.2.1 Dans le domaine des analyses quantitatives 156
5.2.2 Dans le domaine de la génération d'invariants 156

A Chapitre 3 : Analyse des programmes à coûts **159**
A.1 Section 3.4 : Abstraire un programme à coûts 159
A.2 Section 3.6 : Implémentation de la méthode sur un exemple 161

B Chapitre 4 : Génération rapide d'invariants inductifs sous forme d'égalités polynomiales **165**
B.1 Section 4.2 : Les programmes polynomiaux 165

Bibliographie **177**

Chapitre 1

Introduction

1.1 Motivations

De l'intérêt de l'analyse de programmes

On peut dater au début des années 1970 l'apparition de l'ordinateur individuel (ou micro-ordinateur), conséquence directe de l'invention du microprocesseur. Dès lors, l'influence des ordinateurs n'a cessé de grandir dans notre quotidien. Ils se sont tout d'abord installés comme objet incontournable de nos foyers. Ils colonisent depuis peu de nombreux autres domaines qui étaient jusque là dominés par la mécanique. Évidemment, on pense tout d'abord au monde de l'automobile avec le système ABS ou encore le régulateur de vitesse. Cette tendance à embarquer de l'électronique est une tendance lourde que les équipementiers semblent vouloir poursuivre avec, notamment, l'apparition de nouveaux systèmes destinés à assister le conducteur : système d'aide au créneau, système anti-collision. On peut aussi citer le domaine de l'aéronautique et plus récemment le domaine médical avec les premières opérations assistées par ordinateur. Ces exemples n'ont pas été choisis au hasard. En effet, dans chacun de ces trois domaines, une défaillance du système électronique peut avoir de graves conséquences, les accidents pouvant provoquer des dommages matériels importants voire, dans le pire cas, des pertes humaines. Du fait leur potentielle dangerosité, ce genre de systèmes embarqués est qualifié de « critique ». Un grand nombre de procédés différents permettant de caractériser le comportement du programme peuvent alors entrer en jeu. Cet ensemble de procédés est regroupé sous le nom **d'analyse de programmes**. Plus précisément, l'analyse de programmes doit permettre d'empêcher les défaillances critiques en vérifiant ou démontrant que le programme a le comportement souhaité *c.à.d.* satisfait des propriétés de **sûreté**.

Les méthodes d'analyse de programmes

Schématiquement, l'ensemble des méthodes permettant l'analyse de programmes

peut se diviser en deux grandes catégories : le monde de **l'analyse dynamique** (appelé aussi test) et celui de **l'analyse statique**. La distinction entre ces deux catégories réside dans le processus d'analyse en lui-même qui peut s'appuyer, ou non, sur l'exécution du programme. Une analyse dynamique, ou test, consiste à vérifier que l'exécution du programme, sur un jeu de données bien choisi, aboutit au calcul souhaité par le programmeur. Le but d'une telle analyse est donc de révéler la présence d'erreurs dans le programme. Une analyse statique, quant à elle, à pour but d'inférer des propriétés du programme et ce, sans l'exécuter, en se basant uniquement sur son code. La finalité de ce type d'analyses est différente de celle du test. Il ne s'agit plus d'exhiber des erreurs mais plutôt de démontrer l'absence d'erreurs. Évidemment, il existe de nombreux ponts entre ces deux catégories d'analyse. D'une part, une analyse statique peut bénéficier de la connaissance de propriétés dynamiques. D'autre part, les problèmes de couverture des tests amènent naturellement l'utilisation d'analyses statiques dans le monde du test.

De la difficulté de l'analyse de programmes

L'analyse de programmes, qu'elle se fasse de manière statique ou dynamique, se heurte à de nombreuses difficultés. En analyse dynamique, un problème important provient du fait que l'ensemble des données d'entrée potentielles d'un programme est généralement infini ou au moins suffisamment grand pour que le test de toutes les entrées ne soit pas possible en un temps jugé raisonnable. On se restreint donc au test d'un ensemble fini de données d'entrée. Le problème qui se pose lors d'une analyse dynamique est donc celui du bon choix de ce jeu de données. Par exemple, il peut sembler intéressant que le jeu de données soit tel que l'exécution du programme sur celui-ci explore toutes les branches du programme (on parlera alors de couverture des branches). L'analyse statique, quant à elle, est confrontée à un résultat d'indécidabilité. Formellement, le théorème de Rice [Ric53], conséquence directe du problème de l'arrêt, énonce que tout propriété non triviale d'un langage de programmation Turing-complet est indécidable. Afin de contourner cette difficulté, les analyses statiques effectuent des approximations des comportements possibles du programme. La théorie de l'interprétation abstraite permet de donner un cadre formel à ces approximations.

Un bref aperçu de la théorie de l'interprétation abstraite

Le but d'une analyse par interprétation abstraite est d'inférer statiquement certaines propriétés du programme. Une telle analyse commence par la définition d'une **sémantique** : représentation formelle de la signification du programme. Elle s'attaque ensuite au calcul de cette sémantique. Or, généralement, la sémantique d'un programme n'est pas calculable. Il s'agit alors d'effectuer des approximations suffisantes afin de définir une sémantique approchée calculable. Il

est évidemment très important que l'information contenue dans la sémantique approchée soit utilisable pour répondre au problème initial. Pour ce faire, il faut que les propriétés déduites de la sémantique approchée soient des sur-approximations des propriétés que l'on souhaitait calculer initialement. Une analyse qui permet de calculer de telles sur-approximations est dite **correcte**.

Afin de bien appréhender ces notions, attardons-nous sur un exemple. Considérons le problème du calcul de l'ensemble des valeurs $\mathcal{E}_x$ que peut prendre une variable réelle x à un point de programme. Cet ensemble n'est pas calculable en général. Toutefois, l'analyse des intervalles, qui est un exemple d'analyse par interprétation abstraite, permet de calculer, sous forme d'un intervalle I_x, une sur-approximation de $\mathcal{E}_x$. Autrement dit, $I_x \supseteq \mathcal{E}_x$ et ainsi I_x fournit une réponse acceptable[1] à notre problème initial en ce sens qu'il capte, au minimum, toutes les valeurs possibles de x.

L'interprétation abstraite propose un cadre formel dans lequel toute propriété approchée est calculée de manière correcte. Ce cadre se base sur des critères qui semblent assez naturels. Tout d'abord, il est exigé que l'ensemble des propriétés possède une structure ordonnée, ce qui permet de pouvoir comparer toute paire de propriétés. L'idée derrière cette exigence est de pouvoir choisir entre deux approximations correctes celle qui approche au **mieux** la propriété initiale qui, rappelons-le, est non calculable. De ce fait, l'ensemble des propriétés approchées doit posséder une **structure d'ordre partiel** appelée formellement **treillis**. En plus de ce treillis des propriétés approchées dit treillis **abstrait**, il est supposé que l'ensemble des propriétés initiales définisse un treillis appelé treillis **concret**. En fait, le treillis abstrait doit être vu comme une copie grossière du treillis concret. Les calculs des propriétés approchées sont réalisables dans ce monde plus simple. D'autre part, dans le cadre de l'interprétation abstraite, nous devons pouvoir exprimer le fait qu'une propriété approchée est une sur-approximation de la propriété initiale correspondante. Cette notion de correction nécessite que les treillis abstraits et concrets puissent être comparés. Plus formellement, on supposera que ces deux treillis sont reliés par un couple de fonctions (α, γ) appelé **connexion de Galois**.

Revenons à l'exemple précédent afin d'illustrer ces nouvelles notions. La propriété que l'on souhaite calculer est l'ensemble $\mathcal{E}_x$. Comme tout ensemble réel, elle appartient au treillis des parties $\mathcal{P}(\mathbb{R})$. La propriété abstraite I_x est, quant à elle, membre d'un treillis appelé treillis des intervalles. Ces deux treillis sont reliés par le couple (α, γ) : la fonction α associe à tout ensemble réel le plus petit intervalle qui le contient et la fonction γ associe à tout intervalle l'ensemble des points qui le compose. Ce lien permet de comparer les propriétés initiales et approchées.

1. Notons au passage que $I_x =]-\infty, +\infty[$ est un résultat correct que peut rendre l'analyse des intervalles. Ce résultat est d'un intérêt pratique faible puisqu'il ne fournit pas plus d'information qu'initialement.

Nous avons noté précédemment $I_x \supseteq \mathcal{E}_x$. Rigoureusement, nous aurions dû noter $\gamma(I_x) \supseteq \mathcal{E}_x$. Ceci signifie que l'ensemble des points qui composent I_x contient l'ensemble $\mathcal{E}_x$ et démontre que la propriété I_x est bien une sur-approximation de la propriété $\mathcal{E}_x$.

En résumé, la sémantique concrète étant non calculable, une analyse par interprétation abstraite calcule une sémantique approchée définie sur un monde plus simple. Ces calculs sont réalisés de manière correcte, c'est à dire par sur-approximation de la sémantique concrète. Enfin, les mondes concrets et abstraits sont reliés de telle sorte que les propriétés calculées dans le monde abstrait peuvent être comparées aux propriétés concrètes.

Nous nous sommes consacrés, dans ce travail, à l'étude d'analyses statiques dans le cadre de l'interprétation abstraite.

Notre première contribution concerne l'étude de propriétés quantitatives. Nous avons vu, lors du paragraphe précédent, que la théorie de l'interprétation abstraite était très adaptée à des notions qualitatives en ce sens qu'elle définit un cadre dans lequel les propriétés peuvent être comparées et qui permet de définir la propriété qui approche au mieux la propriété initiale. En revanche, cette théorie se désintéresse de la notion de quantité que l'on peut même présenter comme orthogonale à ce cadre. Le premier problème auquel on s'est attaqué consiste en l'étude d'inférence de propriétés quantitatives dans le cadre de l'analyse statique. La question était double. Tout d'abord, il s'agissait de proposer une méthode permettant de définir statiquement un coût de programmes. Il importait ensuite de comparer notre méthode avec le cadre de l'interprétation abstraite. Afin de répondre au premier point, nous avons pris comme point de départ des programmes transportant des coûts. Ce faisant, il nous est apparu essentiel d'imposer une structure à l'ensemble des coûts transportés par les programmes. Nos réflexions ont abouti à la définition de ce que nous avons appelé « dioïdes de coûts ». Cette structure mathématique a été conçue de telle sorte à permettre une manipulation aisée des coûts de programmes. Elle permet aussi, via l'utilisation de la structure de moduloïde, d'exprimer la sémantique de nos programmes à coûts sous forme matricielle. Ce caractère matriciel est alors utilisé pour définir un cadre linéaire d'approximation. Une fois ce cadre défini, nous nous sommes attelés à la tâche définie par la deuxième partie de notre problème initial, à savoir comment notre structure de dioïde de coûts s'intègre dans la théorie de l'interprétation abstraite. Ce dernier point a été publié dans un workshop international avec comité de lecture [CJ10]. Le travail dans son entier a quant à lui été publié dans les actes d'une conférence internationale [CJJS08] et a donné lieu à la parution d'une version étendue dans un journal [CJJS10].

La deuxième contribution concerne le problème de l'inférence d'invariants

polynomiaux de programmes. Ce problème est essentiel dans le domaine de la vérification de programmes et a donc été largement étudié par le passé. Les approches initiales par Karr [Kar76] et Cousot et Halbawchs [CH78] ont démontré la possibilité d'inférer statiquement des invariants linéaires en les variables du programme. Elles ont mené à des implémentations efficaces basées sur des variantes du domaine polyédrique. Le problème de l'inférence d'invariants non-linéaires sous forme d'égalités polynomiales a, quant à lui, suscité un grand intérêt ces dix dernières années. Par exemple, Sankaranarayanan et al. [SSM04b] ont proposé une stratégie basée sur le calcul de contraintes pour générer des invariants non linéaires, dérivée de leur précédent travail sur les invariants linéaires [CSS03]. Une proposition concomitante par Müller-Olm et Seidl [MOS02, MOS04] définit une méthode par interprétation abstraite. Leur analyse consiste en une méthode de propagation arrière : partant d'un polynôme p, elle permet de calculer la plus faible pré-condition de validité de la relation $p = 0$ sur une classe de programmes polynomiaux. Rodríguez-Carbonell et Kapur ont aussi développé une approche par interprétation abstraite, basée sur le domaine des idéaux de variété [RCK04b, RCK04c, RCK07b, RCK04a, RCK07a]. Malgré la diversité de ces approches, les solutions apportées ne nous ont pas semblé complètement satisfaisantes. L'approche arrière de Müller-Olm [MOS04] qui nous semblait la plus adaptée à ce problème nécessite un calcul itératif lourd qui ne peut pas concurrencer les procédures de Rodríguez-Carbonell et Kapur [RCK07b, RCK07a]. Ces procédures, quant à elles, sont basées sur des méthodes avant qui utilisent un domaine d'une complexité qui ne nous semblait pas nécessaire. La question était donc de proposer une analyse aussi élégante que celle proposée par Müller-Olm et Seidl tout en étant aussi efficace que celle de Rodríguez-Carbonell et Kapur. Ceci a été réalisé à l'aide d'un hypothèse d'inductivité formulée dans les travaux de Sankaranarayanan et al. [SSM04b]. Plus précisément, nous avons développé une analyse mêlant les mécanismes d'interprétation abstraite décrit par Müller-Olm et Seidl, à un problème de résolution de contraintes induit par l'hypothèse d'inductivité. Les calculs de points fixes, généralement résolus par des procédures itératives, sont alors obtenus, grâce à ces contraintes, en un pas de calcul. De ce fait, et du fait de la facilité de résolution des contraintes, notre analyse est capable de générer rapidement des invariants polynomiaux de programmes. Ce travail a été résumé dans un rapport technique INRIA [CJJK11] et devrait prochainement donner lieu à une publication.

1.2 Structure de ce manuscrit

Ce document est organisé de la façon suivante.

Le Chapitre 2 décrit la théorie de l'interprétation abstraite, introduite par

Cousot & Cousot dans leur papier fondateur [CC77]. Elle répond à la problématique suivante. La plupart de propriétés que l'on souhaite inférer en analyse statique sont indécidables. Il est donc nécessaire, afin de pouvoir traiter ces propriétés, de réaliser des approximations. L'interprétation abstraite fournit un cadre rigoureux à ces approximations en définissant notamment la notion d'abstraction correcte.

Le Chapitre 3 présente une technique d'analyse statique qui permet de modéliser et d'approcher le comportement asymptotique moyen de programmes via un coût nommé coût *long-run*. Notre premier travail a été de trouver une structure mathématique adaptée à la manipulation des coûts présents dans le programme. Un effort particulier a été conduit afin que cette structure soit la plus générique possible et puisse être utilisée pour manipuler des coûts décrivant différentes ressources des programmes (comme le temps ou la mémoire). On présente ensuite un cadre, analogue à celui de l'interprétation abstraite permettant d'approcher de manière correcte les programmes que l'on considère. Cette abstraction permet de calculer le coût *long-run* sur le programme approché de sorte qu'il constitue une sur-approximation du coût *long-run* réel. Enfin, on étudie la possibilité de relever les abstractions classiques de l'interprétation abstraite dans notre cadre.

Le Chapitre 4 décrit une analyse statique permettant la génération d'invariants de programmes. Dans un premier temps, nous présentons un cadre mathématique adapté à la manipulation d'invariants. Nous présentons ensuite les programmes que l'on traite : il s'agit de programmes impératifs dotés d'une structure conditionnelle ainsi que d'une structure de boucle et dont les affectations sont polynomiales. Ceci étant fait, nous présentons une procédure générale de génération d'invariants ainsi que la procédure `fastind`, qui est un cas particulier de la procédure précédente dans le cas où l'on s'intéresse aux invariants inductifs. Enfin, nous exposons les résultats expérimentaux obtenus et détaillons les implémentations qui ont été faites. Les différences entre notre approche et les approches déjà existantes seront détaillées en fin de chapitre.

Chapitre 2

L'interprétation abstraite : un cadre générique pour l'analyse statique

Le chapitre suivant est consacré à l'étude de **l'interprétation abstraite**, cadre théorique permettant **l'analyse statique** de **programme**. Commençons par faire un survol des notions dont on aura besoin. Cet aperçu doit permettre au lecteur de se faire une première idée des concepts que l'on abordera rigoureusement par la suite.

L'interprétation abstraite, c'est quoi ?

L'interprétation abstraite est un cadre théorique permettant de réaliser des analyses statiques. Elle a été introduite par Cousot & Cousot [CC77]. On a évoqué, en introduction, l'indécidabilité de l'inférence de propriétés de programme [Ric53]. Afin de contourner ce problème, il semble naturel de tenter de résoudre un problème approché, plus simple que le problème initial. La théorie de l'interprétation abstraite permet d'encadrer, de manière rigoureuse, la manière dont ces approximations sont effectuées. Afin de simplifier le propos, introduisons un exemple.

Prenons le cas d'étude d'un programme `Prog` à deux variables réelles `x` et `y`, dont la valeur initiale n'est pas connue (elle peut dépendre, par exemple, d'un autre programme non encore analysé). Supposons qu'il soit critique de savoir qu'à la fin du programme les valeurs de `x` et `y` sont telles que $y < x - 3$. Pour résoudre ce problème, il suffit de connaître l'ensemble des **états accessibles** à la fin du programme c'est à dire l'ensemble des valeurs possibles pour les variables `x` et `y`. Malheureusement, cet ensemble n'est généralement pas calculable. Nous allons donc en calculer une approximation. La Figure 2.1 représente cette situation. Cette figure simple amène de nombreux commentaires.

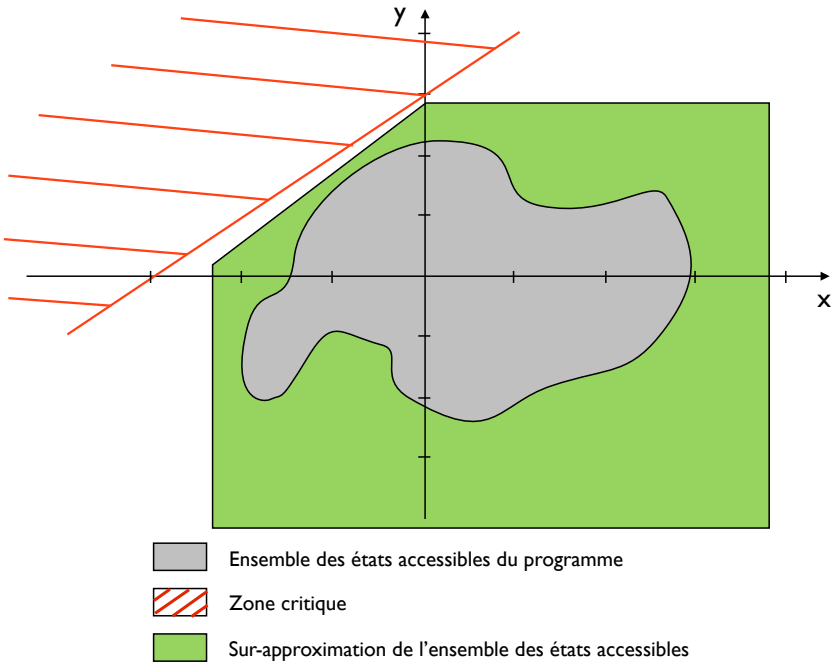


FIGURE 2.1 – Sur-approximation des états accessibles du programme `Prog`

Domaine concret, domaine abstrait

L'ensemble des états accessibles (ensemble gris) admet une représentation complexe. Au contraire, l'approximation (ensemble vert) est représenté sous la forme de l'intérieur d'un polygone, forme géométrique plus simple. Ces représentations ne sont pas anodines. Elles témoignent du fait que le monde dans lequel « vivent » les approximations est beaucoup plus simple que le monde initial. Ce monde des approximations sera appélé **domaine abstrait** tandis que le monde initial sera nommé **domaine concret**. On retient donc, avec ce nouveau vocabulaire, que puisque le domaine concret est trop fin, trop précis pour que les calculs soient effectués, on se place dans un domaine plus simple, plus grossier.

Correction, sur-approximation

D'autre part, on remarque que l'approximation englobe l'ensemble des états accessibles (tout point de l'ensemble gris est un point intérieur du polygone). Ceci est une propriété cruciale appelée **correction**. Les approximations que l'on souhaite calculer sont des **sur-approximations** de l'ensemble des états accessibles. Ceci nous permet d'affirmer que tout état « gris » vérifie la propriété « verte ». On verra que l'interprétation abstraite nous fournit un cadre de calcul des propriétés approchées qui est correct par construction : toute propriété approchée calculée dans ce cadre sera une sur-approximation des états accessibles.

Précision de l'analyse

Remarquons maintenant que l'ensemble $\mathbb{R}^2$ est une approximation correcte et ce quelque soit l'ensemble concret. Ceci signifie que l'approximation $\mathbb{R}^2$ ne nous fournit aucune information sur l'ensemble concret. Ceci est une spécificité de l'interprétation abstraite : on dispose d'un cadre assurant la correction des approximations mais pas d'une notion de mesure qui assurerait que l'on ne s'éloigne pas trop du monde concret. Ainsi le calcul approché effectué n'est pas forcément assez fin pour montrer la propriété que l'on souhaite.

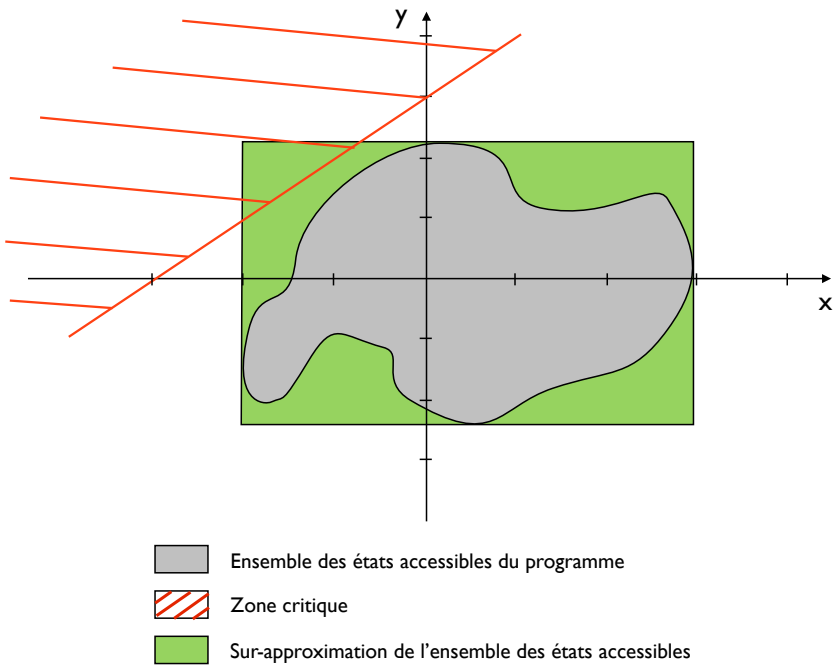


FIGURE 2.2 – Sur-approximation des états accessibles du programme `Prog`

Si l'on revient à la Figure 2.1, on voit que l'approximation calculée, en plus d'être correcte, est suffisamment précise pour montrer la propriété de sécurité que l'on souhaite, à savoir que $y < x - 3$. Ceci se lit aisément sur la figure : l'ensemble vert n'intersecte pas l'ensemble rouge. Il faut bien se rendre compte que ceci est dû au choix judicieux du domaine abstrait. Supposons, par exemple, que nous ayons choisi comme domaine abstrait le domaine des rectangles de côtés parallèles aux axes des coordonnées. La Figure 2.2 nous permet de nous rendre compte que ce domaine n'est pas assez fin pour résoudre le problème initial. En effet, comme on le voit dans cette figure, on ne peut représenter de rectangle englobant la zone grise qui n'intersecterait pas la zone rouge. Une des difficultés de l'interprétation abstraite est de proposer une analyse suffisamment fine pour

que son résultat soit une information d'intérêt mais suffisamment grossière pour qu'elle soit calculable. Les calculs dans le monde des rectangles sont certainement plus simples que ceux effectués dans le domaine des polygones mais ces calculs résultent en une information trop pauvre dans le cas qui nous concerne. Il s'agit donc de trouver un compromis entre calculabilité, rapidité de l'analyse et intérêt de l'information qu'elle produit.

Notion de meilleure sur-approximation

Enfin, la Figure 2.2 permet d'introduire une autre notion classique de l'interprétation abstraite : celle de **meilleure sur-approximation**. La zone verte de cette figure est la meilleure approximation correcte de l'ensemble gris si l'on considère le domaine abstrait des rectangles défini précédemment.

Notion d'ordre

Si on prend un peu de recul sur l'ensemble des notions que l'on vient d'introduire, on se rend compte qu'une bonne partie est liée à une notion de comparaison : approximation correcte ou sur-approximation, notion de meilleure approximation. En fait, la **notion d'ordre** joue un rôle essentiel en interprétation abstraite. Remarquons que plusieurs ordres différents entrent en jeu. En effet, nous avons besoin d'un ordre dans le domaine abstrait afin de pouvoir comparer les abstractions entre elles. Mais nous avons aussi besoin d'un ordre pour pouvoir comparer les propriétés du domaine abstrait avec celles du domaine concret ou, autrement dit, pour vérifier que les zones vertes englobent bien les zones grises.

Interpréter un programme

Jusque là, nous n'avons pas encore décrit la notion de **programme**. Pour cette introduction, contentons-nous de décrire un programme comme une séquence de commandes. Généralement, on compte parmi ces commandes des fonctions d'affectation, de test, des structures conditionnelles ainsi que des boucles. Interpréter un programme, c'est donner du sens à ses commandes. Supposons, par exemple, que le code du programme `Prog` possède une affectation `y` $:= -3x + 2$. L'interprétation de cette commande est donnée par la fonction :

$$\begin{array}{rcccccccc} q & : & \mathbb{R} & \times & \mathbb{R} & \rightarrow & \mathbb{R} & \times & \mathbb{R} \\ & & (x & , & y) & \mapsto & (x & , & -3x+2) \end{array}$$

La fonction q permet d'associer l'état des variables du programme avant et après affectation. On parle ici d'interprétation concrète puisque cette interprétation, qui a lieu dans le monde concret, permet de définir les états accessibles après affectation. On a évoqué le fait qu'il était difficile de calculer dans le domaine concret. On facilite ces calculs en choisissant un domaine abstrait plus grossier. Supposons, par exemple, qu'au lieu de connaître l'ensemble des états accessibles, l'on souhaite juste inférer les signes des variables `x` et `y`. Pour ce faire, on peut utiliser le domaine des signes `Sign` $= \{ \emptyset, \{0\}, \mathbb{R}^+, \mathbb{R}^-, \mathbb{R} \}$. Une analyse utilisant ce domaine permet, au mieux, d'obtenir le signe des variables `x` et `y`. Cette infor-

mation n'établit pas de relation entre les valeurs possibles de `x` et `y`. De ce fait, le domaine des signes est dit non relationnel et, de ce point de vue, est moins précis qu'un domaine relationnel tel que le domaine des polygones évoqué précédemment (figure 2.1). Dans le domaine des signes, la fonction q est interprétée par la fonction $q^\sharp$ suivante :

$$\begin{array}{rcccccccc} q^\sharp & : & \texttt{Sign} & \times & \texttt{Sign} & \rightarrow & \texttt{Sign} & \times & \texttt{Sign} \\ & & (S_1 & , & S_2) & \mapsto & (S_1 & , & \mathbb{R}^- *^\sharp S_1 +^\sharp \mathbb{R}^+) \end{array}$$

où l'on a noté $*^\sharp$ et $+^\sharp$ les fonctions exprimant la règle des signes ($\mathbb{R}^+ *^\sharp \mathbb{R}^- = \mathbb{R}^-$, $\mathbb{R}^- *^\sharp \mathbb{R}^- = \mathbb{R}^+ \ldots$). Ces fonctions sont des interprétations dans le domaine abstrait de la multiplication et de l'addition classique. Supposons que, par une analyse, l'on arrive à prouver que `x` ne peut admettre que des valeurs négatives avant affectation. Cette variable sera alors abstraite, avant affectation par $\mathbb{R}^-$. Avec cette donnée comme entrée, la fonction $q^\sharp$ renvoie $(\mathbb{R}^-, \mathbb{R}^+)$. Ceci signifie que `x` reste négatif après affectation alors que `y`, quelque soit son signe avant affectation, sera positif après. À l'opposé, si l'on détermine que la variable `x` est positive avant affectation, $q^\sharp$ renvoie le couple $(\mathbb{R}^+, \mathbb{R})$. Ceci signifie que l'analyse est trop imprécise pour obtenir le signe de `y` après affectation. D'autre part, notons l'utilisation du symbole $\sharp$ pour représenter les opérations abstraites.

Ce petit exemple permet de montrer comment interpréter une commande dans un domaine abstrait choisi. Une analyse par interprétation abstraite d'un programme définit non seulement l'interprétation de chaque commande dans le domaine abstrait mais aussi comment ces commandes abstraites se composent entre elles. Le mécanisme résultant permet de mimer, dans le monde abstrait, le comportement concret du programme.

Maintenant que nous avons introduit les principaux concepts de l'interprétation abstraite, nous allons décrire son fonctionnement de manière plus précise. Pour ce faire, nous nous appuierons sur les articles de Cousot & Cousot [CC77, CC79, Cou81, CC92b]. De plus, nous illustrerons les résultats de ce chapitre en nous basant sur une analyse par interprétation abstraite particulière : l'analyse polyédrique. Cette analyse a été introduite par Cousot & Halbwachs [CH78] et permet d'inférer des ensemble d'inégalités linéaires vérifiées par les variables du programmes. En Figure 2.3, nous détaillons le résultat de l'analyse polyédrique sur le programme `sqrt` constitué d'une boucle simple. Cette analyse permet d'inférer que l'égalité $-2r + t - 1 = 0$ est vérifiée avant et après exécution de la boucle. Elle prouve aussi la validité de l'inégalité $-n + s - 1 \geq 0$ en fin de programme.

Ce chapitre est structuré comme suit. Dans un premier temps, nous étudierons en Section 2.1 la notion d'ordre partiel. Cette section « boîte à outil » permet de présenter de nombreux résultats que l'on utilisera dans les sections suivantes. En

```
1.  r := 0;
2.  s := 1;
3.  t := 1;
      −2r + t − 1 = 0
4.  while s ≤ n do
5.    r := r + 1;
6.    t := t + 2;
7.    s := s + t;
      −2r + t − 1 = 0 ∧ −n + s − 1 ≥ 0
8.
```

FIGURE 2.3 – L'analyse polyédrique d'un programme `sqrt`

Section 2.2 nous décrirons de manière très générale les langages de programmation et étudierons le comportement concret des programmes. Nous présenterons notamment des méthodes (non calculables en général) pour calculer l'ensemble des états accessibles d'un programme. Enfin, on décrira en Section 2.3 comment effectuer des approximations dans le cadre de l'interprétation abstraite afin de calculer des approximations correctes par construction de l'ensemble des état accessibles.

Les résultats présentés dans ce chapitre seront utilisés dans le Chapitre 4 et 3. Les analyses présentées dans ces deux chapitres sont réalisées dans des cadres favorables au raisonnement par interprétation abstraite. Les théorèmes classiques ci-après peuvent donc être énoncés sous des hypothèses non minimales.

2.1 Autour de la structure d'ordre

En introduction, nous avons vu qu'une analyse par interprétation abstraite permet de générer une sur-approximation des états accessibles du programme. Cette comparaison ne peut se faire que si l'on a préalablement défini un ordre. Cette section explore les résultats de la théorie de l'ordre qui nous serviront par la suite. Nous commençons par définir la notion de treillis. Ce concept est particulièrement important en interprétation abstraite car les domaines concrets et abstraits seront supposés munis d'une structure de treillis.

2.1.1 Les treillis

Un treillis est une structure ordonnée particulière. Nous définissons tout d'abord la notion d'ordre. Du point de vue des notations, nous utiliserons par la suite le symbole $\sqsubseteq$ pour définir une relation binaire notée de manière infixe.

Définition 2.1 (Pré-ordre et ordre partiel).
Soit S un ensemble. Un ordre partiel $\sqsubseteq$ *est une relation binaire qui est :*

- *réflexive :* $\forall x \in S, \quad x \sqsubseteq x$
- *transitive :* $\forall x, y, z \in S, \quad (x \sqsubseteq y \wedge y \sqsubseteq z) \Rightarrow x \sqsubseteq z$
- *anti-symétrique :* $\forall x, y \in S, \quad (x \sqsubseteq y \wedge y \sqsubseteq x) \Rightarrow x = y$

Une relation binaire réflexive et transitive est appelée un pré-ordre.

On notera $S(\sqsubseteq)$ pour indiquer que S est un ensemble muni d'une structure d'ordre partiel.

Dans les ensembles munis d'une structure d'ordre partiel, on distingue certains éléments : les bornes supérieures et inférieures.

Définition 2.2 (Bornes supérieures et inférieures).
Soit $\sqsubseteq$ un ordre partiel sur un ensemble S.

Un élément $u \in S$ est un majorant *d'un sous-ensemble X de S s'il est plus grand que tous les éléments de X : $\forall x \in X, \quad x \sqsubseteq u$.*

Dans le cas où u est un majorant de X plus petit que tous les autres majorants de X, on parle de borne supérieure *:*
$(\forall x \in X,\ x \sqsubseteq u) \ \wedge\ \forall u' \in S : (\forall x \in X : x \sqsubseteq u') \Rightarrow (u \sqsubseteq u')$.

La borne supérieure de X, si elle existe, est unique.

Les notions de minorant *et de* borne inférieure *sont définies de manière duale.*

Par la suite, on utilisera la notation $\mathtt{sup}\{x, y\}$ ou préférentiellement $x \bigsqcup y$ pour représenter la borne supérieure, lorsqu'elle existe, de l'ensemble $\{x, y\}$. Cette notation est étendue aux sous-ensembles quelconques de S : $\mathtt{sup} X = \bigsqcup X$ représente la borne supérieure, lorsqu'elle existe, de l'ensemble X. On utilisera les symboles $\bigsqcap$ et $\mathtt{inf}$ pour représenter la notion duale de borne inférieure. Enfin, on remarque que la borne supérieure u d'un ensemble X n'est pas forcément contenue dans X.

La définition suivante énonce qu'un treillis est un ensemble ordonné muni d'une borne supérieure et d'une borne inférieure.

Définition 2.3 (Notion de treillis).
Soit $S(\sqsubseteq)$ un ensemble partiellement ordonné.

On dira que S est un treillis *si tout couple d'éléments admet une borne supérieure et une borne inférieure : $\forall x, y \in S, \quad x \bigsqcup y$ et $x \bigsqcap y$ existent dans S.*

On dira que S est un treillis complet *si tout ensemble d'éléments admet une borne supérieure et une borne inférieure : $\forall X \subseteq S, \quad \bigsqcup X$ et $\bigsqcap X$ existent dans S.*

En particulier, dans un treillis complet, l'ensemble S possède un un plus petit

élément, nommé infimum *et noté* $\bot$ *(*$= \bigsqcap S$*) ainsi qu'un plus grand élément nommé* supremum *et noté* $\top$ *(*$= \bigsqcup S$*).*
Un ensemble S *muni d'une structure de treillis complet sera alors noté :*

$$S(\sqsubseteq, \bot, \top, \bigsqcup, \bigsqcap)$$

Étude d'un exemple : l'analyse polyédrique [CH78]
On considère un programme à m variables ($\{\mathtt{x_1}, \ldots, \mathtt{x_m}\}$) à valeurs réelles. Dans l'analyse polyédrique, le domaine concret est représenté par un treillis de parties tandis que le treillis abstrait est le treillis des polyèdres convexes. Présentons plus précisément ces deux treillis.

Le domaine concret est le treillis $\mathcal{P}(\mathbb{R}^m)(\subseteq, \emptyset, \mathbb{R}^m, \cup, \cap)$. À chaque point de programme, on associe à chaque variable du programme un élément de ce treillis, à savoir l'ensemble des valeurs possibles de la variable à ce point. On définit ainsi l'ensemble des états accessibles du programme.

Le domaine abstrait, quant à lui, est représenté par le treillis des polyèdres convexes $\mathtt{P}_m(\sqsubseteq_{\mathtt{P}}, \bot_{\mathtt{P}}, \top_{\mathtt{P}}, \bigsqcup_{\mathtt{P}}, \bigsqcap_{\mathtt{P}})$. L'ensemble $\mathtt{P}_m$ représente les polyèdres convexes de dimension au plus m. Un polyèdre convexe peut être défini comme l'ensemble des solutions d'un système fini d'inégalités linéaires. Géométriquement parlant, c'est l'intersection de demi-espaces fermés (chaque inégalité représente un demi-espace fermé). La Figure 2.4 donne la représentation sous forme d'inégalités d'un polyèdre convexe p de $\mathbb{R}^2$ ainsi que son interprétation géométrique.

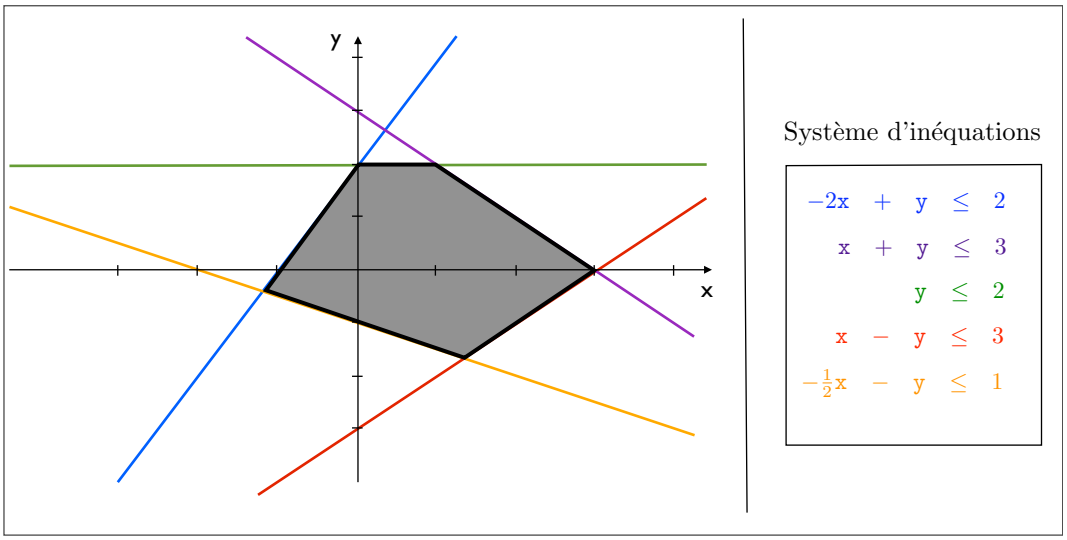


FIGURE 2.4 – Un polyèdre convexe p de $\mathbb{R}^2$

L'infimum $\bot_{\mathtt{P}}$ du treillis $\mathtt{P}_m$ est l'ensemble vide. Le supremum $\top_{\mathtt{P}}$ du treillis $\mathtt{P}_m$ est l'ensemble $\mathbb{R}^m$. L'opérateur de borne inférieure $\bigsqcap_{\mathtt{P}}$ est simplement l'intersection

ensembliste. En effet, l'intersection de deux polyèdres convexes est un polyèdre convexe. De plus, cet opérateur est aisé à calculer : étant donnés deux polyèdres convexes p_1 et p_2 définis par des ensembles d'inégalités, le polyèdre $p_1 \sqcap_{\mathsf{P}} p_2$ est le polyèdre convexe défini en regroupant toutes ces inégalités. Pour revenir à la Figure 2.4, on peut voir le polyèdre p comme intersection des polyèdres p_1 et p_2, où p_1 est défini par le système d'inégalités $\{-2\mathtt{x} + \mathtt{y} \leq -1,\ \mathtt{x} + \mathtt{y} \leq 3,\ \mathtt{y} \leq 2\}$ et p_2 défini par $\{\mathtt{x} - \mathtt{y} \leq 3,\ -\frac{1}{2}\mathtt{x} - \mathtt{y} \leq 2\}$. L'opérateur de borne supérieure $\bigsqcup_{\mathsf{P}}$ est, quant à lui, plus difficile à calculer. L'union ensembliste ne convient pas puisque, partant de deux polyèdres convexes p_1 et p_2, l'ensemble $p_1 \cup p_2$ n'est pas convexe en général. En fait, $p_1 \bigsqcup_{\mathsf{P}} p_2$ est défini comme le plus petit polyèdre convexe contenant p_1 et p_2 *c.à.d.* l'enveloppe convexe des points contenus dans p_1 et p_2. Le calcul d'enveloppe convexe nécessite une autre représentation des polyèdres basée sur l'ensemble de ses sommets. D'un point de vue implémentation, il semble difficile de faire un choix entre ces deux représentations, chacune des deux étant utile pour réaliser certains calculs associés au domaine des polyèdres. De plus, notons que ces deux représentations peuvent être utilisées conjointement pour décider efficacement de l'inclusion de polyèdres. Précisons enfin que la complexité des fonctions permettant la conversion d'une représentation vers l'autre [Lan66, Bal61] augmente très rapidement lorsque le nombre de variables et d'inégalités du polyèdre croît. De ce fait, il semble intéressant d'effectuer l'analyse de manière simultanée sur chacune de ces deux représentations. Cette méthodologie, bien que redondante, évite l'utilisation excessive des fonctions de conversion et semble apporter de meilleurs résultats dans la pratique [CH78]. La Figure 2.5 représente le polyèdre p_1 et ses deux représentations.

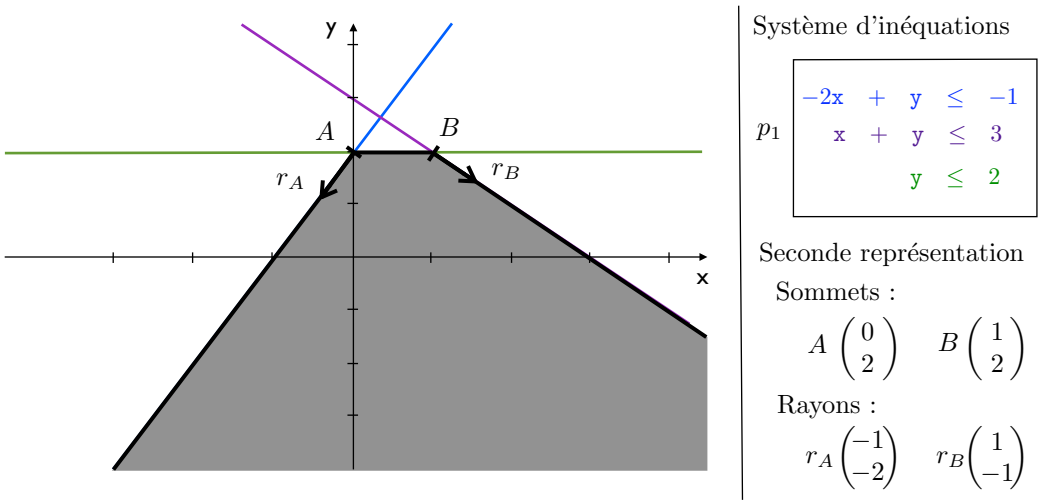


FIGURE 2.5 – Un polyèdre convexe p_1 de $\mathbb{R}^2$

Tirons un premier bilan sur cet exemple. En plus de la structure d'ordre qui est essentielle dans la théorie de l'interprétation abstraite, il faut retenir de cette discussion l'importance de pouvoir représenter les objets de l'analyse de manière concise et de telle sorte que les opérateurs agissant sur ces objets puissent être calculés de manière efficace.

La notion de treillis que nous avons définie ici est importante en interprétation abstraite. En effet, les domaines concret et abstrait seront supposés munis d'une structure de treillis. Jusque là, nous avons argumenté sur l'importance d'une structure ordonnée qui permet notamment de comparer les propriétés abstraites (ou concrètes) entre elles. En plus de la notion d'ordre, nous rajoutons donc l'exigence de l'existence d'un opérateur de borne supérieure et d'un opérateur de borne inférieure. Ces opérateurs permettent de combiner les propriétés entre elles.

2.1.2 Calculer dans les treillis

Nous avons vu que les domaines concrets et abstraits sont représentés par des treillis. Nous étudions maintenant les fonctions définies sur cette structure.

2.1.2.1 Les morphismes d'ordre et d'union

Parmi les fonctions définies sur les treillis, il convient de distinguer celles qui permettent de transporter la structure d'ordre. On parlera alors de morphisme d'ordre ou plus simplement de fonctions monotones.

Définition 2.4 (Propriétés des fonctions sur les treillis)**.**
Soient $S_1(\sqsubseteq_1, \bot_1, \top_1, \bigsqcup_1, \bigsqcap_1)$ *et* $S_2(\sqsubseteq_2, \bot_2, \top_2, \bigsqcup_2, \bigsqcap_2)$ *deux treillis complets. On dira qu'une fonction* $\varphi : S_1 \to S_2$ *est :*

- monotone *si elle vérifie la propriété de transport d'ordre :*

$$\forall x, y \in S_1, \quad x \ \sqsubseteq_1 \ y \ \Rightarrow \ \varphi(x) \ \sqsubseteq_2 \ \varphi(y)$$

  On parlera aussi de morphisme d'ordre.

- additive *ou appelée* morphisme d'union complet *si elle transporte l'opérateur de borne supérieure : pour tout ensemble* $X \subseteq S_1$, $X \neq \emptyset$,

$$\begin{aligned} \varphi(\textstyle\bigsqcup_1 X) &= \textstyle\bigsqcup_2 \varphi(X) \\ &= \textstyle\bigsqcup_2 \{ \ \varphi(x) \mid x \in X \ \} \end{aligned}$$

  Dans ce cas, on pourra utiliser la notation $\bigsqcup$`-mc`.

Tout d'abord, revenons sur le terme *monotone*. Mathématiquement parlant, nous aurions plutôt dû employer le terme de fonction croissante. Nous garderons le terme monotone pour deux raisons. Tout d'abord, cette désignation est classique en théorie de l'ordre. D'autre part, remarquons qu'un fonction décroissante peut facilement être vue comme une fonction croissante : il suffit pour cela de renverser l'ordre du treillis S_2 *c.à.d.* considérer $S_2(\sqsupseteq)$ au lieu de $S_2(\sqsubseteq)$.

Notons aussi que les propriétés de la définition 2.4 sont données dans un ordre croissant d'exigence : un morphisme d'union complet est toujours monotone. On parlera aussi de *morphisme d'union complet strict* pour un morphisme d'union complet φ tel que $\varphi(\bot_1) = \bot_2$. Dans ce cas, on pourra noter que φ est une fonction $\bigsqcup$-`mcs`. Enfin, remarquons que dans le but de simplifier l'écriture de la définition 2.4, nous nous sommes placés dans le cadre de treillis complets, ce qui permet d'assurer l'existence de bornes supérieures pour tout ensemble. On pourrait étendre ces définitions sur des ensembles munis seulement d'ordre partiel : un morphisme d'union est alors une fonction qui transporte les bornes supérieures existantes.

2.1.2.2 Points fixes dans les treillis

En interprétation abstraite, les calculs sont généralement effectués suivant une approche itérative consistant à calculer de manière incrémentale un point fixe. Commençons tout d'abord par définir la notion de point fixe.

Définition 2.5 (Point fixe)**.**
Soit $S(\sqsubseteq)$ un ensemble partiellement ordonnée et $\varphi : S \to S$.

L'élément $x \in S$ est un pré-point fixe *de φ si :* $\quad x \sqsubseteq \varphi(x)$.

L'élément $x \in S$ est un post-point fixe *de φ si :* $\quad \varphi(x) \sqsubseteq x$.

L'élément $x \in S$ est un point fixe *de φ si c'est à la fois un pré-point fixe et un post-point fixe :* $\quad \varphi(x) = x$.

Il convient de noter que la littérature ne s'accorde pas sur le vocabulaire de pré-point fixe et de post-point fixe. Nous avons suivi la dénomination de [CC92b] qui s'oppose à celle que l'on peut trouver dans [DP90, Bac00].

Le théorème suivant connu sous le nom de théorème de Knaster-Tarski [Tar55] est un résultat central de l'approche itérative. Il énonce l'existence du plus petit point fixe de toute fonction monotone dans un treillis complet.

Théorème 2.1 (Knaster-Tarski)**.**
Soit $S(\sqsubseteq, \bot, \top, \bigsqcup, \bigsqcap)$ un treillis complet et $\varphi : S \to S$ une fonction monotone.

L'ensemble $\mathtt{PF}_\varphi$ des points fixes de φ forme un treillis complet. Ainsi, φ possède

un plus petit point fixe (l'infimum de $\mathtt{PF}_\varphi$*) caractérisé par :*

$$\mathtt{lfp}\ \varphi \quad = \quad \bigsqcap \ \{\ x \in S \mid \varphi(x) \sqsubseteq x\ \}$$

et un plus grand point fixe (le supremum de $\mathtt{PF}_\varphi$*), caractérisé par :*

$$\mathtt{gfp}\ \varphi \quad = \quad \bigsqcup \ \{\ x \in S \mid x \sqsubseteq \varphi(x)\ \}$$

En reprenant le vocabulaire de la définition 2.5, le plus petit point fixe de φ est défini comme le plus petit post-point fixe de φ et son plus grand point fixe est défini commme le plus grand de ses pré-points fixes. La répartition des points fixes dans un treillis complet S est généralement représentée par le schéma de la Figure 2.6. Notons, que l'élément $\bot$ est toujours un pré-point fixe ($\bot \sqsubseteq \varphi(\bot)$) et que l'élément $\top$ est toujours un post-point fixe ($\varphi(\top) \sqsubseteq \top$).

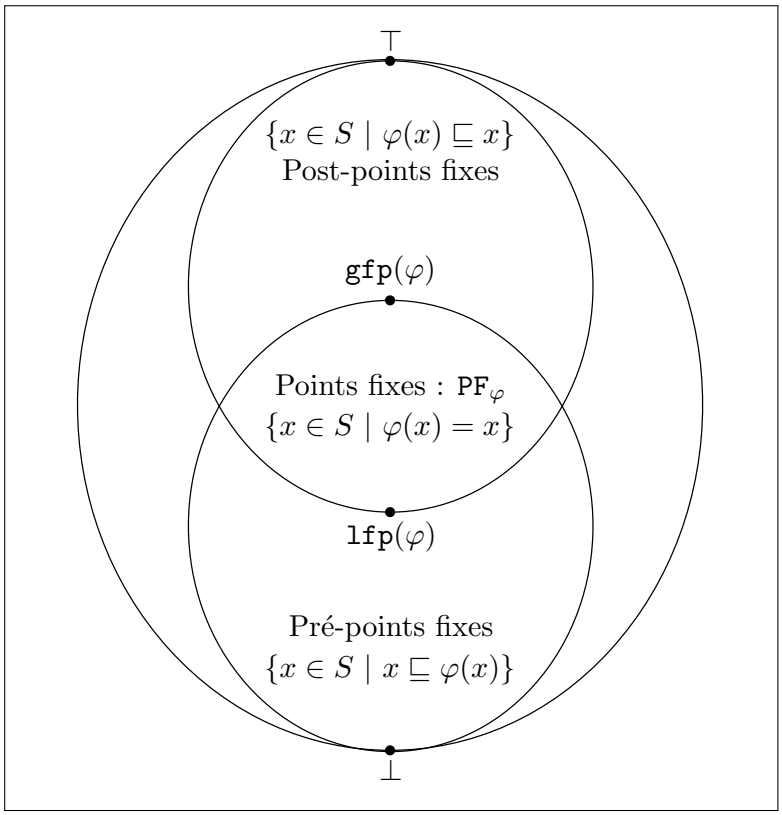


FIGURE 2.6 – Points fixes d'un treillis complet S

Le théorème 2.1 caractérise le plus petit point fixe de φ mais ne fournit pas de méthode constructive qui permettrait son calcul. Le théorème suivant est un premier pas vers cette méthode de calcul.

Théorème 2.2 (Kleene).
Soit $S(\sqsubseteq, \bot, \top, \bigsqcup, \bigsqcap)$ *un treillis complet et* $\varphi : S \to S$ *une fonction* $\bigsqcup$-`mc`.

φ *admet un plus petit point fixe caractérisé par :*

$$\texttt{lfp}\ (\varphi) \ = \ \bigsqcup \ \{ \ \varphi^n(\bot) \mid n \in \mathbb{N} \ \}$$

Il convient de faire quelques remarques sur ce théorème. Tout d'abord, notons que le théorème 2.2 admet un énoncé dual. Dans le cas où la fonction de départ est un morphisme d'intersection complet ($\bigsqcap$-`mc`), son plus grand point fixe peut être obtenu en itérant à partir du supremum $\top$ du treillis :

$$\texttt{gfp}\ (\varphi) = \bigsqcap \ \{ \ \varphi^n(\top) \mid n \in \mathbb{N} \ \}$$

Remarquons ensuite que la suite $(\varphi^n(\bot))_{n\in\mathbb{N}}$ est croissante. Cette propriété est facilement obtenue par récurrence : initialement $\bot \sqsubseteq \varphi(\bot)$ et le caractère monotone de φ permet de conclure en appliquant φ à chaque membre de l'inclusion. Ainsi, pour tout $k \in \mathbb{N}$, nous avons :

$$\bigsqcup \ \{ \ \varphi^n(\bot) \mid n \leq k \ \} = \varphi^k(\bot)$$

Afin de calculer $\texttt{lfp}(\varphi)$ à l'aide de ce théorème, il suffit donc, partant de $\bot$, d'appliquer successivement φ. Cette méthode suppose le caractère stationnaire de la suite $(\varphi^n(\bot))_{n\in\mathbb{N}}$, ce qui n'est pas forcément le cas. On peut apporter deux solutions à ce problème d'arrêt des itérations. La première est structurelle : elle demande que le treillis choisi pour itérer soit tel que toute suite croissante soit stationnaire. Autrement dit, « choisissons un cadre dans lequel ça marche ! ». La seconde technique est algorithmique. Elle consiste, à chaque étape de l'itération, à approcher l'élément calculé par un élément plus « grossier », de sorte à créer une itération approchée qui termine en temps fini. Le résultat obtenu par ce dernier procédé est alors une approximation du point fixe que l'on souhaite calculer. Détaillons ces deux approches.

La condition de chaîne ascendante

Nous commençons par présenter l'approche structurelle : la condition de chaîne ascendante assure, par définition, la stabilité de la suite en un temps fini.

Définition 2.6 (Condition de chaîne ascendante).
Soit $S(\sqsubseteq)$ *un ensemble partiellement ordonné.*

On dit que S *vérifie la* condition de chaîne ascendante *si toute suite croissante* $(x_i)_{i\in\mathbb{N}}$ *est stationnaire :*

$$(\ \forall i \in \mathbb{N},\ x_i \ \sqsubseteq \ x_{i+1}\ ) \ \Rightarrow \ (\ \exists m \in \mathbb{N},\ \forall k \geq m,\ x_k = x_m\ )$$

On dit que S vérifie la condition de chaîne descendante *si toute suite décroissante $(x_i)_{i\in\mathbb{N}}$ est stationnaire :*

$$(\ \forall i \in \mathbb{N},\ x_i \ \sqsupseteq\ x_{i+1}\ )\ \Rightarrow\ (\ \exists m \in \mathbb{N},\ \forall k \geq m,\ x_k = x_m\ )$$

Ainsi, dans un treillis complet, si la condition de chaîne ascendante est vérifiée, le théorème 2.2 fournit un algorithme de calcul de plus petit point fixe de φ par calcul successif des itérées de φ. De manière duale, dans un treillis vérifiant la condition de chaîne descendante, le théorème 2.2 permet le calcul du plus grand point fixe de φ en un nombre fini d'étapes. Un treillis vérifiant une condition de chaîne (ascendante) définit donc un cadre très favorable en interprétation abstraite. Il faut toutefois nuancer ce propos. Même si l'on a obtenu un procédé itératif terminant en temps fini, la condition de chaîne (ascendante) ne précise pas le nombre d'itérations nécessaire avant stabilité. L'algorithme déduit du théorème 2.2 ne s'exécute donc pas forcément en un temps jugé raisonnable. De ce fait, même dans le cas où le treillis vérifie la condition de chaîne ascendante, on pourra se satisfaire d'un calcul approché afin d'améliorer la vitesse de convergence de la méthode.

Les opérateurs d'accélération de convergence [CC92a]
Présentons maintenant l'approche algorithmique. Nous commençons par définir la notion d'opérateur d'élargissement. Partant d'une suite $(x_i)_{i\in\mathbb{N}}$, ce type d'opérateur permet de définir une suite de termes approchés $(y_i)_{i\in\mathbb{N}}$ qui se stabilise en un nombre fini d'itérations.

Définition 2.7 (Opérateur d'élargissement)**.**
Soit $S(\sqsubseteq, \bot, \top, \bigsqcup, \bigsqcap)$ un treillis complet.

Un opérateur d'élargissement *est une fonction $\nabla : S \times S \to S$ telle que :*

(Larg1) $\forall x, y \in S, \quad x \ \sqsubseteq\ x \ \nabla\ y$

(Larg2) $\forall x, y \in S, \quad y \ \sqsubseteq\ x \ \nabla\ y$

(Larg3) *Pour toute suite $(x_i)_{i\in\mathbb{N}}$ croissante la suite $(y_i)_{i\in\mathbb{N}}$ définie par :*

$$\forall i \in \mathbb{N},\ \left\{ \begin{array}{rcl} y_0 & = & x_0 \\ y_{i+1} & = & y_i \ \nabla\ x_{i+1} \end{array} \right.$$

est croissante et stationnaire à partir d'un certain rang.
$(\ \exists m \in \mathbb{N}, \forall k \geq m,\ y_k = y_m\ )$

Notons que, si la condition de chaîne ascendante est vérifiée dans le treillis S, l'opérateur d'union $\bigsqcup$ vérifie les propriétés listées ci-dessus. C'est donc, dans ce cadre, un opérateur d'élargissement.

Le théorème 2.2 permet de définir le plus petit point fixe d'une fonction φ comme limite de la suite $(\varphi(\bot))_{n\in\mathbb{N}}$. En utilisant un opérateur d'élargissement, on peut calculer une limite approchée de cette suite en un nombre fini d'itérations. Le théorème suivant énonce ce résultat.

Théorème 2.3 (Calcul approché de `lfp` par élargissement)**.**
Soit $S(\sqsubseteq, \bot, \top, \bigsqcup, \bigsqcap)$ un treillis complet et $\varphi : S \to S$ une fonction $\bigsqcup$-`mc`.
Soit $\nabla : S \times S \to S$ un opérateur d'élargissement.

Alors, la suite $(x_i)_{i\in\mathbb{N}}$ définie par :

$$\forall i \in \mathbb{N}, \begin{cases} x_0 & = & \bot & \\ x_{i+1} & = & x_i & \textit{si } \varphi(x_i) \sqsubseteq x_i \\ & = & x_i \nabla \varphi(x_i) & \textit{sinon} \end{cases}$$

est croissante et stationnaire à partir d'un certain rang m.

Sa limite x_m est un post-point fixe de φ et de ce fait une sur-approximation de `lfp`(φ) *:*

$$\varphi(x_m) \sqsubseteq x_m \quad \textit{et} \quad \texttt{lfp}(\varphi) \sqsubseteq x_m$$

Notons tout d'abord que le théorème 2.3 est énoncé avec une hypothèse forte sur l'opérateur φ : il est supposé être $\bigsqcup$-`mc`. Les opérateurs définis par une analyse statique par interprétation abstraite ne vérifient pas forcément cette hypothèse. Généralement, de telles analyses proposent des opérateurs d'élargissement ad hoc qui possèdent des propriétés plus strictes que celles présentées en définition 2.7. La combinaison de ces propriétés avec d'autres propriétés spécifiques à l'analyse permet, en pratique, de relâcher l'hypothèse faite sur φ. Nous ne présentons pas ici d'autres opérateurs d'élargissement mais soulignons qu'il existe de nombreuses variantes de la définition 2.7.

D'autre part, notons que si l'on se place dans un treillis vérifiant la condition de chaîne ascendante, et que l'on choisit $\nabla = \bigsqcup$, la suite définie dans le théorème 2.3 permet un calcul exact de `lfp`(φ) correspondant au calcul de la suite des itérées du théorème 2.2. De ce point de vue, l'approche par élargissement peut être vue comme une généralisation de l'approche par chaîne ascendante.

La limite x_m calculée ci-dessus est une approximation « par au-dessus » du plus petit point fixe `lfp`(φ) que l'on souhaite calculer. Cette sur-approximation peut être assez grossière. Il serait donc souhaitable de pouvoir redescendre dans le treillis $\texttt{PF}_\varphi$ pour approcher de plus près `lfp`(φ) tout en restant un post-point fixe et donc une sur-approximation de `lfp`(φ). Ceci est réalisé à l'aide d'un opérateur de rétrécissement.

Définition 2.8 (Opérateur de rétrécissement)**.**
Soit $S(\sqsubseteq, \bot, \top, \bigsqcup, \bigsqcap)$ un treillis complet.

Un opérateur de rétrécissement *est une fonction* $\Delta : S \times S \to S$ *telle que :*

(Ret1) $\forall x, y \in S, \quad (\ y \ \sqsubseteq \ x \ ) \ \Rightarrow \ (\ y \ \sqsubseteq \ (x \ \Delta \ y) \ \sqsubseteq x \ )$

(Ret2) *Pour toute suite* $(x_i)_{i\in\mathbb{N}}$ *décroissante la suite* $(y_i)_{i\in\mathbb{N}}$ *définie par :*

$$\forall i \in \mathbb{N}, \left\{ \begin{array}{rcl} y_0 & = & x_0 \\ y_{i+1} & = & y_i \ \Delta \ x_{i+1} \end{array} \right.$$

est décroissante et stationnaire à partir d'un certain rang.
$(\ \exists m \in \mathbb{N}, \forall k \geq m, \ y_k = y_m \ )$

Nous pouvons effectuer le même type de remarque que celle de la définition 2.7 : si la condition de chaîne descendante est vérifiée dans le treillis S, l'opérateur d'intersection $\bigsqcap$ vérifie les propriétés listées ci-dessus. C'est donc, dans ce cadre, un opérateur de rétrécissement.

Théorème 2.4 (Affinement d'un calcul par rétrécissement)**.**
Soit $S(\sqsubseteq, \bot, \top, \bigsqcup, \bigsqcap)$ *un treillis complet et* $\varphi : S \to S$ *une fonction* $\bigsqcup$-$\mathtt{mc}$.
Soit $\Delta : S \times S \to S$ *un opérateur de rétrécissement.*

Alors, pour tout $\mathtt{a}$ *post-point fixe de* φ *(*$\varphi(\mathtt{a}) \sqsubseteq \mathtt{a}$ *et donc* $\mathtt{lfp}(\varphi) \sqsubseteq \mathtt{a}$*), la suite* $(x_i)_{i\in\mathbb{N}}$ *définie par :*

$$\forall i \in \mathbb{N}, \left\{ \begin{array}{rcl} x_0 & = & \mathtt{a} \\ x_{i+1} & = & x_i \ \Delta \ \varphi(x_i) \end{array} \right.$$

est décroissante et stationnaire à partir d'un certain rang m.

Tous les termes de cette suite sont des post-points fixes de φ *qui vérifient :*

$$\forall i \in \mathbb{N}, \quad \mathtt{lfp}(\varphi) \ \sqsubseteq \ \varphi(x_i) \ \sqsubseteq \ x_i$$

Remarquons que si $\mathtt{a}$ est un point fixe, alors $x_1 = \mathtt{a}\Delta\varphi(\mathtt{a}) = \mathtt{a}\Delta\mathtt{a} = \mathtt{a}$ et l'itération est directement stationnaire. Autrement dit, si l'on part d'un point fixe $\mathtt{a}$, l'opération de rétrécissement n'a pas d'effet.

Cette méthode d'élargissement/rétrécissement peut donc se résumer comme suit. Afin de calculer l'approximation d'un plus petit point fixe d'une fonction φ, on effectue des calculs dans le domaine des pré-points fixes de φ. Ces calculs sont spécifiés de telle sorte à produire, en un nombre fini d'étapes un post-point fixe de φ et donc une sur-approximation de $\mathtt{lfp}(\varphi)$. Il peut alors s'ensuivre un processus permettant d'affiner ce premier calcul. Cette deuxième étape est réalisée par une descente finie dans le domaine des post-points fixes. Son résultat fournit une sur-approximation de $\mathtt{lfp}(\varphi)$ meilleure que la précédente. La Figure 2.7 illustre cette explication.

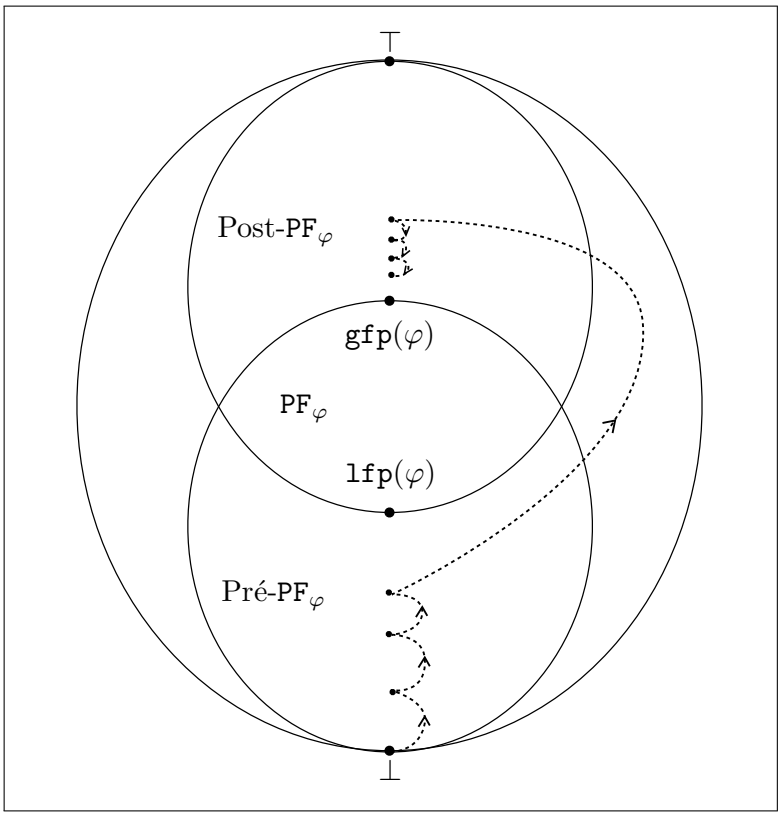


FIGURE 2.7 – Itération croissante par élargissement suivie d'un processus d'affinement par rétrécissement

Étude d'un exemple : l'analyse polyédrique [CH78]
On verra par la suite que dans l'analyse polyédrique entre en jeu un calcul de plus petit point fixe. Or, le treillis $\mathtt{P}_m$ des polyèdres convexes ne vérifie pas la condition de chaîne ascendante. Un opérateur d'élargissement ∇ est donc utilisé pour permettre le calcul de ce point fixe en un nombre fini d'itérations. Étant donné p_1 et p_2 deux polyèdres de $\mathtt{P}_m$, l'ensemble $p_1 \nabla p_2$ représente le polyèdre convexe défini par les inégalités de p_1 vérifiées par tous les points du polyèdre p_2. Cet opérateur d'élargissement agit donc sur p_1 en diminuant son nombre d'inégalités. Tout polyèdre étant défini par un nombre fini d'inégalités, ce processus termine en un nombre fini d'itérations.

2.2 Syntaxe et sémantique des programmes

Dans cette section, nous définissons la notion de programme de manière très générique de sorte à pouvoir montrer comment est réalisée une analyse par inter-

prétation abstraite dans le cas le plus général possible.

2.2.1 Syntaxe des programmes

En introduction, nous avons présenté un programme comme une suite finie de commandes. En fait, on peut définir un programme de manière encore plus large : on représente un programme par un graphe orienté et étiqueté par les commandes du programme, qui ne sont autres que des fonctions de tests ou d'affectations. Les nœuds de ce graphe seront appelés les points de programme.

Définition 2.9 (Notion de programme [CC79]).
Un programme est un triplet $\pi = (\mathcal{U}, \mathcal{G}, L)$ tel que :

- *$\mathcal{U}$ est un ensemble appelé univers : c'est l'espace des valeurs que peuvent prendre les variables du programme,*
- *$\mathcal{G}$ est un graphe. Plus précisément, c'est un quintuplet $\mathcal{G} = (n, \mathcal{E}, n_e, n_s, C)$ tel que :*
  - *$[1, n]$ est l'ensemble des nœuds ou sommets du graphe. Parmi l'ensemble des sommets, on distingue $n_e \in [1, n]$ le point d'entrée du graphe et $n_s \in [1, n]$ le point de sortie,*
  - *$\mathcal{E} \subseteq [1, n]^2$ est l'ensemble des arcs du graphe,*
  - *$C : \mathcal{E} \to L$ est une fonction qui à chaque arc du programme associe une commande.*
- *L est l'ensemble des commandes qui étiquettent le graphe. Il est composé des ensembles L_t et L_a ($L = L_t \cup L_a$) :*
  - *$L_t \subseteq \mathcal{B}^{\mathcal{U}}$ est l'ensemble des tests. Ce sont des fonctions de $\mathcal{U}$ dans l'ensemble des booléens $\mathcal{B}$,*
  - *$L_a \subseteq \mathcal{U}^{\mathcal{U}}$ est l'ensemble des affectations. Ce sont des fonctions de $\mathcal{U}$ dans lui-même.*

On notera de plus $succ_{\mathcal{E}} : [1, n] \to 2^{[1,n]}$ la fonction qui a tout nœud i du graphe associe l'ensemble de ses nœuds successeurs ($succ_{\mathcal{E}}(i) = \{j \in [1, n] \mid (i, j) \in \mathcal{E}\}$). On définit de même la fonction prédécesseur notée $pred_{\mathcal{E}}$.
Enfin, on fait l'hypothèse que $pred_{\mathcal{E}}(n_e) = \emptyset$ et $succ_{\mathcal{E}}(n_s) = \emptyset$.

Il est important de noter ici que cette représentation autorise la présence de boucles. D'autre part, parmi l'ensemble des nœuds, on distingue ceux qui possèdent plus d'un prédécesseur. Ces nœuds particuliers seront appelés nœuds ou *points de jonction*. Notons enfin que les fonctions d'affectations et de tests ne sont pas forcément définies sur $\mathcal{U}$ tout entier. Chacune de ces fonctions n'est donc totale que si nous la restreignons à son domaine de définition.

Étude d'un exemple : l'analyse polyédrique [CH78]
Comme décrit dans la Section 2.1, l'analyse polyédrique étudie des programmes

à m variables réelles. Ainsi, on a : $\mathcal{U} = \mathbb{R}^m$. L'ensemble des étiquettes L est composé des fonctions d'affectations et des fonctions de test. Parmi ces commandes, on distingue celles qui sont linéaires en les variables du programme. Celles-ci seront analysées de manière plus précise que les commandes non linéaires qui ne sont pas adaptées au domaine des polyèdres convexes $\mathtt{P}_m$ choisi pour l'analyse.

Maintenant que nous avons décrit syntaxiquement la notion de programme, nous présentons la notion de sémantique d'un programme *c.à.d.* le processus qui consiste à donner une signification à ce programme.

2.2.2 Sémantique concrète des programmes

Par sémantique concrète, on entend sémantique des états accessibles. Plus précisément, on appellera sémantique concrète d'un programme le processus qui permet d'obtenir l'ensemble des états accessibles à chaque point de programme. Pour ce faire, il faut déjà donner un interprétation en termes d'accessibilité de chaque commande (arc) du programme. Autrement dit, on définit la *sémantique des commandes*. Ensuite, afin d'obtenir les états accessibles à chaque point de programme, il faut aussi expliquer comment combiner les interprétations des commandes de sorte à obtenir la sémantique du programme dans son entier.

2.2.2.1 Le domaine concret

À chaque point d'un programme π, les états possibles de π sont donnés par un ensemble de points de $\mathcal{U}$, appelé assertion concrète.

Définition 2.10 (Notion d'assertion concrète)**.**
Une assertion concrète est une fonction totale de $\mathcal{U}$ dans $\mathcal{B}$.
L'ensemble des assertions est noté $\mathcal{A}$. Il est muni d'une structure de treillis complet $\mathcal{A}(\Rightarrow, fFalse, fTrue, \vee, \wedge)$ où :

- *l'implication $\Rightarrow$ est étendue aux fonctions de la manière classique : on notera $f_1 \Rightarrow f_2$ si $\forall x \in \mathcal{U},\ f_1(x) \Rightarrow f_2(x)$,*
- *$fTrue$ est la fonction telle que : $\forall x \in \mathcal{U}, fTrue(x) = True$,*
- *$fFalse$ est la fonction telle que : $\forall x \in \mathcal{U}, fFalse(x) = False$.*

On utilise dans cette définition des prédicats, *c.à.d.* des fonctions à valeurs dans $\mathcal{B}$ ou, autrement dit, des ensembles de points de $\mathcal{U}$. On aurait donc tout aussi bien pu définir le domaine des assertions concrètes comme étant $\mathcal{P}(\mathcal{U})$.

Étude d'un exemple : l'analyse polyédrique [CH78]
Dans le cas de l'analyse polyédrique, le domaine des assertions concrètes est l'en-

semble $\mathcal{P}(\mathbb{R}^m)$.

2.2.2.2 Sémantique concrète des commandes

On définit maintenant la sémantique concrète des commandes étiquetant un programme. Comme on va le voir, celle-ci peut se faire en avant ou en arrière.

Sémantique concrète avant des commandes
La sémantique concrète avant de la commande $C(i, j)$ permet, connaissant l'assertion vérifiée au point i, de définir l'assertion vérifiée au point j. On donne ci-dessous une interprétation avant particulière, celle qui consiste à déterminer la plus grande assertion vérifiée au point j. On parlera alors de calcul de plus forte post-condition [Flo67].

Définition 2.11 (Sémantique concrète avant des commandes)**.**
La sémantique concrète avant des commandes *est fournie par la fonction suivante.*

$$\begin{array}{rcccccc} t_{av} & : & L & \rightarrow & (\ \mathcal{A} & \rightarrow & \mathcal{A} \ ) \\ & & q & \mapsto & (\ \varphi & \mapsto & t_{av}(q)(\varphi) \ ) \end{array}$$

où $t_{av}(q)(\varphi)$ est défini selon le type de q.
Si q est une fonction de test.

$$\begin{array}{rcccl} t_{av}(q)(\varphi) & : & \mathcal{U} & \rightarrow & \mathcal{B} \\ & & X & \mapsto & \varphi(X) \ \wedge \ X \in dom(q) \ \wedge \ q(X) \end{array}$$

Si q est une fonction d'affectation.

$$\begin{array}{rcccl} t_{av}(q)(\varphi) & : & \mathcal{U} & \rightarrow & \mathcal{B} \\ & & X & \mapsto & \exists Y \in \mathcal{U}, \ \varphi(Y) \ \wedge \ Y \in dom(q) \ \wedge \ X = q(Y) \end{array}$$

Commentons cette définition. Si q est une fonction de test, $t_{av}(q)(\varphi)$ représente l'ensemble obtenu par intersection de l'ensemble des états vérifiant l'assertion précédente φ et de l'ensemble des états qui vérifient le test q. Si q est une affectation, $t_{av}(q)(\varphi)$ est l'ensemble des états qui possèdent un antécédent par q dans l'ensemble φ. Ce que l'on peut noter, en utilisant la fonction réciproque q^{-1}, par l'ensemble $\{ \ X \in \mathcal{U} \mid True \ \in \ \varphi(q^{-1}(\{X\}))\}$. Cette notation fait apparaître une difficulté : la fonction q^{-1} n'est pas forcément facilement calculable. Évidemment, le cas où q est une bijection (dont l'inverse est facilement accessible) est un cas très favorable pour l'analyse avant. Dans ce cas, la fonction t_{av} est directement définie par $t_{av}(q)(\varphi) = \varphi \circ q^{-1}$ (q^{-1} désigne ici la fonction inverse de q).

Étude d'un exemple : l'analyse polyédrique [CH78]
L'analyse polyédrique admet l'interprétation concrète avant définie par la fonction t_{av} suivante :

$$\begin{array}{rcccccc} t_{av} & : & L & \rightarrow & (\ \mathcal{P}(\mathbb{R}^m) & \rightarrow & \mathcal{P}(\mathbb{R}^m) \ ) \\ & & q & \mapsto & (\ S & \mapsto & t_{av}(q)(S) \ ) \end{array}$$

où $t_{av}(q)(S)$ est défini selon le type de q. Illustrons t_{av} sur deux commandes simples : une affectation $q_1 \equiv (\mathtt{x_1} := \mathtt{x_1} + 4)$ et un test $q_2 \equiv (\mathtt{x_1} \geq 0)$. L'affectation est traitée de la manière suivante :

$$\begin{array}{rcl} t_{av}(q_1)(S) & = & S[^{\mathtt{x_1}+4}/_{\mathtt{x_1}}] \\ & = & \{ \ (x_1 + 4, x_2, \ldots, x_m) \in \mathbb{R}^m \mid (x_1, x_2, \ldots, x_m) \in S \ \} \\ & = & \{ \ (x_1, x_2, \ldots, x_m) \mid (x_1 - 4, x_2, \ldots, x_m) \in S \ \} \end{array}$$

La dernière égalité illustre la discussion dans le cas où l'affection q_1 est inversible. La commmande de test est traitée comme suit.

$$\begin{array}{rcl} t_{av}(q_2)(S) & = & S \ \cap \ (\mathtt{x_1} \geq 0) \\ & = & \{ \ (x_1, \ldots, x_m) \in \mathbb{R}^m \mid (x_1, \ldots, x_m) \in S \ \wedge \ x_1 \geq 0 \ \} \end{array}$$

Sémantique concrète arrière des commandes
On peut aussi proposer d'interpréter les commandes en arrière. La sémantique arrière de la commande $C(i, j)$ permet, connaissant l'assertion $\mathtt{a}_j$ vérifiée au point j, de définir l'assertion $\mathtt{a}_i$ vérifiée au point i. Plus précisément, on va déterminer la plus faible pré-condition [Hoa69, Dij76] de validité *c.à.d.* le plus petit ensemble $\mathtt{a}_i$ tel que $\mathtt{a}_j$ est vérifiée.

Définition 2.12 (Sémantique concrète arrière des commandes)**.**
La sémantique concrète arrière des commandes *est fournie par la fonction suivante.*

$$\begin{array}{rcccccc} t_{ar} & : & L & \rightarrow & (\ \mathcal{A} & \rightarrow & \mathcal{A} \ ) \\ & & q & \mapsto & (\ \varphi & \mapsto & t_{ar}(q)(\varphi) \ ) \end{array}$$

où $t_{ar}(q)(\varphi)$ est défini selon le type de q.
Si q est une fonction de test.

$$\begin{array}{rcccl} t_{ar}(q)(\varphi) & : & \mathcal{U} & \rightarrow & \mathcal{B} \\ & & X & \mapsto & \varphi(X) \ \wedge \ X \in dom(q) \ \wedge \ q(X) \end{array}$$

Si q est une fonction d'affectation.

$$\begin{array}{rcccl} t_{ar}(q)(\varphi) & : & \mathcal{U} & \rightarrow & \mathcal{B} \\ & & X & \mapsto & X \in dom(q) \ \wedge \ \varphi(q(X)) \end{array}$$

Commentons cette définition. Si q est une fonction de test, $t_{ar}(q)(\varphi)$ représente l'ensemble obtenu par intersection de l'ensemble des états vérifiant l'assertion φ et de l'ensemble des états qui vérifient le test q. Si q est une affectation, $t_{ar}(q)(\varphi)$ est l'ensemble des états dont l'image par q est dans l'ensemble φ. On remarque, par comparaison avec l'analyse avant, que la définition de t_{ar} n'utilise pas la fonction réciproque q^{-1}.

Nous allons maintenant étudier la manière de combiner les interprétations des commandes afin d'obtenir la sémantique du programme en son entier.

2.2.2.3 Sémantique concrète : approche MOP et approche par plus petit point fixe

Nous avons donné, en section précédente, des instances de la fonction $t : L \to \mathcal{A} \to \mathcal{A}$ qui donne la sémantique des commandes. Il s'agit maintenant de déterminer comment appliquer cette fonction sur les arcs du programme. Nous présentons dans ce qui suit deux processus qui permettent d'obtenir la sémantique d'un programme en son entier.

Sémantique concrète *Merge Over all Paths* (MOP)
La sémantique concrète *Merge Over all Paths* (MOP) permet de déterminer les assertions vérifiées à chaque point de programme en se basant sur la notion de chemin. Plus précisément, l'assertion $\mathtt{a}_i$ vérifiée au point de programme i est définie à l'aide de la sémantique des commandes étiquetant les arcs du chemin menant du point de programme initial n_e au point i. S'il existe plusieurs chemins de n_e à i, il faudra combiner les sémantiques des commandes de chaque chemin. Vu que nous nous intéressons aux états accessibles, $\mathtt{a}_i$ sera alors défini à partir de l'union des interprétations des chemins partant de n_e et menant à i. La définition suivante formalise ce processus.

Définition 2.13 (Sémantique concrète MOP)**.**
Soit π un programme et $\Phi \in \mathcal{A}$ une assertion initiale.
Soit $t : L \to (\mathcal{A} \to \mathcal{A})$ une fonction codant le comportement de π.
La sémantique concrète merge over all paths $\mathtt{MOP}_\pi(t, \Phi)$ *définit l'assertion $\mathtt{a}_i$ vérifiée à chaque point de programme i de la manière suivante :*

$$\forall i \in [1, n],\ \mathtt{a}_i = \bigcup_{p \in \mathtt{chemin}(i)} \tilde{t}(p)(\Phi)$$

où $\mathtt{chemin}(i)$ représente l'ensemble des chemins menant du point d'entrée n_e au point i et t est étendue de manière récursive aux chemins de chemin *:*

$$\begin{array}{ll} \textit{si } p = (i,j) \in \mathcal{E}, & \tilde{t}(p)(\Phi) = t(C(i,j))(\Phi) \\ \textit{si } p = q.(i,j) \in \mathcal{E}^*, & \tilde{t}(p)(\Phi) = t(C(i,j))[\tilde{t}(q)(\Phi)] \end{array}$$

La sémantique concrète MOP semble être destinée à opérer en avant, en utilisant la fonction t_{av}. Cependant, elle peut aussi être utilisée pour une analyse arrière.

Sémantique MOP concrète avant d'un programme π
Étant donnée une assertion initiale Φ, cette sémantique est formellement définie comme le résultat de l'analyse $\mathtt{MOP}_\pi(t_{av}, \Phi, \mathcal{A})$. Elle permet de définir, à chaque point de programme, l'ensemble des états accessibles.

Sémantique MOP concrète arrière d'un programme π
La sémantique MOP concrète arrière se déroule, comme son nom l'indique, en arrière. Cette sémantique n'est donc pas définie sur π mais sur son **programme miroir** π' obtenu en renversant le sens des arcs de $\mathcal{G}$: $\mathcal{G}' = (n, \mathcal{E}', n_s, n_e, C')$ avec $\mathcal{E}' = \{(i,j) \mid (j,i) \in \mathcal{E}\}$ et $C'(i,j) = C(j,i)$ pour tout $(i,j) \in \mathcal{E}'$. Partant de l'assertion finale Φ, la sémantique $\mathtt{MOP}_{\pi'}(t_{ar}, \Phi, \mathcal{A})$ permet de définir, à chaque point de programme, la plus petite assertion qui assure que le programme termine dans un état vérifiant Φ.

On peut alors se poser la question de la calculabilité d'un telle sémantique. La définition de $\mathtt{a}_i$ dépend de l'ensemble $\mathtt{chemin}(i)$, contenant les chemins menant à i. On peut partitionner cet ensemble en groupant les chemins de même longueur : $\mathtt{chemin}(i) = \cup_{k\geq 0} \{p \mid p \in \mathtt{chemin}(i),\ |p| = k\}$. On pourrait alors proposer une procédure incrémentale pour calculer $\mathtt{a}_i$ en calculant successivement le résultat de t sur chacun de ces sous-ensembles. On définit ainsi l'ensemble $\mathtt{a}_i$ comme la limite d'une suite croissante : le k-ième élément $\mathtt{a}_i^k$ de la suite est le calcul de t sur les chemins de longueur au plus k. Remarquons maintenant que, du fait de la possible présence de boucles dans le programme, il peut exister des chemins arbitrairement grands. Ainsi, cette procédure est, en général, non terminante. Dans les cas où la terminaison n'est pas assurée, il est possible de proposer le calcul d'une procédure plus simple, obtenue par approximations de la suite intiale. Plus précisément, la suite $(\mathtt{a}_i^k)_{k\in\mathbb{N}}$ est approchée par une suite $(\tilde{\mathtt{a}}_i^k)_{k\in\mathbb{N}}$ définie dans un treillis $\mathcal{D}$ moins précis que celui des assertions $\mathcal{A}$. Intuitivement, plus le treillis $\mathcal{D}$ sera défini de manière grossière, plus la procédure approchée aura de chances de terminer rapidement. Au contraire, plus ce treillis $\mathcal{D}$ sera fin et précis, plus il y a de risques que la procédure approchée ne termine pas ou termine mais en un temps de calcul jugé non raisonnable. Cette idée simple est utilisée pour effectuer des analyses approchées et est développée rigoureusement en Section 2.3. Nous n'entrons pas plus dans les détails ici et définissons simplement la possibilité d'effectuer une analyse MOP pour tout domaine $\mathcal{D}$ qui est un treillis complet.

Définition 2.14 (Analyse MOP générale)**.**
Soit $\mathcal{D}(\sqsubseteq, \bot, \top, \bigsqcup, \bigsqcap)$ un domaine d'assertion muni d'une structure de treillis

complet. Soit π un programme et $\Phi \in \mathcal{D}$ une assertion initiale.
Soit $t : L \to (\mathcal{D} \to \mathcal{D})$ une fonction codant le comportement de π.
L'analyse merge over all paths $\mathtt{MOP}_\pi(t, \Phi, \mathcal{D})$ *définit l'assertion vérifiée à chaque point de programme de la manière suivante :*

$$\forall i \in [1, n],\ \mathtt{a}_i = \bigsqcup_{p \in \mathtt{chemin}(i)} \tilde{t}(p)(\Phi)$$

Sémantique concrète par point fixe
La sémantique des programmes peut aussi être définie par calcul de point fixe dans le domaine des assertions. Décrivons ce processus. L'assertion $\mathtt{a}_j$ vérifiée à un point de programme j est définie à l'aide des assertions vérifiées en les points i qui sont des prédécesseurs directs de j. Étant donné que la syntaxe de nos programmes autorise la présence de boucles, on définit de cette manière un système d'équations récursives. La sémantique du programme est alors déterminée par résolution de ce système d'équations. La définition suivante formalise le processus d'obtention du système d'équations dans le cas le plus général *c.à.d.* pour un treillis complet quelconque et pas seulement sur le domaine $\mathcal{A}$ des assertions concrètes.

Définition 2.15 (Système d'équations associé à un programme)**.**
Soit $\mathcal{D}(\sqsubseteq, \bot, \top, \bigsqcup, \bigsqcap)$ un domaine d'assertions muni d'une structure de treillis complet. Soit π un programme et $\Phi \in \mathcal{D}$ une assertion initiale.
Soit $t : L \to (\mathcal{D} \to \mathcal{D})$ une fonction codant le comportement de π.
Le système d'équations $\mathtt{a} = F_\pi(t, \Phi, \mathcal{D})(\mathtt{a})$ associé au programme π est défini par :

$$\begin{cases} \mathtt{a}_{n_e} = \Phi & \textit{pour } n_e \textit{ nœud d'entrée} \\ \mathtt{a}_j = \bigsqcup_{i \in pred_{\mathcal{E}}(j)} t(C(i,j))(\mathtt{a}_i) & \textit{pour } j \neq n_e \textit{ et } j \in [1, n] \end{cases}$$

Notons que, comme dans le cas de la sémantique concrète MOP, cette sémantique peut être instanciée à l'aide de la sémantique des commandes avant t_{av} ou la sémantique des commandes arrières t_{ar}.

Sémantique concrète avant par plus petit point fixe du programme π
Étant donnée une assertion initiale Φ, cette sémantique est fomellement définie comme le plus petit point fixe du système d'équations $\mathtt{a} = F_\pi(t_{av}, \Phi, \mathcal{A})(a)$. Elle permet de définir, à chaque point de programme, l'ensemble des états accessibles. En fait, cette sémantique correspond à la sémantique $\mathtt{MOP}_\pi(t_{av}, \Phi, \mathcal{A})$ précédente.

Sémantique concrète arrière par plus petit et par plus grand point fixe du programme π
La sémantique concrète arrière est formellement définie comme point fixe du système d'équations $\mathtt{a} = F_{\pi'}(t_{ar}, \Phi, \mathcal{A})(\mathtt{a})$, où Φ est l'assertion finale du programme π et π' est le programme miroir de π. Nous pouvons définir deux sémantiques différentes : celle par plus petit point fixe et celle par plus grand point fixe.

La sémantique par plus petit point fixe correspond à une notion de *correction totale* : l'assertion $\mathtt{a}_{n_e}$ calculée est la plus petite assertion qui assure que le programme termine dans un état vérifiant Φ. Cette interprétation correspond à la notion de plus faible pré-condition de la logique de Hoare [Hoa69, Dij76] et correspond, de ce fait à la sémantique $\mathtt{MOP}_{\pi'}(t_{ar}, \Phi, \mathcal{A})$.

La sémantique par plus grand point fixe correspond à une notion de *correction partielle* : l'assertion $\mathtt{a}_{n_e}$ calculée est la plus petite assertion qui assure que si le programme termine alors il est dans un état vérifiant Φ [Cou81]. Autrement dit, $\mathtt{a}_{n_e}$ représente le plus petit ensemble d'états initiaux pour lequel l'éxécution du programme π ne termine pas ou mène à un état dans Φ. Cette interprétation correspond à la notion de plus faible pré-condition libérale de la logique de Hoare [Hoa69, Dij76].

Cette définition de la sémantique concrète à l'aide d'un plus petit ou plus grand point fixe dans le treillis des assertions concrètes est à relier à la Section 2.1. Dans cette précédente section, nous avons vu que, sous les hypothèses du théorème 2.2, le plus grand et plus petit point fixe d'une fonction agissant sur un treillis pouvait être décrit de manière itérative. Insistons sur le fait que cette procédure itérative n'est pas forcément terminante et ne permet donc pas, à elle seule, de régler les problèmes de calculabilité de la sémantique concrète.

Il convient maintenant de comparer l'approche `MOP` et l'approche par point fixe qui nous ont permis de définir la sémantique concrète des programmes. Plus précisément, nous nous intéressons aux hypothèses permettant à ces deux approches de coïncider.

Comparaison des approches `MOP` et plus petit point fixe
Le théorème suivant définit un cadre d'égalité des approches `MOP` et par plus petit point fixe.

Théorème 2.5 ([CC79])**.**
Soit $\mathcal{D}(\sqsubseteq, \bot, \top, \bigsqcup, \bigsqcap)$ un treillis complet.
Soit $t : L \to \mathcal{D} \to \mathcal{D}$ tel que $\forall q \in L$, $t(q)$ est une fonction monotone. Alors :

$$\forall \pi,\ \forall \Phi \in \mathcal{D}, \quad \mathtt{MOP}_{\pi}(t, \Phi) \quad \sqsubseteq \quad \mathtt{lfp}(F_{\pi}(t, \Phi))$$

Si de plus, pour tout $q \in L$, $t(q)$ est une fonction $\bigsqcup$-`mcs`*, alors :*

$$\forall \pi,\ \forall \Phi \in \mathcal{D}, \quad \texttt{MOP}_\pi(t, \Phi) \quad = \quad \texttt{lfp}(F_\pi(t, \Phi))$$

L'analyse `MOP` et celle par plus petit point fixe ont toutes les deux leur intérêt. L'analyse `MOP` peut se voir comme une définition formelle. Elle nous semble plus naturelle pour ce qui est de donner du sens au programme. L'analyse par plus petit point fixe, quant à elle, peut être vue comme une alternative pratique. Elle est notamment calculable dès que la condition de chaîne ascendante est vérifiée dans le treillis des assertions.

2.2.2.4 Une autre (?) définition de la sémantique

À la place de l'analyse `MOP`, nous aurions pu opter pour une sémantique *meet over all paths* comme étudié, par exemple, par Kam & Ullman [KU77].

Définition 2.16 (Sémantique `MeetOP` générale)**.**
Soit $\mathcal{D}(\sqsubseteq, \bot, \top, \bigsqcup, \bigsqcap)$ un domaine d'assertions muni d'une structure de treillis complet. Soit π un programme et $\Phi \in \mathcal{D}$ une assertion initiale.
Soit $t : L \to (\mathcal{D} \to \mathcal{D})$ une fonction codant le comportement de π.
La sémantique meet over all paths $\texttt{MeetOP}_\pi(t, \Phi, \mathcal{D})$ *définit l'assertion vérifiée à chaque point de programme de la manière suivante :*

$$\forall i \in [1, n],\ \mathtt{a}_i = \bigsqcap_{p \in \texttt{chemin}(i)} \tilde{t}(p)(\Phi)$$

L'instanciation concrète avant de cette sémantique permet de définir la plus grande assertion qui est vérifiée du point d'entrée du programme au point actuel. Ce genre d'informations revêt de l'importance car elles peuvent permettent de réaliser des optimisations pour la phase de compilation [KU77]. Comme pour la sémantique `MOP` précédente, nous pouvons montrer que l'analyse `MeetOP` peut s'exprimer sous forme de point fixe d'un système d'équations. Plus précisément, cette analyse coïncide avec le plus grand point fixe du système d'équations suivant dès que la sémantique des commandes t est $\bigsqcap$-`mcs`.

Définition 2.17 (Système (bis) d'équations associé à un programme)**.**
Soit $\mathcal{D}(\sqsubseteq, \bot, \top, \bigsqcup, \bigsqcap)$ un treillis complet.
Soit π un programme et $\Phi \in \mathcal{D}$ une assertion initiale.
Soit $t : L \to (\mathcal{D} \to \mathcal{D})$ une fonction codant le comportement de π.
Le système (bis) d'équations $\mathtt{a} = G_\pi(t, \Phi, \mathcal{D})(\mathtt{a})$ associé au programme π est défini par :

$$\begin{cases} \mathtt{a}_{n_e} & = & \Phi & \textit{pour } n_e \textit{ nœud d'entrée} \\ \mathtt{a}_j & = & \displaystyle\bigsqcap_{i \in pred_{\mathcal{E}}(j)} t(C(i,j))(\mathtt{a}_i) & \textit{pour } j \neq n_e \textit{ et } j \in [1, n] \end{cases}$$

Bien qu'ayant un intérêt pratique, l'analyse `MeetOP` ne présente, en fait, pas de nouveauté théorique. En effet, d'un point de vue treillis, cette analyse correspond juste à une analyse `MOP` pour laquelle nous aurions renversé le treillis initial. Plus précisément, l'analyse $\texttt{MOP}_\pi(t, \Phi, \mathcal{D})$ correspond à l'analyse $\texttt{MeetOP}_\pi(t, \Phi, \tilde{\mathcal{D}})$ pour $\tilde{\mathcal{D}}$ défini comme $\mathcal{D}(\sqsupseteq, \top, \bot, \bigsqcap, \bigsqcup)$. Tous les résultats obtenus pour `MOP` sont donc transposables à l'analyse `MeetOP` via cette correspondance. Étant donnée cette correspondance, nous considèrerons uniquement l'analyse `MOP` dans ce qui suit.

Pour conclure cette Section 2.2, notons que les définitions des sémantiques concrètes ci-dessus ne fournissent pas directement d'algorithme permettant de les calculer. Ces sémantiques, très précises, peuvent traiter d'assertions représentant des ensembles infinis de valeurs et sont définies, via l'analyse `MOP`, à l'aide de chemins pouvant être arbitrairement grands. Ceci pose très clairement des problèmes de calculabilité : problèmes de représentation en mémoire et de finitude de temps de calcul. Afin de pouvoir quand même calculer des assertions *c.à.d.* des propriétés décrivant le programme, nous allons réaliser une analyse approchée de programme. La section suivante décrit ce processus.

2.3 Analyse approchée des programmes

Le but d'une analyse par interprétation abstraite est d'inférer des propriétés particulières de programmes. Ces propriétés, que l'on a appelées assertions, sont exprimées par la sémantique concrète, qui est non calculable en général. Nous avons vu, dans la section précédente, que cette sémantique pouvait se définir comme point fixe d'une fonction dans un treillis. Nous avons aussi vu que ce point fixe pouvait se définir de manière itérative comme limite d'une suite croissante. Le défaut de calculabilité de la sémantique concrète s'exprime alors par le fait que cette suite ne converge pas en temps fini. Ceci ne peut se produire dans les treillis vérifiant la condition de chaîne ascendante, puisque cette propriété exprime le fait que toute chaîne croissante est stationnaire. L'idée derrière une analyse par interprétation abstraite est de transporter cette procédure itérative, définissant la sémantique concrète, dans un monde approché, plus grossier, appelé ensemble des assertions abstraites $\mathcal{A}^\sharp$, où la calculabilité pourra être assurée. La question qui se pose alors est celle du choix du monde abstrait. Il faut réussir à concilier au mieux deux exigences antagonistes : l'expressivité et l'efficacité. Plus précisément, plus le monde abstrait choisi est simple, moins il permet d'exprimer de propriétés intéressantes mais plus les calculs itératifs sont rapides. À l'inverse, un monde abstrait plus complexe permet d'exprimer plus de propriétés mais augmente sensiblement le temps nécessaire au calcul de ces propriétés. Afin que les calculs dans le domaine des assertions abstraites $\mathcal{A}^\sharp$ possèdent du sens par rapport aux

assertions concrètes que l'on souhaitait initialement calculer, il reste alors à définir comment les deux domaines $\mathcal{A}$ et $\mathcal{A}^\sharp$ sont reliés. Autrement dit, il s'agit de définir formellement ce que l'on entend par le terme « approcher ». Le but de cette section est de détailler l'ensemble des aspects présentés dans ce paragraphe.

2.3.1 Structure du domaine abstrait et relation avec le domaine concret : un premier aperçu

Nous commençons par décrire sommairement quelques propriétés que nous exigeons sur la structure des assertions abstraites $\mathcal{A}^\sharp$ et la relation entre domaine concret $\mathcal{A}$ et abstrait $\mathcal{A}^\sharp$.

2.3.1.1 Structure de l'ensemble des assertions abstraites

On a vu en Section 2.2 que le domaine des assertions concrètes était muni d'une structure de treillis $\mathcal{A}(\sqsubseteq, \bot, \top, \bigsqcup, \bigsqcap)$. Ceci n'est pas anodin. Grâce à l'ordre défini par $\sqsubseteq$, on peut comparer les assertions concrètes. Grâce à la borne inférieure $\bigsqcap$, on peut définir la plus grande assertion vérifiant deux autres assertions et grâce à la borne supérieure $\bigsqcup$ la plus petite assertion impliquée par deux autres assertions. Par analogie avec le domaine concret, il semble naturel de munir le domaine abtrait d'une structure de treillis, afin de pouvoir effectuer les mêmes manipulations sur les assertions de ce monde. Par la suite, le domaine $\mathcal{A}^\sharp$ ainsi que toutes ces composantes seront différenciés du domaine $\mathcal{A}$ à l'aide du symbole « $\sharp$ » et on fera l'hypothèse suivante.

> Nous considèrons que le domaine des assertions approchées $\mathcal{A}^\sharp$ est muni d'une structure de treillis $\mathcal{A}^\sharp(\sqsubseteq^\sharp, \bot^\sharp, \top^\sharp, \bigsqcup^\sharp, \bigsqcap^\sharp)$ avec plus petit élément $\bot^\sharp$ et plus grand élément $\top^\sharp$.

2.3.1.2 Relation entre assertions concrètes et abstraites [CC92c]

Le minimum que l'on puisse exiger des ensembles $\mathcal{A}$ et $\mathcal{A}^\sharp$ est qu'ils soient reliés par une relation binaire $\sigma \subseteq \mathcal{A} \times \mathcal{A}^\sharp$. Cette relation se lit de la manière suivante : $(\mathtt{a}, \mathtt{a}^\sharp) \in \sigma$ signifie que $\mathtt{a}^\sharp$ est une approximation, dans le domaine abstrait $\mathcal{A}^\sharp$, de l'assertion concrète $\mathtt{a}$. Sous cette définition, un élément $\mathtt{a} \in \mathcal{A}$ peut être en relation, via σ, avec plusieurs éléments de l'ensemble $\mathcal{A}^\sharp$. Étudions de plus près $\overline{\mathtt{a}}$, l'ensemble des éléments approchant un élément $\mathtt{a}$: $\overline{\mathtt{a}} = \{\ \mathtt{a}^\sharp \in \mathcal{A}^\sharp \mid (\mathtt{a}, \mathtt{a}^\sharp) \in \sigma\}$. Plusieurs cas peuvent se produire :

1) l'ensemble $\overline{\mathtt{a}}$ est vide. Ce cas implique, par définition, que notre analyse ne peut construire de propriétés approchant $\mathtt{a}$. Par la suite, nous excluons

ce cas.

Nous supposons que l'hypothèse d'existence d'approximations abstraites est vérifiée : $\forall \mathtt{a} \in \mathcal{A},\ \exists \mathtt{a}^\sharp \in \mathcal{A}^\sharp,\quad (\mathtt{a}, \mathtt{a}^\sharp) \in \sigma$

2) l'ensemble $\overline{\mathtt{a}}$ possède un plus petit élément, noté $\alpha(\mathtt{a})$. Ce plus petit élément est, par définition, la meilleure sur-approximation de $\mathtt{a}$: tout autre élément de $\overline{\mathtt{a}}$ est plus grand que $\alpha(\mathtt{a})$ et est, de ce fait, une sur-approximation de $\mathtt{a}$ moins précise. Par la suite, nous dirons que l'hypothèse de meilleure approximation est vérifiée lorsque, pour tout élément $\mathtt{a} \in \mathcal{A}$, l'ensemble $\overline{\mathtt{a}}$ possède un plus petit élément. Cette hypothèse représente un cadre idéal en interprétation abstraite. À toute assertion concrète $\mathtt{a}$, on peut associer sa meilleure abstraction $\alpha(a)$. On définit ainsi une fonction d'approximation α.
3) l'ensemble $\overline{\mathtt{a}}$ ne possède pas de plus petit élément. C'est notamment le cas lorsque $\overline{\mathtt{a}}$ contient au moins une chaîne strictement décroissante d'élements ou lorsque $\overline{\mathtt{a}}$ contient des éléments non comparables. Notons au passage que ces deux situations peuvent se réaliser au sein du même ensemble $\overline{\mathtt{a}}$. L'hypothèse de meilleure approximation n'est pas vérifiée et il n'est pas possible de construire la fonction d'approximation α de la même manière que précédemment. Plusieurs stratégies peuvent être appliquées pour se ramener au cas où cette hypothèse est vérifiée. Notons toutefois que cette hypothèse, bien que fournissant un cadre agréable, n'est pas strictement nécessaire au développement d'une analyse par interprétation abstraite. Nous présenterons plus en détails ces aspects lors du retour sur l'exemple des polyèdres.

2.3.2 Déroulement d'une analyse par interprétation abstraite dans un cadre idéal

En Section 2.3.2, nous avons présenté la structure de treillis de l'ensemble des assertions abstraites $\mathcal{A}^\sharp$ et montré comment cet ensemble était relié à l'ensemble $\mathcal{A}$ via une relation de correspondance σ. Cette présentation initiale s'est faite dans un cadre assez large où nous avons fait peu d'hypothèses. Nous allons maintenant présenté le déroulement d'une analyse par interprétation abstraite dans un cadre idéal : les ensembles $\mathcal{A}$ et $\mathcal{A}^\sharp$ sont munis de la structure de treillis complet et la relation σ définit une connexion de Galois sur ces ensembles.

2.3.2.1 Connexions de Galois sur treillis complets

Une connexion de Galois est une paire de fonctions permettant de relier deux ensembles ordonnés [DP90]. Nous commençons donc par définir formellement la

notion de connexion de Galois sur des ensembles ordonnés qui ne sont pas forcément des treillis complets.

Connexions de Galois et ensembles ordonnés

Définition 2.18 (Connexion de Galois)**.**
Soit $L_1(\sqsubseteq_1)$ et $L_2(\sqsubseteq_2)$ deux ensembles partiellement ordonnés.
Soit $\alpha : L_1 \to L_2$ et $\gamma : L_1 \to L_2$ deux fonctions.
La paire de fonctions $<\alpha, \gamma>$ forme une connexion de Galois si la condition suivante est vérifiée :

$$\forall x_1 \in L_1, \forall x_2 \in L_2, \quad \alpha(x_1) \sqsubseteq_1 x_2 \;\Leftrightarrow\; x_1 \sqsubseteq_2 \gamma(x_2)$$

Une telle situation sera décrite par la notation $L_1 \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} L_2$.

À partir de cette définition concise de connexion de Galois, nous pouvons lister une série de propriétés vérifiées par de telles paires de fonctions.

Théorème 2.6 (Propriétés des connexions de Galois)**.**
Soient $L_1(\sqsubseteq_1)$ et $L_2(\sqsubseteq_2)$ deux ensembles partiellement ordonnés.
Soit $L_1 \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} L_2$ une connexion de Galois.
Alors :

(Gal1) *Les fonctions α et γ sont monotones :*

$$\forall x_1, x_2 \in L_1, \quad x_1 \sqsubseteq_1 x_2 \;\Rightarrow\; \alpha(x_1) \sqsubseteq_2 \alpha(x_2)$$

$$\forall y_1, y_2 \in L_1, \quad y_1 \sqsubseteq_2 y_2 \;\Rightarrow\; \gamma(y_1) \sqsubseteq_1 \gamma(y_2)$$

(Gal2) *Les fonctions $\gamma \circ \alpha : L_1 \to L_1$ et $\alpha \circ \gamma : L_2 \to L_2$ sont telles que :*

$$\forall x \in L_1, \quad \gamma \circ \alpha(x) \sqsupseteq_1 x$$

$$\forall y \in L_2, \quad \alpha \circ \gamma(y) \sqsubseteq_2 y$$

(Gal3) *Les fonctions $\alpha \circ \gamma \circ \alpha : L_1 \to L_2$ et $\gamma \circ \alpha \circ \gamma : L_2 \to L_1$ sont telles que :*

$$\alpha \circ \gamma \circ \alpha = \alpha$$

$$\gamma \circ \alpha \circ \gamma = \gamma$$

Réciproquement, toute paire de fonctions $<\alpha, \gamma>$ *vérifiant les propriétés* **(Gal1)** *et* **(Gal2)** *définit une connexion de Galois.*

Commentons les propriétés vérifiées par les connexions de Galois. La propriété **(Gal1)** énonce la monotonie des fonctions α et γ. Cette propriété exprime le transport de la structure d'ordre de l'ensemble L_1 vers L_2 et inversement le transport de la structure d'ordre de L_2 vers L_1. Il faut donc retenir que deux ensembles reliés par une connexion de Galois ont un structure d'ordre « proche ».

La propriété **(Gal2)** énonce le caractère extensif de la fonction $\gamma \circ \alpha$. Autrement dit, l'image par $\gamma \circ \alpha$ d'un élément $x \in L_1$ est une sur-approximation de l'élément x. L'idée derrière cette propriété est qu'il y a une perte d'information liée à l'abstraction de l'élément x mais que cette perte est contrôlée : l'élément x est approché par au-dessus. On parlera alors d'abstraction *correcte*. La Figure 2.8 illustre cette propriété sur un exemple simple. On considère comme domaine concret le treillis des parties de l'ensemble $\{-1, 0, 1\}$. On choisit d'abstraire ce domaine par le treillis des signes. La fonction d'abstraction α liant ces deux domaines agit comme suit : $\alpha(X) = \{0\}$ si $X = \{0\}$; $\alpha(X) = \geq 0$ si tous les éléments de X sont positifs ; $\alpha(X) = \leq 0$ si tous les éléments de X sont négatifs ; $\alpha(\emptyset) = \bot$; $\alpha(X) = \top$ dans les autres cas. La fonction γ associe à un signe le plus grand ensemble du domaine concret qui ne contient que des éléments de ce signe.

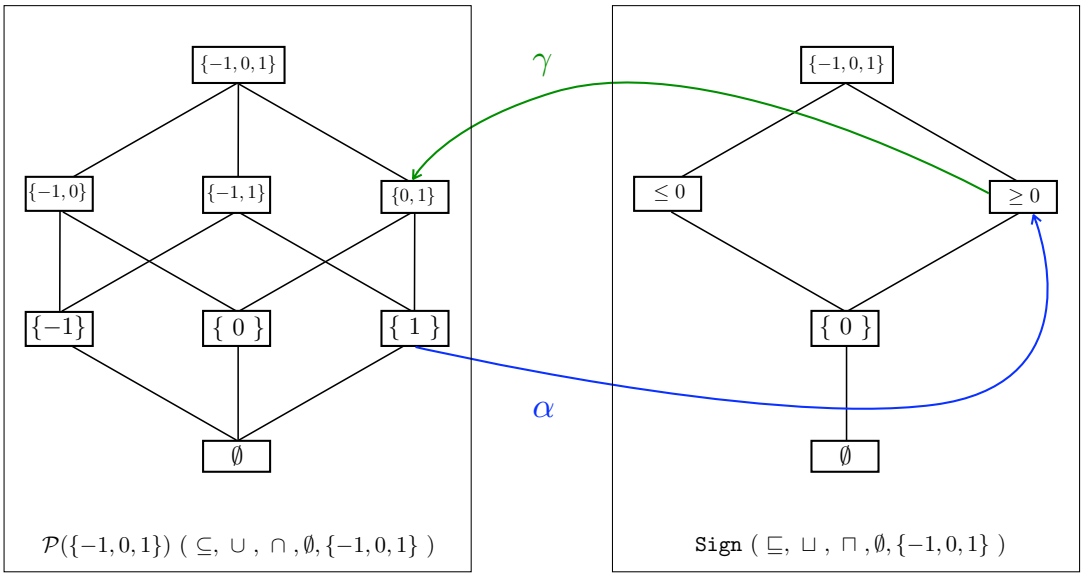


FIGURE 2.8 – Connexion de Galois reliant le treillis de parties $\mathcal{P}(\{-1, 0, 1\})$ et le treillis des signes

Nous inspectons maintenant le cas où les connexions de Galois sont définies sur des treillis complets.

Connexions de Galois et treillis complets
Dans le cas où les ensembles L_1 et L_2 sont des treillis complets, une connexion de Galois $L_1 \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} L_2$ vérifie les propriétés supplémentaires suivantes.

Théorème 2.7 (Propriétés des connexions de Galois sur treillis complets)**.**
Soient $L_1(\sqsubseteq_1, \bot_1, \top_1, \bigsqcup_1, \bigsqcap_1)$ *et* $L_2(\sqsubseteq_2, \bot_2, \top_2, \bigsqcup_2, \bigsqcap_2)$ *deux treillis complets.*
Soit $L_1 \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} L_2$ *une connexion de Galois.*
Alors :

(Gal4) *La fonction* α *est* $\bigsqcup$`-mcs` *:*

$$\forall S \subseteq L_1, \quad \alpha(\bigsqcup_1 S) \;=\; \bigsqcup_2 \alpha(S) \quad \textit{et} \quad \alpha(\bot_1) = \bot_2$$

La fonction γ *est* $\bigsqcap$`-mcs` *:*

$$\forall S \subseteq L_2, \quad \gamma(\bigsqcap_2 S) \;=\; \bigsqcap_1 \gamma(S) \quad \textit{et} \quad \gamma(\top_2) = \top_1$$

(Gal5) *La donnée d'une fonction de la paire* $<\alpha, \gamma>$ *permet de définir l'autre :*

$$\begin{aligned} \forall x \in L_1, \quad \alpha(x) &= \bigsqcap_2 \{y \in L_2 \mid x \sqsubseteq_1 \gamma(y)\} \\ \forall y \in L_2, \quad \gamma(y) &= \bigsqcup_1 \{x \in L_1 \mid \alpha(x) \sqsubseteq_2 y\} \end{aligned}$$

La propriété **(Gal4)** stipule que α et γ sont des morphismes d'union complets stricts. Cette propriété est, comme la propriété **(Gal1)** précédente, une propriété de transport. Plus précisément, le fait que α soit $\bigsqcup$`-mcs` signifie que l'image par α du treillis complet L_1 est un $\bigsqcup$ semi-treillis complet muni d'un plus petit élément. Ainsi, c'est un treillis complet [BCOQ92, Théorème 4.27]. Évidemment, nous aurions pu faire une remarque analogue concernant la fonction γ.

Terminons avec la propriété **(Gal5)**. Elle exprime que la donnée d'une seule des deux fonctions de la paire $<\alpha, \gamma>$ est suffisante pour définir le cadre de connexion de Galois sur treillis complets. Cet aspect est souligné dans le théorème suivant.

Théorème 2.8. *[GM01, Proposition 8.5]*
Soient $L_1(\sqsubseteq_1, \bot_1, \top_1, \bigsqcup_1, \bigsqcap_1)$ *et* $L_2(\sqsubseteq_2, \bot_2, \top_2, \bigsqcup_2, \bigsqcap_2)$ *deux treillis complets.*
Soit $\alpha : L_1 \to L_2$ *une fonction* $\bigsqcup$`-mcs` *et définissons la fonction* $\gamma : L_2 \to L_1$ *par :*

$$\forall y \in L_2, \quad \gamma(y) \;=\; \bigsqcup_1 \{x \in L_1 \mid \alpha(x) \sqsubseteq_2 y\}$$

Alors la paire $<\alpha, \gamma>$ *définit une connexion de Galois* $L_1 \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} L_2$.

Nous aurions pu donner une version duale de ce théorème en mettant l'accent sur la donnée de la fonction γ. Il suffirait pour cela de remplacer L_1 par L_2, $\bigsqcup_1$ par $\bigsqcap_2$, α par γ et $\sqsubseteq_2$ par $\sqsupseteq_1$. En fait, la dualité inhérente à la notion de connexion de Galois implique que la plupart des théorèmes de l'interprétation abstraite admet une version duale qui s'obtient en utilisant le dictionnaire que nous venons d'évoquer.

Nous avons présenté, dans cette section, les principales propriétés des connexions de Galois. En interprétation abstraite, les connexions de Galois sont utilisées pour relier le domaine des assertions concrètes $\mathcal{A}$ et abstraites $\mathcal{A}^\sharp$. Pour une connexion $\mathcal{A} \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{A}^\sharp$, la fonction α qui permet de passer du domaine concret à l'abstrait, est appelée *opérateur d'abstraction* alors que la fonction γ est nommée *opérateur de concrétisation*. Nous avons mentionné plus haut que le cadre défini par des connexions de Galois sur treillis complets était idéal en interprétaion abstraite. Expliquons pourquoi. Rappelons tout d'abord que le treillis concret présenté en Section 2.2.2, à savoir $\mathcal{A}(\Rightarrow, fFalse, fTrue, \vee, \wedge)$, est un treillis complet. Comme nous l'avons remarqué, si α vérifie les hypothèses du Théorème 2.8, l'image par α d'un treillis complet est un treillis complet. Dans ce cas, si α est surjective ($\alpha(\mathcal{A}) = \mathcal{A}^\sharp$) et $\mathcal{A}$ est un treillis complet, alors l'ensemble $\mathcal{A}^\sharp$ est aussi un treillis complet. Le fait que les ensembles concrets et abstraits possèdent la même structure permet d'instancier la procédure itérative présentée en Section 2.2.2 dans le domaine abstrait $\mathcal{A}^\sharp$. La connexion de Galois permet alors de redescendre dans le monde concret les observations réalisées grâce aux calculs effectués dans le monde abstrait. Notons enfin que la surjectivité de α, équivalente à l'injectivité de γ [1], permet d'affirmer que deux propriétés abstraites différentes admettent toujours des concrétisations différentes. Ainsi, l'hypothèse de meilleure approximation est vérifiée.

2.3.2.2 Calculs corrects dans le domaine abstrait

Dans cette section, nous nous plaçons dans un cadre idéal de la théorie de l'interprétation abstraite. Plus précisément, le domaine concret $\mathcal{A}$ et le domaine abstrait $\mathcal{A}^\sharp$ sont munis d'une structure de treillis complets et reliés par une connexion de Galois $<\alpha, \gamma>$ telle que $\mathcal{A}^\sharp = \alpha(\mathcal{A})$.

La question qui se pose à présent est de savoir comment définir les assertions abstraites vérifiées par le programme. Elles ne peuvent être obtenues comme image par α des assertions concrètes vérifiées par le programme. En effet, ce procédé requiert d'avoir calculé au préalable les assertions concrètes, ce qui, comme nous l'avons mentionné précédemment n'est pas toujours possible ou requiert un

1. Cette équivalence est une autre illustration de la dualité présente dans la théorie des connexions de Galois et pourrait être ajoutée au dictionnaire de dualité précédemment évoqué.

temps de calcul jugé trop important. D'autre part, nous avons justement introduit le domaine abstrait dans le but d'y effectuer des calculs de manière plus simple et plus rapide en bénéficiant du caractère plus grossier de ce domaine. Ces calculs doivent refléter ceux que l'on aimerait effectuer dans le domaine concret. Ils sont censés approcher les calculs concrets par au-dessus. Nous commençons par donner la définition d'approximation correcte.

Définition 2.19 (Approximation correcte)**.**
Soient $\mathcal{A}(\sqsubseteq, \bot, \top, \bigsqcup, \bigsqcap)$ *et* $\mathcal{A}^\sharp(\sqsubseteq^\sharp, \bot^\sharp, \top^\sharp, \bigsqcup^\sharp, \bigsqcap^\sharp)$ *les treillis complets des assertions concrètes et abstraites, et* $<\alpha, \gamma>$ *une connexion de Galois entre ces domaines.*
La fonction $t^\sharp : L \to \mathcal{A}^\sharp \to \mathcal{A}^\sharp$ *est une* sur-approximation correcte *de* $t : L \to \mathcal{A} \to \mathcal{A}$ *si :*

$$\forall q \in L, \qquad \alpha \circ t_q \circ \gamma \quad \sqsubseteq^\sharp \quad t^\sharp_q$$

$$\left(\begin{array}{llll} \Leftrightarrow & \forall q \in L, & \alpha \circ t_q & \sqsubseteq^\sharp & t^\sharp_q \circ \alpha \\ \Leftrightarrow & \forall q \in L, & t_q \circ \gamma & \sqsubseteq & \gamma \circ t^\sharp_q \end{array} \right)$$

Pour simplifier la lisibilité de cet énoncé, nous avons opté pour la notation t_q pour représenter la fonction $t(q)$ (et $t^\sharp_q$ pour $t^\sharp(q)$). De plus, afin de ne pas alourdir les notations, nous avons noté de la même manière l'ordre $\sqsubseteq^\sharp$ du domaine $\mathcal{A}^\sharp$ et son extension point à point dans le domaine $\mathcal{A}^\sharp \to \mathcal{A}^\sharp$ et procédé de même pour l'ordre $\sqsubseteq$.

Le principe d'un analyse par interprétation abstraite consiste à effectuer les calculs itératifs sur le domaine abstrait. L'analyse est effectuée à l'aide d'une fonction $t^\sharp : L \to \mathcal{A}^\sharp \to \mathcal{A}^\sharp$, approximant la fonction $t : L \to \mathcal{A} \to \mathcal{A}$ donnant le sens des commandes du programmes. Le théorème suivant stipule que dans le cas où $t^\sharp$ est une sur-approximation correcte de t, l'analyse par point fixe est correcte : son résultat est une sur-approximation du point fixe concret.

Théorème 2.9 ([CC79])**.**
Soient $\mathcal{A}(\sqsubseteq, \bot, \top, \bigsqcup, \bigsqcap)$ *et* $\mathcal{A}^\sharp(\sqsubseteq^\sharp, \bot^\sharp, \top^\sharp, \bigsqcup^\sharp, \bigsqcap^\sharp)$ *les treillis complets des assertions concrètes et abstraites.*
Soit $<\alpha, \gamma>$ *une connexion de Galois entre ces domaines.*
Soient $t : L \to \mathcal{A} \to \mathcal{A}$ *et* $t^\sharp : L \to \mathcal{A}^\sharp \to \mathcal{A}^\sharp$ *deux fonctions monotones telles que* $t^\sharp$ *est une sur-approximation correcte de* t*. Alors :*

$\forall \pi,\ \forall \Phi \in \mathcal{A},\ \Phi^\sharp \in \mathcal{A}^\sharp$ *telles que* $\Phi \sqsubseteq \gamma(\Phi^\sharp)$

$$\alpha\ (\texttt{lfp}(F_\pi(t, \Phi))) \quad \sqsubseteq^\sharp \quad \texttt{lfp}(F_\pi(t^\sharp, \Phi^\sharp))$$

Un théorème analogue peut être énoncé dans le cas d'une analyse `MOP`.

Théorème 2.10 ([CC79]).
Soient $\mathcal{A}(\sqsubseteq, \bot, \top, \bigsqcup, \bigsqcap)$ *et* $\mathcal{A}^\sharp(\sqsubseteq^\sharp, \bot^\sharp, \top^\sharp, \bigsqcup^\sharp, \bigsqcap^\sharp)$ *les treillis complets des assertions concrètes et abstraites.*
Soit $<\alpha, \gamma>$ *une connexion de Galois entre ces domaines.*
Soit $t^\sharp : L \to \mathcal{A}^\sharp \to \mathcal{A}^\sharp$ *une sur-approximation correcte de* $t : L \to \mathcal{A} \to \mathcal{A}$.

Alors, $\forall \Phi \in \mathcal{A},\ \forall \Phi^\sharp \in \mathcal{A}^\sharp, \forall \pi,$

$$\alpha(\Phi) \ \sqsubseteq^\sharp \ \Phi^\sharp \quad \Rightarrow \quad \alpha(\mathtt{MOP}_\pi(t, \Phi)) \ \sqsubseteq^\sharp \ \mathtt{MOP}_\pi(t^\sharp, \Phi^\sharp)$$

Étant données les propriétés vérifiées par les connexions de Galois, cette notion de correction peut tout aussi bien être écrite à l'aide de la fonction de concrétisation :

$$\Phi \ \sqsubseteq \ \gamma(\Phi^\sharp) \quad \Rightarrow \quad \mathtt{MOP}_\pi(t, \Phi) \ \sqsubseteq \ \gamma(\mathtt{MOP}_\pi(t^\sharp, \Phi^\sharp))$$

Remarque 2.1 (Cas d'égalité dans les théorèmes).
Les théorèmes de cette section ont été énoncés à l'aide d'inclusions. On dira que $t^\sharp : L \to \mathcal{A}^\sharp \to \mathcal{A}^\sharp$ *est une* abstraction exacte *de* $t : L \to \mathcal{A} \to \mathcal{A}$ *en cas d'égalité dans la définition 2.19 (*$\alpha \circ t_q \circ \gamma = t_q^\sharp$*). Dans le cas d'une approximation exacte, les inégalités des théorème 2.9 et théorème 2.10 deviennent des égalités dès que l'on choisit* $\Phi^\sharp$ *tel que* $\Phi^\sharp = \alpha(\Phi)$.

2.3.2.3 Notion de meilleure fonction abstraite

Le cadre de l'interprétation abstraite permet de comparer les approximations correctes entre elles. En fait, on peut même définir la fonction abstraite $t^\sharp$ qui approche de manière la plus précise la fonction t exprimant la signification concrète des commandes.

Théorème 2.11 (Meilleure approximation).
Soient $\mathcal{A}(\sqsubseteq, \bot, \top, \bigsqcup, \bigsqcap)$ *et* $\mathcal{A}^\sharp(\sqsubseteq^\sharp, \bot^\sharp, \top^\sharp, \bigsqcup^\sharp, \bigsqcap^\sharp)$ *les treillis complets des assertions concrètes et abstraites.*
Soit $<\alpha, \gamma>$ *une connexion de Galois entre ces domaines et* $t : L \to \mathcal{A} \to \mathcal{A}$.
Soient $t_1^\sharp, t_2^\sharp : L \to \mathcal{A}^\sharp \to \mathcal{A}^\sharp$ *deux sur-approximations correctes de* t.
On dira que $t_1^\sharp$ *est* meilleure *que* $t_2^\sharp$ *si :*

$$\forall q \in L, \quad t_1^\sharp(q) \quad \sqsubseteq^\sharp \quad t_2^\sharp(q)$$

La fonction $t_{best}^\sharp : L \to \mathcal{A}^\sharp \to \mathcal{A}^\sharp$ *définie par :*

$$\forall q \in L, \quad t_{best}^\sharp(q) \ = \ \alpha \circ t(q) \circ \gamma$$

est la meilleure *sur-approximation correcte de* t.

Dans $\mathcal{A}^\sharp$, la meilleure approximation correcte est celle qui fournit la sémantique abstraite la plus précise *c.à.d.* la plus proche de la sémantique concrète. Lors d'une analyse par interprétation abstraite, il faudrait donc toujours choisir $t^\sharp_{best}$ comme sur-approximation de t. Cependant, même dans le cadre idéal d'interprétation abstraite dans lequel nous sommes, les fonctions α et γ ne sont pas forcément faciles à calculer. Pour une analyse par interprétation abstraite, la question du calcul de ces fonctions n'est pas vraiment légitime : α et γ ne sont jamais calculées lors de l'analyse. Ces fonctions servent juste de tuteurs et contraignent la sémantique abstraite à être une sur-approximation de la sémantique concrète. D'autre part, nous allons voir que les analyses par interprétation abstraite peuvent être instanciées dans des cadres moins strict que celui défini par des connexions de Galois sur treillis complets. Nous présentons brièvement cet aspect dans la section suivante.

2.3.3 Analyse par interprétation abstraite dans un cadre plus général

Afin de comprendre comment effectuer des analyses par interprétation abstraite dans un cadre autre que le cadre idéal défini précédemment, revenons sur l'exemple de l'analyse polyédrique.

Étude d'un exemple : l'analyse polyédrique [CH78]
Rappelons que le domaine concret est défini par le treillis complet $\mathcal{P}(\mathbb{R}^m)(\subseteq, \emptyset, \mathbb{R}^m, \cup, \cap)$. Le domaine abstrait, quant à lui est défini par le treillis $\mathtt{P}_m(\sqsubseteq_{\mathtt{P}}, \bot_{\mathtt{P}}, \top_{\mathtt{P}}, \bigsqcup_{\mathtt{P}}, \bigsqcap_{\mathtt{P}})$. Malheureusement, ce treillis n'est pas complet. En effet, la propriété stipulant l'existence d'une borne inférieure pour toute partie de $\mathtt{P}_m$ n'est pas vérifiée. Il suffit pour s'en convaincre d'étudier la partie formée de tous les polyèdres incluant le disque unité. La borne inférieure de cette partie est le disque lui-même, qui n'est pas un élément de $\mathtt{P}_m$. Nous ne pouvons donc pas utiliser les théorèmes présentés en Section 2.3.2. Cependant, tout n'est pas perdu. Par exemple, nous pouvons définir la fonction de concrétisation $\gamma : \mathtt{P}_m \to \mathcal{P}(\mathbb{R}^m))$ comme étant la fonction qui à tout polyèdre convexe associe l'ensemble des points qui le compose. À l'opposé, il semble difficile de définir une fonction d'abstraction α pour la raison que nous avons déjà mentionnée. En effet, comment définir l'abstraction de l'ensemble des points constituant le disque unité ? On peut approcher cet ensemble par une infinité de polyèdres mais il n'existe pas, dans le domaine $\mathtt{P}_m$, de meilleure approximation de cet ensemble.

L'exemple de l'analyse polyèdrique soulève une question légitime : que faire dans le cas où l'hypothèse de meilleure approximation n'est pas vérifiée ? En fait, on peut résoudre se problème par l'utilisation de diverses stratégies [CC92c]. Nous

présentons ici deux possibilités permettant de se ramener au cas où l'hypothèse de meilleure approximation est vérifiée par des manipulations sur le domaine abstrait.

Affaiblir les propriétés abstraites [CC92c]
On peut analyser la difficulté rencontrée dans le cas de l'analyse polyédrique par le fait que le domaine abstrait est trop précis. Ce point de vue consiste à dire que l'on approche le disque unité de trop près, si bien que la meilleure approximation de cet ensemble est le disque lui-même. L'idée est alors de choisir un domaine abstrait plus grossier. Ainsi, afin d'analyser le programme `sqrt` de la figure 2.3, nous aurions pu choisir le domaine abstrait des intervalles. Pour un programme à m variables, un élément du domaine des intervalles est la donnée, pour chaque variable $\mathtt{x_i}$, d'un intervalle encadrant cette variable. Autrement dit, le domaine abstrait est constitué de l'ensemble des hyper rectangles de dimension au plus m. Dans ce domaine, le problème de l'approximation du disque unité est trivial : la meilleure approximation du disque unité est l'hypercube unité. Ce domaine permet de se placer dans le cadre idéal de l'existence de meilleure approximation. Cependant, le domaine des intervalles est trop grossier pour inférer les propriétés que l'on souhaite, à savoir $-2r + t - 1 = 0$ et $-n + s - 1 \geq 0$ (voir figure 2.3). En effet, ce domaine n'est pas relationnel. Il est donc impossible d'énoncer, dans ce domaine, des propriétés liant les variables du programme. Pour résoudre ce nouveau problème, nous pouvons proposer d'effectuer les calculs dans le domaine abstrait des octogones [Min04, Min06]. Un élément de ce domaine est constitué d'inégalités de la forme $\pm\mathtt{x}_i \pm \mathtt{x}_j \leq c$. Ainsi, une analyse effectuée dans ce domaine pourra inférer la propriété $-n + s - 1 \geq 0$. Cependant, la propriété $-2r + t - 1 = 0$ reste trop précise et sera approchée, dans ce domaine par la conjonction des inégalités $t \geq 1$ et $t - r \geq 1$.

Cette première stratégie nous permet de souligner l'importance du choix du domaine abstrait. Rappelons que plus le domaine abstrait est précis, plus les calculs de points fixes seront difficiles à effectuer mais plus les propriétés obtenues seront précises. L'exemple du domaine des octogones illustre parfaitement cet aspect puisqu'il propose une précision intermédiare entre les intervalles et les polyèdres pour un coût lui aussi intermédiaire entre les intervalles (coût linéaire) et les polyèdres (coût exponentiel).

Renforcer les propriétés abstraites [CC92c]
Cette stratégie se base sur un point de vue opposé à la stratégie précédente. Il consiste à dire que le problème provient du manque de précision du domaine abstrait. Dans le cas des polyèdres, cela revient à dire que le problème lors de l'approximation du disque unité est dû à l'absence du disque unité dans le

domaine des polyèdres. L'idée est alors de « rajouter » le disque unité dans le domaine des polyèdres tout en conservant la structure de polyèdre. Pour cela, on peut utiliser un mécanisme de complétion conjonctive qui consiste à prendre pour domaine abstrait l'ensemble des parties constituées d'éléments incomparables du domaine abstrait initial. Cette méthode permet de vérifier l'hypothèse de meilleure approximation. Cependant, elle a tendance à produire des explosions combinatoires, ce qui est notamment le cas pour la complétion du domaine des polyèdres. Cette stratégie est donc peu usitée dans la pratique et plutôt considérée comme une approche théorique.

Ces deux stratégies ne sont pas complètement satisfaisantes puisqu'elles ne permettent pas d'inférer l'une des propriétés souhaitées, à savoir $-2r + t - 1 = 0$. En fait, dans le cas de l'analyse polyèdrique, le cadre permettant de vérifier l'hypothèse de meilleure approximation est trop exigeant. L'idée est alors de se détacher de cette obligation en proposant un cadre plus large. Par exemple, une analyse par interprétation abstraite peut être définie par la simple donnée d'une fonction d'abstraction α, d'un ordre abstrait $\sqsubseteq^\sharp$ et d'un mécanisme de rétrécissement $\Delta^\sharp$ [CC92c]. La notion de correction présentée en définition 2.19 s'écrit alors simplement $\alpha \circ t_q \sqsubseteq^\sharp t_q^\sharp \circ \alpha$. L'inégalité de correction du défini par le théorème 2.9 se réécrit tel quel et est assurée, pour toute sur-approximation correcte $t^\sharp$ de t, par le mécanisme de rétrécissement utilisé. Dans le cas des polyèdres, le cadre de l'interprétation abstraite est instancié par la donnée d'une fonction de concrétisation γ, d'un ordre concret $\sqsubseteq$ et d'un mécanisme d'élargissement. Nous détaillons davantage cette analyse dans le paragraphe suivant.

Étude d'un exemple : l'analyse polyédrique [CH78]
Commençons par décrire la fonction $t^\sharp$ qui définit comment les commandes du programme agissent sur les polyèdres de $\mathtt{P}_m$. Les commandes linéaires *c.à.d.* les affectations linéaires et les fonctions de test définies par des égalités ou inégalités linéaires sont particulièrement adaptées à cette analyse. En revanche, les commandes non-linéaires sont approchées de manière très grossière. Décrivons un peu plus précisément comment s'opère cette analyse. On commence par décrire l'effet de la fonction $t^\sharp$ dans le cas où la commande considérée est une affectation.

Cas des affectations non-linéaires.
Si le programme comporte une affectation non-linéaire d'une variable $\mathtt{x_i}$, l'analyse abstraite considérera grossièrement que toute valeur peut être affectée à $\mathtt{x_i}$. Ceci revient à dire que l'on perd toute information sur la variable $\mathtt{x_i}$ après cette affectation. Pour ce type d'affectations, $t^\sharp$ est définie comme la fonction qui à tout polyèdre p (représenté par un système S_p d'inégalités linéaires) associe le polyèdre p' obtenu par élimination de la variable $\mathtt{x_i}$ dans le système S_p. Ceci est réalisé par projection des inégalités de S_p suivant $\mathtt{x_i}$.

Cas des affectations linéaires.
Si l'on considère maintenant que $\mathtt{x_i}$ subit une affectation linéaire, deux sous-cas se présentent en fonction du caractère inversible ou non de l'affectation.

Une affectation linéaire est dite inversible si elle est du type $\mathtt{x_i} = a\mathtt{x_i} + \ldots$ avec $a \neq 0$. Prenons un cas simple pour comprendre comment est défini $t^\sharp$ dans ce cas. Considérons l'affectation $\mathtt{x_1} = 2\mathtt{x_1} + 3\mathtt{x_2}$ et étudions comment cette affectation agit sur un polyèdre p vérifiant l'inégalité $\mathtt{x_1} \leq \mathtt{x_2} + 2\mathtt{x_3}$ (*c.à.d.* que l'inégalité $\mathtt{x_1} \leq \mathtt{x_2} + 2\mathtt{x_3}$ est contenue dans le système S_p décrivant p). Si x_1 et x'_1 dénotent la valeur de $\mathtt{x_1}$ avant et après affectation, on a $x'_1 = 2x_1 + 3x_2$. On peut inverser ce système de sorte à connaître la valeur de x_1 en fonction de celle de x'_1 : $x_1 = \frac{1}{2}x'_1 - \frac{3}{2}x_2$. Ainsi, le fait que x_1 vérifie l'inégalité $\mathtt{x_1} \leq \mathtt{x_2} + 2\mathtt{x_3}$ est équivalent au fait que x'_1 vérifie l'inégalité $\frac{1}{2}\mathtt{x_1} - \frac{3}{2}\mathtt{x_2} \leq \mathtt{x_2} + 2\mathtt{x_3}$ soit $\mathtt{x_1} \leq 5\mathtt{x_2} + 4\mathtt{x_3}$. Pour résumer et généraliser, si une variable $\mathtt{x_i}$ est affectée de manière linéaire inversible $\mathtt{x_i} := f(\mathtt{x_1}, \ldots, \mathtt{x_m})$, $t^\sharp$ associe au polyèdre p le polyèdre $p[f_{inv}(\mathtt{x_1}, \ldots, \mathtt{x_m})/\mathtt{x_i}]$ [2] (obtenu en remplaçant $\mathtt{x_i}$ par $f_{inv}(\mathtt{x_1}, \ldots, \mathtt{x_m})$ dans toute inégalité définissant S_p, avec f_{inv} telle que $f_{inv} \circ f(\mathtt{x_1}, \ldots, \mathtt{x_m}) = f \circ f_{inv}(\mathtt{x_1}, \ldots, \mathtt{x_m}) = \mathtt{x_i}$). En résumé, le cas des affectations linéaires inversibles est très facilement et rapidement traitable. Il convient de noter, de plus, que le traitement de ces affectations se fait sans perte d'information : pour toute variable, l'appartenance au polyèdre avant l'affectation est équivalente à l'appartenance au nouveau polyèdre après affectation. L'analyse de ces affectations est de ce fait particulièrement élégante.

En revanche, le traitement des affectations linéaires non-inversibles produit une perte d'information. La variable affectée est, dans un premier temps, éliminée du polyèdre courant par projection suivant cette variable (de la même façon que dans le cas des affectations non-linéaires). L'information de l'affectation est alors rajoutée à ce nouveau polyèdre par ajout de l'égalité correspondant à cette affectation.

Cas des tests.
Nous ne développons pas de manière précise comment sont traitées les fonctions de test. Notons tout de même qu'une disjonction de cas analogue à la précédente entre en jeu. Les tests non-linéaires sont tout bonnement ignorés par l'analyse. Dans le cas d'une fonction de test linéaire, l'analyse calcule l'intersection du polyèdre courant avec celui défini par le test.

Maintenant que nous avons décrit le traitement de chaque commande par la fonction $t^\sharp$, il ne reste plus qu'à décrire comment est obtenue la sémantique abstraite du programme. L'approche par plus petit point fixe décrite par le système d'équations de la définition 2.15 ne peut être utilisée telle quelle. Le problème provient du fait que le domaine des polyèdres $\mathtt{P}_m$ ne vérifie pas la condition de

2. Cette écriture correspond à la notation $\varphi \circ q^{-1}$ introduite lors du commentaire de la définition 2.11.

chaîne ascendante. Un calcul de plus petit point fixe n'est donc pas directement réalisable à l'aide du théorème 2.2. Afin de contourner ce problème de terminaison, une opération d'élargissement est ajoutée à l'analyse. Plus précisément, un opérateur d'élargissment est introduit dans le système d'équations dans chaque calcul concernant un point de jonction d'une boucle. Cet opérateur d'élargissement ∇ a été introduit en Section 2.1 Étant donné p_1 et p_2 deux polyèdres de $\mathtt{P}_m$, l'ensemble $p_1 \nabla p_2$ représente le polyèdre convexe défini par les inégalités de p_1 vérifiées par tous les points du polyèdre p_2. Cet opérateur d'élargissement agit donc sur p_1 en diminuant son nombre d'inégalités. Tout polyèdre étant défini par un nombre fini d'inégalités, ce processus termine en un nombre fini d'itérations.

Chapitre 3

Analyse de programmes à coûts

Un grand nombre des objets du quotidien embarquent des systèmes électroniques. Cette tendance, comme nous l'avons déjà mentionné, continue de se développer dans les domaines de l'automobile, de l'aéronautique mais aussi dans le domaine médical. Dans ce chapitre, nous nous intéressons particulièrement aux systèmes embarqués dont l'utilisation de **ressources** est limitée. Ceci peut se traduire par une limitation de l'espace mémoire ou du temps durant lequel le programme peut s'exécuter. Si l'on considère l'exemple des technologies « portables » (tels que les téléphones, ordinateurs, tablettes), on peut aussi citer une limitation des ressources énergétiques disponibles (batteries limitées). L'analyse de tels systèmes, pour lesquels la consommation de ressources est un problème critique, doit permettre d'inférer des **propriétés quantitatives** assurant que les ressources disponibles seront suffisantes lors de l'exécution.

Afin de répondre à ce problème, nous étudions, dans ce chapitre, la possibilité de quantifier par interprétation abstraite, l'utilisation des ressources d'un programme. La difficulté de ce problème est liée à la nature de la théorie de l'interprétation abstraite. Nous avons vu dans le Chapitre 2, que cette théorie propose un cadre d'approximation basé sur la notion de treillis. À l'aide de ces treillis, nous avons vu que nous pouvions comparer entre elles les abstractions. On peut ainsi définir la qualité des approximations effectuées et notamment la notion de meilleure approximation. À l'opposé, les notions quantitatives n'apparaissent pas naturellement dans ce cadre. Nous cherchons, dans ce chapitre, à réconcilier ces deux aspects.

Plus précisément, nous présentons une méthode permettant d'approcher, de manière correcte, un mesure sur les **programmes transportant des coûts**. La première étape de notre méthode consiste à définir ce genre de programmes. Dans la suite, ces programmes seront représentés par une relation de transition étiquetée par des coûts. On notera $\sigma \rightarrow^q \sigma'$ une transition entre l'état σ et l'état σ' possédant le coût q. Afin de pouvoir combiner les coûts des transitions, nous

supposerons que l'ensemble Q des coûts admet une structure particulière. Plus précisément, nous supposerons l'existence d'un opérateur noté multiplicativement $\otimes$, permettant d'accumuler les coûts le long d'un chemin (séquence de transitions) ainsi que l'existence d'un opérateur noté additivement $\oplus$, permettant de combiner les coûts de différents chemins. Ces deux coûts munissent l'ensemble Q d'une structure appelée dioïde. Nous verrons que cette structure permet de définir une autre structure, appelée moduloïde. Cette dernière structure est particulièrement importante car elle permet de représenter la sémantique étiquetée de nos programmes sous forme matricielle. Cette matrice, indexée par l'ensemble des états du programme, transporte, en coefficients, les coûts du programme.

Notre analyse se concentre sur l'étude de programmes dont le comportement est cyclique (tels que les systèmes réactifs) dont la propriété d'intérêt est le comportement asymptotique le long des cycles plutôt que le coût global du programme en son entier. Ce comportement est décrit par une mesure nommée **coût long-run**. Cette dénomination provient du cas classique où l'opération $\otimes$ du dioïde correspond à l'opération d'addition $+$ arithmétique. Dans ce cas, on montre que le coût long-run correspond à la limite du coût moyen maximum sur des traces arbitrairement longues.

Le coût long-run étant généralement non calculable, nous définissons un **cadre d'approximation** permettant d'inférer une sur-approximation de ce coût. Ce cadre tire parti de l'expression linéaire de la sémantique de nos programmes. Plus précisément, nous verrons que toute abstraction α sur les états d'un programme peut être relevée en un opérateur matriciel $\alpha^{\uparrow}$. Cette abstraction linéaire possède alors l'avantage d'admettre une fonction résiduée $\gamma^{\uparrow}$, ce qui revient à dire que le couple $(\alpha^{\uparrow}, \gamma^{\uparrow})$ forme une connexion de Galois. Ainsi, via ce relèvement linéaire, nous définissons une analyse par interprétation abstraite.

Notons au passage que nous ne supposons pas de propriété particulière sur les abstractions initiales α et que c'est le relèvement linéaire qui permet de voir ces abstractions comme des éléments d'une connexion de Galois. Ainsi, les abstractions initiales les plus élémentaires consistent en un partitionnement des états par fusion. Toutefois, nous nous sommes aussi interrogés sur la possibilité d'utiliser des abstractions α plus complexes comme celles issues de connexions de Galois. Dans ce cas, nous verrons que le relèvement linéaire n'est pas très efficace et qu'il ne permet pas le transport des structures ordonnées. Nous nous attacherons alors à proposer un relèvement plus performant, qui respecte la notion d'ordre sur les états tout en restant dans le cadre linéaire défini par notre méthode.

Ce chapitre est structuré comme suit. En Section 3.1, nous présentons la notion de programmes quantitatifs. En Section 3.2, nous décrivons les propriétés qui munissent l'ensemble des coûts Q d'une structure de dioïde particulière nommée

dioïde de coûts. En Section 3.3 nous présenterons le cadre linéaire induit par les opérations du dioïde de coûts. Nous présenterons alors la notion de coût *long-run*. Ce coût n'étant pas directement calculable, nous procéderons à des approximations. Nous détaillerons en Section 3.4, notre cadre d'approximation ainsi que le relèvement linéaire sur lequel il se base. En Section 3.5, nous étudierons alors un relèvement plus adapté lorsque l'abstraction initiale est issue d'une connexion de Galois. Autrement dit, nous verrons comment l'on peut intégrer les abstractions classiques de l'interprétation abstraite dans notre cadre linéaire. Enfin, nous illustrerons notre analyse en l'implémentant sur un exemple en Section 3.6 et finirons par un état de l'art en Section 3.7.

3.1 Les programmes quantitatifs

L'analyse présentée dans ce chapitre a pour but d'inférer une sur-approximation du coût asymptotique de programmes porteurs d'informations quantitatives. Dans cette section, nous présentons les programmes qui sont traités par notre analyse.

3.1.1 Syntaxe des programmes quantitatifs

Les programmes quantitatifs sont définis à l'aide d'une relation de transition étiquetée par des coûts, éléments de l'ensemble Q.

Définition 3.1 (Programmes quantitatifs)**.**
Un programme quantitatif *est un quadruplet* $P = <\Sigma, I, Q, \rightarrow^{\cdot}>$ *tel que :*

- Σ *désigne un ensemble dénombrable d'états,*
- $I \subseteq \Sigma$ *est l'ensemble des états initiaux du programme* P*,*
- Q *est un ensemble représentant les coûts du programme,*
- $\rightarrow^{\cdot} \subseteq \Sigma \times \Sigma \rightarrow Q$ *est une relation de transition étiquetée par des éléments de l'ensemble* Q*.*

Une transition $\sigma \rightarrow^{q} \sigma'$ *définit une transition de l'état* σ *à l'état* σ' *de coût* q*.*

Cette définition des programmes diffère de la définition 2.9 donnée dans le chapitre traitant de la théorie de l'interprétation abstraite. En effet, dans le Chapitre 2, nous avons défini les programmes comme une relation de transition étiquetée non pas par des coûts mais par les commandes du programme. Le passage de la représentation « commandes » à la représentation « coûts » peut se faire à l'aide d'un processus attribuant un coût à chaque commande. Pour être réaliste, le coût de chaque commande doit dépendre de l'état du programme dans lequel la commande est appelée. Autrement dit, à chaque point de programme précédant un appel de la commande, l'ensemble des valeurs possibles des variables influe sur

le coût de la commande. D'autre part, ce coût doit aussi dépendre de l'application visée. Par exemple, si l'on s'intéresse à la complexité spatiale du programme, le coût d'une commande dépendra naturellement de l'espace mémoire alloué lors de l'exécution de la commande. Dans ce chapitre, nous souhaitons inférer un coût de programme générique. Notre méthode entrera donc en jeu une fois le processus d'attribution de coûts réalisé et cette phase initiale ne sera pas traitée ici. Autrement dit, nous nous intéressons directement aux programmes quantitatifs dans ce chapitre.

Il convient de noter que la définition de nos programmes autorise la présence de non-déterminisme et de boucles. Par la suite, nous noterons $\Pi_{\sigma,\sigma'}$ l'ensemble des transitions (chemins) menant de σ à σ'.

3.1.2 Sémantique des programmes quantitatifs

Dans un premier temps, nous présentons la sémantique d'un programme quantitatif comme étant la donnée de ses traces d'exécution. Dans la Section 3.3, nous présenterons une sémantique plus adaptée à notre analyse.

Définition 3.2 (Sémantique de traces d'un programme quantitatif)**.**
Soit $P = <\Sigma, I, Q, \rightarrow^{\cdot}>$ un programme quantitatif. La sémantique de traces de P est notée $[\![P]\!]_{tr}$ et est définie comme suit.

$$[\![P]\!]_{tr} = \{\sigma_0 \rightarrow^{q_0} \ldots \sigma_{n-1} \rightarrow^{q_{n-1}} \sigma_n \mid \sigma_0 \in I, \sigma_i \rightarrow^{q_i} \sigma_{i+1}\}$$

Nous définissons donc la sémantique des programmes quantitatifs sous une forme opérationnelle. Nous nous distinguons de la sémantique dénotationnelle présentée dans le Chapitre 2. Cette différence de point de vue souligne ce qui nous importe ici : nous souhaitons inférer un coût de programme, qui dépend, évidemment, du coût de chaque opération du programme.

La notion de programme quantitatif étant maintenant définie, nous étudions la structure permettant de gérer l'ensemble des coûts Q.

3.2 Une structure mathématique adaptée : les dioïdes de coûts

Dans cette section, nous définissons les opérateurs $\oplus$ et $\otimes$ permettant de manipuler et combiner les éléments de l'ensemble des coûts Q. Ces opérateurs munissent Q d'une structure de dioïde. En plus de ces opérateurs, nous supposerons aussi l'existence d'une opération racine `n-ieme`, définissant ainsi la notion de dioïde de coûts.

3.2.1 Les opérateurs $\oplus$ et $\otimes$

Les coûts de transitions unitaires $\sigma \rightarrow^q \sigma'$ peuvent être combinés à l'aide de deux opérateurs. Le premier, noté multiplicativement $\otimes : Q \times Q \rightarrow Q$, permet d'accumuler les coûts rencontrés le long d'une suite de transitions. Le second, noté additivement $\oplus : Q \times Q \rightarrow Q$, permet de combiner les coûts de différents chemins. Ces opérateurs peuvent être instanciés de nombreuses manières. Par exemple, si Q permet de représenter des temps d'exécution et que l'on souhaite connaître le temps d'exécution dans le pire cas, on prendra pour $\otimes$ l'addition arithmétique classique $+$ et pour $\oplus$ l'opérateur max. Si l'on s'intéresse à des questions d'accessibilité comme de connaître le plus grand ensemble de coûts rencontrés dans toute exécution du programme, Q sera choisi comme étant un ensemble de parties $\mathcal{P}(S)$ et on prendra l'union ensembliste $\cup$ pour $\otimes$ et l'intersection ensembliste $\cap$ pour $\oplus$. Introduisons maintenant quelques notations.

Définition 3.3 (Coût d'une suite de transitions)**.**
Soit $P = <\Sigma, I, Q, \rightarrow^{\cdot}>$ un programme quantitatif et soit $\pi = \sigma \rightarrow^{q_1} \ldots \rightarrow^{q_n} \sigma'$ une suite de transitions.
Le coût de la transition π est noté $q(\pi)$ ou simplement q_π et est défini par :

$$q_\pi = q_1 \otimes \ldots \otimes q_n$$

Le coût global q de σ à σ' est alors défini à l'aide de l'opérateur $\oplus$ par :

$$q = \bigoplus_{\pi \in \Pi_{\sigma,\sigma'}} q_\pi$$

L'existence d'une transition π de coût q_π pourra être notée de manière concise :

$$\sigma \overset{\pi}{\Rightarrow}^{q_\pi} \sigma'$$

Notons que cette définition est écrite sans utiliser de parenthésage. Ceci provient de l'associativité des opérateurs qui est l'une des propriétés que nous développons maintenant.

3.2.2 Notion de dioïde et ordre associé

Afin de pouvoir facilement manipuler les coûts d'un programme, nous exigeons que les opérateurs $\oplus$ et $\otimes$ possèdent certaines propriétés qui munissent Q d'une structure de dioïde commutatif.

Définition 3.4 (Dioïde commutatif)**.**
Un dioïde commutatif *est une structure $Q(\oplus, \otimes)$ qui vérifie les propriétés suivantes.*

1. *L'opérateur $\otimes$ est associatif, commutatif et possède un élément neutre* $\mathtt{e}$. *La quantité* $\mathtt{e}$ *est utilisée pour les transitions de coût nul.*
2. *L'opérateur $\oplus$ est associatif, commutatif et possède un élément neutre $\perp$. La quantité $\perp$ représente l'impossibilité d'une transition.*
3. *$\otimes$ est distributive par rapport à $\oplus$, et $\perp$ est un élément absorbant pour $\otimes$ ($\forall x. x \otimes \perp = \perp \otimes x = \perp$).*
4. *Le pré-ordre défini par $\oplus$ ($a \leq b \Leftrightarrow \exists c : a \oplus c = b$) est une relation d'ordre (*c.à.d. *elle satisfait $a \leq b$ et $b \leq a \Rightarrow a = b$).*

Revenons à la structure de dioïde. Tout d'abord, remarquons que $\perp$, l'élément neutre de l'opérateur $\oplus$, est le plus petit élément du dioïde. En effet, tout élément a de Q vérifie $\perp \oplus a = a$. Par définition de l'ordre $\leq$, on a donc $\perp \leq a$. D'autre part, notons qu'un dioïde ne peut être un anneau. Ceci est une conséquence du fait que la relation d'ordre contredit l'existence d'inverses par $\oplus$. En effet, si tout élément a possédait un inverse $-a$ par la loi $\oplus$, on aurait alors $a \oplus (-a) = \perp$. Cette dernière égalité implique que $a \leq \perp$ et on aurait donc, par anti-symétrie, $a = \perp$ (de la même manière $-a = \perp$).

Le lemme suivant est un résultat classique de la théorie des dioïdes [GM01, proposition 6.1.7][BMR97, théorème 2.4].

Lemme 3.1.
Soit $Q(\oplus, \otimes)$ un dioïde commutatif.
Les opérateurs $\oplus$ et $\otimes$ préservent l'ordre $\leq$:

$$\begin{array}{rrcl} \forall a, b, c \in Q, & a \leq b & \Rightarrow & a \otimes c \leq b \otimes c \\ \textit{et} & a \leq b & \Rightarrow & a \oplus c \leq b \oplus c \end{array}$$

S'il existe différents chemins menant de σ à σ' et possédant le même coût q, le coût global est défini comme étant égal à q. Cette propriété est connue sous le nom d'idempotence.

Définition 3.5 (Idempotence)**.**
Un dioïde $Q(\oplus, \otimes)$ est idempotent *si son opérateur $\oplus$ l'est :*

$$\forall q \in Q, \quad q \oplus q = q$$

Les dioïdes idempotents les plus fréquemment rencontrés dans la littérature sont certainement $\overline{\mathbb{R}}(\mathtt{max}, +)$ et $\overline{\mathbb{R}}(\mathtt{min}, +)$, où $\overline{\mathbb{R}}$ désigne l'ensemble $\mathbb{R} \cup \{-\infty, +\infty\}$. Les ordres induits sont respectivement les ordres $\leq$ et $\geq$ sur les nombres réels, étendus à $\overline{\mathbb{R}}$ de la manière habituelle.

Il est important de noter que, dans un dioïde idempotent, l'ordre induit vérifie la propriété :

$$\boxed{a \leq b \quad \Leftrightarrow \quad a \oplus b = b} \tag{3.1}$$

Cette propriété permet de faire le lien avec la notion de treillis.

Remarque 3.1 (Addition idempotente versus treillis)**.**
L'équivalence 3.1 permet de montrer qu'il y a un lien fort entre la notion d'addition idempotente et la structure de semi-treillis supérieur[1]*. En effet, une addition idempotente induit une structure d'ordre (comme déjà mentionné en définition 3.4) dans laquelle chaque paire d'éléments* (a, b) *admet la borne supérieure* $a \oplus b$*. Inversement, si l'on considère un semi-treillis supérieur, on peut définir l'opération d'addition de deux éléments par la borne supérieure de ces éléments. Ceci définit une addition idempotente [BCOQ92]. Notons que les dioïdes idempotents sont aussi nommés semi-anneaux tropicaux dans la littérature.*

Notre analyse manipule des ensembles d'états pouvant être infinis dénombrables. Nous supposerons donc le caractère complet des dioïdes.

Définition 3.6 (Dioïde idempotent complet)**.**
Un dioïde idempotent est complet *s'il est clos par passage aux sommes infinies et que la propriété de distributivité de la loi* $\oplus$ *est satisfaite pour un nombre potentiellement infini de termes : pour tout ensemble* $X \subseteq Q$*, la somme (infinie ou non)*

$$\bigoplus_{x \in X} x$$

existe dans le dioïde et :

$$\forall a \in Q, \quad a \otimes (\bigoplus_{x \in X} x) = \bigoplus_{x \in X} (a \otimes x)$$

Remarque 3.2 (Addition idempotente versus treillis (suite))**.**
Complétons la Remarque 3.1. Elle nous permet d'affirmer qu'un dioïde idempotent complet définit un semi-treillis supérieur complet sur Q*. Soulignons au passage qu'un dioïde complet possède naturellement un plus grand élément, noté* $\top$*, défini par la somme de tous les éléments du dioïde. On a vu précédemment que l'élément neutre* $\perp$ *est, par définition, le plus petit élément du dioïde. Ainsi, un dioïde idempotent complet est un semi-treillis supérieur complet qui admet un plus petit élément. C'est donc un treillis complet [BCOQ92, Théorème 4.27]. De ce fait, tout dioïde de ce type est muni d'un opérateur d'intersection* $\wedge$*. En résumé :*

$Q(\oplus, \otimes)$		$Q(\leq, \oplus, \wedge, \perp, \top)$
dioïde complet idempotent	$\equiv$	*treillis complet*

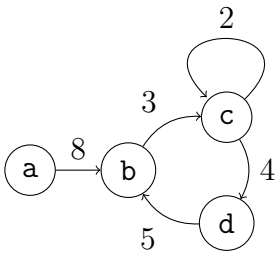
1. Rappelons qu'un semi-treillis supérieur est un ensemble partiellement ordonné tel que toute paire d'éléments possède une borne supérieure.

3.2.3 Les dioïdes de coûts

En introduction, nous avons indiqué que notre analyse permet de calculer un coût, nommé coût long-run, qui est une sur-approximation des coûts moyens des cycles du programme. De ce fait, en plus de la structure de dioïde précédemment présentée, notre analyse nécessite l'existence d'un opérateur de moyenne.

3.2.3.1 L'opérateur racine n-ième

Nous nous intéressons ici à la définition d'un opérateur de moyenne qui permettrait de calculer le coût moyen d'une suite de transitions $\pi = \sigma \rightarrow^{q_1} \cdots \rightarrow^{q_n} \sigma'$. Nous avons vu que le coût accumulé d'une suite de transitions était défini grâce à l'opérateur $\otimes$: par $q(\pi) = q_1 \otimes \cdots \otimes q_n$. Étant donné la notation multiplicative de cet opérateur, l'opérateur moyenne est la racine n-ième $\sqrt[n]{.} : Q \rightarrow Q$. La figure 3.1 illustre la notion de racine n-ième sur un exemple simple pour lequel l'opérateur $\otimes$ est l'opérateur d'addition classique.



Coût moyen du chemin `abc` : $\sqrt[2]{8 \otimes 3} = (8+3)/2 = 5.5$

Coût moyen du cycle `bcdb` : $\sqrt[3]{3 \otimes 4 \otimes 5} = (3+4+5)/3 = 4$

Coût moyen du cycle `bccdb` : $\sqrt[4]{3 \otimes 2 \otimes 4 \otimes 5} = 3.5$

Coût moyen du cycle `cc` : $\sqrt[1]{2} = 2$

FIGURE 3.1 – Calcul de moyenne par l'opérateur racine n-ième

Définition 3.7 (Racine n-ième et coût moyen)**.**
Soit $n > 0$ et $Q(\oplus, \otimes)$ un dioïde.
Supposons que, pour tout élément q de Q, l'équation $X^n = q$ possède une unique *solution dans Q et notons $\sqrt[n]{q}$ cette solution.*
Dans ce cas, la fonction racine n-ième *est bien définie et vaut :*

$$\begin{array}{rcl} \sqrt[n]{.} : Q & \rightarrow & Q \\ q & \mapsto & \sqrt[n]{q} \end{array}$$

On dira que le dioïde Q est muni de l'opération racine `n-ieme` *si la fonction racine n-ième est bien définie pour tout $n > 0$.*
Enfin, si π un chemin de longueur $|\pi|$, on notera : $\tilde{q}(\pi) = \sqrt[|\pi|]{q(\pi)}$ le coût moyen de π.

Par la suite, nous ferons la distinction entre la notion de racine n-ième pour un n donné et la notation « opérateur racine `n-ieme` » qui nous permet de définir une propriété de la racine n-ième pour tout $n > 0$. Cette même distinction sera faite entre « opérateur puissance `n-ieme` » et puissance n-ième pour un $n > 0$ donné.

3.2.3.2 Les dioïdes de coûts : définition

Avant de définir la notion de dioïde de coûts, nous définissons la notion de $\oplus$-morphisme complet. Les fonctions vérifiant cette propriété commutent avec l'opérateur $\oplus$.

Définition 3.8 ($\oplus$-morphisme complet)**.**
Soit $Q(\oplus, \otimes)$ un dioïde complet et $f : Q \to Q$ une fonction.
La fonction f est un $\oplus$-morphisme complet *(abrégé en $\oplus$-`mc`) si on a :*

$$\forall X \subseteq Q, \quad f(\bigoplus_{x \in X} x) \;=\; \bigoplus_{x \in X} f(x) \tag{3.2}$$

Notons que, dans la théorie des dioïdes, cette notion est souvent appelée « semi-continuité inférieure » [BCOQ92]. Nous préférons utiliser l'appellation de $\oplus$-morphisme complet par analogie avec la notion de $\bigsqcup$-morphisme complet définie dans le Chapitre 2. En fait, ces deux notions coïncident dès lors que le dioïde est idempotent complet car l'opérateur $\oplus$ permet alors de définir un treillis complet dont l'opérateur de borne supérieure est $\oplus$ (d'après la Remarque 3.2). Notons que l'égalité 3.2, satisfaite pour tout ensemble X est, a fortiori, vérifiée pour les ensembles X finis. Dans ce cas, on parlera simplement de $\oplus$-*morphisme*.

Nous pouvons maintenant définir la notion de dioïde qui sous-tend notre analyse. La définition suivante résume les conditions vérifiées par notre structure.

Définition 3.9 (Dioïde de coûts)**.**
Un dioïde $Q(\oplus, \otimes)$ est un dioïde de coûts *si les conditions suivantes sont vérifiées.*

- *$Q(\oplus, \otimes)$ est un dioïde idempotent complet et commutatif,*
- *$Q(\oplus, \otimes)$ est muni de l'opérateur racine* `n-ieme`*,*
- *l'opérateur puissance* `n-ieme` *est* $\oplus$-`mc`*.*

On remarque une asymétrie dans cette définition entre l'exigence souhaitée pour l'opérateur racine `n-ieme` (existence) et celle souhaitée pour l'opérateur puissance `n-ieme` (caractère $\oplus$-`mc`). En fait, dans un dioïde de coûts, l'opérateur racine `n-ieme` est aussi $\oplus$-`mc`.

Proposition 1.
Soit Q un dioïde de coût. On a :

(i) L'opérateur racine `n-ieme` *est* $\oplus$-`mc` *:*

$$\forall X \subseteq Q, \forall n > 0, \sqrt[n]{\bigoplus_{x \in X} x} = \bigoplus_{x \in X} \sqrt[n]{x} \tag{3.3}$$

(ii) Pour tout $a, b \in Q$ *et* $n, m > 0$*,*

$$\sqrt[n]{a} \oplus \sqrt[m]{b} \geq \sqrt[n+m]{a \otimes b} \tag{3.4}$$

La propriété (3.3) est une conséquence directe du fait que la puissance puissance `n-ieme` est $\oplus$-`mc`. Il suffit pour cela d'appliquer la racine n-ième à chaque membre de l'égalité :

$$(\bigoplus_{x \in X} \sqrt[n]{x})^n = \bigoplus_{x \in X} x$$

La propriété (3.4) est quant à elle une conséquence des deux lemmes suivants. Le premier est connu sous le nom d'inégalité de Cauchy [DS87, DS92].

Lemme 3.2 (Inégalité de Cauchy)**.**
Soit Q *un dioïde de coûts. On a alors :*

$$\forall n > 0,\ \forall x_1, \cdots, x_n \in Q, \quad x_1 \otimes \cdots \otimes x_n \ \leq\ x_1^n \oplus \cdots \oplus x_n^n$$

Démonstration.
Le produit $x_1 \otimes \cdots \otimes x_n$ apparaît dans le développement de $(x_1 \oplus \cdots \oplus x_n)^n$. Q étant un dioïde de coûts, la puissance n-ième est un $\oplus$-morphisme. Ainsi, nous pouvons écrire $x_1^n \oplus \cdots \oplus x_n^n = (x_1 \oplus \cdots \oplus x_n)^n = x_1 \otimes \cdots \otimes x_n \oplus d$ où d représente le reste du développement. On conclut par définition de la relation d'ordre. □

Lemme 3.3.
Soit Q *un dioïde de coûts. On a :*

$$\forall a, b \in Q, \forall n > 0, \quad a \leq b \ \Leftrightarrow\ a^n \leq b^n$$

Démonstration.
Par double implication.

(⇒) Ce premier sens de l'équivalence est l'expression de la monotonie de la puissance n-ième, qui est une conséquence directe du lemme 3.1.

(⇐) Dans un dioïde de coûts, la racine n-ième est un $\oplus$-morphisme. De ce fait, cet opérateur est monotone [BCOQ92] ce qui permet de conclure ce cas.

□

Montrons maintenant la propriété 3.4 de la proposition 1.

Démonstration.
On procède par équivalence, en appliquant le lemme 3.3 aux deux éléments de Q suivant : $\sqrt[n+m]{a \otimes b}$ et $\sqrt[n]{a} \oplus \sqrt[m]{b}$, et en prenant $mn(m+n)$ (> 0) pour puissance.

$$\begin{array}{rrcl} & \sqrt[n+m]{a \otimes b} & \leq & \sqrt[n]{a} \oplus \sqrt[m]{b} \\ \Longleftrightarrow & a^{mn} \otimes b^{mn} & \leq & a^{m(m+n)} \oplus b^{n(m+n)} \\ \Longleftrightarrow & \underbrace{a^m \otimes \cdots \otimes a^m}_{\text{n fois}} \otimes \underbrace{b^n \otimes \cdots \otimes b^n}_{\text{m fois}} & \leq & (a^m)^{m+n} \oplus (b^n)^{m+n} \end{array}$$

La dernière inégalité est toujours vérifiée dans un dioïde de coût. En effet, c'est une instanciation du lemme 3.2 avec les $m+n$ termes suivants : $x_1 = \cdots = x_n = a^m$ et $x_{n+1} = \cdots = x_{n+m} = b^n$. La propriété (3.4) est équivalente à cette inégalité, ce qui termine la démonstration. □

Notre définition de dioïde de coûts peut paraître plutôt restrictive. Cependant, nous allons montrer qu'elle englobe une large variété de classes de dioïdes présents dans la littérature.

3.2.4 Exemples de dioïdes de coûts

Tout d'abord, nous commençons par rappeler quelques définitions.

Définition 3.10.
Un dioïde $(Q, \oplus, \otimes)$ est dit :
- sélectif *si pour tout élément a, b de Q,*

$$a \oplus b = a \quad \text{ou} \quad a \oplus b = b$$

- double-idempotent *si les deux lois $\oplus$ et $\otimes$ sont idempotentes.*
- intègre *si pour tout a, b, c éléments de Q,*

$$(a \otimes b = a \otimes c \ \wedge \ a \neq \bot) \Rightarrow b = c$$

La propriété de sélectivité implique que deux éléments a, b de Q sont toujours comparables. Autrement dit, un dioïde sélectif induit une structure d'ordre totale.

Comme annoncé, la notion de dioïde de coûts recouvre une large variété de dioïdes. La proposition suivante liste plusieurs catégories de dioïdes de coûts.

Proposition 2.
Soit $(Q, \oplus, \otimes)$ un dioïde complet et commutatif. Pour que Q soit un dioïde de coûts, il suffit que l'une des propriétés suivantes soit vérifiée.

(1) *Q est double-idempotent.*

(2) *Q est sélectif et muni d'un opérateur racine* `n-ieme`.

(3) *Q est idempotent et intègre, et pour tout élément q de Q, pour tout $n > 0$, l'équation $X^n = q$ admet au moins une solution.*

Avant de démontrer cette proposition, nous listons dans la Figure 3.2 quelques exemples de dioïdes des trois types précédents et qui sont donc des dioïdes de coûts.

	ensemble sous-jacent	$\oplus$	$\otimes$	$\sqrt[n]{q}$
Double-idempotent	$\mathbb{Q} \cup \{+\infty, -\infty\}$	min	max	q
	$\mathbb{R} \cup \{+\infty, -\infty\}$	max	min	q
	$\mathcal{P}(S)$	$\cap$	$\cup$	q
	$\mathcal{P}(S)$	$\cup$	$\cap$	q
Sélectif	$\mathbb{R}_+ \cup \{+\infty\}$	max	$\times$	$q^{\frac{1}{n}}$
	$\mathbb{Q} \cup \{+\infty, -\infty\}$	max	$+$	$\frac{q}{n}$
	$\mathbb{R} \cup \{+\infty, -\infty\}$	min	$+$	$\frac{q}{n}$
Intègre	$\mathbb{R}_+^m \cup \{+\infty\}$	min	$+$	$\frac{q}{n}$

FIGURE 3.2 – Quelques exemples de dioïdes de coûts

L'exemple $\overline{\mathbb{R}}(\mathtt{max}, +)$ est un dioïde de coût qui peut être utilisé pour la définition du pire temps d'exécution (WCET) : quand deux états peuvent être joints par différentes suites de transitions de coûts différents, seul le pire temps est considéré. Pour calculer le coût d'une suite de transitions, on somme les coûts de chaque transition. Le dioïde $\mathcal{P}(S)(\cap, \cup)$ est un autre exemple de dioïde de coûts (S étant un ensemble fini ou infini). Dans ce dioïde, le coût pour atteindre un état donne une information sur les éléments de S qui ont été parcourus. Ceci peut être utilisé pour connaître les parties du code qui ont été exécutées.

Passons maintenant à la démonstration de la proposition 2. Afin d'améliorer la lisibilité de cette démonstration, nous utilisons plusieurs lemmes intermédiaires. Tout d'abord, montrons que les dioïdes de type **(1)** sont naturellement munis d'un opérateur racine `n-ieme`.

Lemme 3.4.
Soit $Q(\oplus, \otimes)$ un dioïde double-idempotent. La fonction identité est un opérateur racine `n-ieme` *de Q.*

Démonstration.
Soit $n > 0$. L'opérateur $\otimes$ étant idempotent, l'équation $X^n = q$ admet q comme solution évidente. D'autre part, cette solution est unique car toute solution $\tilde{q}$ vérifie $\tilde{q}^{\,n} = q$ par définition, et aussi $\tilde{q}^{\,n} = \tilde{q}$ par idempotence. $\square$

Pour chacune de ces trois classes de dioïdes, nous avons à montrer que l'opérateur puissance `n-ieme` est $\oplus$-`mc`. En fait, grâce au lemme suivant, il suffit de démontrer que cet opérateur est un $\oplus$-morphisme.

Lemme 3.5.
Dans un dioïde complet et idempotent muni d'une racine `n-ieme`*, la puissance* `n-ieme` *est* $\oplus$-`mc` *si et seulement si c'est un* $\oplus$*-morphisme.*

Démonstration.
Soit $n > 0$. Le sens direct de l'équivalence étant évident, nous montrons qu'être un $\oplus$-morphisme suffit pour être $\oplus$-`mc`. Supposons donc que la puissance n-ième est un $\oplus$-morphisme. La racine n-ième est alors aussi un $\oplus$-morphisme. En effet, on a, pour tout $m > 0$,

$$(\oplus_{i=1}^{m} \sqrt[n]{x_i})^n = \oplus_{i=1}^{m} (\sqrt[n]{x_i})^n = \oplus_{i=1}^{m} x_i$$

En appliquant la racine n-ième aux membres de gauche et de droite de cette égalité, on conclut que cette fonction est un $\oplus$-morphisme. Ceci permet d'affirmer que cette fonction est monotone [BCOQ92].

Soit maintenant X un sous-ensemble non vide de Q. Si $x \in X$, on a :

$$\begin{array}{rcll} x & \leq & \bigoplus_{x\in X} x & \\ x^n & \leq & (\bigoplus_{x\in X} x)^n & \text{(par monotonie de la puissance } n\text{-ième)} \\ (\bigoplus_{x\in X} x^n) & \leq & (\bigoplus_{x\in X} x)^n & \text{(par idempotence)} \end{array}$$

Prouvons maintenant l'inégalité inverse. Si $x \in X$, on a :

$$\begin{array}{rcll} x^n & \leq & \bigoplus_{x\in X} x^n & \\ x & \leq & \sqrt[n]{\bigoplus_{x\in X} x^n} & \text{(par monotonie de la racine } n\text{-ième)} \\ (\bigoplus_{x\in X} x) & \leq & \sqrt[n]{\bigoplus_{x\in X} x^n} & \text{(par idempotence)} \\ (\bigoplus_{x\in X} x)^n & \leq & \bigoplus_{x\in X} x^n & \text{(par monotonie de la puissance } n\text{-ième)} \end{array}$$

Ainsi, $(\bigoplus_{x\in X} x)^n = \bigoplus_{x\in X} x^n$, ce qui prouve, par définition, que la puissance n-ième est $\oplus$-`mc`. □

Pour les dioïdes de type **(3)**, nous devons aussi montrer que la puissance `n-ieme` est un $\oplus$-morphisme [DS87, DS92].

Lemme 3.6.
Soit $Q(\oplus, \otimes)$ *un dioïde idempotent commutatif et intègre. Alors :*

$$\forall n \in \mathbb{N},\ \forall a, b \in Q,\ (a \oplus b)^n = a^n \oplus b^n$$

Démonstration.
Soit $n > 0$. Dans un premier temps, remarquons que si $a = \perp$ alors $(a \oplus b)^n = b^n$. Ainsi, si $a = \perp$ ou $b = \perp$ l'égalité est trivialement satisfaite. Supposons maintenant que $a \neq \perp$ et $b \neq \perp$ et raisonnons par récurrence sur n. Pour $n = 0$ et $n = 1$, la propriété est trivialement satisfaite. On suppose alors $n \geq 1$, et on démontre la propriété au rang $n + 1$. On a :

$$\begin{aligned}
(a \oplus b)^{n+1} \otimes (a \oplus b) &= ((a \oplus b)^n \otimes (a \oplus b)) \otimes (a \oplus b) \\
&= ((a^n \oplus b^n) \otimes (a \oplus b)) \otimes (a \oplus b) && \text{(par hyp. d'ind.)} \\
&= (a^{n+1} \oplus ab^n \oplus a^n b \oplus b^{n+1}) \otimes (a \oplus b) \\
&= a^{n+2} \oplus ab^{n+1} \oplus a^{n+1}b \oplus b^{n+2} \\
&\quad \oplus a^2 b^n \oplus a^n b^2 && (*)
\end{aligned}$$

$$(a^{n+1} \oplus b^{n+1}) \otimes (a \oplus b) = a^{n+2} \oplus ab^{n+1} \oplus a^{n+1}b \oplus b^{n+2} \qquad (**)$$

Il suffit alors de démontrer que les termes $(*)$ et $(**)$ sont égaux pour conclure. En effet, si c'est le cas, nous avons $(a \oplus b)^{n+1} \otimes (a \oplus b) = (a^{n+1} \oplus b^{n+1}) \otimes (a \oplus b)$. Comme $a \oplus b \neq \perp$ et que le dioïde est intègre, on peut simplifier cette égalité à gauche et à droite par $(a \oplus b)$, ce qui montre la propriété au rang $n + 1$. Regardons maintenant de plus près les termes $(*)$ et $(**)$. On a $a^2 b^n \oplus a^n b^2 = ab(ab^{n-1} \oplus a^{n-1}b)$. De plus, par hypothèse d'induction, on a $a^n \oplus b^n = (a \oplus b)^n = ab^{n-1} \oplus a^{n-1}b \oplus (a^n \oplus b^n \oplus (\oplus_{k=2}^{n-2} a^k b^{n-k}))$, ce qui permet d'affirmer que $ab^{n-1} \oplus a^{n-1}b \leq a^n \oplus b^n$. L'opérateur $\otimes$ préservant l'ordre, on peut multiplier chaque membre de cette inégalité par $a \otimes b$. On obtient $a^2 b^n \oplus a^n b^2 \leq a^{n+1}b \oplus ab^{n+1}$ et ainsi $a^2 b^n \oplus a^n b^2 \oplus a^{n+1}b \oplus ab^{n+1} = a^{n+1}b \oplus ab^{n+1}$ par définition de l'ordre dans un dioïde idempotent. Cette égalité suffit pour démontrer l'égalité des termes $(*)$ et $(**)$, ce qui conclut cette démonstration. □

On montre finalement que les dioïdes de type **(3)** sont munis d'une racine n-ième.

Lemme 3.7.
Dans les dioïdes idempotents commutatifs et intègres, si, pour $n > 0$, l'équation $X^n = q$ admet une solution, alors cette solution est unique.

Démonstration.
Soit $n > 0$. Tout d'abord, considérons le cas $q = \perp$. L'équation ci-dessus se réécrit alors $x^n = \perp$. L'unique solution de cette équation est $\lambda = \perp$. En effet, $\lambda = \perp$ est solution, et si $\lambda \neq \perp$ est solution alors $\lambda^n = \perp = \perp \otimes \lambda$, ce qui prouve par intégrité que $\lambda^{n-1} = \perp$ et, en itérant ce procédé, que $\lambda = \perp$. Supposons maintenant que $q \neq \perp$. Si λ_1 et λ_2 sont deux solutions de l'équation $X^n = q$, on a $\lambda_1^n = \lambda_2^n = q$ et $\lambda_1 \neq \perp$, $\lambda_2 \neq \perp$. Comme la puissance n-ième est un $\oplus$-morphisme (lemme 3.6), on peut utiliser le lemme 3.2 pour montrer :

$$\lambda_1^{n-1} \otimes \lambda_2 = \lambda_1 \otimes \cdots \otimes \lambda_1 \otimes \lambda_2 \leq \lambda_1^n \oplus \lambda_2^n = q \oplus q = q = \lambda_1^n$$

c'est à dire

$$\lambda_1^{n-1} \otimes \lambda_2 \leq \lambda_1^n$$

On a alors $\lambda_2 \leq \lambda_1$ par divisions successives par λ_1. En utilisant un raisonnement symétrique, on montre $\lambda_1 \leq \lambda_2$. Ainsi, $\lambda_1 = \lambda_2$ et l'équation $X^n = q$ admet une unique solution. □

Nous pouvons maintenant démontrer la proposition 2.

Démonstration.
Dans un dioïde double-idempotent, l'opérateur puissance `n-ieme` est la fonction identité. Ainsi, cet opérateur est $\oplus$-`mc`, ce qui prouve que les dioïdes de type **(1)** sont des dioïdes de coûts.

Dans le cas des dioïdes sélectifs, pour tout $n > 0$, $(a \oplus b)^n =$ soit a^n ou b^n et ainsi la puissance `n-ieme` est un $\oplus$-morphisme. Ceci permet de conclure, par le lemme 3.5, que les dioïdes de type **(2)** sont des dioïdes de coûts.

Comme les dioïdes de type **(3)** sont munis d'une racine `n-ieme`, les lemmes 3.5 et 3.6 permettent de conclure que la puissance `n-ieme` est $\oplus$-`mc`. Les dioïdes de type **(3)** sont ainsi des dioïdes de coûts. □

3.3 Sémantique linéaire et coût long-run

3.3.1 Matrice de transition et moduloïde

L'utilisation de dioïdes permet d'exprimer les opérations de coûts sous forme de calculs matriciels. L'ensemble des transitions directes (en un pas) peut être représenté par une *matrice de transition* $M \in \mathcal{M}_{\Sigma\times\Sigma}(Q)$ telle que

$$M_{\sigma,\sigma'} = \begin{cases} q \text{ si } \sigma \rightarrow^q \sigma' \\ \perp \text{ sinon} \end{cases}$$

L'ensemble $\mathcal{M}_{\Sigma\times\Sigma}(Q)$ désigne l'ensemble des matrices dont les lignes et colonnes sont indicées par l'ensemble des états Σ et dont les coefficients sont dans Q. On étend, de manière habituelle, les opérateurs $\oplus$ et $\otimes$ de sorte à les faire agir sur $\mathcal{M}_{\Sigma\times\Sigma}(Q)$. Notons au passage que les matrices itérées M^n transportent les coûts des chemins de longueur n. Le dioïde initial Q étant idempotent et complet, $\mathcal{M}_{\Sigma\times\Sigma}(Q)(\oplus,\otimes)$ possède aussi une structure de dioïde idempotent et complet. L'ordre induit par $\oplus$ correspond alors à l'extension point à point de l'ordre sur Q :

$$M \leq M' \quad \Leftrightarrow \quad \forall i,j,\ M_{i,j} \leq M'_{i,j}$$

Notons qu'une matrice de transition peut être vue comme un opérateur linéaire. Cet opérateur agit sur l'ensemble des vecteurs à coefficients dans Q et indicés par

l'ensemble des états Σ. Par la suite, on notera $Q(\Sigma)$ cet ensemble de vecteurs. $Q(\Sigma)$ est muni d'une structure de moduloïde, structure analogue à celle d'espace vectoriel dans laquelle la notion de corps est remplacée par celle de dioïde, ainsi que précisé par la définition suivante.

Définition 3.11 (Moduloïde)**.**
Soit $E(\oplus, \otimes)$ un dioïde commutatif. Un E-moduloïde est un ensemble V muni d'une loi interne $\oplus$ et d'une loi externe $\odot$ telles que :

1. *$V(\oplus)$ est un monoïde commutatif, possédant un élément neutre noté 0 ;*
2. *la loi $\odot$ définie de $E \times V$ sur V, satisfait les axiomes suivants*
   (a) $\forall \lambda \in E, \forall (x, y) \in V^2, \lambda \odot (x \oplus y) = (\lambda \odot x) \oplus (\lambda \odot y)$,
   (b) $\forall (\lambda, \mu) \in E^2, \forall x \in V, (\lambda \oplus \mu) \odot x = (\lambda \odot x) \oplus (\mu \odot x)$,
   (c) $\forall (\lambda, \mu) \in E^2, \forall x \in V, \lambda \odot (\mu \odot x) = (\lambda \otimes \mu) \odot x$,
   (d) $\forall x \in V, \mathsf{e} \odot x = x$ *et* $\perp \odot x = 0$,
   (e) $\forall \lambda \in E, \lambda \odot 0 = 0$.

Comme dans le cas des espaces vectoriels, si n est un entier, E^n, l'ensemble des vecteurs à n coefficients dans E est un moduloïde. De manière plus générale, un vecteur $u \in E(\Sigma)$, où Σ est un ensemble fini de cardinal n ($|\Sigma| = n$) peut être vu comme une fonction $\delta_u : [1, n] \to E$. Cette notion de vecteurs peut-être généralisée dans le cas où l'ensemble Σ est infini (dénombrable) : δ_u est alors une fonction de $\mathbb{N}$ dans E. On peut définir de la même façon des matrices indicées par un ensemble infini. Le produit d'une matrice par un vecteur est alors défini par : $(Mu)_i = \bigoplus_{j=1}^{+\infty} \delta_M(i, j) \otimes \delta_u(j)$. Par la suite, pour ne pas alourdir les notations, nous utiliserons la notation matricielle même dans le cas d'un ensemble infini d'indices.

Notons que si $E(\oplus, \otimes)$ est un dioïde commutatif, tout E-moduloïde V est naturellement muni d'une structure d'ordre donnée par l'extension de l'ordre induit par $\oplus$.

D'autre part, nous avons vu que, pour tout dioïde idempotent complet $Q(\oplus, \otimes)$, l'ensemble $\mathcal{M}_{\Sigma \times \Sigma}(Q)$ est muni d'une structure de dioïde idempotent complet. Si la loi $\otimes$ est commutative, cet ensemble $\mathcal{M}_{\Sigma \times \Sigma}(Q)$ est aussi un Q-moduloïde. Ces deux structures sont ordonnées par l'extension de l'ordre induit par la loi $\oplus$.

Lorsqu'un programme à coûts est donné directement par sa forme matricielle et non sous forme de relation de transition étiquetée, nous parlerons de *programme quantitatif linéaire.*

Définition 3.12 (Programme quantitatif linéaire)**.**
Soit $Q(\oplus, \otimes)$ un dioïde de coûts.

Un programme quantitatif linéaire *est un quadruplet $T = <\Sigma, I, Q, M>$ tel que :*

– *Σ désigne un ensemble dénombrable d'états,*
– *$I \subseteq \Sigma$ est l'ensemble des états initiaux du programme P,*
– *$Q(\oplus, \otimes)$ est un dioïde de coûts représentant les coûts du programme,*
– *$M \in \mathcal{M}_{\Sigma\times\Sigma}(Q)$ est une matrice de transition à coefficients dans Q.*

Le coefficient $M_{\sigma,\sigma'}$ de la matrice M représente une transition de l'état σ à l'état σ' de coût q.

Nous avons vu dans cette section que les coûts des transitions directes peuvent être récapitulés et transportés dans une matrice de transition. Nous allons maintenant utiliser cette matrice pour définir une notion de coût long-run sur l'ensemble du programme.

3.3.2 Coût long-run

Cachera, Jensen & Sotin ont proposé le calcul d'un coût global défini comme le « maximum » (obtenu par l'opérateur $\oplus$) de l'ensemble des coûts des exécutions finies du programme [SCJ06]. Plus précisément, ce coût n'étant pas calculable en général, les auteurs proposent d'effectuer un calcul approché permettant d'obtenir une sur-approximation de ce coût.

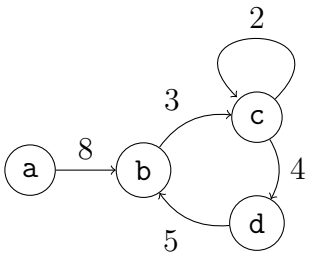
Ce coût n'est pas forcément satisfaisant. Tout d'abord, ce coût est relativement « gros ». Si, de plus, il est calculé en utilisant une abstraction trop grossière, le calcul de ce coût rendra $\top$. Ceci n'apporte pas d'information concernant le coût réel du programme. D'autre part, ce coût n'est pas adapté aux programmes qui ne sont pas censés terminer tels que les systèmes réactifs.

Dans ce qui suit, nous proposons un coût qui répond à ces deux attentes, nommé le coût *long-run* , et qui représente le maximum des coûts moyens des cycles du programme. Cette terminologie est tirée de [Alf98, BEK05], dans le contexte des processus probabilistes modélisés par des processus de décision markoviens.

Nous nous intéressons au coût moyen « maximum » de tous les cycles du graphe : cette quantité est le *coût long-run* . Nous reprenons l'exemple déjà utilisé précédemment pour illustrer ce coût.

Grâce aux propriétés des dioïdes que l'on étudie, la matrice M^k transporte les coûts des transitions de tous les chemins de longueur k. La diagonale de cette matrice contient alors les coûts des cycles de longueur k. La trace de cette matrice calcule donc le maximum des coûts des cycles de longueur k. Le coût long-run est défini en utilisant cette trace.

Définition 3.13 (Coût long-run)**.**
Soit $T = <\Sigma, I, Q, M>$ un programme quantitatif linéaire.
Soit R la restriction de M à l'ensemble des états Σ_I, accessibles depuis I.



Coût moyen du chemin `abc` $= (8+3)/2 = 5.5$
Coût moyen du cycle `bcdb` $= (3+4+5)/3 = 4$
Coût moyen du cycle `bccdb` $= 14/4 = 3.5$
Coût moyen du cycle `cc` $= 2/1 = 2$
Coût long-run $= 4$

FIGURE 3.3 – Calcul du coût long-run sur un graphe élémentaire

Le coût long-run *de T est défini par :*

$$\boxed{\rho(T) = \bigoplus_{k=1}^{|\Sigma_I|} \sqrt[k]{tr\, R^k}} \quad \textit{où} \quad tr\, R = \bigoplus_{i=1}^{|\Sigma_I|} R_{i,i}$$

Notons au passage que cette définition est valide même lorsque l'on considère un nombre infini d'états. Ceci provient du fait que l'on travaille dans des dioïdes complets. Par exemple, si nous travaillons dans le dioïde $(Time, \mathtt{max}, +)$, où $Time$ est isomorphe à $\overline{\mathbb{R}}$, $\rho(T)$ est le maximum des temps moyens passés par instruction, la moyenne étant calculée sur chaque cycle en divisant le temps total passé dans le cycle par le nombre d'instructions du cycle. Dans le cas où l'ensemble des états est fini, le coût long-run est calculable. Il est important de noter que, dans le cas où la matrice est irréductible, la définition du coût long-run correspond à la plus grande valeur propre de la matrice [CTCG+98]. D'un point de vue implémentation, cette propriété est fondamentale puisqu'elle nous fournit un algorithme effectif de calcul de coût long-run.

Dans cette définition, le terme $\sqrt[k]{tr\, R^k}$ représente le coût moyen du maximum des coûts des cycles de longueur k. Ceci peut sembler étonnant et ne pas être en accord avec la définition plus informelle « maximum des coûts moyens des cycles ». La proposition suivante permet d'éclaircir ce point.

Proposition 3.
Soit $T = <\Sigma, I, Q, q>$ un programme quantitatif.
Soit Γ l'ensemble des cycles de T.
On a alors :

$$\boxed{\rho(T) \;=\; \bigoplus_{c \in \Gamma} \tilde{q}(c)}$$

Afin d'améliorer la lisibilité des preuves, nous commençons par établir les lemmes 3.8 et 3.9.

Rappelons que, par définition, un cycle c est un chemin fini qui commence et finit par le même état, et contient au moins une transition. Comme c est défini

par rapport à la sémantique de trace, il contient seulement des états de l'ensemble dénombrable Σ des états accessibles de la sémantique. Notons $\Gamma_{\leq|\Sigma|}$ l'ensemble des cycles dont la longueur est plus petite ou égale à $|\Sigma|$. Les lemmes suivants permettent de montrer que ces cycles sont les seuls à considérer. Ceci provient essentiellement du fait que les cycles plus grands se décomposent par rapport à ces derniers.

Lemme 3.8.
Soit $T = <\Sigma, I, Q, q>$ un programme quantitatif.
Soit Γ l'ensemble des cycles de T.

$$\forall c \in \Gamma, \quad \exists \Gamma_c \ \subseteq \ \Gamma_{\leq|\Sigma|}, \quad \tilde{q}(c) \leq \bigoplus_{c_e \in \Gamma_c} \tilde{q}(c_e)$$

Démonstration.
Par récurrence forte sur la longueur de c.

- Si $|c| \leq |\Sigma|$ (vérifié en particulier si Σ est infini), alors en posant $\Gamma_c = \{c\}$ l'inégalité est trivialement vérifiée.
- Si $|c| > |\Sigma|$, alors il existe un état σ' tel que

$$c = \sigma \overset{\pi_1}{\Rightarrow} \sigma' \overset{\pi_2}{\Rightarrow} \sigma' \overset{\pi_3}{\Rightarrow} \sigma \quad \text{avec} \begin{cases} \pi_1\pi_3 \in \Gamma & \wedge \quad |\pi_1\pi_3| < |c| \\ \pi_2 \in \Gamma & \wedge \quad |\pi_2| < |c| \end{cases}$$

Avec ces notations, on a

$$\begin{aligned} \tilde{q}(c) = \tilde{q}(\pi_1\pi_2\pi_3) &= \sqrt[|c|]{q(\pi_1\pi_2\pi_3)} \\ &= \sqrt[|c|]{q(\pi_1\pi_3) \otimes q(\pi_2)} && (3.5) \\ &\leq \sqrt[|\pi_1\pi_3|]{q(\pi_1\pi_3)} \oplus \sqrt[|\pi_2|]{q(\pi_2)} = \tilde{q}(\pi_1\pi_3) \oplus \tilde{q}(\pi_2) && (3.6) \end{aligned}$$

L'égalité (3.5) est justifiée par le caractère commutatif de $\otimes$. L'inégalité (3.6) est une conséquence de la propriété (3.4) de la proposition 1. L'hypothèse de récurrence s'applique aux cycles $\pi_1\pi_3$ et π_2 fournissant les ensembles $\Gamma_{\pi_1\pi_3}$ et Γ_{π_2}. On conclut cette preuve en posant :

$$\Gamma_c = \Gamma_{\pi_1\pi_3} \cup \Gamma_{\pi_2}$$

□

Lemme 3.9.
Soit $T = <\Sigma, I, Q, q>$ un programme quantitatif.
Soit Γ l'ensemble des cycles de T.

$$\bigoplus_{c \in \Gamma} \tilde{q}(c) \ = \ \bigoplus_{c_e \in \Gamma_{\leq|\Sigma|}} \tilde{q}(c_e)$$

Démonstration.
Remarquons tout d'abord que :

$$\forall c \in \Gamma, \quad \tilde{q}(c) \ \leq \bigoplus_{c_e \in \Gamma_c} \tilde{q}(c_e) \leq \bigoplus_{c_e \in \Gamma_{\leq |\Sigma|}} \tilde{q}(c_e)$$

La première inégalité provient du lemme 3.8. La seconde est une conséquence de l'inclusion $\Gamma_c \subseteq \Gamma_{\leq|\Sigma|}$ et du fait que, pour tout a, b, c éléments du dioïde de coût, on a $a \leq b \Rightarrow a \leq b \oplus c$.

On somme alors les membres de l'inégalité sur tous les éléments c de Γ. Par idempotence de $\oplus$, on obtient que le maximum des coûts moyens sur tous les cycles est majoré par le maximum des coûts moyens sur un ensemble plus petit de cycles.

$$\bigoplus_{c \in \Gamma} \tilde{q}(c) \ \leq \bigoplus_{c_e \in \Gamma_{\leq |\Sigma|}} \tilde{q}(c_e) \tag{3.7}$$

Comme $\Gamma_{\leq|\Sigma|} \subseteq \Gamma$, l'inégalité est aussi vérifiée dans l'autre sens :

$$\bigoplus_{c_e \in \Gamma_{\leq |\Sigma|}} \tilde{q}(c_e) \ \leq \bigoplus_{c \in \Gamma} \tilde{q}(c) \tag{3.8}$$

En combinant les deux inégalités, on obtient l'égalité souhaitée. □

Montrons maintenant la proposition 3.

Démonstration.
En regroupant tous les cycles de même longueur et contenant le même nœud σ, on obtient une partition de l'ensemble $\Gamma_{\leq|\Sigma|}$. Notons que $n \geq 1$, et que l'état σ est accessible. Nous notons $\mathcal{C}_n^\sigma$ un tel ensemble.

$$\Gamma_{\leq|\Sigma|} = \bigcup_{\substack{n \leq |\Sigma| \\ \sigma \in \Sigma}} \mathcal{C}_n^\sigma$$

Ainsi, l'égalité du lemme 3.9 permet d'affirmer :

$$\begin{aligned}\bigoplus_{c \in \Gamma_{\leq|\Sigma|}} \tilde{q}(c) = \bigoplus_{c \in \Gamma_{\leq|\Sigma|}} \sqrt[|c|]{q(c)} &= \bigoplus_{n \leq |\Sigma|} \bigoplus_{\sigma \in \Sigma} \bigoplus_{c \in \mathcal{C}_n^\sigma} \sqrt[n]{q(c)} \\ &= \bigoplus_{n \leq |\Sigma|} \sqrt[n]{\bigoplus_{\sigma \in \Sigma} \bigoplus_{c \in \mathcal{C}_n^\sigma} q(c)}\end{aligned}$$

La dernière égalité est obtenue par le caractère $\oplus$-`mc` de la racine n-ième (Propriété (3.3) de la proposition 1).

Si σ est un état accessible, alors tous les états des chemins débutant par σ le sont aussi. Notons R la restriction de la matrice M aux indices de Σ_I (ensemble des états accessibles depuis I). Nous avons alors :

$$\begin{aligned}
R^n_{\sigma,\sigma} &= \bigoplus_{c \in \mathcal{C}^\sigma_n} q(c) \\
\bigoplus_{c \in \Gamma} \sqrt[|c|]{q(c)} &= \bigoplus_{n \leq |\Sigma|} \sqrt[n]{\bigoplus_{\sigma \in \Sigma} R^n_{\sigma,\sigma}} \\
&= \bigoplus_{n \leq |\Sigma|} \sqrt[n]{tr\, R^n} \\
&= \rho(T)
\end{aligned}$$

□

Comme nous souhaitons donner une caractérisation du comportement asymptotique d'un programme, on aurait pu définir le coût long-run grâce à la définition alternative suivante :

$$lrc(T) = \limsup_{n \to \infty} \bigoplus_{\substack{t \in [\![T]\!]_{tr} \\ |t| = n}} \tilde{q}(t)$$

Au lieu de définir le coût long-run par rapport aux cycles du programme, cette définition considère les traces arbitrairement longues. Contrairement à $\rho(T)$, la définition $lrc(T)$ n'est pas calculable, même lorsque l'ensemble des états est fini. Nous montrons ci-dessous que ces deux notions coïncident pour une classe restreinte de dioïdes de coûts lorsque l'on considère un ensemble fini d'états.

3.3.2.1 Quand les traces rejoignent les cycles

Le but de cette section est de faire le lien entre la définition 3.13 qui décrit le coût long-run comme le maximum des coûts moyens des cycles et la dénomination de ce coût : coût « longue exécution », *c.à.d.* un coût évalué sur les traces de plus en plus grandes du programme (on fait tendre leur longueur vers plus l'infini). Nous allons montrer, dans un cadre restreint, que ces deux notions coïncident. Plus précisément, on suppose que l'ensemble des états Σ est fini, et l'on se place dans un dioïde dont l'ensemble sous-jacent est $\overline{\mathbb{R}}$ et dont la loi $\otimes$ est l'opérateur arithmétique $+$ (la racine n-ième correspond de ce fait à la division par n). Afin d'établir ce résultat, nous devons montrer que le coût de tout préfixe fini d'une trace devient négligeable quand cette trace devient arbitrairement longue. Nous devons donc imposer l'hypothèse suivante.

Hypothèse 1.
Toute transition δ qui n'est pas dans un cycle vérifie $q(\delta) \neq +\infty$.

L'hypothèse 1 permet d'exclure le cas pathologique des matrices de transition dont certains coefficients représenteraient des coûts infinis. Remarquons que cette hypothèse n'exclut pas la présence de coût infini dans les cycles. Cette propriété n'est pas nécessaire pour la démonstration puisqu'une telle présence est répercutée sur la valeur de ρ.

Théorème 3.1.
Soit $T = <\Sigma, I, Q, M>$ un programme quantitatif linéaire tel que :
- *Σ est un ensemble fini d'états,*
- *l'ensemble sous-jacent du dioïde Q est $\overline{\mathbb{R}}$,*
- *la loi $\otimes$ du dioïde Q est l'opérateur arithmétique $+$.*

Alors, sous l'hypothèse 1, on a :

$$\rho(T) \;=\; \lim_{n\to\infty} \bigoplus_{\substack{t\in[\![T]\!]_{tr}\\ |t|=n}} \tilde{q}(t)$$

Démonstration.
Soit $[\![T]\!]^n_{tr}$ le sous-ensemble des traces de longueur n appartenant à la sémantique de trace de T. Nous montrons le théorème en bornant la quantité $\bigoplus_{t\in[\![T]\!]^n_{tr}} \tilde{q}(t)$. Nous écrivons la preuve en utilisant l'arithmétique « usuelle » afin qu'elle soit plus lisible et intuitive. La loi $\otimes$ est définie par l'opérateur $+$, de telle sorte que la multiplication $q \otimes \cdots \otimes q$ de n fois la même valeur q produit $n.q$. La racine n-ième correspond alors à l'opérateur de division par n. La loi $\oplus$ et son ordre associé $\leq$ peut être interprété par `max` et $\leq$ ou par `min` et $\geq$.

Nous traitons d'abord le cas où la valeur $\top$ apparaît dans les cycles (accessibles) de la sémantique de trace. Dans ce cas, il existe une longueur n_0 telle que, pour tout n plus grand que n_0, il est possible de construire une trace de longueur n qui inclut cette transition de coût infini. Le coût global et moyen de cette trace sont aussi égaux à $\top$, et il en est alors de même pour $\rho(T)$. L'égalité souhaitée est alors trivialement vérifiée.

Plaçons-nous maintenant dans le cas où $\top$ n'apparaît pas dans l'ensemble des transitions accessibles de la sémantique. On s'intéresse à la quantité $\bigoplus_{t\in[\![T]\!]^n_{tr}} \frac{q(t)}{|t|}$. Notons N la longueur de la plus grande trace de $[\![T]\!]_{tr}$ qui ne contient pas de

cycle [2]. Toute trace t de taille $n > N$ peut être décomposée sous la forme suivante :

$$t = p_1.c_1 \dots c_{k-1}.p_k \text{ où } \begin{cases} c_i \text{ est un cycle} \\ r = \sum_i |p_i| \leq N \\ p_1.\dots.p_k \text{ est acyclique} \end{cases}$$

Pour simplifier les notations, nous traitons le cas où $k = 1$, le cas général étant traité de la même façon. La trace t peut ainsi être écrite

$$t = p.c.s \text{ où } \begin{cases} c \text{ est un cycle} \\ r = |p| + |s| \leq N \\ p.s \text{ est acyclique} \end{cases}$$

On cherche à majorer le coût $q(t)$. Pour ce faire, on va considérer q_- (resp. q_+) le coût minimum (resp. maximum) des transitions accessibles de T. L'existence de ces extrema est assuré par le fait que l'ensemble Σ est fini. Notons que par définition des transitions $q_- \neq \bot$ et, par hypothèse, $q_+ \neq \top$. Grâce à la proposition 3, on sait que le coût moyen de tout cycle de T est majoré par $\rho(T)$, le coût long-run de T. On a alors

$$q(t) = (q(p) + q(s)) + q(c) \leq r.q_+ + (n - r).\rho(T) \tag{3.9}$$

Par définition, r et $n - r$ sont des entiers positifs. On a donc

$$q(t) = (q(p) + q(s)) + q(c) \leq N.|q_+| + (n - A).\rho(T) \tag{3.10}$$

où la valeur de A est donnée par N ou $-N$ en fonction du signe de $\rho(T)$. En sommant (3.10) pour tout t de taille n, on obtient

$$\bigoplus_{t \in [\![T]\!]^n_{tr}} q(t) \leq N.|q_+| + (n - A).\rho(T) \tag{3.11}$$

Cherchons maintenant une borne inférieure.

Puisque l'on se restreint à l'ensemble des états accessibles, on remarque que pour tout n, il existe une trace dans $[\![T]\!]^n_{tr}$ telle que tout cycle de cette trace est critique *c.à.d.* de coût moyen égal à $\rho(T)$. Considérons $t_{\mathtt{max}}$ l'une des traces vérifiant cette propriété. Comme précédemment, on écrit $t_{\mathtt{max}}$ sous la forme $t_{\mathtt{max}} = p.c.s$ (le cas général se traitant de la même façon), où p et s sont des chemins acycliques et c est seulement composé de cycles critiques. On a alors

$$q(t_{\mathtt{max}}) \leq \bigoplus_{t \in [\![T]\!]^n_{tr}} q(t) \tag{3.12}$$

2. Il convient de noter que N est bien défini car nous avons supposé que l'ensemble des états Σ est fini.

Cette inégalité est trivialement vérifiée puisque $t_{\texttt{max}} \in [\![T]\!]^n_{tr}$ et $\oplus$ est idempotente. Minorons maintenant le coût de cette trace.

$$\begin{aligned} q(t_{\texttt{max}}) &= (q(p) + q(s)) + q(c) \\ q(t_{\texttt{max}}) &\geq N.|q_-| + (n - A).\rho(T) \end{aligned} \tag{3.13}$$

Cette inégalité est une conséquence du fait que $r.q_-$ minore $q(p.s)$, et que $(n - r).\rho(T)$ est le coût exact du chemin c. En combinant les inégalités (3.12) et (3.13), on obtient que :

$$N.|q_-| + (n - A).\rho(T) \leq \bigoplus_{t \in [\![T]\!]^n_{tr}} q(t) \tag{3.14}$$

On a ainsi prouvé que :

$$N.|q_-| + (n - A).\rho(T) \leq \bigoplus_{t \in [\![T]\!]^n_{tr}} q(t) \leq N.|q_+| + (n - A).\rho(T)$$

En divisant chaque membre de cette inégalité par n, on obtient

$$F(n) = \frac{N.|q_-|}{n} + \frac{n - A}{n}.\rho(T) \leq \frac{\bigoplus_{t \in [\![T]\!]^n_{tr}} q(t)}{n} \leq \frac{N.|q_+|}{n} + \frac{n - A}{n}.\rho(T) = G(n)$$

Finalement, comme $\frac{a}{n} \oplus \frac{b}{n} = \frac{a \oplus b}{n}$, on obtient :

$$F(n) \leq \bigoplus_{t \in [\![T]\!]^n_{tr}} \frac{q(t)}{n} \leq G(n) \tag{3.15}$$

Étudions maintenant le comportement asymptotique de (3.15).

$$\lim_{n \to \infty} F(n) = \lim_{n \to \infty} G(n) = \rho(T)$$

Ceci implique :

$$\lim_{n \to \infty} \bigoplus_{t \in [\![T]\!]^n_{tr}} \frac{q(t)}{n} = \rho(T)$$

c.à.d. en utilisant les termes propres aux dioïdes

$$\lim_{n \to \infty} \bigoplus_{t \in [\![T]\!]^n_{tr}} \sqrt[n]{q(t)} = \rho(T)$$

ce qui permet de conclure la preuve. □

Tous les dioïdes ne vérifient pas toujours l'égalité du théorème 3.1. On peut citer, par exemple, le dioïde $(\mathcal{P}(S), \cap, \cup)$ qui peut être utilisé pour analyser si certaines portions de code sont sûres d'être exécutées. Le diagramme de la figure 3.4 représente un programme T dont le coût long-run $\rho(T)$ ne coïncide pas avec le coût asymptotique de T. En effet, dans cet exemple, on a $\rho(T) = \{\mathtt{a}\} \oplus \{\mathtt{b}\} = \{\mathtt{a}\} \cap \{\mathtt{b}\} = \emptyset$, alors que toute trace suffisamment grande contient a, ce qui implique que leur somme (*c.à.d.* leur intersection) contient ce coût (le nœud marqué en noir est l'état initial du graphe).

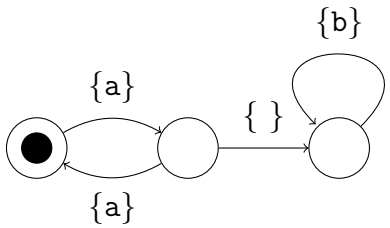


FIGURE 3.4 – Programme T constituant un contre-exemple au théorème 3.1 dans le dioïde $(\mathcal{P}(S), \cap, \cup)$

3.4 Abstraire un programme à coûts

La matrice de transition représentant un programme est, de manière générale, de dimension infinie. Ainsi, sa trace ne peut être calculée en temps fini. Le coût long-run n'est donc pas calculable. Pour résoudre ce problème, nous définissons une matrice abstraite qui sera utilisée pour obtenir un coût approché du coût long-run. Dans cette section, nous présentons ce cadre d'abstraction linéaire. Il correspond à une reformulation la théorie de l'interprétation abstraite du Chapitre 2 en termes d'opérateurs linéaires agissant sur des moduloïdes. Nous montrons aussi comment ce cadre nous permet de définir une sur-approximation correcte du coût long-run.

3.4.1 Connexions de Galois dans les moduloïdes

Nous avons vu dans le Chapitre 2 que les connexions de Galois étaient un outil essentiel pour exprimer la correction des analyses. Dans le cadre des dioïdes idempotents, la notion de connexion de Galois est généralement nommée théorie de la résiduation.

Proposition 4 (Résiduation [BCOQ92]).
Soient $E(\oplus_1, \otimes_1)$ et $F(\oplus_2, \otimes_2)$ deux dioïdes idempotents complets et $f : E \to F$

une fonction monotone.
Nous appelons sous-solution de l'équation $f(x) = b$ un élément y tel que $f(y) \leq_2 b$. Les propositions suivantes sont équivalentes :

(Res1) *Pour tout élément $b \in F$, l'équation $f(x) = b$ admet une plus grande sous-solution.*

(Res2) *La fonction f est un $\oplus$-morphisme complet strict :*

$$f \text{ est } \oplus\text{-}\mathtt{mc} \quad \text{et} \quad f(\bot_E) = \bot_F$$

(Res3) *Il existe une fonction $f^\dagger : F \to E$ monotone et $\wedge$-$\mathtt{mc}$ telle que*[3] *:*

$$\begin{array}{lll} f \circ f^\dagger & \leq_2 & Id_F \\ f^\dagger \circ f & \geq_1 & Id_E \end{array}$$

Par conséquence, $f^\dagger$ est unique. Lorsque f satisfait l'une de ces propriétés, elle est dite résiduée*, et $f^\dagger$ est appelée sa* résiduée*.*

Le terme *résiduation* est fréquent en théorie des Systèmes à Événements Discrets (SED), théorie où sont étudiés les propriétés des algèbres « Max Plus ». Cependant, il ne faut pas s'y tromper : la théorie de la résiduation de la théorie SED correspond à la notion de connexion de Galois en théorie de l'ordre [EKMS92]. La différence de vocabulaire provient de la différence de point de vue entre ces deux théories : en théorie SED l'élément de base est le couple $(\oplus, \otimes)$ et l'ordre n'est qu'une propriété induite par la loi $\oplus$ alors que la théorie de l'ordre, comme son nom l'indique, a comme point de départ un ordre $\leq$. Au final, la Proposition 4 n'est rien d'autre qu'une réécriture du théorème 2.7 pour une instance particulière de treillis complet (structure induite par la structure de dioïde idempotent complet). La propriété **(Res1)** correspond à la propriété **(Gal5)** du théorème 2.7. La propriété **(Res2)** correspond quant à elle à la propriété **(Gal4)** de ce même théorème 2.7. Enfin, la propriété **(Res3)** correspond à la propriété **(Gal2)** du théorème 2.6.

Remarque 3.3.
La proposition 4 peut être exprimée sous une forme moins restrictive en prenant comme hypothèse des ordres partiels complets quelconques [GM01]. De ce fait, nous pourrons utiliser cette proposition dans le cadre particulier de la structure de moduloïde issue d'un dioïde de coûts. Autrement dit, pour un dioïde de coûts $Q(\oplus, \otimes)$ donné, pour tout couple d'ensembles $(\Sigma, \Sigma^\sharp)$, pour toute fonction $f : Q(\Sigma) \to Q(\Sigma^\sharp)$ définie sur les Q-moduloïdes $Q(\Sigma)$ et $Q(\Sigma^\sharp)$, les propriétés **(Res1)**, **(Res2)** *et* **(Res3)** *sont vérifiées.*

3. Rappelons que le cadre de dioïde idempotent complet assure l'existence d'un opérateur d'intersection $\wedge$ (Remarque 3.2).

3.4.2 Relèvement des abstractions

Dans cette section, nous montrons comment exprimer les notions d'abstraction et de concrétisation dans le cadre des moduloïdes. Par la suite, Σ désignera un ensemble d'états *concrets* et $\Sigma^\sharp$ un ensemble d'états *abstraits*. On appellera alors *fonction d'abstraction*, toute fonction $\alpha : \Sigma \to \Sigma^\sharp$ définie de l'ensemble des états concrets dans l'ensemble des états abstraits. Nous insistons sur le fait que l'on ne suppose ici aucune propriété particulière sur les ensembles Σ et $\Sigma^\sharp$ et sur les fonctions d'abstraction.

La définition suivante montre comment toute fonction d'abstraction α peut être relevée en un opérateur linéaire $\alpha^\uparrow \in \mathcal{M}_{\Sigma^\sharp \times \Sigma}(Q)$.

Définition 3.14 (Relèvement linéaire)**.**
Soit $Q(\oplus, \otimes)$ un dioïde de coûts et soient Σ et $\Sigma^\sharp$ deux ensembles et $\alpha : \Sigma \to \Sigma^\sharp$. On appelle relèvement linéaire *de la fonction α, la fonction $\alpha^\uparrow \in \mathcal{M}_{\Sigma^\sharp \times \Sigma}(Q)$ suivante :*

$$\alpha^\uparrow_{\sigma^\sharp,\sigma} = \begin{cases} \mathsf{e} \text{ si } \alpha(\sigma) = \sigma^\sharp \\ \bot \text{ sinon} \end{cases}$$

(on rappelle que e *désigne l'élément neutre de la loi $\otimes$)*

En figure 3.5, nous illustrons la définition du relèvement sur un petit exemple pour lequel Σ et $\Sigma^\sharp$ sont finis.

$$\begin{cases} \alpha(\sigma_1) &= {\sigma_3}^\sharp \\ \alpha(\sigma_2) &= {\sigma_1}^\sharp \\ \alpha(\sigma_3) &= {\sigma_3}^\sharp \\ \alpha(\sigma_4) &= {\sigma_2}^\sharp \\ \alpha(\sigma_5) &= {\sigma_2}^\sharp \end{cases} \qquad \alpha^\uparrow = \begin{pmatrix} \bot & \mathsf{e} & \bot & \bot & \bot \\ \bot & \bot & \bot & \mathsf{e} & \mathsf{e} \\ \mathsf{e} & \bot & \mathsf{e} & \bot & \bot \end{pmatrix}$$

FIGURE 3.5 – Une abstraction α entre $\Sigma = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5\}$ et $\Sigma^\sharp = \{{\sigma_1}^\sharp, {\sigma_2}^\sharp, {\sigma_3}^\sharp\}$ et son relèvement linéaire $\alpha^\uparrow$

Dans ce qui suit, on notera de manière indifférenciée $\leq$ l'ordre défini sur $\mathcal{M}_{\Sigma \times \Sigma}(Q)$ et celui défini sur $\mathcal{M}_{\Sigma^\sharp \times \Sigma^\sharp}(Q)$. Rappelons que ces ordres sont obtenus par extension point à point de l'ordre défini sur Q. Il est important de noter que nous ne supposons pas de structure d'ordre sur l'ensemble des états concrets Σ ni sur l'ensemble des états abstraits $\Sigma^\sharp$.

Théorème 3.2.
Soient Σ et $\Sigma^\sharp$ les domaines concrets et abstraits d'états et $\alpha : \Sigma \to \Sigma^\sharp$. Soit

$\alpha^\uparrow \in \mathcal{M}_{\Sigma^\sharp \times \Sigma}(Q)$ la fonction linéaire obtenue par relèvement de la fonction α. Alors, il existe une unique fonction monotone $\gamma^\uparrow$ telle que :

$$\begin{array}{rcl} \alpha^\uparrow \circ \gamma^\uparrow & \leq & Id_{\Sigma^\sharp} \\ \gamma^\uparrow \circ \alpha^\uparrow & \geq & Id_\Sigma \end{array}$$

où Id_Σ (resp. $Id_{\Sigma^\sharp}$) désigne la matrice identité de $\mathcal{M}_{\Sigma\times\Sigma}(Q)$ (resp. $\mathcal{M}_{\Sigma^\sharp\times\Sigma^\sharp}(Q)$).

Démonstration.
Le dioïde Q étant complet, les ordres définis sur les moduloïdes $Q(\Sigma)$ et sur $Q(\Sigma^\sharp)$ par extension de l'ordre défini sur Q, sont complets. D'autre part, la fonction d'abstraction étant linéaire les hypothèses **(Res2)** de la Proposition 4 (version moduloïde) sont trivialement vérifiées. Le résultat souhaité est donc démontré en prenant $\gamma^\uparrow = (\alpha^\uparrow)^\dagger$.

□

Si l'on suppose α surjective[4], la matrice $\alpha^\uparrow$ est particulièrement simple : elle possède un unique élément différent de $\perp$ par colonne et au moins un élément e par ligne. De ce fait, la matrice $\gamma^\uparrow$ est tout simplement la transposée de la matrice $\alpha^\uparrow$.

3.4.3 Sémantique abstraite induite

Soit $T = <\Sigma, I, Q, M>$ un programme linéaire. Nous souhaitons définir un système de transition abstrait sur le domaine abstrait $\Sigma^\sharp$ qui soit « compatible » avec T, du point de vue de ses traces ainsi que des coûts que nous serons amenés à calculer. La définition suivante d'une abstraction correcte permet d'assurer que le coût long-run sera correctement sur-approché durant le processus d'abstraction.

Définition 3.15 (Abstraction linéaire correcte)**.**
Soit $T = <\Sigma, I, Q, M>$ un programme quantitatif linéaire.
Soit $T^\sharp = <\Sigma^\sharp, I^\sharp, Q, M^\sharp>$ un programme quantitatif linéaire sur le domaine abstrait $\Sigma^\sharp$ et soit α une abstraction de Σ dans $\Sigma^\sharp$.
Le triplet $(T, T^\sharp, \alpha)$ est une abstraction linéaire correcte *si :*

$$\alpha^\uparrow \circ M \leq M^\sharp \circ \alpha^\uparrow$$
$$et \quad \{\alpha(\sigma) \mid \sigma \in I\} \subseteq I^\sharp$$

Le cadre classique de l'interprétation abstraite permet de définir la notion de meilleure abstraction correcte pour une sémantique concrète donnée. De la même manière, partant d'une fonction d'abstraction α et d'une sémantique concrète linéaire, nous pouvons définir une notion de meilleure sémantique abstraite, correcte par construction. La proposition suivante détaille ce résultat.

4. Si α n'est pas surjective, il suffit de considérer sa restriction sur son image.

Proposition 5 (Meilleure abstraction linéaire)**.**
Soit $T = <\Sigma, I, Q, M>$ un programme quantitatif linéaire.
Soit $T^\sharp = <\Sigma^\sharp, I^\sharp, Q, M^\sharp>$ un programme quantitatif linéaire tel que :

$$M^\sharp = \alpha^\uparrow \circ M \circ \gamma^\uparrow \quad \textit{et} \quad I^\sharp = \{\alpha(\sigma) \mid \sigma \in I\}$$

Alors $(T, T^\sharp, \alpha)$ est une abstraction linéaire correcte.
De plus, si T et α sont donnés, $T^\sharp$ fournit la meilleure abstraction *: si le triplet $(T, <\Sigma', I', Q, M'>, \alpha)$ est une autre abstraction, alors*

$$M^\sharp \leq M' \quad \textit{et} \quad I^\sharp \subseteq I'$$

Démonstration. La preuve est une conséquence directe des inégalités $Id \leq \gamma^\uparrow \circ \alpha^\uparrow$ et $\alpha^\uparrow \circ \gamma^\uparrow \leq Id$ résultantes de la définition de $\gamma^\uparrow$ par $\gamma^\uparrow = (\alpha^\uparrow)^\dagger$. □

Ce meilleur choix $M^\sharp$ permet de réécrire l'inégalité de la définition 3.15 en une égalité. Cette propriété est souvent représentée par le diagramme commutatif suivant.

$$\begin{array}{ccc} Q(\Sigma) & \xrightarrow{\alpha^\uparrow} & Q(\Sigma^\sharp) \\ \downarrow M & & \downarrow M^\sharp \\ Q(\Sigma) & \xrightarrow{\alpha^\uparrow} & Q(\Sigma^\sharp) \end{array}$$

Maintenant que le cadre d'abstraction linéaire est décrit, nous montrons que ce cadre permet de définir une sur-approximation du coût long-run.

3.4.4 Analyse de coût long-run

La notion de coût long-run est préservée par abstraction linéaire. Plus précisément, le théorème suivant énonce qu'une abstraction linéaire correcte produit une sur-approximation du coût concret long-run.

Théorème 3.3 (Correction)**.**
Soit $(T, T^\sharp, \alpha)$ une abstraction linéaire correcte. Alors :

$$\rho(T) \leq \rho(T^\sharp)$$

Afin d'améliorer la lisibilité de la preuve de ce théorème, on la décompose en les lemmes 3.10 et 3.11. D'autre part, pour ne pas alourdir les notations, on se permettra dans la démonstration de ces lemmes l'abus de notation consistant à noter α pour désigner son relèvement linéaire $\alpha^\uparrow$.

Le lemme suivant démontre que pour tout couple $(M, M^\sharp)$ définissant une abstraction linéaire correcte, les matrices itérées de M et $M^\sharp$ vérifient l'inégalité de correction. Ainsi, un grand nombre de propriétés vérifiées pour $(M, M^\sharp)$ le seront aussi pour $(M^n, (M^\sharp)^n)$, $n \geq 1$.

Lemme 3.10.
Soit $(T, T^\sharp, \alpha)$ une abstraction linéaire correcte. Alors :

$$\forall n \geq 1, \quad \alpha^\uparrow \circ M^n \leq (M^\sharp)^n \circ \alpha^\uparrow$$

Démonstration.
Par récurrence sur n. Par définition d'abstraction linéaire correcte, le cas $n = 1$ est vérifié. Supposons maintenant que $\alpha \circ M^n \leq (M^\sharp)^n \circ \alpha$. Les lois $\oplus$ et $\otimes$ respectant l'ordre (lemme 3.1), on a :

$$(\alpha \circ M^n) \circ M \leq ((M^\sharp)^n \circ \alpha) \circ M$$

On a alors

$$\begin{aligned}
\alpha \circ M^{n+1} = \alpha \circ (M^n \circ M) &\leq (M^\sharp)^n \circ (\alpha \circ M) && (\textit{associativité de } \circ) \\
&\leq (M^\sharp)^n \circ (M^\sharp \circ \alpha) && (\textit{d'après le cas } n = 1) \\
&\leq (M^\sharp)^{n+1} \circ \alpha && (\textit{associativité de } \circ)
\end{aligned}$$

La propriété est donc vérifiée dans le cas $n + 1$ et par le principe de récurrence, pour tout $n \geq 1$. □

Lemme 3.11.
Soit $(T, T^\sharp, \alpha)$ une abstraction linéaire correcte. On a alors :

$$\forall \sigma_0, \sigma_1 \in \Sigma,\ \forall \sigma_0^\sharp, \sigma_1^\sharp \in \Sigma^\sharp, \quad \left.\begin{array}{rcl} \alpha(\sigma_0) & = & \sigma_0^\sharp \\ \alpha(\sigma_1) & = & \sigma_1^\sharp \end{array}\right\} \Rightarrow M_{\sigma_0,\sigma_1} \leq (M^\sharp)_{\sigma_0^\sharp,\sigma_1^\sharp} \tag{3.16}$$

D'autre part, on a :

$$\forall n \geq 1, \quad \bigoplus_{\substack{\sigma_0 \in \Sigma \\ \sigma_1 \in \Sigma}} M^n_{\sigma_0,\sigma_1} \leq \bigoplus_{\substack{\sigma_0^\sharp \in \Sigma^\sharp \\ \sigma_1^\sharp \in \Sigma^\sharp}} (M^\sharp)^n_{\sigma_0^\sharp,\sigma_1^\sharp} \tag{3.17}$$

Ainsi que :

$$\forall n \geq 1, \quad \bigoplus_{\sigma \in \Sigma} M^n_{\sigma,\sigma} \leq \bigoplus_{\sigma^\sharp \in \Sigma^\sharp} (M^\sharp)^n_{\sigma^\sharp,\sigma^\sharp} \tag{3.18}$$

La démonstration de ce lemme n'a pas d'intérêt théorique majeur. Elle est relativement longue et répétitive. Afin d'aérer notre présentation, nous préférons placer cette démonstration en Annexe A.

Prouvons maintenant le théorème 3.3.

Démonstration.
L'inégalité 3.18 du lemme 3.11 nous assure que :

$$\bigoplus_{\sigma\in\Sigma} M^n_{\sigma,\sigma} \leq \bigoplus_{\sigma^\sharp\in\Sigma^\sharp} (M^\sharp)^n_{\sigma^\sharp,\sigma^\sharp}$$

On en déduit, par définition de l'opérateur trace, que :

$$tr\, M^n \leq tr(M^\sharp)^n$$

D'après la proposition 1, la racine n-ième est un $\oplus$-morphisme. De ce fait, cet opérateur préserve l'ordre [BCOQ92] et l'on a donc :

$$\sqrt[n]{tr\, M^n} \leq \sqrt[n]{tr(M^\sharp)^n}$$

En sommant cette dernière inégalité pour tout n de 1 à $|\Sigma|$, on obtient :

$$\bigoplus_{n=1}^{|\Sigma|} \sqrt[n]{tr\, M^n} \leq \bigoplus_{n=1}^{|\Sigma|} \sqrt[n]{tr(M^\sharp)^n}$$

et ainsi $\rho(M) \leq \rho(M^\sharp)$. □

Les propositions précédentes traitent de notre sémantique du point de vue matriciel. Peut-on exprimer des propriétés équivalentes pour notre sémantique de traces initiale et relier ces deux points de vue ? La proposition suivante stipule que la définition alternative du coût long-run (à l'aide des traces dont la longueur tend vers l'infini) est aussi préservée par abstraction.

Proposition 6.
Soit $(T, T^\sharp, \alpha)$ une abstraction correcte. Alors :

$$\limsup_{n\to\infty} \bigoplus_{\substack{t\in[\![T]\!]_{tr}\\ |t|=n}} \tilde{q}(t) \quad \leq \quad \limsup_{n\to\infty} \bigoplus_{\substack{t^\sharp\in[\![T^\sharp]\!]_{tr}\\ |t^\sharp|=n}} \tilde{q}^\sharp(t^\sharp)$$

Démonstration.
Soit $\sigma_0 \in I$. Notons $\sigma_0^\sharp = \alpha(\sigma_0)$. Comme $\alpha(I) \subseteq I^\sharp$, on a $\sigma_0^\sharp \in I^\sharp$. Soit $n \geq 1$. D'après l'inégalité 3.16, on a, $\forall \sigma_1 \in \Sigma$ et $\sigma_1^\sharp = \alpha(\sigma_1)$:

$$M^n_{\sigma_0,\sigma_1} \quad \leq \quad (M^\sharp)^n_{\sigma_0^\sharp,\sigma_1^\sharp}$$

On somme alors chaque membre de cette inégalité sur l'ensemble $G = \{(\sigma_0, \sigma_1, \sigma_0^\sharp, \sigma_1^\sharp) \mid \sigma_0 \in I,\ \sigma_0^\sharp = \alpha(\sigma_0),\ \sigma_1^\sharp = \alpha(\sigma_1)\}$:

$$\begin{array}{ccc}
\displaystyle\bigoplus_{(\sigma_0,\sigma_1,\sigma_0^\sharp,\sigma_1^\sharp)\in G} M^n_{\sigma_0,\sigma_1} & \leq & \displaystyle\bigoplus_{(\sigma_0,\sigma_1,\sigma_0^\sharp,\sigma_1^\sharp)\in G} (M^\sharp)^n_{\sigma_0^\sharp,\sigma_1^\sharp} \\
\| & & |\wedge \\
\displaystyle\bigoplus_{(\sigma_0,\sigma_1)\in I\times\Sigma} M^n_{\sigma_0,\sigma_1} & & \displaystyle\bigoplus_{(\sigma_0^\sharp,\sigma_1^\sharp)\in I^\sharp\times\Sigma^\sharp} (M^\sharp)^n_{\sigma_0^\sharp,\sigma_1^\sharp}
\end{array}$$

Les matrices itérées M^n et $(M^\sharp)^n$ contiennent les coûts des chemins de longueur n. Ainsi, cette dernière inégalité est équivalente à :

$$\bigoplus_{\substack{t \in [\![T]\!]_{tr} \\ |t| = n}} q(t) \leq \bigoplus_{\substack{t^\sharp \in [\![T^\sharp]\!]_{tr} \\ |t^\sharp| = n}} q^\sharp(t^\sharp)$$

Enfin, la racine n-ième étant $\oplus$-`mc`, on obtient :

$$\bigoplus_{\substack{t \in [\![T]\!]_{tr} \\ |t| = n}} \sqrt[n]{q(t)} \leq \bigoplus_{\substack{t^\sharp \in [\![T^\sharp]\!]_{tr} \\ |t^\sharp| = n}} \sqrt[n]{q^\sharp(t^\sharp)}$$

Cette inégalité étant vérifiée pour tout $n \geq 1$, la proposition est prouvée par passage à la limite supérieure. □

Nous avons présenté dans cette section un cadre d'approximation qui permet le calcul d'une sur-approximation du coût *long-run*. Ce cadre tire parti de la possibilité d'exprimer la sémantique des programmes et de la fonction d'abstraction sous forme de matrices. Ceci est réalisé par l'utilisation du relèvement linéaire. Dans la section suivante, nous étudions les limites de ce relèvement.

3.5 Intégration des abstractions classiques dans le modèle linéaire

Dans la Section 3.4.2, nous avons présenté une méthode de relèvement linéaire. Ce relèvement est basé sur l'utilisation d'un dioïde de coûts $Q(\oplus, \otimes)$. Plus précisément, cette méthode permet de relever toute fonction d'abstraction $\alpha : \Sigma \to \Sigma^\sharp$ en une matrice $\alpha^\uparrow \in \mathcal{M}_{\Sigma^\sharp \times \Sigma}(Q)$ à coefficients dans $\{\bot, \mathtt{e}\}$. La fonction $\alpha^\uparrow$ est alors une fonction linéaire entre les Q-moduloïdes $\Sigma^\uparrow$ et $(\Sigma^\sharp)^\uparrow$. Il est important de noter que les ensembles initiaux Σ et $\Sigma^\sharp$ ne sont pas supposés munis d'une structure d'ordre. Les ensembles relevés, eux, sont naturellement ordonnés par l'ordre induit par la loi $\oplus$.

Cette méthode de relèvement est bien adaptée aux abstractions α assez simples qui consistent essentiellement à fusionner des états sans tenir compte de la structure des ensembles Σ et $\Sigma^\sharp$. Toutefois, si nous souhaitons bénéficier pleinement des abstractions plus intéressantes telles que celles issues de connexions de Galois, nous devons définir une méthode de relèvement plus fine. Nous nous intéressons, dans cette section, à définir ce nouveau relèvement. Nous commençons par expliciter les inconvénients provoqués par l'utilisation du relèvement linéaire défini en Section 3.4.2.

3.5.1 Critique du relèvement linéaire

Le relèvement linéaire agit comme suit : un état σ de Σ est relevé en un vecteur de la forme $(\bot, \ldots, \bot, \mathsf{e}, \bot, \ldots, \bot)^T$ où e apparaît à la σ-ième position (on rappelle que Σ est dénombrable). L'ensemble des états concrets Σ est ainsi représenté en utilisant le Q-moduloïde $\Sigma^{\uparrow} = (\{\bot, \mathsf{e}\}^{|\Sigma|}, \oplus, \otimes)$. L'inconvénient le plus évident de cette méthode est qu'elle peut créer des matrices de dimension très importante.

Essayons maintenant d'utiliser cette méthode dans le cas où Σ est un treillis. Un autre inconvénient lié à cette méthode de relèvement apparaît alors : la structure ordonnée de Σ n'est pas transportée. Ou, en termes mathématiques, cette méthode de relèvement n'est pas un morphisme d'ordre. Ceci est d'autant plus regrettable que l'ensemble $\Sigma^{\uparrow}$ possède naturellement une structure d'ordre induite par la loi $\oplus$, et ceci que quel soit l'ensemble Σ.

Nous illustrons ces deux inconvénients en figure 3.6 sur l'exemple simple du treillis de parties à trois éléments $\mathcal{P}(\{1,2,3\})$. On remarque, tout d'abord,

$$\begin{array}{rcl@{\qquad}rcl}
\{\}^{\uparrow} & = & (\mathsf{e}, \bot, \bot, \bot, \bot, \bot, \bot, \bot)^T & \{1,2\}^{\uparrow} & = & (\bot, \bot, \bot, \bot, \mathsf{e}, \bot, \bot, \bot)^T \\
\{1\}^{\uparrow} & = & (\bot, \mathsf{e}, \bot, \bot, \bot, \bot, \bot, \bot)^T & \{1,3\}^{\uparrow} & = & (\bot, \bot, \bot, \bot, \bot, \mathsf{e}, \bot, \bot)^T \\
\{2\}^{\uparrow} & = & (\bot, \bot, \mathsf{e}, \bot, \bot, \bot, \bot, \bot)^T & \{2,3\}^{\uparrow} & = & (\bot, \bot, \bot, \bot, \bot, \bot, \mathsf{e}, \bot)^T \\
\{3\}^{\uparrow} & = & (\bot, \bot, \bot, \mathsf{e}, \bot, \bot, \bot, \bot)^T & \{1,2,3\}^{\uparrow} & = & (\bot, \bot, \bot, \bot, \bot, \bot, \bot, \mathsf{e})^T
\end{array}$$

FIGURE 3.6 – Relèvement linéaire $\mathcal{P}(\{1,2,3\})^{\uparrow}$ du treillis de parties à trois éléments

que le codage de la figure 3.6 ne respecte pas l'ordre initial. Par exemple, l'ensemble vide est le plus petit ensemble du treillis mais son relèvement n'est pas plus petit que le relèvement des autres éléments (il est même non comparable à ces éléments). D'autre part, ce codage est de grande dimension. Vu que le treillis $\mathcal{P}(\{1,2,3\})$ possède huit éléments, son relèvement $\mathcal{P}(\{1,2,3\})^{\uparrow}$ est de dimension huit. Ceci ne semble pas raisonnable. On peut, par exemple, faire descendre cette dimension à six en prenant $(\bot, \bot, \bot, \bot, \bot, \bot)^T$ comme relèvement de l'ensemble vide et $(\mathsf{e}, \mathsf{e}, \mathsf{e}, \mathsf{e}, \mathsf{e}, \mathsf{e})^T$ comme relèvement de $\{1,2,3\}$. Cette première tentative est aussi un premier pas vers le respect de l'ordre initial.

Il apparaît donc que le relèvement linéaire n'est pas adapté au traitement de structures ordonnées et a fortiori au traitement de fonctions définies sur de telles structures. Il s'agit maintenant de proposer un relèvement qui apporte des solutions aux inconvénients présentés précédemment. Formellement, un tel relèvement doit satisfaire aux exigences de la définition suivante.

Définition 3.16 (Relèvement parfait)**.**
Soient $L(\leq)$ un ensemble ordonné et $Q(\oplus, \otimes)$ un dioïde de coûts.
On appelle relèvement parfait *de L, et on note $\overline{L}$, un relèvement qui satisfait aux exigences suivantes :*

(Exig1) *$\overline{L}$ est un Q-moduloïde,*

(Exig2) *le relèvement respecte l'ordre initial :*

$$\forall \sigma_1, \sigma_2 \in L, \qquad \sigma_1 \leq \sigma_2 \;\Rightarrow\; \overline{\sigma_1} \leq \overline{\sigma_2}$$

(Exig3) *le codage est de dimension « raisonnable ».*

Soit maintenant $\alpha : L_1 \to L_2$ une fonction d'abstraction.
On appelle relèvement parfait *de $L_1 \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} L_2$, un relèvement qui satisfait aux exigences suivantes :*

(Exig4) *$\overline{L_1}$ et $\overline{L_2}$ sont des relèvements parfaits de L_1 et L_2,*

(Exig5) $\forall X \in L_1, \quad \overline{\alpha(X)} = \overline{\alpha}(\overline{X})$,

(Exig6) *$\overline{\alpha} : \overline{L_1} \to \overline{L_2}$ est une fonction $\oplus$-*`mc`*.*

Les exigences **(Exig5)** et **(Exig6)** sont des exigences de préservation : l'exigence **(Exig5)** stipule que la signification de la fonction α est préservée par relèvement tandis que l'exigence **(Exig6)** énonce la préservation de la loi $\oplus$ par la fonction relevée $\overline{\alpha}$.

3.5.2 Relèvement parfait des connexions de Galois

L'interprétation abstraite considère souvent des connexions de Galois de la forme $\mathcal{P}(E) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} L$ où $\mathcal{P}(E)$ est l'ensemble des parties[5] d'un certain ensemble E représentant le domaine sémantique concret et L est un treillis complet représentant le domaine abstrait. Nous noterons dans toute cette section $Q(\oplus, \otimes)$ un dioïde de coûts qui nous permettra de coder des vecteurs à l'aide de ses éléments $\perp$ et `e`.

Avant de présenter notre méthode dans le cas général, nous commençons par le cas où L est un treillis de parties, dont la structure est naturellement adaptée à un relèvement de taille réduite et au transport de la propriété d'ordre.

3.5.2.1 Relèvement parfait dans le cas des parties d'ensemble

Avant de relever la fonction d'abstraction α, on s'intéresse au relèvement de la structure des parties.

5. L'ensemble des parties d'un ensemble est naturellement muni d'une structure de treillis complet nommé treillis booléen [DP90].

Relèvement parfait de $\mathcal{P}(E)$.
Reprenons l'exemple précédent du treillis $\mathcal{P}(\{1,2,3\})$. Tout élément X de $\mathcal{P}(\{1,2,3\})$ peut se coder à l'aide d'un vecteur v à trois coefficients. Chaque coefficient v_i de v permet d'indiquer si l'élément i est présent ou non dans X. Plus précisément, si i est dans X, nous prendrons $v_i = \mathsf{e}$ et si i n'est pas dans X, nous prendrons $v_i = \bot$. La figure 3.7 résume ce codage.

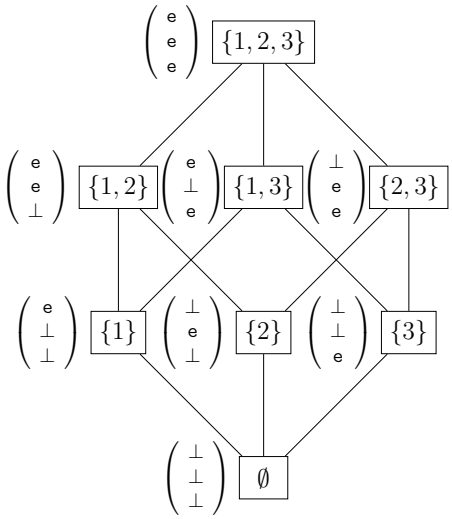


FIGURE 3.7 – Treillis de parties $\mathcal{P}(\{1,2,3\})$ et son relèvement parfait

Ce relèvement, très naturel, satisfait trivialement aux exigences **(Exig1)** et **(Exig3)**. D'autre part, on remarque que si le vecteur v code l'ensemble X et w code l'ensemble Y, alors le vecteur $v \oplus w$ code l'ensemble $X \cup Y$. Autrement dit, l'opérateur $\oplus$ est la traduction en terme de moduloïde de l'opérateur ensembliste $\cup$. De ce fait, ce codage préserve l'ordre initial du treillis et satisfait donc à l'exigence **(Exig2)**. On définit donc ainsi un relèvement parfait.

Nous pouvons facilement généraliser cette méthode pour un treillis de parties $\mathcal{P}(E)$ quelconque. Il suffit pour cela de coder les éléments de E (en utilisant le relèvement linéaire) et de déduire le codage des éléments de $\mathcal{P}(E)$ en utilisant le fait que les lois $\oplus$ et $\cup$ coïncident. Formellement, $\overline{\mathcal{P}(E)}$ est défini par :

$$\begin{array}{lrcl}
\forall e \in E, & \overline{e} & = & e^{\uparrow} \\
\forall X \in \mathcal{P}(E), & \overline{X} & = & \oplus \ \{\, \overline{e} \mid e \in X \,\} \\
 & & & \updownarrow \\
 & (X & = & \cup \ \{\, e \mid e \in X \,\})
\end{array}$$

Nous insistons sur la correspondance des lois $\oplus$ et $\cup$. Ceci se traduit par l'égalité :

$$\forall r \geq 1, \qquad \overline{\bigcup_{i=1}^{r} X_i} \;=\; \bigoplus_{i=1}^{r} \overline{X_i} \tag{3.19}$$

où, pour tout $i \in [1, r]$, X_i est un élément de $\mathcal{P}(E)$. Par facilité d'écriture, cette égalité est présentée dans le cas fini. Notons toutefois qu'elle est aussi vérifiée pour une union quelconque.

Relèvement parfait d'une connexion de Galois de $\mathcal{P}(E)$ dans $\mathcal{P}(F)$.
On considère une connexion de Galois $\mathcal{P}(E) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{P}(F)$, où E et F sont des ensembles quelconques. Par le relèvement précédent, les treillis de parties sont relevés en deux Q-moduloïdes $\overline{\mathcal{P}(E)}$ et $\overline{\mathcal{P}(F)}$. Définissons alors $\overline{\alpha}$ à l'aide de l'exigence **(Exig5)** :

$$\forall \overline{X} \in \overline{\mathcal{P}(E)}, \qquad \overline{\alpha}(\overline{X}) \;=\; \overline{\alpha(X)}$$

Afin de satisfaire à l'exigence **(Exig6)**, il faut montrer que $\overline{\alpha}$ est une fonction $\oplus$-`mc`. Ceci se démontre à l'aide de la correspondance entre les lois $\oplus$ et $\cup$, ainsi que du caractère $\cup$-`mc` de α. Démontrons[6] que :

$$\forall r \geq 1, \qquad \overline{\alpha}\Big(\bigoplus_{i=1}^{r} \overline{X_i}\Big) \;=\; \bigoplus_{i=1}^{r} \overline{\alpha}(\overline{X_i})$$

On a :

$$\begin{aligned}
& \overline{\alpha}\ \big(\textstyle\bigoplus_{i=1}^{r} \overline{X_i}\big) & \\
= \;& \overline{\alpha}\ \big(\overline{\textstyle\cup_{i=1}^{r} X_i}\big) & \text{par l'égalité (3.19)} \\
= \;& \overline{\alpha\ \big(\textstyle\cup_{i=1}^{r} X_i\big)} & \text{par définition de } \overline{\alpha} \\
= \;& \overline{\textstyle\cup_{i=1}^{r} \alpha(X_i)} & \text{car } \alpha \text{ est } \cup\text{-}\mathtt{mc} \\
= \;& \textstyle\bigoplus_{i=1}^{r} \overline{\alpha(X_i)} & \text{par l'égalité (3.19)} \\
= \;& \textstyle\bigoplus_{i=1}^{r} \overline{\alpha}(\overline{X_i}) & \text{par définition de } \overline{\alpha}
\end{aligned}$$

Nous avons démontré que $\overline{\alpha}$ est $\oplus$-`mc`. Cette fonction $\overline{\alpha}$ est donc un relèvement parfait de α et le cas des connexions de Galois sur des treillis de parties est réglé. Intéressons-nous maintenant au cas général.

6. Par facilité d'écriture, nous traitons ici le cas fini. La démonstration est la même dans le cas d'une somme quelconque.

3.5.2.2 Relèvement parfait dans le cas général

En général, L est un treillis complet qui n'est pas forcément un treillis de parties et le relèvement est alors moins direct. Cependant, on peut souvent se ramener au cas précédent. En effet, même si un treillis quelconque n'est généralement pas un treillis de parties, on peut cependant le voir comme un sous-treillis de parties. La proposition suivante énonce ce résultat [DP90].

Proposition 7.
Soit L un treillis. Si L est fini, on a :

$$\begin{array}{ll} & L \text{ est distributif.} \\ \Leftrightarrow & L \text{ est isomorphe à un treillis d'ensembles.} \\ \Leftrightarrow & L \text{ est isomorphe à un sous-treillis de parties.} \end{array}$$

Ce résultat est connu sous le nom de théorème de représentation des treillis distributifs finis et est dû à Birkhoff. Il peut être étendu au cas quelconque en supposant le treillis algébrique. Cette propriété permet de passer d'un ensemble dirigé à un élément de cet ensemble et est donc, d'une certaine manière, une propriété de finitude (pour plus de détails nous renvoyons le lecteur à [DP90]).

Proposition 8.
Soit L un treillis. Si L est algébrique, on a :

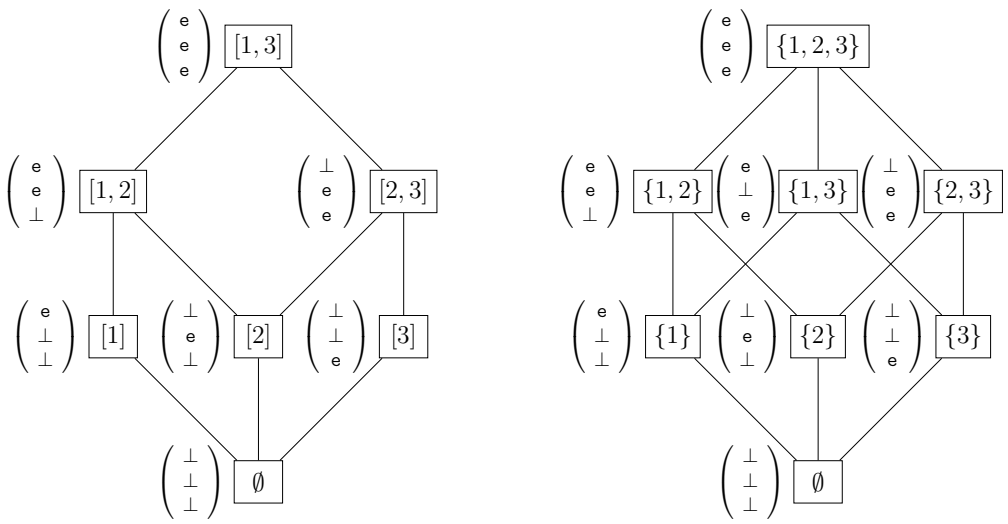
$$\begin{array}{ll} & L \text{ est distributif infini.} \\ \Leftrightarrow & L \text{ est isomorphe à un treillis d'ensembles.} \\ \Leftrightarrow & L \text{ est isomorphe à un sous-treillis de parties.} \end{array}$$

Notons que la propriété de distributivité infinie étend la propriété de distributivité à des ensembles d'éléments quelconques (éventuellement infinis).

Dans tout ce qui suit, on supposera que le treillis L vérifie les hypothèses d'une de ces deux propositions. La définition 3.17 introduit une notation permettant de se référer au treillis de parties mentionné dans les deux propositions différentes.

Définition 3.17.
Soit L un treillis satisfaisant l'une des hypothèses de la proposition 7 ou de la proposition 8. Alors L est isomorphe à un sous-treillis de parties.
Ce treillis de parties sera noté $\mathcal{B}(L)$.

On peut alors appliquer la manière de coder précédente à $\mathcal{B}(L)$ et *a fortiori* à L. Illustrons cette méthode par l'exemple du treillis des intervalles sur $\{1, 2, 3\}$. On note L ce treillis. Le plus grand treillis de parties qui contient L est le treillis de parties de l'ensemble $\{1, 2, 3\}$ que nous avons étudié précédemment. Le codage de $\overline{L}$ se déduit donc de celui de $\overline{\mathcal{P}(\{1, 2, 3\})}$. La figure 3.8 résume cette situation.



(a) Le treillis L des intervalles sur $\{1, 2, 3\}$

(b) Le treillis $\mathcal{B}(L)$. Par définition, $\mathcal{B}(L)$ est le treillis des parties $\mathcal{P}(\{1, 2, 3\})$

FIGURE 3.8 – Un exemple de treillis L et son treillis $\mathcal{B}(L)$

Notons que le codage obtenu pour L est de dimension moindre que si l'on avait utilisé le codage de la Section 3.4.2 qui, lui, aurait nécessité des vecteurs à six coefficients (un coefficient par élément du treillis).

Revenons maintenant à notre problème initial. L'ensemble des vecteurs $\overline{L}$ que l'on vient de construire n'est plus muni d'une structure de Q-moduloïde, contrairement à $\overline{\mathcal{B}(L)}$. Ceci est illustré dans notre exemple par le vecteur $(\mathsf{e}, \bot, \mathsf{e})^T$. Ce vecteur de $\overline{\mathcal{B}(L)} \setminus \overline{L}$ est obtenu par somme des deux vecteurs $(\mathsf{e}, \bot, \bot)^T$ et $(\bot, \bot, \mathsf{e})^T$ qui sont tous les deux éléments de $\overline{L}$. En fait, le problème provient du fait que l'opérateur d'union du treillis $\cup$ ne correspond plus à $\oplus$. Cette difficulté se traduit aussi au niveau du relèvement de la fonction d'abstraction α. Si l'on revient à notre exemple, on a :

$$\overline{\alpha(\{1, 3\})} = \overline{[1, 3]} = (\mathsf{e}, \mathsf{e}, \mathsf{e})^T$$

ainsi que :

$$\overline{\alpha}(\overline{\{1, 3\}}) = \overline{\alpha}(\overline{\{1\}}) \oplus \overline{\alpha}(\overline{\{3\}}) = (\mathsf{e}, \bot, \mathsf{e})^T$$

Ceci prouve que $\overline{\alpha(\{1, 3\})} \neq \overline{\alpha}(\overline{\{1, 3\}})$. Ainsi, même si la fonction $\overline{\alpha}$ est bien $\oplus$-`mc`, une perte d'information est réalisée durant ce relèvement et $\overline{\alpha}$ n'est donc pas un relèvement parfait.

Nos exigences de départ étaient trop élevées. Toutefois, nous montrons dans la section suivante que ce relèvement à perte d'information $\overline{\alpha}$ permet de définir un nouveau relèvement qui satisfait à l'exigence de sens **(Exig5)** mais pas à celle de linéarité **(Exig6)**.

3.5.3 Relèvement semi-parfait de connexions de Galois

On définit la notion de relèvement semi-parfait par la figure 3.9. Plus précisé-

$$\begin{tikzcd}
\mathcal{P}(E) \arrow[rr, bend left, "\alpha"] \arrow[d, dashed] & & L \arrow[ll, bend left, "\gamma"] \arrow[d, dashed] \\
\overline{\mathcal{P}(E)} \arrow[rr, bend left, "\overline{\alpha}"] & \overline{\mathcal{B}(L)} \arrow[l, bend left, "\overline{\gamma}"] \arrow[r, bend left, "\rho"] & \overline{L} \arrow[l, bend left, "\iota"]
\end{tikzcd}$$

FIGURE 3.9 – Relèvement semi-parfait d'une connexion de Galois

ment, le relèvement semi-parfait de α est obtenu par composition de la fonction (à perte d'information) $\overline{\alpha}$ qui est $\oplus$-`mc` avec une fonction ρ. Cette fonction ρ permet de rétablir la signification initiale de α. Elle est définie comme suit :

$$\forall x \in \overline{\mathcal{B}(L)}, \quad \rho(x) \ = \ \wedge\{z \in \overline{L} \mid z \geq x\}$$

Notons que l'existence de l'opérateur $\wedge$ est assurée par la nature complète de $\overline{L}$. D'autre part, soulignons que l'opérateur ρ défini de cette façon est un opérateur de clôture sur $\overline{\mathcal{B}(L)}$.

Comme $\overline{\alpha}$ est une fonction linéaire entre deux moduloïdes complets, la proposition 4 permet d'affirmer qu'il existe une fonction résiduée $\overline{\gamma}$ pour $\overline{\alpha}$, *c.à.d.* $\overline{\alpha}\circ\overline{\gamma} \leq Id_{\overline{\mathcal{B}(L)}}$ et $\overline{\gamma}\circ\overline{\alpha} \geq Id_{\overline{\mathcal{P}(E)}}$. Le passage de $\overline{L}$ à $\overline{\mathcal{B}(L)}$ est réalisé par simple injection canonique ι. Finalement, nous démontrons la propriété suivante, qui énonce que la notion de connexion de Galois est préservée par relèvement semi-parfait.

Proposition 9.
Les fonctions $\rho \circ \overline{\alpha}$ et $\overline{\gamma} \circ \iota$ telles que définies ci-dessous forment une connexion de Galois entre les moduloïdes $\overline{\mathcal{P}(E)}$ et $\overline{L}$.

Démonstration.
Notons tout d'abord que $\rho\circ\overline{\alpha}$ et $\overline{\gamma}\circ\iota$ sont monotones par composition de fonctions monotones. Nous montrons ensuite que $(\overline{\gamma} \circ \iota) \circ (\rho \circ \overline{\alpha}) \geq Id_{\overline{\mathcal{P}(E)}}$: pour tout $a \in \overline{\mathcal{P}(E)}$, $\rho(\overline{\alpha}(a)) \geq \overline{\alpha}(a)$ car ρ est extensive. Comme $\overline{\gamma}$ est monotone et pseudo-inverse de $\overline{\alpha}$, on a $\overline{\gamma} \circ \rho(\overline{\alpha}(a)) \geq \overline{\gamma}(\overline{\alpha}(a)) \geq a$. Finalement, nous montrons que $(\rho\circ\overline{\alpha})\circ(\overline{\gamma}\circ\iota) \leq Id_{\overline{\mathcal{B}(L)}}$: comme $< \overline{\alpha}, \overline{\gamma} >$ est une connexion de Galois, $\overline{\alpha}\circ\overline{\gamma}\circ\iota(x) = \overline{\alpha}\circ\overline{\gamma}(x) \leq x$ pour tout $x \in \overline{L}$. Par application de la fonction monotone ρ à chaque membre de cette inégalité, , nous obtenons $\rho(\overline{\alpha} \circ \overline{\gamma}(x)) \leq \rho(x)$. Comme $x \in \overline{L}$, $\rho(x) = x$, ce qui permet de conclure la preuve. $\square$

Il semble donc assez difficile d'obtenir un relèvement parfait de toute connexion de Galois. La principale difficulté provient du fait qu'on ne peut, généralement, relever l'opérateur $\cup$ en un opérateur matriciel. Nous avons donc proposé la notion de relèvement semi-parfait qui répond presque à toutes nos exigences initiales. Toutefois, le relèvement semi-parfait d'une abstraction ne définit pas un opérateur matriciel. D'un point de vue implémentation, ceci pose un problème puisque le coût long-run se calcule, en pratique, comme plus grande valeur propre de la matrice abstraite. Notons tout de même que le relèvement semi-parfait d'une connexion de Galois (α, γ) permet de définir un couple $(\overline{\alpha}, \overline{\gamma})$ composé de deux opérateurs matriciels. Ce couple définit la partie linéaire du relèvement. Notre cadre d'approximation peut s'instancier à l'aide de ce couple. Dans ce cas, soulignons toutefois que le remplacement de la fonction $\rho \circ \overline{\alpha}$ par la fonction $\overline{\alpha}$ produit une perte d'information.

3.6 Implémentation de la méthode sur un exemple

Cette section présente un exemple qui illustre notre méthode. Son contenu est une contribution du travail de thèse de Pascal Sotin, encadré par Thomas Jensen et David Cachera, et ne doit donc pas être mis au crédit de l'auteur. Cependant, afin de faciliter la rédaction de cette section, nous continuons à utiliser la première personne du pluriel.

Nous proposons ici un langage impératif simple équipé *d'opérations de tableaux* dont le coût dépend de la taille du tableau en question. Les coûts sont composés de deux attributs : un attribut *temps* qui exprime le nombre de cycles nécessaires à la réalisation d'une opération, et un attribut *énergie* qui exprime la puissance consommée par cette opération. Un programme peut s'exécuter sous différents *modes d'énergie*, chacun de ces modes déterminant la puissance consommée par opération. On supposera qu'il est possible de changer dynamiquement de mode. Grâce à une conversion sur les temps de transitions, les coûts énergétiques sont exprimés à l'aide du dioïde $(\mathbb{Q}, \max, +)$.

3.6.1 Syntaxe

Notre langage est un langage impératif simple et explicitement étiqueté, inspiré par le langage Simple défini par Miné dans [Min04], et équipé d'opérations de manipulation de tableaux. On l'appelle ArraySimple, et on donne sa syntaxe en Figure 3.6.1.

Nous ne détaillons pas les constructions classiques du langage telles que **if** ou `while`, qui ont un comportement standard. La taille des tableaux peut être af-

expr	::=	X	$X \in \mathcal{V}$
	|	$-$ *expr*	
	|	*expr* $\diamond$ *expr*	$\diamond \in \{+, -, \times, /\}$
	|	`length` A	$A \in \mathcal{A}$
	|	A[*expr*]	$A \in \mathcal{A}$
test	::=	*expr* $\bowtie$ *expr*	$\bowtie \in \{=, \neq, <, \leq\}$
	|	`not` *test*	
	|	*test* `and` *test*	
	|	*test* `or` *test*	
inst	::=	$X \leftarrow$ *expr*	$X \in \mathcal{V}$
	|	**if** *test* {*block*} `else` {*block*}	
	|	`while` pc_1 *test* {*block*}	$pc_1 \in \mathcal{L}$
	|	A[*expr*] $\leftarrow$ *expr*	$A \in \mathcal{A}$
	|	`apply` *op* A	$A \in \mathcal{A}$, $op \in \mathcal{O}$
	|	`setlength` *expr* A	$A \in \mathcal{A}$
	|	`setmode` M	$M \in \mathbb{M}$
block	::=	pc_1 *inst*; pc_2 ... *inst* pc_n	$pc_1 \ldots pc_n \in \mathcal{L}$

$\mathcal{L}$ est un ensemble fini d'étiquettes de programme
$\mathcal{V}$ est un ensemble fini de variables scalaires
$\mathcal{A}$ est un ensemble fini de variables de tableau
$\mathcal{O}$ est un ensemble fini d'opérations de tableau
$\mathbb{M}$ est un ensemble fini de modes d'énergie.

FIGURE 3.10 – Syntaxe du langage ArraySimple

fectée par l'instruction `setlength` et accédée via l'opération `length`. En plus des opérations classiques d'affectation des cellules des tableaux, les tableaux peuvent être manipulés par des opérateurs *globaux* via l'instruction `apply` instruction. Par exemple, on peut calculer l'élément maximum d'un tableau, ou d'autres opérations plus complexes telles que le tri ou la permutation. Ces opérations seront considérées comme des primitives du langage, et seul leur comportement quantitatif sera décrit. Enfin, l'instruction `setmode` permet de passer du mode d'énergie courant à celui spécifié.

3.6.2 Sémantique opérationnelle quantitative

Nous définissons une sémantique opérationnelle qui prend en compte les coûts des opérations sur les tableaux. Cette sémantique néglige les valeurs des cellules des tableaux et retient seulement la taille de ces tableaux.

La sémantique des expressions et tests est donnée par une sémantique collectrice non-déterministe inspirée des travaux de Miné [Min04]. Plus précisément, les environnements sont composés de fonctions des variables scalaires dans un ensemble de valeurs $\mathbb{I}$ (soit $\mathbb{Z}$, $\mathbb{Q}$ ou $\mathbb{R}$), et de fonctions de l'ensemble des variables de tableaux dans $\mathbb{N}$, qui servent à représenter leur taille : $\mathcal{E} = \mathcal{V} \to \mathbb{I} \cup \mathcal{A} \to \mathbb{N}$. Nous notons $[\![e]\!]$ la sémantique de l'expression e, qui est une fonction associant des environnements à des ensembles de valeurs numériques : $[\![expr]\!] : \mathcal{E} \to \mathcal{P}(\mathbb{I}) \cup \mathbb{N}$. La sémantique collectrice pour les instructions est notée $\{\!|.|\!\}$ et est une fonction de $\mathcal{P}(\mathcal{E})$ dans lui-même. Par exemple, la sémantique d'une affectation est définie par $\{\!|X \leftarrow e|\!\}R = \{\rho[X \mapsto v] \mid \rho \in R, v \in [\![e]\!]\rho\}$.

Concentrons-nous sur le système de transition muni de l'aspect quantitatif de la sémantique. Les états sont composés d'un compteur de programme, d'un environnement et d'un mode d'énergie : formellement, nous avons $\sigma = \mathcal{L} \times \mathcal{E} \times \mathbb{M}$. Les transitions de la sémantique opérationnelle sont instrumentés par des coûts qui sont des couples dans $\mathbb{N} \times \mathbb{Q}$, le premier élément comptant le temps (dans les cycles) et le second l'énergie par cycle, *c.à.d.* la puissance.

Nous présentons maintenant les règles définissant les transitions qui impactent la consommation d'énergie. Les instructions sont munies de leur étiquette de programme initiale et finale. L'instruction `setmode` permet de changer le mode d'énergie courant, et coûte $c_m(m, m')$ qui est une paire dépendant uniquement du mode d'énergie précédant l'instruction ainsi que du nouveau mode.

$$\frac{pc \ \texttt{setmode} \ m' \ pc'}{(pc, \rho, m) \to^{c_m(m,m')} (pc', \rho, m')}$$

Le coût d'une instruction d'affectation dépend du mode d'énergie. Plus précisément, il est extrait d'une table associant chaque mode d'énergie à une paire (temps, énergie). Cette table possède trois colonnes : la première qui décrit le

mode, la deuxième pour le temps (compté en cycles) et la dernière pour la puissance consommée (en mWh par cycle). On accède à cette table via une fonction *lookup* qui rend le couple $(temps, puissance)$ associé au mode actuel. Dans l'exemple que nous développons ici, nous utilisons cinq modes, variant de **A** à **E**. Le mode **A** est le mode le plus lent mais aussi le plus économe en énergie. Au contraire, **E** est le mode le plus rapide mais aussi le plus gourmand en puissance instantanée. Notons cependant que le mode **E** peut permettre de consommer globalement moins d'énergie que le **A** si l'on considère l'opération en entier (temps $\times$ puissance instantanée). Nous présentons ci-dessous un exemple de table que l'on peut utiliser pour calculer les coûts d'une affectation.

$$T_a = \begin{array}{|c|c|c|} \hline \mathbf{A} & 7 & 1 \\ \mathbf{B} & 4 & 1.5 \\ \mathbf{E} & 2 & 4 \\ \hline \end{array}$$

Cette table se lit de la manière suivante : dans le mode **B**, une opération d'affectation nécessite un temps de 4 cycles et consomme une énergie de 1.5 mWh par cycle[7]. Étant donné la table T_a, la règle sémantique pour l'affectation est la suivante.

$$\frac{pc\ X \leftarrow expr\ pc' \quad \rho' \in \{\!|X \leftarrow expr|\!\}\{\rho\} \quad c_e = lookup(T_a, m)}{(pc, \rho, m) \rightarrow^{c_e} (pc', \rho', m)}$$

Nous présentons maintenant la règle de transition d'une opération globale sur les tableaux. Les coûts de ce genre d'opération sont aussi inférés d'une table et sont dépendants de la taille du tableau sur lequel l'opération est effectuée.

$$\frac{pc\ \texttt{apply}\ op\ A\ pc' \quad size = \rho(A) \quad c_a = lookup([\![op]\!]_c(size), m)}{(pc, \rho, m) \rightarrow^{c_a} (pc', \rho, m)}$$

La sémantique de coût d'une opération de tableaux est notée entre crochets $[\![.]\!]_c$ et associe à la variable de taille n une table où la consommation d'énergie est dépendante de n. Notons que le nombre de cycles doit rester constant dans ce modèle (ce point sera discuté plus loin).

En guise d'illustration, la table suivante donne le coût de calcul associé à l'opérateur `read`.

$$[\![\texttt{read}]\!]_c = \lambda n. \begin{array}{|c|c|c|} \hline \mathbf{B} & 5 & n \\ \mathbf{D} & 3 & 2+3n \\ \hline \end{array}$$

Lecture de n entrées et stockage de celles-ci dans un tableau.

7. Ces quantités sont données ici dans un but d'illustration et ne reflètent pas des données issues d'exécutions réelles.

Les tailles des tableaux sont déterminées par une analyse statique utilisant des domaines numériques abstraits. Une telle analyse renverra le résultat $\top$ si elle ne parvient pas à borner la taille d'un tableau. Un tel résultat sera aussi renvoyé si la taille d'un tableau est exprimé par une expression qui ne peut pas être déterminée de manière précise dans le domaine numérique abstrait considéré.

3.6.3 Abstraction

L'abstraction utilisée sur notre langage est basée sur une analyse déjà existante, dont le domaine numérique abstrait est l'ensemble des polyèdres. Plus précisément, nous utilisons l'analyseur `Concurinterproc` [AJL]. On obtient alors, pour tout couple (compteur de programme, mode), un polyèdre convexe qui représente une sur-approximation relationnelle des valeurs des variables du programme, chaque dimension du polyèdre représentant une variable. Rappelons maintenant que le coût des opérations de tableaux dépend de la taille des tableaux. L'environnement abstrait précédent nous permet d'obtenir une sur-approximation de la taille d'un tableau `A`. Pour ce faire, nous projetons le polyèdre sur la dimension correspondant à la variable `A`, obtenant ainsi un intervalle $[a, b]$ contenant la taille de `A`. Le calcul du coût des opérations sur `A` est effectué à l'aide de la valeur b. Ce procédé est correct dès lors que nous considérons des opérations de tableaux dont le coût croît avec la taille, ce que nous supposons dorénavant.

Rappelons que nous souhaitons calculer un coût long-run pour un programme du langage ArraySimple. Jusqu'ici, les coûts ont été représentés par deux composants : le premier est un entier naturel qui exprime le nombre de cycles que nécessite l'instruction, le second représente l'énergie consommée par cycle. Afin de passer à une représentation de coûts avec une unique composante, nous procédons comme suit : pour toute transition requérant n cycles et de coût w (coût par cycle), nous créons un chemin de longueur n dont chaque transition est étiquetée uniquement par w, vu comme un coût indépendant du temps.

3.6.4 Résultats expérimentaux

L'abstraction et les calculs du coût long-run sont effectués en suivant trois étapes : premièrement, l'outil `Concurinterproc` est utilisé pour calculer des approximations polyédriques, ce qui permet de borner les tailles des tableaux. Les états abstraits correspondent à des couples de $\mathcal{L} \times \mathbb{M}$. On les énumère et on ne retient que les états d'intérêt, *c.à.d.* ceux qui sont accessibles depuis les états initiaux. Dans un deuxième temps, une matrice est construite, indexée par les états abstraits, et composée de valeurs qui sont déterminées par les tables de coût et l'approximation du coût inféré par l'analyse polyédrique. Cette étape a été implémentée en OCaml. La troisième étape consiste à calculer le coût long-run. Ceci

est réalisé avec l'aide de la librairie max-plus de Scilab, qui permet de calculer le coût long-run efficacement par utilisation de matrices creuses et de l'algorithme de calcul de valeur propre d'Howard [CTCG+98]. Cette utilisation nous a permis des calculs pour un ensemble d'états contenant jusqu'à 20k états.

3.7 Comparaison avec les approches existantes

Tout d'abord notons que la terminologie de coût *long-run* est tirée de [Alf98, BEK05], dans le contexte des processus probabilistes. Plus précisément, de Alfaro [Alf98] définit la notion de « propriété moyenne *long-run* » comme étant l'ensemble des propriétés sur le comportement moyen du programme, mesurées sur une période de temps tendant vers l'infini. Ces propriétés sont spécifiées par le concept d'expériences qui sont des graphes étiquetés. Ces expériences ont pour but de décrire des schémas de comportement d'intérêt et sont appliqués aux programmes probabilistes représentés par des chaînes de Markov temporisées. La réalisation d'une expérience fournit un *retour*, nombre réel représentant le succès ou la durée de l'expérience. Afin de spécifier les propriétés moyennes *long-run*, les notions de logiques temporelles sont étendues aux expériences.

Notre travail est inspiré par le cadre quantitatif d'interprétation abstraite développé par Di Pierro et Wiklicky [DW00]. Nous avons suivi leur approche en modélisant les programmes par des opérateurs linéaires sur un espace de vecteurs. En revanche, la structure liée à cet espace est très différente. Nos opérateurs agissent sur des dioïdes idempotents alors que les leurs agissent sur un semi-anneau de probabilités. Le fait de travailler avec des dioïdes idempotents nous a permis d'exploiter des résultats de la théorie des Systèmes à Événements Discrets [BCOQ92] où l'utilisation de telles structures est courante. Une autre différence avec l'approche de Di Pierro et Wiklicky est le type de programmes analysé puisqu'ils considèrent des langages déclaratifs (programmes probabilistes concurrents à contraintes et lambda calcul [DHW05]).

Dans le travail de Di Pierro et Wiklicky, la relation avec l'interprétation abstraite est justifiée par l'utilisation du pseudo-inverse d'un opérateur linéaire, mécanisme similaire à celui de connexion de Galois et qui permet d'assurer la correction des abstractions. Dans notre travail, c'est la théorie de la résiduation qui nous permet d'obtenir les pseudos-inverses de fonctions abstraites linéaires. Cette terminologie classique en SED coïncide en fait avec la notion de connexion de Galois. Les ordres partiels sont ici fournis par les structures de dioïdes. Ce sont ces structures partiellement ordonnées de dioïdes qui nous permettent de garantir la correction de nos abstractions, sous l'hypothèse que $\alpha \circ M \leq_D M^\sharp \circ \alpha$, qui est une exigence classique en interprétation abstraite.

D'autres travaux utilisent des semi-anneaux idempotents pour décrire des as-

pects quantitatifs et ceci sous la forme de semi-anneaux de contraintes [BMR97, Bis04], et particulièrement sous le nom de *contraintes softs*. Ce type de contraintes a été utilisé dans le cadre de la Qualité de Service [DFM+05, San08], et notamment pour des systèmes modélisés par des mécanismes de réécriture de graphe [HT05]. Dans toutes ces approches, les opérateurs $\oplus$ et $\otimes$ du semi-anneau de contraintes sont utilisés pour combiner les contraintes. Parmi ces travaux, deux approches similaires méritent une attention particulière puisqu'elles traitent de mécanismes d'abstraction. Aziz [Azi06] utilise les semi-anneaux dans le cadre du *mobile process calculus* dérivé du π-calcul, et ceci dans le but de modéliser le coût d'actions communicantes. Il définit aussi un cadre d'analyse statique, en abstrayant les semi-anneaux « concrets » en des semi-anneaux de cardinalité réduite, et en définissant les opérateurs de semi-anneaux abstraits en conséquence. Par exemple, le semi-anneau $(\mathbb{R}_+ \cup \{+\infty\}, \min, +)$ peut être abstrait par $(\{low, medium, high\}, \min, \max)$. Bistarelli *et al.* [BCR02] définissent un cadre basé interprétation abstraite afin d'abstraire des problèmes de satisfaction de contraintes softs (SCSPs). Similairement à l'approche d'Aziz, ils obtiennent un SCSP abstrait en changeant juste le semi-anneau associé, en laissant inchangé le reste de la structure. Les semi-anneaux concrets et abstraits sont reliés au moyen d'une insertion de Galois, ce qui permet d'obtenir les résultats de correction. La plus grande différence entre ces approches et la notre est que, dans ces modèles, le semi-anneau est abstrait et le système lui-même est laissé inchangé. Au contraire, dans notre approche, nous abstrayons les structures d'ensembles d'états et gardons le même dioïde. D'autre part, même si ces approches traitent aussi de dioïdes, aucune des deux n'utilise de notion de coût long-run pour exprimer le comportement quantitatif moyen d'un système considéré.

Notons aussi que des travaux récents utilisent des algèbres Max-plus dans le cadre de l'interprétation abstraite [AGG08, AGG10, AGK, All09]. Allamigeon *et al.* étudient la notion de polyèdre tropicaux. Cette notion est l'analogue de la notion de polyèdre classique transportée dans le cadre des algèbres Max-plus. Nous avons vu en Chapitre 2 que l'analyse polyédrique classique permettait d'obtenir des invariants linéaires. Par analogie avec ce cas, le domaine des polyèdre tropicaux permet d'obtenir des invariants min et max sur les variables du programme. Notons que le cadre d'algèbre Max-plus apporte un certain nombre de spécificités, comme le fait que la notion de système d'égalités et système d'inégalités coïncident dans ce monde, mais aussi certaines analogies avec l'analyse polyédrique classique. Ainsi, les auteurs montrent que les polyèdres tropicaux admettent eux aussi une double représentation, comme c'est le cas avec les polyèdres classiques qui peuvent être représentés par un nombre fini de contraintes inégalitaires ou de façon géométrique par la donnée de leurs sommets et de leurs rayons [AGG10].

Le travail présenté dans ce chapitre a donné lieu à trois publications. Le pre-

mier article a été publié dans une conférence internationale [CJJS08] et ne contenait pas l'étude sur la possibilité d'intégrer des abstractions classiques de l'interprétation abstraite dans notre cadre. Cette étude a été publiée dans un workshop international avec comité de lecture [CJ10]. Enfin, l'intégralité de ce travail a donné lieu à la publication d'une version étendue dans un journal [CJJS10].

Chapitre 4

Génération rapide d'invariants inductifs sous forme d'égalités polynomiales

Dans le Chapitre 2, nous avons illustré la théorie de l'interprétation abstraite par l'analyse polyédrique [CH78]. Cette analyse permet de fournir, à chaque point de programme, une sur-approximation de l'ensemble (potentiellement infini) des états accessibles. Ces sur-approximations sont des éléments du treillis des polyèdres *c.à.d.* des ensembles d'inégalités linéaires liant les variables du programme. Autrement dit, l'analyse polyédrique génère des informations linéaires sur les états accessibles du programme. Elle est particulièrement adaptée à l'analyse de programmes dont les éléments de syntaxe sont de nature linéaire. Afin de nuancer ce propos, considérons le programme `sqrt` présenté en figure 4.1. Ce programme est construit à l'aide d'affectations et de tests linéaires. Il calcule la valeur entière de la racine carrée d'une valeur n donnée en entrée. Ce calcul est effectué par affectations successives d'une variable `r`. En figure 2.3, nous présentons le résultat de l'analyse polyédrique effectuée sur ce programme grâce à l'outil `concurinterproc` [AJL]. Cette analyse infère l'égalité $\mathtt{t} = 2\mathtt{r} + 1$ en fin de programme (point de programme 8). Toutefois, la structure de polyèdre n'étant pas adaptée à l'expression d'information de nature non-linéaire, l'analyse polyédrique ne génère pas l'égalité $\mathtt{s} = (\mathtt{r} + 1)^2$. Cette égalité est nécessaire pour démontrer que le programme `sqrt` calcule bien une racine carrée. Ainsi, même pour un programme de nature linéaire, une information linéaire n'est pas toujours suffisante pour vérifier que le résultat d'un programme est en accord avec sa spécification. Évidemment, ce constat sur les « programmes linéaires » est a fortiori vrai pour les « programmes polynomiaux » qui sont construits à l'aide d'affectations et de tests polynomiaux.

Le problème de la génération d'invariants linéaires a donné lieu à de nom-

```
1.  r := 0;
2.  s := 1;
3.  t := 1;
      −2r + t − 1 = 0
4.  while s ≤ n do
5.    r := r + 1;
6.    t := t + 2;
7.    s := s + t;
      −2r + t − 1 = 0 ∧ −n + s − 1 ≥ 0
8.
```

FIGURE 4.1 – L'analyse polyédrique d'un programme `sqrt` réalisée par l'outil `concurinterproc` [AJL]

breuses méthodes [Kar76, CH78, CSS03, SSM04a]. L'étude de l'inférence d'invariants polynomiaux de programme sous forme **d'égalités polynomiales** a naturellement suivi et de nombreuses approches ont aussi été proposées [MOS02, MOS04, SSM04b, RCK04a, RCK07a, RCK07b] [1]. Ces approches différentes se basent toutes sur un domaine particulier, celui des **idéaux**, ainsi que sur l'utilisation de **bases de Gröbner** qui permet de décider de l'inclusion sur ce domaine.

Dans ce chapitre, nous proposons une analyse statique par interprétation abstraite permettant de calculer des invariants polynomiaux de programme sous forme d'égalités. Pour cela, nous nous appuyons sur l'approche de Müller-Olm & Seidl [MOS02, MOS04, MOPS06] dont nous donnons ici une idée générale. Cette analyse consiste en une analyse `MOP` arrière (cf Section 2.2.2.3 du Chapitre 2). Les programmes considérés sont décrits sous forme d'une relation de transition dont les arcs sont étiquetés par des commandes polynomiales. Ces commandes particulières consistent en des affectations polynomiales (de type $\mathtt{x_i} := p(\mathtt{x_1}, \ldots, \mathtt{x_m})$ avec $p \in \mathbb{R}[\mathtt{x_1}, \ldots, \mathtt{x_m}]$) et des tests polynomiaux sous forme de déséglités polynomiales ($p_1 \neq p_2$ avec p_1 et p_2 éléments de $\mathbb{R}[\mathtt{x_1}, \ldots, \mathtt{x_m}]$). Cette analyse est une méthode de propagation arrière : partant d'un polynôme p, elle consiste à calculer la plus faible précondition d'appartenance à l'ensemble des zéros du polynôme g. En prouvant que cette précondition n'impose pas de contrainte sur les états initiaux du programme, on démontre ainsi que toute exécution aboutit dans un état qui est un zéro de g. On démontre ainsi que g est un invariant polynomial à la fin du programme. En résumé, pour tout polynôme g, on peut vérifier que g est invariant. Cette méthode de vérification peut être adaptée en une méthode de génération en considérant un polynôme g_{par} paramétré. Les paramètres de ce polynôme sont alors contraints de sorte que, sous ces contraintes, g_{par} est inva-

1. Les méthodes proposées dans cet état de l'art seront présentées et critiquées en fin de chapitre.

riant en fin de programme. La plus faible précondition est obtenue par un calcul de point fixe itératif dans le domaine des idéaux. Des calculs de bases de Gröbner sont nécessaires pour démontrer l'arrêt de l'itération. Ce procédé est connu pour posséder une complexité doublement exponentielle dans le pire cas [MM82].

Dans ce chapitre, notre problématique est de proposer une analyse dont les opérateurs abstraits admettent un calcul aussi simple que ceux développés par Müller-Olm et Seidl [MOS04] et dont les calculs itératifs peuvent être effectués aussi efficacement que dans l'analyse proposée par Rodríguez-Carbonell et Kapur [RCK07a]. L'analyse `fastind` que nous avons développée satisfait ces exigences [CJJK11]. Elle s'inspire de l'analyse arrière de Müller-Olm et Seidl. Les calculs itératifs sont grandement allégés grâce à **l'hypothèse d'inductivité** qui consiste à ne rechercher que des invariants de boucles. Ce point de vue naturel a notamment été adopté dans l'analyse avant développée par Sankaranarayanan *et al.* [SSM04b]. Cette hypothèse d'inductivité s'exprime, pour chaque boucle du programme, sous forme d'une contrainte d'égalité d'idéaux. Nous montrons, sous cette hypothèse, que la procédure itérative de calcul de points fixes termine dès le premier pas de calcul. Précisons, d'autre part, que les contraintes générées par notre méthode s'expriment généralement sous forme de systèmes d'équations linéaires et admettent donc une résolution aisée. Il en résulte une analyse efficace qui a donné lieu à une implémentation à l'aide du logiciel Maple ainsi qu'à un développement Ocaml. Enfin, nous prouvons la rapidité d'exécution de notre méthode en comparant très favorablement son temps d'exécution à ceux donnés dans les articles résumant les travaux de Müller-Olm et Seidl [MOS04] et Rodríguez-Carbonell et Kapur [RCK07b, RCK07a].

Ce chapitre est structuré comme suit. En Section 4.1, nous présentons la structure des idéaux, particulièrement adaptée au problème d'inférence d'égalités polynomiales. Nous détaillons ensuite, en Section 4.2, la syntaxe (structurée) et la sémantique des programmes que nous considérons. Nous exposons dans la section suivante (Section 4.3), une analyse approchée permettant de générer des invariants polynomiaux de manière itérative. La Section 4.4 constitue la principale contribution de ce chapitre. Nous présentons dans cette section l'analyse `fastind`. Nous expliquons comment l'hypothèse d'inductivité nous permet de nous affranchir des calculs itératifs et par conséquent des calculs de bases de Gröbner. Ceci permet d'améliorer, de manière radicale, le temps d'exécution de l'analyse. Enfin en Section 4.5 nous comparons notre approche avec celles déjà existantes dans l'état de l'art, et comparons les temps d'exécution.

4.1 Une structure mathématique adaptée : les idéaux

L'analyse statique que nous présentons dans ce chapitre consiste à inférer des invariants sous forme d'égalités polynomiales. Nous présentons dans cette section le domaine qui permet la manipulation de telles égalités.

Dans ce qui suit, nous considérons des polynômes de $\mathbb{R}[\mathtt{x_1}, \ldots, \mathtt{x_m}]$ où m représente le nombre de variables du programme [2]. Nous ferons la différence entre la notation x, qui représente un élément de $\mathbb{R}^m$, la notation x_i qui représente un élément de $\mathbb{R}$, et $\mathtt{x_i}$ utilisée pour une variable du programme. Commençons par définir l'ensemble des zéros d'un polynôme. Cette notion, très classique, sera utilisée pour définir la notion d'invariant.

Définition 4.1 (Zéro d'un polynôme)**.**
Soit $x \in \mathbb{R}^m$ et $g \in \mathbb{R}[\mathtt{x_1}, \ldots, \mathtt{x_m}]$. On dit que x est un zéro *de g si $g(x) = 0$.*

Un ensemble d'égalités polynomiales $\{p_1 = 0, \ldots, p_s = 0\}$ possède des propriétés de stabilité intéressantes. Plus précisément, si une relation polynomiale $p = 0$ est satisfaite, alors la relation $q.p = 0$ l'est aussi pour tout $q \in \mathbb{R}[\mathtt{x_1}, \ldots, \mathtt{x_m}]$; de plus, si $p_1 = 0$ et $p_2 = 0$ sont satisfaites, alors $p_1 + p_2 = 0$ est elle aussi vérifiée. Ces propriétés définissent la structure d'idéal.

Définition 4.2 (Idéal polynomial)**.**
Un ensemble $I \subseteq \mathbb{R}[\mathtt{x_1}, \ldots, \mathtt{x_m}]$ est un idéal polynomial *(ou simplement* idéal*) si les propriétés suivantes sont vérifiées :*

- *l'ensemble $\mathcal{I}$ contient* 0*,*
- *l'ensemble $\mathcal{I}$ est stable par addition :*

$$\forall p_1, p_2 \in I, \quad p_1 + p_2 \in I$$

- *l'ensemble $\mathcal{I}$ est stable par multiplication externe :*

$$\forall q \in \mathbb{R}[\mathtt{x_1}, \ldots, \mathtt{x_m}],\ \forall p \in I, \quad q.p \in I$$

Nous noterons $\mathcal{I}$ l'ensemble des idéaux polynomiaux de $\mathbb{R}[\mathtt{x_1}, \ldots, \mathtt{x_m}]$.

La notation $<S>$ désignera l'idéal polynomial généré par un ensemble S de polynômes. Par définition, $<S>$ est le plus petit idéal contenant tous les polynômes de S.

2. Rigoureusement, le domaine de l'analyse peut être défini à l'aide de $\mathbb{F}[\mathtt{x_1}, \ldots, \mathtt{x_m}]$ où $\mathbb{F}$ est un anneau nœthérien *c.à.d.* un anneau dans lequel la condition de chaîne ascendante est vérifiée.

L'ensemble $\mathcal{I}$ peut être muni d'une structure d'ordre partiel en utilisant *l'inclusion ensembliste inverse* sur les idéaux. La borne supérieure d'un ensemble d'idéaux polynomiaux est donnée par l'intersection de tous les éléments qu'il contient. La borne inférieure est, quant à elle, l'idéal généré par l'union de tous les éléments de l'ensemble [3].

Définition 4.3 (Structure de treillis de $\mathcal{I}$)**.**
Nous notons $\bigsqcup^\sharp$, $\bigsqcap^\sharp$, $\sqsubseteq^\sharp$, $\bot^\sharp$ *et* $\top^\sharp$ *les opérateurs et ensembles suivants :*

- $\bigsqcup^\sharp$ *désigne l'intersection d'idéaux :*

$$\forall I, J \quad I \bigsqcup^\sharp J = I \bigcap J$$

- $\bigsqcap^\sharp$ *désigne le plus petit idéal engendré par l'union d'idéaux :*

$$\forall I, J \quad I \bigsqcap^\sharp J = <I \bigcup J>$$

- $\sqsubseteq^\sharp$ *désigne l'inclusion ensembliste inverse :*

$$\forall I, J, \quad I \sqsubseteq^\sharp J \;\Leftrightarrow\; I \supseteq J$$

- $\bot^\sharp$ *définit le plus petit élément de* $\mathcal{I}$ *par l'opérateur* $\sqsubseteq^\sharp$ *:*

$$\bot^\sharp = <1>$$

- $\top^\sharp$ *définit le plus grand élément de* $\mathcal{I}$ *par l'opérateur* $\sqsubseteq^\sharp$ *:*

$$\top^\sharp = <0>$$

Les opérateurs $\bigsqcup^\sharp$ *et* $\bigsqcap^\sharp$ *sont étendus de manière classique sur les ensembles d'idéaux polynomiaux. Sous ces définitions,* $\mathcal{I}(\bigsqcup^\sharp, \bigsqcap^\sharp, \sqsubseteq^\sharp, \bot^\sharp, \top^\sharp)$ *est un treillis complet.*

La structure d'idéal présente admet une propriété de finitude, ce qui en fait une structure facile à manipuler. Cette propriété, connue sous le nom de *théorème de Hilbert* est rappelée ci-dessous.

Théorème 4.1 (Hilbert)**.**
Tout idéal polynomial $I \in \mathcal{I}$ *est finiment engendré,* c.à.d. $I = <S>$ *pour un sous-ensemble* S *de* I*.*

Un résultat standard de l'algèbre multivariée stipule que le théorème de Hilbert est équivalent au fait que l'ensemble $\mathcal{I}$ satisfait la condition de chaîne ascendante [CLO07]. Comme nous l'avons mentionné au Chapitre 2, cette propriété

3. Notons que l'union ensembliste de deux idéaux n'est pas un idéal en général.

joue un rôle clé en interprétation abstraite : elle assure la terminaison des algorithmes itératifs de calcul de plus petits points fixes. D'autre part, notons que nous parlons ici de condition de chaîne ascendante car, dans la littérature, l'ordre utilisé sur les idéaux est l'ordre $\subseteq$. Pour notre part, nous considérons l'ordre inverse $\supseteq$. Relativement à cet ordre, l'ensemble $\mathcal{I}$ vérifie la condition de chaîne descendante.

Le théorème 4.1 permet aussi de mettre en valeur la relation étroite qui existe entre les ensembles d'invariants et la structure d'idéal. En effet, nous avons vu précédemment qu'un ensemble d'invariants pouvait être naturellement représenté par un idéal. D'autre part, tout idéal polynomial est généré par un ensemble fini de polynômes qui peuvent être vus comme un ensemble d'invariants.

Définition 4.4 (Variété, radical d'un idéal)**.**
Soit I un idéal polynomial de $\mathbb{R}[\mathtt{x_1}, \ldots, \mathtt{x_m}]$ et V un sous-ensemble de $\mathcal{P}(\mathbb{R}^m)$.
La variété *générée par I est notée $\mathtt{V}(I)$ et est définie par :*

$$\mathtt{V}(I) = \{(a_1, \ldots, a_m) \in \mathbb{R}^m \mid f(a_1, \ldots, a_n) = 0 \text{ pour tout } f \in I\}$$

L'idéal généré par V est noté $\mathtt{I}(V)$ et est défini par :

$$\mathtt{I}(V) = \{f \in \mathbb{R}[\mathtt{x_1}, \ldots, \mathtt{x_m}] \mid f(a_1, \ldots, a_m) = 0 \text{ pour tout } (a_1, \ldots, a_m) \in V\}$$

Si I est un idéal polynomial, nous notons $\sqrt{I}$ le radical *de I, défini par :*

$$\sqrt{I} = \{f \in \mathbb{R}[\mathtt{x_1}, \ldots, \mathtt{x_m}] \mid \text{il existe } e \in \mathbb{N} \text{ tel que } f^e \in \mathbb{N}\}$$

Contrairement au cas univarié, $\mathbb{R}[\mathtt{x_1}, \ldots, \mathtt{x_m}]$ n'est pas muni d'une division euclidienne. Cependant, il est aisé de définir une division à l'aide d'un ordre monomial [CLO07]. Dans notre cas, nous utiliserons une définition élargie de cette notion de division.

Définition 4.5 (Opérateur de division, reste)**.**
Un opérateur de division `div` *est une fonction, qui, à tout couple de polynômes $(g, p) \in \mathbb{R}[\mathtt{x_1}, \ldots, \mathtt{x_m}]^2$, associe un couple $(q, r) \in \mathbb{R}[\mathtt{x_1}, \ldots, \mathtt{x_m}]^2$ tel que $g = pq + r$. Le polynôme r est appelé le* reste *de g par p selon* `div`*, et est noté $\mathtt{Rem}(g, p, \mathtt{div})$ ou seulement $\mathtt{Rem}(g, p)$.*

Il convient de remarquer que la définition classique de division multivariée obéit aux exigences d'opérateur de division.

Notre sémantique concrète (Section 4.2.2) manipule des ensembles de $\mathbb{R}^m$ tandis que notre sémantique abstraite traite de polynômes (Section 4.3.1). Rappelons que $\mathcal{P}(\mathbb{R}^m)$ est naturellement muni d'une structure de treillis complet $\mathcal{P}(\mathbb{R}^m)(\subseteq, \bigcup, \bigcap)$ où $\subseteq$, $\bigcup$ et $\bigcap$ représentent respectivement l'inclusion, l'union et

l'intersection d'ensembles. Les deux treillis complets $\mathcal{P}(\mathbb{R}^m)$ et $\mathcal{I}$ sont reliés par la connexion de Galois $\mathcal{P}(\mathbb{R}^m) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{I}$ où :

$$\begin{array}{lrcl} \alpha : & \mathcal{P}(\mathbb{R}^m) & \rightarrow & \mathcal{I} \\ & X & \mapsto & \mathtt{I}(X) \;=\; \{u \in \mathbb{R}[\mathtt{x_1}, \ldots, \mathtt{x_m}] \mid \forall x \in X,\; u(x) = 0\} \end{array}$$

$$\begin{array}{lrcl} \gamma : & \mathcal{I} & \rightarrow & \mathcal{P}(\mathbb{R}^m) \\ & I & \mapsto & \mathtt{V}(I) = \{x \in \mathbb{R}^m \mid \forall u \in I,\; u(x) = 0\} \end{array}$$

On rappelle, comme l'exige la Définition 2.18 que :

$$\forall X \in \mathcal{P}(\mathbb{R}^m),\; \forall I \in \mathcal{I} : \quad X \subseteq \gamma(I) \;\Leftrightarrow\; \alpha(X) \sqsubseteq^\sharp I$$

Par le suite, dans le cas où I est un idéal principal *c.à.d.* s'il existe un polynôme g tel que $I = <g>$, on se permettra de confondre I et le polynôme g l'engendrant. On fera alors l'abus de notation consistant à écrire $\gamma(g)$ au lieu de $\gamma(<g>)$. Notons maintenant que la connexion de Galois définie ci-dessus permet de faire le lien entre un polynôme g et l'ensemble de ses zéros $\gamma(g)$.

En résumé, nous sommes dans un cadre très favorable de la théorie de l'interprétation abstraite : les domaines concrets et abstraits sont munis de structures de treillis complets ; ces deux domaines sont reliés par une connexion de Galois ; le domaine abstrait vérifie la condition de chaîne descendante. Cette dernière propriété assure la terminaison de la procédure itérative permettant de calculer les points fixes de la sémantique abstraite (Section 4.3.1).

4.2 Les programmes polynomiaux

Nous considérons une variante du langage IMP [Win93] dans lequel les fonctions d'affectations sont polynomiales et les tests conditionnels sont des égalités ou déséglités polynomiales.

4.2.1 Syntaxe des programmes polynomiaux

Dans le Chapitre 2, nous avons présenté les programmes comme étant des graphes étiquetés par des commandes. Dans cette section, nous délaissons cette approche et donnons la syntaxe des programmes polynomiaux sous une forme structurée. La principale différence entre ces deux approches réside dans la présentation de la structure de boucles. Les boucles se lisent sur les graphes grâce à la présence d'arcs retours. Dans l'approche structurée, les boucles sont des opérateurs à part entière. Nous faisons ici le choix de l'approche structurée car nous pensons que ceci permet de mieux comprendre et présenter les opérateurs concrets et abstraits de boucles.

Définition 4.6 (Syntaxe des programmes polynomiaux)**.**

$p \in \mathbb{R}[\mathtt{x_1}, \ldots, \mathtt{x_m}]$			*polynômes*
$\mathbb{V} \ni var$	$::=$	$\mathtt{x_1} \mid \ldots \mid \mathtt{x_m}$	*variables du programme*
$\mathbb{T} \ni test$	$::=$	$p \bowtie 0$	*gardes polynomiales*
$\mathbb{P} \ni c$	$::=$	$var := p$	*affectations polynomiales*
		$\mid c \; ; \; c$	*séquence*
		$\mid$ ***if*** $test$ ***then*** c ***else*** c	*structure conditionnelle*
		$\mid$ ***while*** $test$ ***do*** c	*structure de boucle*
		$\mid$ ***skip***	*skip*

le symbole $\bowtie$ *est utilisé pour représenter un élément de l'ensemble* $\{=, \neq\}$*. Nous utilisons aussi la notation* $\not\bowtie$ *pour définir la négation de* $\bowtie$*.*

Notons que nous pourrons adjoindre des numéros de ligne à la syntaxe des programmes décrite ci-dessus. La sémantique de nos programmes est décrite dans la section suivante.

4.2.2 Sémantique concrète des programmes polynomiaux

Afin d'inférer une égalité polynomiale $g = 0$ vérifiée en fin de programme, nous allons montrer que l'exécution du programme depuis tout état initial mène à un état qui est un élément de l'ensemble des zéros de g. Pour ce faire, nous nous inspirons de la procédure développée par Müller-Olm et Seidl [MOS04] qui consiste à montrer que la plus faible précondition d'appartenance à l'ensemble des zéros de g n'impose pas de contrainte sur les états initiaux du programme. Cette plus faible précondition se définit à l'aide d'une sémantique collectrice arrière qui n'est pas tout à fait classique : elle opère en arrière et est obtenue par un plus grand point fixe. Afin de faciliter le décryptage de notre sémantique par un lecteur non expert, nous commençons par donner le sens de nos programmes à l'aide d'une sémantique opérationnelle petit pas (avant) très classique. Nous définirons ensuite la sémantique collectrice arrière puis montrerons comment elle s'exprime à l'aide de cette sémantique avant.

Commençons donc par définir la sémantique opérationnelle petit pas de nos programmes.

4.2.2.1 Sémantique opérationnelle petit pas et accessibilité

Sémantique opérationnelle petit pas

Nous exprimons la sémantique du langage IMP à l'aide d'une sémantique opérationnelle petit pas (`SOPP`). Notons $\rightarrow \subseteq (\mathbb{P} \times \mathbb{R}^m) \times (\mathbb{P} \times \mathbb{R}^m)$ cette sémantique. Elle est formellement définie par l'ensemble des règles d'inférences suivantes.

Définition 4.7 (Sémantique opérationnelle petit pas (SOPP)).
Soit c un programme polynomial, et $\sigma, \sigma' \in \mathbb{R}^m$ des états du programme c.
La sémantique opérationnelle petit pas est définie par les règles du de la Figure 4.2.

(i) $$\frac{}{\langle \textbf{skip}, \sigma\rangle \to \langle \bullet, \sigma\rangle}\ \textbf{\textit{skip}}$$

(ii) $$\frac{p(\sigma) = v}{\langle \mathtt{x_j} := p, \sigma\rangle \to \langle \bullet, \sigma[\![v]\!]_{\mathtt{j}}\rangle}\ \textbf{\textit{assign}}$$

(iii) $$\frac{\langle c_1, \sigma\rangle \to \langle c_1', \sigma'\rangle}{\langle c_1; c_2, \sigma\rangle \to \langle c_1'; c_2, \sigma'\rangle}\ \textbf{\textit{seq_red}} \qquad \frac{\langle c_1, \sigma\rangle \to \langle \bullet, \sigma'\rangle}{\langle c_1; c_2, \sigma\rangle \to \langle c_2, \sigma'\rangle}\ \textbf{\textit{seq_nred}}$$

$$\frac{}{\langle \bullet; c, \sigma\rangle \to \langle c, \sigma\rangle}\ \textbf{\textit{seq_point}}$$

(iv) $$\frac{p(\sigma) \bowtie 0 \equiv \mathit{true}}{\langle \textbf{if } p \bowtie 0 \textbf{ then } c_1 \textbf{ else } c_2, \sigma\rangle \to \langle c_1, \sigma\rangle}\ \textbf{\textit{if_true}}$$

$$\frac{p(\sigma) \bowtie 0 \equiv \mathit{false}}{\langle \textbf{if } p \bowtie 0 \textbf{ then } c_1 \textbf{ else } c_2, \sigma\rangle \to \langle c_2, \sigma\rangle}\ \textbf{\textit{if_false}}$$

(v) $$\frac{p(\sigma) \bowtie 0 \equiv \mathit{true}}{\langle \textbf{while } p \bowtie 0 \textbf{ do } c, \sigma\rangle \to \langle c; \textbf{while } b \textbf{ do } c, \sigma\rangle}\ \textbf{\textit{while_true}}$$

$$\frac{p(\sigma) \bowtie 0 \equiv \mathit{false}}{\langle \textbf{while } p \bowtie 0 \textbf{ do } c, \sigma\rangle \to \langle \textbf{skip}, \sigma\rangle}\ \textbf{\textit{while_false}}$$

Notation : étant donné un état $\sigma = (\sigma_1, \ldots, \sigma_m)$, nous notons $\sigma[\![v]\!]_{\mathtt{j}}$ l'état modifié $(\sigma_1, \ldots, \sigma_{j-1}, v, \sigma_{j+1}, \ldots, \sigma_m)$.

FIGURE 4.2 – Sémantique opérationnelle petit pas.

On remarque, dans la Figure 4.2, l'apparition d'un programme noté $\bullet$. Le couple $\langle \bullet, \sigma\rangle$ permet de représenter un état $\sigma \in \mathbb{R}^m$ comme un élément de $\mathbb{P} \times \mathbb{R}^m$. Le programme $\bullet$ est donc juste un artifice de notation permettant de définir la relation $\to$ comme sous-ensemble de $(\mathbb{P} \times \mathbb{R}^m) \times (\mathbb{P} \times \mathbb{R}^m)$. De ce fait, il n'est pas utile de le faire apparaître comme élément de syntaxe de nos programmes. Du fait de ce choix, l'accessibilité est définie comme suit.

Si la relation $\langle c, \sigma\rangle \to \langle c', \sigma'\rangle$ est vérifiée, on dira que le programme c dans l'état σ se réduit en un pas en le programme c' dans l'état σ'. D'autre part, on note $\to^* \subseteq (\mathbb{P} \times \mathbb{R}^m) \times (\mathbb{P} \times \mathbb{R}^m)$ la relation qui définit les réductions en zéro, un, ou un nombre fini de pas et $\to^\infty \subseteq \mathbb{P} \times \mathbb{R}^m$ la relation qui permet de représenter les réductions infinies. Enfin on note $\to^{co*} \subseteq ((\mathbb{P} \times \mathbb{R}^m) \times (\mathbb{P} \times \mathbb{R}^m)) \cup (\mathbb{P} \times \mathbb{R}^m)$ la relation qui représente les réductions en zéro, un, un nombre fini ou un nombre infini de pas. Ces relations seront utilisées dans le reste de ce chapitre dans de nombreuses définitions et théorèmes.

État accessible et polynôme invariant

Nous commençons par définir la notion d'état accessible. Cet ensemble d'états est déterminé par l'exécution du programme sur l'ensemble des états initiaux.

Définition 4.8 (État accessible)**.**
Soit $\mathbb{P}$ et $\sigma' \in \mathbb{R}^m$. L'état σ' est un état accessible *du programme c si :*

$$\exists \sigma \in \mathbb{R}^m, \quad \langle c, \sigma\rangle \to^* \langle \bullet, \sigma'\rangle$$

Intéressons-nous maintenant aux états terminants. Cette notion peut être vue comme duale de celle d'accessibilité. Les états terminants correspondent à l'image réciproque par la relation $\to^*$ des états accessibles. Réciproquement, les états accessibles correspondent à l'image directe, par $\to^*$ des états terminants.

Définition 4.9 (État terminant)**.**
Un état $\sigma \in \mathbb{R}^m$ est dit terminant *pour un programme c s'il vérifie la propriété suivante :*

$$\exists \sigma_f \in \mathbb{R}^m, \quad \langle c, \sigma\rangle \to^* \langle \bullet, \sigma_f\rangle$$

Un programme c dont tous les états $\sigma \in \mathbb{R}^m$ sont terminants est dit terminant.

D'un point de vue calculatoire, états accessibles et états terminants sont situés à deux extrémités du programme : l'une de ses extrémités est calculable par sémantique avant (les états accessibles) tandis que l'autre extrémité l'est par sémantique arrière (les états terminants).

Nous pouvons alors définir la notion d'invariant de programme. On dira qu'un polynôme g est invariant pour un programme si chacun des états accessibles du programme est un zéro de g. Ou, autrement dit, si l'exécution du programme sur chaque état terminant mène à un état qui est un zéro du polynôme g.

Définition 4.10 (Invariant polynomial)**.**
Un polynôme $g \in \mathbb{R}[\mathtt{x}_1, \ldots, \mathtt{x}_\mathtt{m}]$ est dit invariant *à la fin d'un programme c si la propriété suivante est vérifiée :*

$$\boxed{\forall \sigma \in \mathbb{R}^m,\ \forall \sigma_f \in \mathbb{R}^m, \quad \langle c, \sigma\rangle \to^* \langle \bullet, \sigma_f\rangle \quad \Rightarrow \quad \sigma_f \in \gamma(g)}$$

On remarque, d'après cette définition, que pour les programmes ne possédant aucun état terminant, tout polynôme est considéré comme étant invariant. Ceci met en valeur l'absence d'information pour de tels programmes. Notre analyse a pour objectif de calculer de tels invariants de programme. Théoriquement, on peut proposer deux techniques duales pour démontrer qu'un polynôme g est invariant. Soit par une analyse avant : on calcule l'ensemble des états accessibles et on démontre qu'ils sont des zéros de g. Soit par une analyse arrière : partant des zéros de g, on calcule l'ensemble des états terminants et on démontre que cet ensemble vaut $\mathbb{R}^m$. Dans tout ce qui suit, nous nous attelons à exposer cette analyse arrière.

4.2.2.2 Sémantique concrète arrière

Nous pouvons maintenant décrire la sémantique concrète qui est utilisée pour prouver la correction de notre analyse.

Définition 4.11 (Sémantique collectrice arrière (`SCA`))**.**

$B^\nu[\![c]\!] \ : \mathcal{P}(\mathbb{R}^m) \to \mathcal{P}(\mathbb{R}^m)$

$B^\nu[\![\mathtt{x_j} := p]\!] \ S = \{x \in \mathbb{R}^m \mid x[\![p(x)]\!]_\mathtt{j} \in S\}$
où $x[\![p(x)]\!]_\mathtt{j}$ est l'élément $(x_1, \ldots, x_{j-1}, p(x), x_{j+1}, \ldots, x_m)$

$B^\nu[\![\textbf{\textit{skip}}]\!] \ S = S$

$B^\nu[\![c_1; c_2]\!] \ S = B^\nu[\![c_1]\!] \ (B^\nu[\![c_2]\!] \ S)$

$B^\nu[\![\textbf{\textit{if}} \ p \bowtie 0 \ \textbf{\textit{then}} \ c_1 \ \textbf{\textit{else}} \ c_2]\!] \ S = (B^\nu[\![c_1]\!] \ S \bigcap [\![p \bowtie 0]\!]) \bigcup (B^\nu[\![c_2]\!] \ S \bigcap [\![p \not\bowtie 0]\!])$
où $[\![p \bowtie 0]\!] = \{x \in \mathbb{R}^m \mid p(x) \bowtie 0\}$

$B^\nu[\![\textbf{\textit{while}} \ p \bowtie 0 \ \textbf{\textit{do}} \ c]\!] \ S = \nu F_{c,p,S}$
où $F_{c,p,S} = \lambda X.([\![p \not\bowtie 0]\!] \bigcap S) \bigcup ([\![p \bowtie 0]\!] \bigcap B^\nu[\![c]\!] \ X)$

Notons que l'interprétation des affectations et des tests correspond à l'analyse concrète arrière de la définition 2.12 du Chapitre 2. Cependant, remarquons que la sémantique est relativement peu habituelle, puisqu'elle est donnée par un plus grand point fixe. Cette définition correspond à un calcul de plus faible précondition libérale et donc à une notion de correction partielle (cf Section 2.2.2.2). En comparaison, l'utilisation d'un plus petit point fixe correspondrait à un calcul de plus faible précondition et à une notion de correction totale et de ce fait, apporterait plus d'informations. Cependant, cette information supplémentaire, liée à des conditions de terminaison, n'a pas de rôle à jouer pour notre analyse. Afin de bien

comprendre les différences entre les approches par plus petit et plus grand points fixes nous développons chacune de ces deux sémantiques sur l'exemple simple du programme présenté en Figure 4.3.

Nous commençons par définir formellement la sémantique $\mathrm{B}^{\mu}[\![.]\!]$. Elle est obtenue à partir de la sémantique $\mathrm{B}^{\nu}[\![.]\!]$ en remplaçant les calculs de plus grands points fixes par des calculs de plus petits points fixes.

Définition 4.12 (Sémantique collectrice arrière par plus petit point fixe)**.**

$B^{\mu}[\![c]\!] \ : \mathcal{P}(\mathbb{R}^m) \to \mathcal{P}(\mathbb{R}^m)$

$B^{\mu}[\![\mathtt{x_j} := p]\!] \ S = \{x \in \mathbb{R}^m \mid x[\![p(x)]\!]_{\mathtt{j}} \in S\}$
où $x[\![p(x)]\!]_{\mathtt{j}}$ est l'élément $(x_1, \ldots, x_{j-1}, p(x), x_{j+1}, \ldots, x_m)$

$B^{\mu}[\![\textbf{\textit{skip}}]\!] \ S = S$

$B^{\mu}[\![c_1; c_2]\!] \ S = B^{\mu}[\![c_1]\!] \ (B^{\mu}[\![c_2]\!] \ S)$

$B^{\mu}[\![\textbf{\textit{if}} \ p \bowtie 0 \ \textbf{\textit{then}} \ c_1 \ \textbf{\textit{else}} \ c_2]\!] \ S = (B^{\mu}[\![c_1]\!] \ S \bigcap [\![p \bowtie 0]\!]) \bigcup (B^{\mu}[\![c_2]\!] \ S \bigcap [\![p \not\bowtie 0]\!])$
où $[\![p \bowtie 0]\!] = \{x \in \mathbb{R}^m \mid p(x) \bowtie 0\}$

$B^{\mu}[\![\textbf{\textit{while}} \ p \bowtie 0 \ \textbf{\textit{do}} \ c]\!] \ S = \mu G_{c,p,S}$
où $G_{c,p,S} = \lambda X.([\![p \not\bowtie 0]\!] \bigcap S) \bigcup ([\![p \bowtie 0]\!] \bigcap B^{\mu}[\![c]\!] \ X)$

Illustrons maintenant ces deux sémantiques sur l'exemple `small` de la Figure 4.3.

Calcul de $\mathrm{B}^{\mu}[\![\mathtt{small}]\!]$ et $\mathrm{B}^{\nu}[\![\mathtt{small}]\!]$.
Le programme présenté en Figure 4.3 permet de calculer le carré d'un entier n par additions successives du contenu de la variable `n` à la variable `x` qui contient initialement 0. Ainsi, le polynôme $g = \mathtt{x} - \mathtt{n}^2$ est un polynôme invariant à la fin du programme `small`.

```
small  ≡  x := 0;
          y := n;
          while y ≠ 0 do
             x := x + n;
             y := y − 1;
          ;;
```

FIGURE 4.3 – Le programme polynomial `small`.

La plus faible précondition de satisfiabilité de la relation $g = 0$ à la fin du programme est caractérisée à l'aide de l'ensemble $\mathrm{B}^{\mu}[\![\mathtt{small}]\!] \ \gamma(g)$. Commençons le

calcul de cette sémantique par sa valeur sur la structure de boucle : $\mathrm{B}^{\mu}[\![\textbf{while } \mathtt{y} \neq 0 \textbf{ do } \mathtt{x} := \mathtt{x} + \mathtt{n}; \mathtt{y} := \mathtt{y} - 1]\!]\ \gamma(g)$. Le calcul du plus petit point fixe est obtenu par itération de la fonction $F = \lambda X.([\![\mathtt{y} \neq 0]\!] \bigcap \gamma(g)) \bigcup ([\![\mathtt{y} = 0]\!] \bigcap \mathrm{B}^{\mu}[\![c]\!]\ X)$ à partir de l'ensemble vide $\emptyset$.

$$\mathrm{B}^{\mu}[\![\textbf{while } \mathtt{y} \neq 0 \textbf{ do } \mathtt{x} := \mathtt{x} + \mathtt{n}; \mathtt{y} := \mathtt{y} - 1]\!]\ \gamma(g) \ = \bigcup_{k \geq 0} F^k(\emptyset)$$

où $F^k(\emptyset)$ est défini par l'ensemble :

$$\begin{array}{rcl} F^k(\emptyset) & = & \{(x,y,n) \in \mathbb{R}^3 \mid \quad (y = 0 \ \wedge \ x = n^2) \ \vee \ (y = 1 \ \wedge \ x + n = n^2) \\ & & \qquad \vee \qquad \ldots \qquad \vee \ (y = k \ \wedge \ x + kn = n^2) \ \} \end{array}$$

Ainsi, nous avons :

$$\begin{array}{rcl} \mathrm{B}^{\mu}[\![\mathtt{small}]\!]\ \gamma(g) & = & \mathrm{B}^{\mu}[\![\mathtt{x} := 0; \mathtt{y} := \mathtt{n}]\!]\ (\ \bigcup_{k \geq 0} F^k(\emptyset)\ ) \\ & = & \{(x,y,n) \in \mathbb{R}^3 \mid \exists k \in \mathbb{N} \text{ tel que } (0,n,n) \in F^k(\emptyset)\} \\ & = & \{(x,y,n) \in \mathbb{R}^3 \mid \exists k \in \mathbb{N} \text{ tel que } n = k\} \\ & = & \mathbb{R} \times \mathbb{R} \times \mathbb{N} \end{array}$$

Ainsi, si la condition initiale $\sigma_i = (x,y,n) \in \mathbb{R} \times \mathbb{R} \times \mathbb{N}$ est satisfaite, la fin du programme est atteinte dans un état σ_f tel que $g(\sigma_f) = 0$. La condition $x \in \mathbb{R}$ peut aisément s'exprimer sous forme polynomiale (par l'ensemble $\{x \in \mathbb{R} \mid 0 = 0\}$). À l'opposé, la condition $n \in \mathbb{N}$ est plus difficile à exprimer dans $\mathbb{R}$ et nécessite l'utilisation d'un produit infini de polynômes ($\{n \in \mathbb{R} \mid \prod_{k \geq 0}(n - k) = 0\}$). L'utilisation de ce genre de polynômes ferait diverger notre analyse.

Intéressons-nous maintenant au calcul de $\mathrm{B}^{\nu}[\![\mathtt{small}]\!]\ \gamma(g)$. Il est facile de montrer que $\mathrm{B}^{\nu}[\![\mathtt{small}]\!]\ \gamma(g) = \mathbb{R}^3$. C'est bien une sur-approximation du plus petit point fixe. Ce point fixe regroupant tous les états du domaine concret, sa connaissance peut sembler apporter très peu d'information. Cependant, la précédente égalité doit être lue de la manière suivante : « la relation $g = 0$ est satisfaite *pour tout état initial* et ce, sans condition initiale ».

Pour résumer, un calcul de plus petit point fixe permet d'obtenir une information très précise et inclut notamment des conditions de terminaison qui doivent être vérifiées afin que l'invariant soit satisfait à la fin du programme. À l'opposé, un calcul de plus grand point fixe fournit une information moins précise mais plus pertinente : on renverse le point de vue en en se concentrant sur les relations satisfaites uniquement par les états finaux.

Dans le but de mieux appréhender notre sémantique concrète arrière, nous la comparons, dans la section suivante, avec la sémantique opérationnelle qui est plus classique. Nous effectuons aussi cette démarche avec la sémantique collectrice arrière par plus petit point fixe. Cette dernière sémantique, non adaptée à notre analyse, ne sera plus utilisée par la suite. Néanmoins, la comparaison avec la sémantique opérationnelle permet de traduire formellement les observations de terminaison que nous venons de décrire.

4.2.2.3 Comparaison de la sémantique concrète arrière avec la SOPP

Le théorème 4.2 compare notre sémantique concrète avec la sémantique SOPP. Il énonce que la sémantique $\mathrm{B}^{\nu}[\![.]\!]$ coïncide avec la relation $\rightarrow^{co*}$. Autrement dit, en plus de collecter des états terminants, la sémantique $\mathrm{B}^{\nu}[\![.]\!]$ contient aussi des états qui admettent une réduction infinie.

Théorème 4.2 (Sémantique concrète vs SOPP)**.**
Soit c un programme polynomial, S un sous-ensemble $\mathbb{R}^m$ et σ un élément de $\mathbb{R}^m$. On a :

$$\sigma \in B^{\nu}[\![c]\!]\ S \quad \Leftrightarrow \quad \begin{cases} \exists \sigma_f \in S, & \langle c, \sigma \rangle \rightarrow^* \langle \bullet, \sigma_f \rangle \\ \text{ou} & \langle c, \sigma \rangle \rightarrow^{\infty} \end{cases}$$

Avant de détailler la démonstration de ce théorème, commentons-le. Ce théorème est à comparer avec le théorème 4.3, plus classique, qui énonce ci-dessous que la sémantique par plus petit point fixe $\mathrm{B}^{\mu}[\![.]\!]$ coïncide avec la relation $\rightarrow^*$. Ceci revient à dire que la sémantique $\mathrm{B}^{\mu}[\![.]\!]$ collecte uniquement des états terminants. Ce type de sémantique est donc adapté à une notion de correction totale et non partielle comme celle que l'on développe.

Théorème 4.3 ($\mathrm{B}^{\mu}[\![.]\!]$ vs SOPP)**.**
Soit c un programme polynomial, S un sous-ensemble $\mathbb{R}^m$ et σ un élément de $\mathbb{R}^m$. On a :

$$\forall c \in \mathbb{P}, \forall \sigma \in B^{\mu}[\![c]\!]\ S \quad \Leftrightarrow \quad \exists \sigma_f \in S,\ \langle c, \sigma \rangle \rightarrow^* \langle \bullet, \sigma_f \rangle$$

Le théorème 4.3 est présenté ici dans un but pédagogique puisqu'il permet d'accentuer la condition de terminaison implicitement exprimée par notre sémantique concrète. Il ne sera pas utilisé par la suite et sa démonstration (proche de celle du théorème 4.2) sera uniquement développée dans l'Annexe B.

Le théorème 4.2 a lui aussi un intérêt pédagogique. Il permet de bien comprendre ce que regroupe la sémantique $\mathrm{B}^{\nu}[\![.]\!]$. D'autre part, il permet de traduire les propriétés énoncées à l'aide de la sémantique SOPP en des propriétés sur $\mathrm{B}^{\nu}[\![.]\!]$. Le résultat suivant illustre cet aspect : il énonce la définition des invariants polynomiaux à l'aide de la sémantique $\mathrm{B}^{\nu}[\![.]\!]$ et est un corollaire direct du théorème 4.2.

Corollaire 4.1 (Caractérisation des invariants polynomiaux par $\mathrm{B}^{\nu}[\![.]\!]$)**.**
Soit $c \in \mathbb{P}$ et $g \in \mathbb{R}[\mathtt{x_1}, \ldots, \mathtt{x_m}]$. Alors :

$$\boxed{\begin{array}{l} g \text{ est invariant polynomial} \\ \text{à la fin du programme } c \end{array} \quad \Leftrightarrow \quad B^{\nu}[\![c]\!]\ \gamma(g) = \mathbb{R}^m}$$

Démonstration. Soit $c \in \mathbb{P}$ et $g \in \mathbb{R}[\mathtt{x_1}, \ldots, \mathtt{x_m}]$. Nous démontrons séparément chaque sens de l'équivalence.

Sens ($\Rightarrow$)
Supposons que g est un invariant polynomial à la fin de c. Soit $\sigma \in \mathbb{R}^m$. Montrons que σ appartient à $\mathrm{B}^\nu[\![c]\!]\ \gamma(g)$. Étant donné que la sémantique opérationnelle n'admet pas d'état bloquant, alors soit il existe σ' tel que $\langle c, \sigma\rangle \to^* \langle \bullet, \sigma'\rangle$ soit $\langle c, \sigma\rangle \to^\infty$. Notons au passage que ce dernier résultat est lui aussi démontré en *Coq*. Dans le cas où $\langle c, \sigma\rangle \to^\infty$, en instanciant la propriété 4.2 avec $S = \gamma(g)$, on peut conclure que σ appartient à $\mathrm{B}^\nu[\![c]\!]\ \gamma(g)$. Dans le cas où $\langle c, \sigma\rangle \to^* \langle \bullet, \sigma'\rangle$, le fait que g est un invariant polynomial permet d'affirmer que $\sigma' \in \gamma(g)$. On en conclut, par le lemme 4.1, que l'état σ appartient à $\mathrm{B}^\nu[\![c]\!]\ \gamma(g)$ ce qui termine la démonstration de ce sens de l'équivalence.

Sens ($\Leftarrow$)
Supposons que $\mathrm{B}^\nu[\![c]\!]\ \gamma(g) = \mathbb{R}^m$. Soit σ' tel que $\langle c, \sigma\rangle \to^* \langle \bullet, \sigma'\rangle$. Il s'agit alors de démontrer que l'état σ' appartient à $\gamma(g)$ pour conclure ce cas. Comme $\mathrm{B}^\nu[\![c]\!]\ \gamma(g) = \mathbb{R}^m$, l'état σ est dans l'ensemble $\mathrm{B}^\nu[\![c]\!]\ \gamma(g)$. Par le lemme 4.2 on obtient soit l'existence de $\sigma_f \in \gamma(g)$ tel que $\langle c, \sigma\rangle \to^* \langle \bullet, \sigma_f\rangle$ soit le fait que $\langle c, \sigma\rangle \to^\infty$. Ce dernier cas est écarté par le déterminisme de la relation $\to^*$ (cas **(Det2)** de la propriété 4.1 de l'Annexe B) puisque l'on a supposé que $\langle c, \sigma\rangle \to^* \langle \bullet, \sigma'\rangle$. Si l'on considère maintenant le cas où $\langle c, \sigma\rangle \to^* \langle \bullet, \sigma_f\rangle$, le déterminisme de la relation $\to^*$ (cas **(Det1)** de la propriété 4.1 de l'Annexe B) permet d'affirmer que $\sigma' = \sigma_f$. Ainsi, l'état σ' appartient bien à l'ensemble $\gamma(g)$, ce qui conclut la démonstration de ce sens de l'équivalence. □

Passons maintenant à la démonstration du théorème 4.2. Afin d'améliorer la lisibilité de sa démonstration, nous la découpons en deux lemmes énonçant chacun un sens de l'équivalence.

Lemme 4.1.
Soit c un programme polynomial, S un sous-ensemble $\mathbb{R}^m$ et σ un élément de $\mathbb{R}^m$. On a :

$$\sigma \in B^\nu[\![c]\!]\ S \quad \Leftarrow \quad \begin{cases} \exists \sigma_f \in S, & \langle c, \sigma\rangle \to^* \langle \bullet, \sigma_f\rangle \\ \text{ou} & \langle c, \sigma\rangle \to^\infty \end{cases}$$

Ce résultat a donné lieu à une démonstration réalisée grâce à l'assistant de preuve *Coq*. Nous ne détaillerons pas plus la démonstration de ce résultat ici et renvoyons le lecteur intéressé à la lecture de la Section 4.3.3 qui présente brièvement la preuve réalisée en *Coq*.

Traitons maintenant le second sens de l'équivalence.

Lemme 4.2.
Soit c un programme polynomial, S un sous-ensemble $\mathbb{R}^m$ et σ un élément de $\mathbb{R}^m$. On a :

$$\sigma \in B^\nu[\![c]\!]\ S \quad \Rightarrow \quad \begin{cases} \exists \sigma_f \in S, & \langle c, \sigma\rangle \to^* \sigma_f \\ \text{ou} & \langle c, \sigma\rangle \to^\infty \end{cases}$$

La démonstration du lemme 4.2 se fait par induction. Cependant, contrairement au cas de propriétés classiques comme celle énonçant le déterminisme de $\rightarrow^*$ (propriété 4.1 de l'Annexe B), ce lemme manipule la notion de clôture infinie $\rightarrow^\infty$. Sa démonstration est, de ce fait, plus technique et plus intéressante. Nous avons donc décidé de l'intégrer à ce manuscrit. De manière informelle, la possibilité d'effectuer une réduction infinie apparaît naturellement lors de l'étude de la structure de boucle. Sans entrer trop dans les détails, les boucles sont dépliées jusqu'au moment où l'on en sort. En cas de non sortie sortie de boucle, on démontre que l'on peut exhiber une réduction du type $\langle c, \sigma\rangle \rightarrow^i \langle c', \sigma'\rangle$ avec i arbitrairement grand. Ceci prouve que l'état σ se réduit infiniment par c. La démonstration suivante présente rigoureusement ces aspects. Notons que nous avons aussi réalisé une démonstration de ce lemme à l'aide de l'assistant de preuve *Coq*.

Démonstration.
Soit $c \in \mathbb{P}$, $S \subseteq \mathbb{R}^m$ et $\sigma \in \mathbb{R}^m$. On raisonne par induction sur la syntaxe du programme polynomial c.

Si $c \equiv$ skip alors on a $\mathrm{B}^\nu[\![c]\!]\ S = S$, et pour tout $\sigma \in \mathbb{R}^m$,

$$\frac{}{\langle \mathbf{skip}, \sigma\rangle \rightarrow \langle \bullet, \sigma\rangle}\ \textbf{\textit{skip}}$$

La propriété est donc vérifiée en prenant $\sigma_f = \sigma$.

Si $c \equiv \mathtt{x_j} := p$ alors on a $\mathrm{B}^\nu[\![c]\!]\ S = \{x \in \mathbb{R}^m \mid x[\![p(x)]\!]_\mathtt{j} \in S\}$, et pour tout $\sigma \in \mathbb{R}^m$,

$$\frac{p(\sigma) = v}{\langle \mathtt{x_j} := p, \sigma\rangle \rightarrow \langle \bullet, \sigma[\![v]\!]_\mathtt{j}\rangle}\ \textbf{\textit{assign}} \quad \textit{où} \quad \sigma[\![v]\!]_\mathtt{j} = (\sigma_1, \ldots, \sigma_{j-1}, v, \sigma_{j+1}, \ldots, \sigma_m)$$

La propriété est donc vérifiée en prenant $\sigma_f = \sigma[\![v]\!]_\mathtt{j}$.

Si $c \equiv c_1; c_2$ alors on a

$$\mathrm{B}^\nu[\![c_1; c_2]\!]\ S = \mathrm{B}^\nu[\![c_1]\!]\ (\mathrm{B}^\nu[\![c_2]\!]\ S)$$

Par hypothèse d'induction, on a :

1. $\sigma' \in \mathrm{B}^\nu[\![c_2]\!]\ S \ \Rightarrow\ \begin{cases} \exists \sigma'' \in S, & \langle c_2, \sigma'\rangle \rightarrow^* \langle \bullet, \sigma''\rangle \\ \quad\quad ou & \langle c_2, \sigma'\rangle \rightarrow^\infty \end{cases}$
2. $\sigma \in \mathrm{B}^\nu[\![c_1]\!]\ (\mathrm{B}^\nu[\![c_2]\!]\ S) \ \Rightarrow\ \begin{cases} \exists \sigma' \in \mathrm{B}^\nu[\![c_2]\!]\ S, & \langle c_1, \sigma\rangle \rightarrow^* \langle \bullet, \sigma'\rangle \\ \quad\quad ou & \langle c_1, \sigma\rangle \rightarrow^\infty \end{cases}$

Par composition des règles ***seq*** et ***seq_point*** on obtient les dérivations suivantes :

$$\begin{tikzcd}
\langle c_1; c_2, \sigma\rangle \arrow[r, "*"] \arrow[dr] & \langle \bullet; c_2, \sigma'\rangle \arrow[r] & \langle c_2, \sigma'\rangle \arrow[r, "*"] \arrow[dr] & \sigma'' \\
& \infty & & \infty
\end{tikzcd}$$

Ainsi, soit c diverge, soit $\langle c, \sigma\rangle \rightarrow^* \langle \bullet, \sigma_f\rangle$ en prenant $\sigma_f = \sigma''$.

<u>**Si** $c \equiv$ **if** $p \bowtie 0$ **then** c_1 **else** c_2</u> alors on a :

$$\mathrm{B}^{\nu}[\![c]\!]\ S = (\mathrm{B}^{\nu}[\![c_1]\!]\ S \bigcap [\![p \bowtie 0]\!]) \bigcup (\mathrm{B}^{\nu}[\![c_2]\!]\ S \bigcap [\![p \not\bowtie 0]\!])$$

Considérons maintenant $\sigma \in \mathrm{B}^{\nu}[\![c]\!]\ S$; deux cas peuvent se produire :

soit $p(\sigma) \bowtie 0 \equiv \mathit{true}$ alors $\sigma \in \mathrm{B}^{\nu}[\![c_1]\!]\ S$. L'hypothèse d'induction permet d'affirmer que $\exists \sigma' \in S, \langle c_1, \sigma \rangle \rightarrow^* \langle \bullet, \sigma' \rangle$ ou $\langle c_1, \sigma \rangle \rightarrow^{\infty}$. Et la sémantique petit pas fournit le premier pas de réduction :

$$\frac{p(\sigma) \bowtie 0 \equiv \mathit{true}}{\langle \textbf{if}\ p \bowtie \mathit{0}\ \textbf{then}\ c_1\ \textbf{else}\ c_2, \sigma \rangle \rightarrow \langle c_1, \sigma \rangle}\ \textbf{\textit{if}}$$

soit $p(\sigma) \bowtie 0 \equiv \mathit{false}$. C'est le cas miroir. On a $\sigma \in \mathrm{B}^{\nu}[\![c_2]\!]\ S$. Par hypothèse d'induction on a $\exists \sigma' \in S, \langle c_2, \sigma \rangle \rightarrow^* \sigma'$ ou $\langle c_2, \sigma \rangle \rightarrow^{\infty}$. De plus :

$$\frac{p(\sigma) \bowtie 0 \equiv \mathit{false}}{\langle \textbf{if}\ p \bowtie \mathit{0}\ \textbf{then}\ c_1\ \textbf{else}\ c_2, \sigma \rangle \rightarrow \langle c_2, \sigma \rangle}\ \textbf{\textit{if}}$$

Ainsi, soit c diverge, soit $\langle c, \sigma \rangle \rightarrow^* \langle \bullet, \sigma_f \rangle$ en prenant $\sigma_f = \sigma'$.

<u>**Si** $c \equiv$ **while** $p \bowtie 0$ **do** c_1</u> alors la sémantique arrière fournit $\sigma \in \nu F_{c_1,p,S}$ où :

$$F_{c_1,p,S} = \lambda X.([\![p \not\bowtie 0]\!] \bigcap S) \bigcup ([\![p \bowtie 0]\!] \bigcap \mathrm{B}^{\nu}[\![c_1]\!]\ X)$$

La sémantique concrète arrière de l'instruction *while* est obtenue par itération depuis le plus grand élément $\mathbb{R}^m$. Ainsi, on a :

$$\mathrm{B}^{\nu}[\![c]\!]\ S = \bigcap\nolimits_{n \geq 1} F^n_{c_1,p,S}(\mathbb{R}^m)$$

et, comme $\sigma \in \mathrm{B}^{\nu}[\![c]\!]\ S$, alors pour tout $i \in \mathbb{N}$, $\sigma \in F^i_{c_1,p,S}(\mathbb{R}^m)$.

Puisque $\mathrm{B}^\nu[\![c_1]\!]$ est un $\bigcup$-morphisme, on a :

$$
\begin{array}{rcll}
F^i_{c_1,p,S}(\mathbb{R}^m) & = & [\![p \not\bowtie 0]\!] \bigcap S & \\
 & & & (0) \\
\bigcup & & (\ [\![p \bowtie 0]\!] \bigcap \mathrm{B}^\nu[\![c_1]\!] \ ([\![p \not\bowtie 0]\!] \bigcap S) \ ) & \\
 & & & (1) \\
\bigcup & & (\ [\![p \bowtie 0]\!] \bigcap \mathrm{B}^\nu[\![c_1]\!] \ ([\![p \bowtie 0]\!] \bigcap \mathrm{B}^\nu[\![c_1]\!] \ ([\![p \not\bowtie 0]\!] \bigcap S)) \ ) & \\
 & & & (2) \\
\bigcup & & \cdots & \\
 & & \vdots & \\
\bigcup & & \cdots & \\
\bigcup & & (\ [\![p \bowtie 0]\!] \bigcap \mathrm{B}^\nu[\![c_1]\!] \ ([\![p \bowtie 0]\!] \bigcap \cdots \bigcap \mathrm{B}^\nu[\![c_1]\!] \ ([\![p \not\bowtie 0]\!] \bigcap S)) \cdots)) \ ) & \\
 & & & (i-1) \\
\bigcup & & (\ [\![p \bowtie 0]\!] \bigcap \mathrm{B}^\nu[\![c_1]\!] \ ([\![p \bowtie 0]\!] \bigcap \cdots \bigcap \mathrm{B}^\nu[\![c_1]\!] \ ([\![p \bowtie 0]\!]) \bigcap \mathrm{B}^\nu[\![c_1]\!] \ (\mathbb{R}^m)) \cdots)) \ ) & \\
 & & & (i)
\end{array}
$$

$F^i_{c_1,p,S}(\mathbb{R}^m)$ est défini par l'union de $i+1$ ensembles, correspondant aux dépliages successifs de la boucle. Dans le reste de la démonstration, nous devons savoir précisément dans lequel de ces $i+1$ ensembles nous pouvons trouver σ. Dans ce but, nous notons ${}_j.F^i$ l'ensemble apparaissant en ligne j de la décomposition de $F^i_{c_1,p,S}(\mathbb{R}^m)$. Afin de supprimer toute ambiguïté, nous définissons formellement ${}_j.F^i$ par la récurrence suivante :

$$
\left\{
\begin{array}{rcl}
{}_0.F^1 & = & [\![p \not\bowtie 0]\!] \bigcap S \\
{}_j.F^{j+1} & = & [\![p \bowtie 0]\!] \bigcap \mathrm{B}^\nu[\![c_1]\!] \ ({}_{j-1}.F^j), \ \forall j \geq 1 \\
 & & \\
{}_1.F^1 & = & [\![p \bowtie 0]\!] \bigcap \mathrm{B}^\nu[\![c_1]\!] \ (\mathbb{R}^m) \\
{}_j.F^j & = & [\![p \bowtie 0]\!] \bigcap \mathrm{B}^\nu[\![c_1]\!] \ ({}_{j-1}.F^{j-1}), \ \forall j \geq 2 \\
 & & \\
{}_t.F^{j+1} & = & {}_t.F^{t+1}, \ \forall 1 \leq t \leq j
\end{array}
\right.
$$

La démonstration est basée sur l'idée suivante.
Si nous pouvons trouver une dérivation de $\langle c, \sigma \rangle$ qui atteint l'ensemble $[\![p \not\bowtie 0]\!]$, alors, en dépliant la boucle une fois de plus, nous rendons cette réduction terminante avec ce pas supplémentaire. Ce cas est caractérisé par le fait que σ appartient à l'ensemble ${}_t.F^i$ pour un certain t tel que $0 \leq t \leq i-1$. Notons que, grâce à la dernière ligne de la récurrence ci-dessus, cette propriété peut être reformulée par $\sigma \in {}_{i-1}.F^i$ pour un certain $i \geq 1$.
Si nous ne pouvons pas trouver de telle dérivation, cela signifie que l'on se réduit infiniment dans l'ensemble $[\![p \bowtie 0]\!]$ et qu'on ne quitte jamais la

boucle. Ce cas est caractérisé par le fait que, pour tout $i \geq 1$, l'état σ appartient à ${}_i.F^i$.
Notons que l'une ou l'autre de ces caractérisations est réalisée. Le reste de la démonstration consiste à formaliser cette idée.

Le lemme suivant exprime ce qui se déroule si le premier cas est vérifié.

Lemme 4.3 (Premier cas)**.**
Soit $c \equiv$ ***while*** $p \bowtie 0$ ***do*** c_1. *Alors :*

$$\forall \sigma \in \mathbb{R}^m, \forall j \geq 1, \left(\sigma \in {}_{j-1}.F^j \Rightarrow \left\{ \begin{array}{rl} \exists \sigma_f \in S, & \langle c, \sigma \rangle \rightarrow^* \langle \bullet, \sigma_f \rangle \\ \text{ou} & \langle c, \sigma \rangle \rightarrow^\infty \end{array} \right. \right)$$

Démonstration.
Par récurrence sur j. Soit $\sigma \in \mathbb{R}^m$.
– Si $j = 1$. Alors $\sigma \in {}_0.F^1 = [\![p \not\bowtie 0]\!] \bigcap S$ et

$$\frac{p(\sigma) \bowtie 0 \equiv \mathit{false}}{\langle c, \sigma \rangle \rightarrow \langle \bullet, \sigma \rangle} \textbf{\textit{if}}$$

On conclut ce cas en prenant $\sigma_f = \sigma$.
– Supposons maintenant la propriété vérifiée au rang j et soit $\sigma \in {}_j.F^{j+1} = [\![p \bowtie 0]\!] \bigcap \mathrm{B}^\nu[\![c_1]\!]\ ({}_{j-1}.F^j)$. Alors, on a :

$$\frac{p(\sigma) \bowtie 0 \equiv \mathit{true}}{\langle c, \sigma \rangle \rightarrow \langle c_1; c, \sigma \rangle} \textbf{\textit{if}}$$

et finalement

$$\begin{array}{ccccccc} \langle c_1; c, \sigma \rangle & \longrightarrow^* & \langle \bullet; c, \sigma' \rangle & \longrightarrow & \langle c, \sigma' \rangle & \longrightarrow^* & \sigma'' \\ & \searrow \infty & & & & \searrow \infty & \end{array}$$

La première étape de réduction est le résultat de l'induction syntaxique : comme $\sigma \in {}_j.F^{j+1} \subseteq \mathrm{B}^\nu[\![c_1]\!]\ ({}_{j-1}.F^j)$, soit il existe un état σ' tel que $\sigma' \in {}_{j-1}.F^j$ et $\langle c_1, \sigma \rangle \rightarrow^* \langle \bullet, \sigma' \rangle$ soit $\langle c_1, \sigma \rangle \rightarrow^\infty$. Le pas suivant est donné par la règle ***seq_point***. La dernière étape provient de l'hypothèse de récurrence. On conclut ce cas en prenant $\sigma_f = \sigma''$.

□

Le second cas se traite à l'aide du lemme suivant.

Lemme 4.4 (Second cas)**.**
Soit $c \equiv$ ***while*** $p \bowtie 0$ ***do*** c_1. *Alors :*

$$\forall \sigma \in \mathbb{R}^m, \forall j \geq 1, \left(\sigma \in {}_j.F^j \Rightarrow \left\{ \begin{array}{l} \exists i_j \geq j-1, \sigma' \in [\![p \bowtie 0]\!], \quad \langle c, \sigma \rangle \rightarrow^{i_j} \langle c, \sigma' \rangle \\ \text{ou} \quad \langle c, \sigma \rangle \rightarrow^{\infty} \end{array} \right. \right)$$

Démonstration.
Par récurrence sur j. Soit $\sigma \in \mathbb{R}^m$.
- Si $j = 1$. Alors $\sigma \in {}_1.F^1 = [\![p \bowtie 0]\!] \bigcap \mathrm{B}^\nu[\![c_1]\!]\ (\mathbb{R}^m)$. Comme $\langle c, \sigma \rangle \rightarrow^0 \langle c, \sigma \rangle$, on conclut ce cas en prenant $\sigma' = \sigma$.
- Supposons la propriété vérifiée au rang j et prenons σ tel que $\sigma \in {}_{j+1}.F^{j+1} = [\![p \bowtie 0]\!] \bigcap \mathrm{B}^\nu[\![c_1]\!]\ ({}_j.F^j)$. Alors, on a :

$$\frac{p(\sigma) \bowtie 0 \equiv \mathit{true}}{\langle c, \sigma \rangle \rightarrow \langle c_1; c, \sigma \rangle}\ \textbf{\textit{if}}$$

ce qui fournit, de manière analogue au lemme précédent :

$$\begin{array}{ccccccc} \langle c_1; c, \sigma \rangle & \longrightarrow^* & \langle \bullet; c, \sigma' \rangle & \longrightarrow & \langle c, \sigma' \rangle & \longrightarrow^{i_j} & \langle \bullet, \sigma'' \rangle \\ \searrow & & & & \searrow & & \\ & \infty & & & & \infty & \end{array}$$

La première étape de réduction est le résultat de l'induction syntaxique : comme $\sigma \in {}_{j+1}.F^{j+1} \subseteq \mathrm{B}^\nu[\![c_1]\!]\ ({}_j.F^j)$, soit il existe un état σ' tel que $\sigma' \in {}_j.F^j$ et $\langle c_1, \sigma \rangle \rightarrow^* \sigma'$ soit $\langle c_1, \sigma \rangle \rightarrow^\infty$. L'étape suivante résulte de l'utilisation de la règle ***seq_step***. Enfin, la dernière étape provient de l'hypothèse de récurrence. Notons que la dérivation entière est réalisée en plus de j étapes, ce qui permet de conclure ce cas.

□

Maintenant que ces deux lemmes sont énoncés et démontrés, il reste à conclure la preuve. Dans ce but, considérons l'ensemble $Aux = \{i \in \mathbb{N} \mid \sigma \in {}_{i-1}.F^i\}$. Deux cas peuvent alors se produire.
1. Soit Aux est non vide, et il existe donc $k \in Aux$.
   Alors, $\sigma \in {}_{k-1}.F^k$ et le lemme 4.3 permet de conclure ce cas.
2. Soit Aux est vide.
   Alors, $\forall j \geq 1, \sigma \in {}_j.F^j$. Par le lemme 4.4, nous obtenons que, soit il existe $j \geq 1$ tel que $(\sigma \in {}_j.F^j \Rightarrow \langle c, \sigma \rangle \rightarrow^\infty)$ soit pour tout $j \geq 1$, il existe $i_j \geq j-1, \langle c, \sigma \rangle \rightarrow^{i_j} \langle c, \sigma' \rangle$. Ceci signifie que nous pouvons exhiber des dérivations arbitrairement longues. Autrement dit, $\langle c, \sigma \rangle \rightarrow^\infty$. Ceci permet de conclure ce cas et cette démonstration.

□

4.3 Analyse approchée : génération d'invariants polynomiaux par procédure itérative

4.3.1 Sémantique abstraite

Rappelons que l'idée fondamentale derrière l'abstraction est d'exporter de manière correcte l'analyse concrète dans un monde plus grossier dans lequel les calculs sont plus simples. Dans notre cas, la sémantique concrète n'est pas calculable, du fait de la présence de points fixes dans le treillis $\mathbb{R}^m$ qui ne vérifie pas la condition de chaîne ascendante. Dans cette section, nous proposons une sémantique abstraite qui approche la sémantique concrète et dont l'ensemble sous-jacent est $\mathcal{I}$, l'ensemble des idéaux polynomiaux. L'ensemble $\mathcal{I}$ est muni d'une structure de treillis qui vérifie la condition de chaîne descendante, ce qui rend possible les calculs de plus grands points fixes.

Nous commençons par définir la sémantique abstraite des programmes polynomiaux et prouvons que cette abstraction approche correctement la sémantique concrète.

4.3.1.1 Abstraction des programmes polynomiaux

Définition 4.13 (Abstraction des programmes polynomiaux)**.**

$$[\![c]\!]^\sharp : \mathcal{I} \to \mathcal{I}$$

$$[\![\mathtt{x_j} := p]\!]^\sharp I = \{q[\![\mathtt{x_j} \mapsto p]\!], q \in I\}$$
où $q[\![\mathtt{x_j} \mapsto p]\!]$ *est le polynôme* $q(\mathtt{x_1}, \ldots, \mathtt{x_{j-1}}, p(\mathtt{x_1}, \ldots, \mathtt{x_m}), \mathtt{x_{j+1}}, \ldots, \mathtt{x_m})$

$$[\![\textbf{\textit{skip}}]\!]^\sharp I = I$$

$$[\![s_1; s_2]\!]^\sharp I = [\![s_1]\!]^\sharp([\![s_2]\!]^\sharp I)$$

$$[\![\textbf{\textit{if}}\ p \neq 0\ \textbf{\textit{then}}\ c_1\ \textbf{\textit{else}}\ c_2]\!]^\sharp I = <p.([\![c_1]\!]^\sharp I), \mathtt{Rem}([\![c_2]\!]^\sharp I, p)>$$

$$[\![\textbf{\textit{if}}\ p = 0\ \textbf{\textit{then}}\ c_1\ \textbf{\textit{else}}\ c_2]\!]^\sharp I = <p.([\![c_2]\!]^\sharp I), \mathtt{Rem}([\![c_1]\!]^\sharp I, p)>$$

$$[\![\textbf{\textit{while}}\ p \neq 0\ \textbf{\textit{do}}\ c]\!]^\sharp I = \nu(F^\sharp_{c,p,I})$$
où $F^\sharp_{c,p,I} = \lambda J.<p.([\![c]\!]^\sharp J), \mathtt{Rem}(I, p)>$

$$[\![\textbf{\textit{while}}\ p = 0\ \textbf{\textit{do}}\ c]\!]^\sharp I = \nu(\overline{F}^\sharp_{c,p,I})$$
où $\overline{F}^\sharp_{c,p,I} = \lambda J.<p.I, \mathtt{Rem}([\![c]\!]^\sharp J, p)>$

Il convient de faire quelques remarques sur cette sémantique. Tout d'abord, remarquons que, puisque notre sémantique agit en arrière, la sémantique des affectations consiste en une simple substitution et ne nécessite pas l'introduction de nouvelles variables. La présence de l'opérateur `Rem` dans la sémantique du **if** doit être détaillée. Considérons donc une instruction **if** gardée par une (dés)égalité polynomiale p. Sa sémantique est basée sur l'idée simple suivante : si l'on souhaite prouver que la relation $g = 0$ est satisfaite sachant que la relation $p = 0$ l'est, il suffit de calculer $\texttt{Rem}(g, p) = g - pq$ pour un certain polynôme q et de montrer que la relation $\texttt{Rem}(g, p) = 0$ est satisfaite. Soulignons que cette propriété ne dépend pas du choix de q ; en particulier, ce choix n'a pas d'influence sur la correction de notre approche. Cependant, certains choix ne sont pas très pertinents et pourraient mener plus tard à la génération de l'invariant trivial "$0 = 0$". Des détails additionnels sur la façon de trouver un opérateur de division adapté sont donnés plus loin (Section 4.4). Finalement, notons que puisque $\mathcal{I}$ est un treillis satisfaisant la condition de chaîne descendante, les plus grands points fixes sont calculables dans ce treillis.

Remarque 4.1.
*Dans le cas spécial du programme $c \equiv$ **if** $p \neq 0$ **then** c_1 **else** skip, le meilleur choix de quotient q est fourni par $q = 0$, ce qui permet de définir $\texttt{Rem}(g, p) = g$. La sémantique abstraite de c est alors donnée par $[\![c]\!]^\sharp I = <I, p.([\![c_1]\!]^\sharp I)>$. Ceci coïncide avec la fonction abstraite développée dans le travail de Müller-Olm et Seidl, qui est prouvée être une abstraction exacte de la fonction de transfert concrète [MOS04].*

La sémantique du **while** est dérivée de celle du **if**. Notons que, similairement au cas de la sémantique abstraite, cette sémantique est définie à l'aide d'un plus grand point fixe.

La fonction de transfert abstraite du **while** est calculée en utilisant une itération à la Kleene, partant de l'élément $\top^\sharp = <0>$, le plus grand élément du treillis $\mathcal{I}$. Plus précisément, pour tout $V \in \mathcal{I}$,

$$[\![\textbf{while } p \neq 0 \textbf{ do } c]\!]^\sharp V \quad = \quad \bigsqcap^\sharp_{n \geq 0} (F^\sharp_{c,p,V})^n (\top^\sharp)$$

D'après le théorème 4.1, cette itération termine en un nombre fini de pas. Comme $\{(F^\sharp_{c,p,V})^n(<0>)\}_{n \in \mathbb{N}}$ est une suite croissante[4] pour l'inclusion des idéaux, le critère d'arrêt consiste à vérifier si les polynômes calculés à l'étape $n + 1$ appartiennent à l'idéal généré à l'étape n.

Remarque 4.2.
*Puisque la fonction de transfert abstraite de l'instruction **while** provient de l'écriture $c \equiv$ **if** $p \neq 0$ **then** c_1 **else** skip, et en accord avec la Remarque 4.1, la définition abstraite de toute instruction **while** gardée par une déséglalité polynomiale*

4. On remarque que cette suite est décroissante si l'on considère l'ordre induit par $\sqsubseteq^\sharp$.

est donnée par l'opérateur de division trivial qui ne modifie pas son polynôme d'entrée. Ceci permet de réécrire $F^{\sharp}_{c,p,I}$ *en* $\lambda J.<p.([\![c]\!]^{\sharp}J), I>$.

4.3.1.2 Correction de l'abstraction

Nous devons prouver que notre sémantique abstraite est correcte relativement à la sémantique concrète. Ces deux sémantiques sont comparées grâce à la connexion de Galois définie dans la Section 4.1.

Théorème 4.4 (Correction)**.**
Soit g *un polynôme de* $\mathbb{R}[\mathtt{x_1}, \ldots, \mathtt{x_m}]$ *et* c *un programme polynomial. Alors :*

$$\boxed{\gamma([\![c]\!]^{\sharp}<g>) \quad \subseteq \quad B^{\nu}[\![c]\!]\ \gamma<g>} \tag{4.1}$$

La démonstration du théorème 4.4 se fait par induction sur la syntaxe des programmes polynomiaux.

Démonstration.
Soit I un idéal. Notons que, par le théorème de Hilbert (théorème 4.1), I est finiment engendré. Ainsi, il existe $g_1, \ldots, g_s \in \mathbb{R}[\mathtt{x_1}, \ldots, \mathtt{x_m}]$ tels que $I = <g_1, \ldots, g_s>$.

Si $c \equiv$ **skip** alors

$$\begin{array}{rcl} [\![c]\!]^{\sharp}I & = & I \\ \mathrm{B}^{\nu}[\![c]\!]\ \gamma(I) & = & \gamma(I) \end{array}$$

Et ainsi, ce cas est trivial.

Si $c \equiv \mathtt{x_j} := p$ alors

$$\begin{array}{rcl} [\![c]\!]^{\sharp}I & = & I[\![\mathtt{x_j} \mapsto p]\!] \\ \mathrm{B}^{\nu}[\![c]\!]\ \gamma(I) & = & \{y \in \mathbb{R}^m \mid y[\![p(y)]\!]_{\mathtt{j}} \in \gamma(I)\} = \gamma(I[\![\mathtt{x_j} \mapsto p]\!]) \end{array}$$

où nous définissons $I[\![\mathtt{x_j} \mapsto p]\!] = <g_1[\![\mathtt{x_j} \mapsto p]\!], \ldots, g_s[\![\mathtt{x_j} \mapsto p]\!]>$.
Ce cas est lui aussi trivial.

Si $c \equiv c_1; c_2$ alors

$$\begin{array}{rcl} [\![c]\!]^{\sharp}I & = & [\![c_1]\!]^{\sharp}([\![c_2]\!]^{\sharp}(I)) \\ \mathrm{B}^{\nu}[\![c]\!]\ \gamma(I) & = & \mathrm{B}^{\nu}[\![c_1]\!]\ (\mathrm{B}^{\nu}[\![c_2]\!]\ \gamma(I)) \end{array}$$

Par hypothèse d'induction sur c_1, on a :

$$\gamma([\![c_1]\!]^{\sharp}([\![c_2]\!]^{\sharp}I)) \quad \subseteq \quad \mathrm{B}^{\nu}[\![c_1]\!]\ \gamma([\![c_2]\!]^{\sharp}I)$$

Maintenant, par hypothèse d'induction sur c_2, on a :

$$\gamma([\![c_2]\!]^{\sharp}I) \quad \subseteq \quad \mathrm{B}^{\nu}[\![c_2]\!]\ \gamma(I)$$

Comme la sémantique $\mathrm{B}^{\nu}[\![c]\!]$ est une fonction croissante, nous concluons ce cas en appliquant $\mathrm{B}^{\nu}[\![c_1]\!]$ à chaque membre de l'inclusion précédente.

Si $c \equiv$ **if** $p \neq 0$ **then** c_1 **else** c_2 alors :

$$\begin{array}{rcl} [\![c]\!]^\sharp I & = & < p.([\![c_1]\!]^\sharp I),\ \mathtt{Rem}([\![c_2]\!]^\sharp I, p) > \\ \mathrm{B}^\nu[\![c]\!]\ \gamma(I) & = & ([\![p \neq 0]\!] \bigcap \mathrm{B}^\nu[\![c_1]\!]\ \gamma(I)) \\ & \bigcup & ([\![p = 0]\!] \bigcap \mathrm{B}^\nu[\![c_2]\!]\ \gamma(I)) \end{array}$$

La variété engendrée par la sémantique abstraite est :

$$\begin{array}{rcccl} & \gamma(<p.([\![c_1]\!]^\sharp I),\ \mathtt{Rem}([\![c_2]\!]^\sharp I, p)>) & & & \\ = & \gamma(p.([\![c_1]\!]^\sharp I)) & \bigcap & \gamma(\mathtt{Rem}([\![c_2]\!]^\sharp I, p)) & \\ = & (\gamma(p) \bigcup \gamma([\![c_1]\!]^\sharp I)) & \bigcap & \gamma(\mathtt{Rem}([\![c_2]\!]^\sharp I, p)) & \\ = & \gamma(p) & \bigcap & \gamma(\mathtt{Rem}([\![c_2]\!]^\sharp I, p)) & \} \ A \\ \bigcup & \gamma([\![c_1]\!]^\sharp I) & \bigcap & \gamma(\mathtt{Rem}([\![c_2]\!]^\sharp I, p)) & \} \ B \end{array}$$

Dans le but d'améliorer la lisibilité de cette démonstration, nous introduisons les notations A et B (sur la droite de la formule précédente). Maintenant, en décomposant $\mathbb{R}^m$ comme $[\![p \neq 0]\!] \bigcup \gamma(p)$, on a :

$$\begin{array}{rlcl} & A \bigcup B & & \\ = & (A \bigcup B) \bigcap ([\![p \neq 0]\!] \bigcup \gamma(p)) & & \\ = & (\ A \bigcap ([\![p \neq 0]\!] \bigcup \gamma(p))\ ) & \bigcup & (\ B \bigcap ([\![p \neq 0]\!] \bigcup \gamma(p))\ ) \\ = & A & \bigcup & (B \bigcap [\![p \neq 0]\!]) \bigcup (B \bigcap \gamma(p)) \\ = & A & \bigcup & (B \bigcap [\![p \neq 0]\!]) \end{array}$$

Cette dernière égalité est une conséquence directe du fait que :

$$B \bigcap \gamma(p) \ = \ A \bigcap \gamma([\![c_1]\!]^\sharp I) \ \subseteq \ A$$

Afin de conclure cette démonstration, nous montrons que :

$$B \bigcap [\![p \neq 0]\!] \ \subseteq \ ([\![p \neq 0]\!] \bigcap \mathrm{B}^\nu[\![c_1]\!]\ \gamma(I)) \tag{4.2}$$

$$A \ \subseteq \ ([\![p = 0]\!] \bigcap \mathrm{B}^\nu[\![c_2]\!]\ \gamma(I)) \tag{4.3}$$

Montrons d'abord l'inégalité (4.2). Par hypothèse d'induction sur c_1, on a :

$$\gamma([\![c_1]\!]^\sharp I) \subseteq \mathrm{B}^\nu[\![c_1]\!]\ \gamma(I)$$

Alors, en intersectant chaque membre de l'inclusion par $[\![p \neq 0]\!]$, on obtient :

$$([\![p \neq 0]\!] \bigcap \gamma([\![c_1]\!]^\sharp I)) \subseteq ([\![p \neq 0]\!] \bigcap \mathrm{B}^\nu[\![c_1]\!]\ \gamma(I))$$

ce qui démontre (1).

Nous montrons maintenant l'inégalité (4.3). Soit x un élément de A et u un élément de $[\![c_2]\!]^\sharp I$. Nous notons $t = \mathtt{Rem}(u, p)$ et q le quotient associé à

cette division : $u = q.p + t$. Comme $x \in A$, $x \in \gamma(p)$ et ainsi $p(x) = 0$. Nous avons aussi $x \in \gamma(\texttt{Rem}([\![c_2]\!]^\sharp I, p))$. Ainsi, $t(x) = 0$, ce qui implique $u(x) = 0$. Nous avons démontré que pour tout $u \in [\![c_2]\!]^\sharp I$, $u(x) = 0$, ce qui signifie que $x \in \gamma([\![c_2]\!]^\sharp I)$ et prouve que $A \subseteq [\![p = 0]\!] \bigcap \gamma([\![c_2]\!]^\sharp I)$. Nous concluons la démonstration de l'inégalité (4.3) grâce à l'hypothèse d'induction sur c_2.

Si $c \equiv$ **if** $p = 0$ **then** c_1 **else** c_2 : ce cas est symétrique au précédent est peut être traité de la même façon. Nous ne le développerons donc pas.

Si $c \equiv$ **while** $p \neq 0$ **do** c_1 alors

$$\begin{array}{rcl} [\![c]\!]^\sharp I & = & \nu\lambda J. < p.([\![c_1]\!]^\sharp(J)), \texttt{Rem}(I, p) > \\ \mathrm{B}^\nu[\![c]\!]\ \gamma(I) & = & \nu\lambda X.([\![p = 0]\!] \bigcap \gamma(I)) \bigcup ([\![p \neq 0]\!] \bigcap \mathrm{B}^\nu[\![c_1]\!]\ X) \end{array}$$

Afin de traiter ce cas, très similaire au précédent, nous utilisons le théorème suivant sur les points fixes [Bac00].

Lemme 4.5 (Lemme de transfert)**.**
Soit $(\mathcal{A}, \sqsubseteq), (\mathcal{A}^\sharp, \sqsubseteq^\sharp)$ *deux treillis complets et* $\gamma : \mathcal{A}^\sharp \to \mathcal{A}$ *une fonction. Soit* $f : \mathcal{A} \to \mathcal{A}$ *et* $f^\sharp : \mathcal{A}^\sharp \to \mathcal{A}^\sharp$ *deux fonctions croissantes telles que :*

$$\gamma \circ f^\sharp \dot{\sqsubseteq} f \circ \gamma$$

Alors, on a :

$$\gamma(\nu f^\sharp) \sqsubseteq \nu f$$

Démonstration.
Appliquons l'hypothèse à $\nu f^\sharp$. Nous obtenons $\gamma(f^\sharp(\nu f^\sharp)) \sqsubseteq f(\gamma(\nu f^\sharp))$. L'élément $\nu f^\sharp$ étant un point fixe de $f^\sharp$, nous avons $f^\sharp(\nu f^\sharp) = \nu f^\sharp$. Ainsi, on a $\gamma(\nu f^\sharp) \sqsubseteq f(\gamma(\nu f^\sharp))$ ce qui signifie que $\gamma(\nu f^\sharp)$ est un pré-point fixe de f. Par définition de νf, nous concluons que $\gamma(\nu f^\sharp) \sqsubseteq \nu f$. □

Notons f la fonction concrète et $f^\sharp$ la définition abstraite définie par :

$$\begin{array}{rcl} f & = & \lambda X.([\![p = 0]\!] \bigcap \gamma(I)) \bigcup ([\![p \neq 0]\!] \bigcap \mathrm{B}^\nu[\![c_1]\!]\ X) \\ f^\sharp & = & \lambda J. < p.([\![c_1]\!]^\sharp(J)), \texttt{Rem}(I, p) > \end{array}$$

et montrons que $\gamma \circ f^\sharp \dot{\subseteq} f \circ \gamma$.
Soit J_0 un idéal. D'une part, nous avons :

$$\begin{array}{rl} & f \circ \gamma(J_0) \\ = & ([\![p = 0]\!] \bigcap \gamma(I)) \bigcup ([\![p \neq 0]\!] \bigcup \mathrm{B}^\nu[\![c_1]\!]\ \gamma(J_0)) \end{array}$$

D'autre part, nous avons :

$$\begin{array}{ll} & \gamma \circ f^\sharp(J_0) \\ = & \gamma(<p.([\![c_1]\!]^\sharp J_0), \mathtt{Rem}(I,p)>) \\ = & (\ \gamma(p) \bigcup \gamma([\![c_1]\!]^\sharp J_0)\ ) \quad \bigcap \quad \gamma(\mathtt{Rem}(I,p)) \\ = & (\ \gamma(p) \bigcap \gamma(\mathtt{Rem}(I,p))\ ) \quad \bigcup \quad (\ \gamma([\![c_1]\!]^\sharp J_0) \bigcap \gamma(\mathtt{Rem}(I,p))\ ) \\ = & (\ \gamma(p) \bigcap \gamma(\mathtt{Rem}(I,p))\ ) \quad \bigcup \quad (\ [\![p \neq 0]\!] \bigcap \gamma([\![c_1]\!]^\sharp J_0) \bigcap \gamma(\mathtt{Rem}(I,p))\ ) \end{array}$$

La dernière égalité est obtenue en raisonnant de la même manière que dans le cas du **if**, en notant $\mathbb{R}^m = [\![p = 0]\!] \bigcup [\![p \neq 0]\!]$. Maintenant, par hypothèse d'induction sur c_1, on a $\gamma([\![c_1]\!]^\sharp J_0) \subseteq \mathrm{B}^\nu[\![c_1]\!]\ \gamma(J_0)$. En raisonnant encore de la même manière que dans le cas **if**, nous obtenons $\gamma(p) \bigcap \gamma(\mathtt{Rem}(I,p)) \subseteq \gamma(p) \bigcap \gamma(I)$. En combinant ces deux inclusions, nous obtenons $\gamma \circ f^\sharp \dot{\subseteq} f \circ \gamma$, ce qui prouve le lemme 4.5 :

$$\begin{array}{ccc} \gamma(\nu f^\sharp) & \subseteq & \nu f \\ \shortparallel & & \shortparallel \\ \gamma([\![c]\!]^\sharp I) & \subseteq & \mathrm{B}^\nu[\![c]\!]\ \gamma(I) \end{array}$$

Ceci permet de conclure ce cas.

Si $c \equiv$ **while** $p = 0$ **do** c_1 : ce cas est symétrique au précédent et traité de la même manière.

□

Notons que ce théorème, et son corollaire ci-dessous sont des propriétés clés pour montrer que notre analyse produit bien des invariants polynomiaux.

Corollaire 4.2.
Soit g un polynôme de $\mathbb{R}[\mathtt{x}_1, \ldots, \mathtt{x}_\mathtt{m}]$ et c un programme polynomial.
Si g satisfait $[\![c]\!]^\sharp<g> = <0>$, alors g est un invariant polynomial à la fin du programme c.

Démonstration.
Pour un polynôme donné g, si $[\![c]\!]^\sharp<g> = <0>$, alors $\gamma([\![c]\!]^\sharp<g>) = \mathbb{R}^m$. Comme $\gamma([\![c]\!]^\sharp<g>) \subseteq \mathrm{B}^\nu[\![c]\!]\ \gamma<g>$, nous avons $\mathrm{B}^\nu[\![c]\!]\ \gamma<g> \supseteq \mathbb{R}^m$, ce qui prouve que $\mathrm{B}^\nu[\![c]\!]\ \gamma<g> = \mathbb{R}^m$. En utilisant le théorème 4.2, ceci implique que, pour tout état initial $\sigma \in \mathbb{R}^m$, s'il existe σ_f tel que $\langle c, \sigma \rangle \rightarrow^+ \sigma_f$, alors $\sigma_f \in \gamma(g)$. Ainsi, g est un invariant polynomial. □

Le Corollaire 4.2 fournit une méthode de vérification permettant de prouver qu'un polynôme g est invariant. Pour ce faire, il suffit de calculer la sémantique abstraite appliquée à $<g>$ et de comparer ce résultat avec l'idéal nul $<0>$.

4.3.2 Génération d'invariants par procédure itérative

Le Corollaire 4.2 montre que la sémantique abstraite fournit une manière de valider un candidat à l'invariance. Nous montrons maintenant comment l'utilisation de polynômes paramétrés permet de générer automatiquement des invariants.

Définition 4.14 (Polynôme a_i-paramétré linéairement, polynôme générique)**.** *Notons* $\boldsymbol{a} = \{ a_i \mid i \in \mathbb{N} \}$ *un ensemble de paramètres.*

On dira qu'un polynôme $g \in \mathbb{R}[\mathtt{x_1}, \ldots, \mathtt{x_m}]$ *est un* polynôme a_i-paramétré linéairement *(noté* a_i-`pl`*) si chacun de ses coefficients est une combinaison linéaire en les paramètres* a_i*.*

Un idéal est dit idéal a_i-paramétré linéairement *s'il est généré par des polynômes* a_i*-paramétrés linéairement.*

On appellera polynôme a_i-générique de degré d *le polynôme* a_i*-paramétré qui contient tous les monômes de degré inférieur à* d*. Par exemple,* $a_0 + a_1.\mathtt{x_1} + a_2.\mathtt{x_2} + a_3.\mathtt{x_1}\mathtt{x_2} + a_4.{\mathtt{x_1}}^2 + a_5.{\mathtt{x_2}}^2$ *est le polynôme* a_i*-générique de degré* 2 *dans* $\mathbb{R}[\mathtt{x_1}, \mathtt{x_2}]$*.*

Le polynôme a_i-générique de degré d (noté g) peut être vu comme une représentation de $\mathbb{R}_d[\mathtt{x_1}, \ldots, \mathtt{x_m}]$, l'ensemble des polynômes de degré inférieur ou égal à d. Plus précisément, tout polynôme de $\mathbb{R}_d[\mathtt{x_1}, \ldots, \mathtt{x_m}]$ est obtenu en instanciant l'ensemble des paramètres a_i. Une telle instanciation sera nommée a_i-*instanciation complète*. Une a_i*-instanciation partielle* de g permet, quant à elle, de représenter tout un ensemble de polynômes de $\mathbb{R}_d[\mathtt{x_1}, \ldots, \mathtt{x_m}]$.

L'algorithme 1 décrit les différentes étapes de la génération itérative d'invariants polynomiaux de degré d pour le programme c. Cette procédure est dite itérative car elle est définie à l'aide du calcul de la sémantique abstraite $[\![c]\!]^\sharp{<}g{>}$ qui est définie par un point fixe calculé par itérations.

Attardons-nous sur cet algorithme. La ligne 4 contient la différence la plus significative avec la procédure de vérification détaillée par le Corollaire 4.2. Nous partons du polynôme a_i-générique de degré d. L'ensemble $[\![c]\!]^\sharp{<}g{>}$ est donc composé uniquement de polynômes paramétrés. Plus précisément, on peut montrer par induction sur la syntaxe des programmes polynomiaux que l'idéal $[\![c]\!]^\sharp{<}g{>}$ est a_i-paramétré linéairement. Prouver l'égalité $[\![c]\!]^\sharp{<}g{>} = {<}0{>}$ revient à montrer que tous les polynômes de $[\![c]\!]^\sharp{<}g{>}$ sont identiquement nuls. Or, un polynôme est identiquement nul si l'ensemble de ses coefficients sont nuls. Les coefficients de $[\![c]\!]^\sharp{<}g{>}$ sont des combinaisons linéaires en les paramètres a_i. Ainsi, $\mathscr{C}_{g,c}$ est système homogène d'équations linéaires en les a_i et $\mathscr{S}_{g,c}$ peut être obtenu par un calcul de pivot de Gauss. Grâce à $\mathscr{S}_{g,c}$, chaque a_i s'écrit comme combinaison linéaire de certains des a_i (la base du système linéaire). L'ensemble $\mathcal{G}_{inv}$ est alors obtenu en considérant l'ensemble des a_i-instanciations complètes par des

input : $c \in \mathbb{P}$, $d \in \mathbb{N}$ et $\boldsymbol{a} = \{a_i \mid i \in \mathbb{N}\}$ un ensemble de paramètres
output: un ensemble de polynômes $\mathcal{G}_{inv}$

1 **begin**
2 $g :=$ le polynôme a_i-générique de degré d;
3 calcul de la sémantique abstraite $[\![c]\!]^\sharp{<}g{>}$;
4 génération de $\mathscr{C}_{g,c}$, l'ensemble des contraintes équivalentes à : $[\![c]\!]^\sharp{<}g{>} = {<}0{>}$;
5 calcul de $\mathscr{S}_{g,c}$, l'ensemble des solutions de $\mathscr{C}_{g,c}$;
6 $\mathcal{G}_{inv} :=$ ensemble de polynômes obtenu par a_i-instanciations complètes de g par des éléments de $\mathscr{S}_{g,c}$;
7 **end**

Algorithm 1: Procédure itérative pour générer des invariants polynomiaux

éléments solutions du système linéaire $\mathscr{C}_{g,c}$. Le théorème 4.5 énonce que tout polynôme de l'ensemble $\mathcal{G}_{inv}$ calculé par cette procédure itérative est invariant à la fin du programme c.

Théorème 4.5.
Soit $c \in \mathbb{P}$ et $d \in \mathbb{N}$. Les polynômes calculés par l'algorithme 1 sont des invariants du programme c dont le degré est inférieur ou égal à d.

La preuve de ce théorème est une conséquence directe du résultat énoncé par le théorème 4.4.

Démonstration.
Soit $c \in \mathbb{P}$, $d \in \mathbb{N}$ et soit $\boldsymbol{a} = \{a_i \mid i \in \mathbb{N}\}$ un ensemble de paramètres. Notons $\mathcal{G}$ l'ensemble des polynômes calculés par l'algorithme 1.
Soit $h \in \mathcal{G}$. Par définition de $\mathcal{G}$, h est tel que :

$$[\![c]\!]^\sharp{<}h{>} = {<}0{>}$$

D'autre part, par le théorème 4.4 on a :

$$\gamma([\![c]\!]^\sharp{<}h{>}) \ \subseteq \ \mathrm{B}^\nu[\![c]\!]\ \gamma(h)$$

On en conclut que $\mathrm{B}^\nu[\![c]\!]\ \gamma(h) \supseteq \gamma({<}0{>}) = \mathbb{R}^m$ et finalement que $\mathrm{B}^\nu[\![c]\!]\ \gamma(h) = \mathbb{R}^m$. Ainsi, par la caractérisation du corollaire 4.1 le polynôme h est invariant à la fin du programme c, ce qui conclut la démonstration. □

Il est important de remarquer ici que la génération de l'ensemble $\mathscr{C}_{g,c}$ peut s'avérer très coûteuse puisque le critère d'arrêt nécessite le calcul de bases de Gröbner sur des ensembles de polynômes paramétrés. Plusieurs pistes peuvent

```
1.  r := 0;
2.  s := 1;
3.  t := 1;
4.  while s ≤ n do
5.    r := r + 1;
6.    t := t + 2;
7.    s := s + t;
8.
```

I_1	$I_4[^1/_t; ^1/_s; ^0/_r]$
I_4	$I_8 \bigsqcap^\sharp I_5^1 \bigsqcap^\sharp I_5^2 \bigsqcap^\sharp I_5^3 \bigsqcap^\sharp I_5^4 \bigsqcap^\sharp I_5^5$
I_5^1	$<a_0 + a_1(r+1) + a_2(s+t+2) + a_3(t+2) + a_4 n + a_5(r+1)^2 + a_6(r+1)(s+t+2) + a_7(r+1)(t+2) + a_8(r+1)n + a_9(s+t+2)^2 + a_{10}(s+t+2)(t+2) + a_{11}(s+t+2)n + a_{12}(t+2)^2 + a_{13}(t+2)n + a_{14}n^2>$
I_8	$<a_0 + a_1 r + a_2 s + a_3 t + a_4 n + a_5 r^2 + a_6 rs + a_7 rt + a_8 rn + a_9 s^2 + a_{10} st + a_{11} sn + a_{12} t^2 + a_{13} tn + a_{14} n^2>$

FIGURE 4.4 – Un programme polynomial pour calculer une racine carrée : `sqrt`.

être abordées afin d'apporter une solution à ce problème de complexité. La solution la plus naïve est de n'effectuer qu'un nombre fixé d'itérations. Ceci permettrait de produire des polynômes *candidats* à l'invariance, qui sont plus simples à vérifier. Une autre solution, que nous développons en Section 4.4, est basée sur une hypothèse naturelle sur les relations polynomiales que l'on souhaite générer. Sous cette hypothèse dite d'inductivité des invariants, on peut prouver qu'une itération suffit pour calculer les points fixes des boucles **while** gardées par une déséglalité polynomiale.

Dans le paragraphe suivant, nous illustrons l'application de l'algorithme 1 sur l'exemple didactique de la Figure 4.4. Nous faisons délibérément le choix de présenter un exemple simple dont la garde n'est pas prise en compte, afin de pouvoir dérouler chaque étape de l'algorithme 1.

Calcul d'invariant pour le programme `sqrt`

Le programme `sqrt` a pour but de calculer la racine carrée d'une variable `n` dont la valeur est donnée en entrée. Nous instancions donc l'algorithme 1 avec le polynôme a_i-`pl` de degré 2 le plus générique :

$$\begin{array}{rcll} g & = & a_0 & + \; a_1 r + a_2 s + a_3 t + a_4 n \\ & & & + \; a_5 r^2 + a_6 rs + a_7 rt + a_8 rn \\ & & & + \; a_9 s^2 + a_{10} st + a_{11} sn \\ & & & + \; a_{12} t^2 + a_{13} tn \\ & & & + \; a_{14} n^2 \end{array}$$

Il s'agit alors de calculer $[\![\texttt{sqrt}]\!]^\sharp <g>$. Le tableau à droite de la Figure 4.4 contient les valeurs abstraites clés de notre analyse. Détaillons ces calculs. Notre analyse opérant en arrière, la première commande analysée est le **while**. La garde de cette boucle n'étant pas une (dés)égalité polynomiale, elle est remplacée par un choix

non déterministe. La boucle est dépliée jusqu'à obtention d'un point fixe. Plus précisément la sémantique abstraite de cette boucle est donnée par :

$$I_4 = [\![\textbf{while } (*) \textbf{ do } \ \mathtt{r} := \mathtt{r}+1; \mathtt{t} := \mathtt{t}+2; \mathtt{s} := \mathtt{s}+\mathtt{t}]\!]^\sharp g = <g_0, g_1, g_2, g_3, \ldots>$$

où les polynômes g_i sont définis par itération du corps de la boucle abstraite sur le polynôme g :

$$\begin{array}{rcl} g_0 & = & g \\ g_1 & = & [\![\ \mathtt{r} := \mathtt{r}+1; \mathtt{t} := \mathtt{t}+2; \mathtt{s} := \mathtt{s}+\mathtt{t}]\!]^\sharp\ g \\ g_2 & = & [\![\ \mathtt{r} := \mathtt{r}+1; \mathtt{t} := \mathtt{t}+2; \mathtt{s} := \mathtt{s}+\mathtt{t}]\!]^\sharp\ g_1 \\ \vdots & \vdots & \vdots \\ g_{i+1} & = & [\![\ \mathtt{r} := \mathtt{r}+1; \mathtt{t} := \mathtt{t}+2; \mathtt{s} := \mathtt{s}+\mathtt{t}]\!]^\sharp\ g_i \end{array}$$

Le théorème 4.1 permet d'affirmer que cette itération termine en un nombre fini de pas. Un calcul de base de Gröbner montre que $g_5 \in <g, g_1, g_2, g_3, g_4>$, ce qui prouve que :

$$I_4 = <g, g_1, g_2, g_3, g_4>$$

Il reste alors à calculer l'effet des affectations initiales sur I_4. Ceci s'effectue en opérant sur I_4 les substitutions correspondant à ces affectations. On calcule ainsi l'idéal I_1 :

$$I_1 \ = < g[^1/_t; {}^1/_s; {}^0/_r], \ \ldots \ , g_4[^1/_t; {}^1/_s; {}^0/_r] >$$

Le théorème 4.4 permet d'affirmer que cet idéal I_1 est une sous-approximation de la plus faible précondition d'appartenance à l'ensemble des zéros de g. Plus précisément :

$$\gamma(I_1) \quad \subseteq \quad \mathrm{B}^\nu[\![\mathtt{sqrt}]\!]\ (\ \gamma(g)\ )$$

Supposons maintenant $\gamma(I_1) = \mathbb{R}^m$. Dans ce cas, on a $\mathrm{B}^\nu[\![\mathtt{sqrt}]\!]\ (\gamma(g)) = \mathbb{R}^m$. Autrement dit, il n'y a pas de condition à l'appartenance aux zéros de g. Ceci revient à dire que tout état initial terminant aboutit, par exécution de `sqrt`, dans un état qui est un zéro de g qui est alors, par définition un invariant polynomial à la fin de `sqrt`.

L'algorithme 1 consiste alors à prouver que cette égalité $\gamma(I_1) = \mathbb{R}^m$ est vérifiée. On doit donc démontrer que tout polynôme de I_1 admet $\mathbb{R}^m$ comme ensemble de zéros. Cette propriété n'est satisfaite que pour le polynôme nul. Autrement dit, on cherche à démontrer :

$$(S) \left\{ \begin{array}{rcl} g[^1/_t; {}^1/_s; {}^0/_r] & = & 0 \\ \vdots & \vdots & \vdots \\ g_4[^1/_t; {}^1/_s; {}^0/_r] & = & 0 \end{array} \right.$$

Or un polynôme est nul si et seulement si tous ses coefficients sont nuls. Du fait de notre approche paramétrée, les coefficients des $g_i[^1/_t; {}^1/_s; {}^0/_r]$ sont des

combinaisons linéaires en les a_i. Le système (S) précédent est donc équivalent à un système d'équations linéaires en les a_i, noté $\mathscr{C}_{g,\texttt{sqrt}}$ dans l'algorithme 1. Par exemple, l'équation $a_0 + a_2 + a_3 + a_9 + a_{10} + a_{12} = 0$ fait partie de $\mathscr{C}_{g,\texttt{sqrt}}$ car elle est obtenue par égalisation à zéro du coefficient de degré 0 de $g[{}^1/_t; {}^1/_s; {}^0/_r]$.

La dernière étape de l'algorithme consiste à résoudre ce système. Pour cela, on exprime l'ensemble des a_i en fonction d'un ensemble base. Ici, en prenant comme base $\{a_0, a_2, a_7, a_{10}, a_{12}\}$, nous obtenons :

$$\mathscr{S}_{g,\texttt{sqrt}} = \begin{cases} a_1 = 2a_0 - a_7 - 2a_{12} & a_6 = -2a_{10} \\ a_3 = -a_0 - a_2 - a_{10} - a_{12} & a_8 = -2a_{13} \\ a_4 = -a_{13} & a_9 = a_{11} = a_{14} = 0 \\ a_5 = -a_2 - 2a_7 - a_{10} - 4a_{12} & \end{cases}$$

Cet ensemble de solutions représente les contraintes à respecter afin que tout état initial aboutisse, par exécution de `sqrt`, à un état qui est un zéro de g. Sous ces contraintes, g est invariant polynomial à la fin de `sqrt`. En substituant les valeurs de $a_1, a_3, a_4, a_5, a_6, a_8, a_9$ dans le polynôme paramétré g initial, on obtient que l'égalité suivante est vérifiée à la fin du programme et ce, pour tout $a_0, a_2, a_7, a_{10}, a_{12} \in \mathbb{R}$:

$$\begin{array}{ccccccc} & a_0(1+2r-t) & + & a_2(s-t-r^2) & + & a_7(-r-2r^2+rt) & \\ + & a_{10}(-t-r^2-2rs+st) & + & a_{12}(-2r-t-4r^2+t^2) & + & a_{13}(-n-2rn+tn) & = 0 \end{array}$$

Ceci permet de conclure qu'en fin du programme :

$$\begin{array}{ccccc} 1+2r-t=0 & , & s-t-r^2=0 & , & -r-2r^2+rt=0 \\ -t-r^2-2rs+st=0 & , & -2r-t-4r^2+t^2=0 & , & -n-2rn+tn=0 \end{array}$$

Ces invariants peuvent être résumés par les deux égalités suivantes :

$$t = 2r + 1 \quad \text{et} \quad s = (r+1)^2$$

4.3.3 Des preuves *Coq*

En plus des démonstrations mathématiques présentes dans ce manuscrit, un certain nombre de preuves ont été réalisées grâce à l'assistant de preuve *Coq*. Nous les regroupons et les décrivons brièvement dans cette section. Nous mettrons en particulier en valeur les preuves nécessitant le principe de co-induction.

Définition des sémantique `SOPP`, $\mathbf{B}^\mu[\![.]\!]$ et $\mathbf{B}^\mu[\![.]\!]$

La première étape de nos preuves *Coq* a été de formaliser les sémantiques que

l'on a étudiées en *Coq*. Dans un premier temps, nous avons décrit de manière inductive notre sémantique `SOPP`. Une fois ce travail réalisé, les opérateurs de clôture $\rightarrow^*$, $\rightarrow^\infty$ et $\rightarrow^{co*}$ se définissent comme suit.

La relation $\rightarrow^*$ est la plus petite relation vérifiant les propriétés de réflexivité et de transitivité. Elle est obtenue par interprétation inductive des règles `refl` et `trans` de la Figure 4.5.

$$\frac{}{\langle c,\sigma\rangle \rightarrow^* \langle c,\sigma\rangle}\ \texttt{refl} \qquad \frac{\langle c,\sigma\rangle \rightarrow \langle c',\sigma'\rangle \qquad \langle c',\sigma'\rangle \rightarrow^* \langle c'',\sigma''\rangle}{\langle c,\sigma\rangle \rightarrow^* \langle c'',\sigma''\rangle}\ \texttt{trans}$$

FIGURE 4.5 – Définition de la relation $\rightarrow^*$.

Les relations $\rightarrow^\infty$ et $\rightarrow^{co*}$ sont, quant à elles, définies par interprétation co-inductive [LG08] des règles précédentes. Afin, de ne pas confondre interprétation inductive et co-inductive, on renommera les règles précédentes et on représentera par une double barre ces règles qui doivent être interprétées de manière co-inductive.

La relation $\rightarrow^\infty$ est formellement définie par interprétation co-inductive des règles décrites en Figure 4.6.

$$\frac{\langle c,\sigma\rangle \rightarrow \langle c',\sigma'\rangle \qquad \langle c',\sigma'\rangle \rightarrow^\infty}{\langle c,\sigma\rangle \rightarrow^\infty}\ \texttt{inftrans}$$

FIGURE 4.6 – Définition co-inductive de la relation $\rightarrow^\infty$.

La relation $\rightarrow^{co*}$ est, quant à elle, définie par interprétation co-inductive des règles de la Figure 4.7.

$$\frac{}{\langle c,\sigma\rangle \rightarrow^{co*} \langle c,\sigma\rangle}\ \texttt{corefl} \qquad \frac{\langle c,\sigma\rangle \rightarrow \langle c',\sigma'\rangle \qquad \langle c',\sigma'\rangle \rightarrow^{co*} \langle c'',\sigma''\rangle}{\langle c,\sigma\rangle \rightarrow^{co*} \langle c'',\sigma''\rangle}\ \texttt{cotrans}$$

FIGURE 4.7 – Définition co-inductive de la relation $\rightarrow^{co*}$.

Les définitions de ces clôtures sont tirées du travail de Leroy & Grall [LG08]. En plus de ces définitions, des théorèmes liant ces clôtures sont développés et démontrés à l'aide de l'assistant de preuve *Coq*. Il est notamment démontré (et prouvé en *Coq*) que ces trois relations de clôture sont liées par l'égalité suivante :

$$\rightarrow^{co*} \quad = \quad \rightarrow^* \cup \rightarrow^\infty$$

Il est important de noter ici que les preuves liant les opérateurs de clôture sont généralement vérifiées pour toute définition de la sémantique opérationnelle. Nous avons donc pu bénéficier directement de ce type de preuves.

Nous nous sommes appuyés sur ce travail pour démontrer en *Coq* des résultats décrivant notre sémantique concrète. Par exemple, nous avons démontré que la sémantique opérationnelle `SOPP` induit une relation $\rightarrow^*$ déterministe.

Propriété 4.1 (Déterminisme de $\rightarrow^*$)**.**
La relation de clôture définie en figure 4.5 est déterministe. Plus précisément, elle vérifie les propriétés suivantes :

(Det1) $\forall c \in \mathbb{P}, \forall \sigma, \sigma', \sigma'' \in \mathbb{R}^m, \quad \langle c, \sigma \rangle \rightarrow^* \sigma' \wedge \langle c, \sigma \rangle \rightarrow^* \sigma'' \quad \Rightarrow \quad \sigma' = \sigma''$

(Det2) $\forall c \in \mathbb{P}, \forall \sigma, \sigma', \in \mathbb{R}^m, \quad \langle c, \sigma \rangle \rightarrow^* \sigma' \quad \Rightarrow \quad not(\ \langle c, \sigma \rangle \rightarrow^\infty\ )$

Démonstration. La démonstration est une conséquence directe du fait que la relation $\rightarrow$ est elle-même déterministe. Ce résultat se montre par induction structurelle sur la syntaxe des programmes polynomiaux. □

Une fois ces preuves réalisées sur la sémantique `SOPP`, nous avons défini les sémantiques $\mathrm{B}^\mu[\![.]\!]$ et $\mathrm{B}^\nu[\![.]\!]$. Ces deux sémantiques partagent les même règles. La principale différence est que la sémantique $\mathrm{B}^\mu[\![.]\!]$ est obtenue par interprétation inductive de ces règles tandis que la sémantique $\mathrm{B}^\nu[\![.]\!]$ est obtenue par interprétation co-inductive de ces même règles. Nous avons alors prouvé des propriétés telles que la monotonie sur ces deux sémantiques. Une fois ce travail réalisé, nous nous sommes intéressés à la comparaison de ces sémantiques par rapport à la sémantique `SOPP`.

Comparaison des sémantiques $\mathrm{B}^\nu[\![.]\!]$ et $\mathrm{B}^\mu[\![.]\!]$ avec la sémantique `SOPP`
La première étape de cette comparaison a été de montrer que les sémantiques $\mathrm{B}^\mu[\![.]\!]$ et $\rightarrow^*$ sont équivalentes, comme l'énonce le théorème 4.3. Ce résultat se démontre par une induction classique. Nous ne développerons donc pas cet aspect ici.

La seconde comparaison (énoncée par le théorème 4.2) consiste à démontrer l'équivalence des sémantiques $\mathrm{B}^\nu[\![.]\!]$ et $\rightarrow^{co*}$. Autrement dit, pour tout programme c, ensemble S de $\mathbb{R}^m$ et σ élément de $\mathbb{R}^m$, on a :

$$\sigma \in \mathrm{B}^\nu[\![c]\!]\ S \quad \Leftrightarrow \quad \begin{cases} \exists \sigma_f \in S, & \langle c, \sigma \rangle \rightarrow^* \langle \bullet, \sigma_f \rangle \\ \quad\quad ou & \langle c, \sigma \rangle \rightarrow^\infty \end{cases}$$

Afin de démontrer ce théorème, nous l'avons décomposé en plusieurs sous-lemmes. En particulier, nous avons démontré, en utilisant le principe de co-induction, le résultat suivant :

$$\forall \sigma \in \mathbb{R}^m, \qquad \sigma \in \mathrm{B}^\nu[\![c]\!]\ \emptyset \quad \Leftrightarrow \quad \langle c, \sigma \rangle \rightarrow^\infty$$

Ce dernier résultat, associé à la monotonie de la sémantique $\mathrm{B}^{\nu}[\![.]\!]$, permet de démontrer que les états non-terminants sont inclus dans la sémantique abstraite et ce quel que soit l'ensemble S de départ, comme l'énonce la propriété suivante.

Propriété 4.2.

$$\forall S \in \mathcal{P}(\mathbb{R}^m), \forall \sigma \in \mathbb{R}^m, \qquad \sigma \in B^{\nu}[\![c]\!]\ S \quad \Leftarrow \quad \langle c, \sigma \rangle \rightarrow^{\infty}$$

Preuve *Coq* du théorème de correction Une preuve *Coq* du théorème 4.4 est actuellement en cours de finition. Elle est à mettre à l'actif de Frédéric Besson. Le seul rôle de l'auteur a été d'expliquer l'analyse. De ce fait, nous n'entrerons pas plus dans les détails ici.

4.4 Le cas particulier des invariants inductifs

Cette section présente la principale contribution de ce chapitre. Elle est basée sur l'idée simple suivante : afin d'accélérer la convergence de la méthode, nous nous limitons à un certain type d'invariants. Plus précisément, nous nous intéressons aux invariants inductifs ou autrement dit aux invariants de boucle.

4.4.1 L'hypothèse d'inductivité

Définition 4.15 (Invariants inductifs (de boucle))**.**
Soit $c \in \mathbb{P}$, $b \in \mathbb{T}$ et considérons la boucle $\mathtt{w} \equiv$ ***while*** b ***do*** c.
Un invariant est dit inductif pour la boucle $\mathtt{w}$ *si :*

$$\boxed{\forall \sigma \in \gamma(g), \forall \sigma' \in \mathbb{R}^m, \qquad \langle c, \sigma \rangle \rightarrow^{*} \langle \bullet, \sigma' \rangle \quad \Rightarrow \quad \sigma' \in \gamma(g)}$$

Encore une fois, nous avons décrit un élément de notre analyse à l'aide de la sémantique `SOPP` afin de bien comprendre sa définition. Toutefois, nous pouvons tout aussi bien caractériser la propriété d'inductivité en utilisant seulement notre sémantique concrète $\mathrm{B}^{\nu}[\![.]\!]$: le lemme suivant énonce ce résultat. La démonstration est analogue à celle du corollaire 4.1.

Lemme 4.6.
Soit $c \in \mathbb{P}$, $b \in \mathbb{T}$ et $g \in \mathbb{R}[\mathtt{x}_1, \ldots, \mathtt{x}_m]$.
Considérons la boucle $\mathtt{w} \equiv$ ***while*** b ***do*** c. *Alors :*

$$\boxed{\begin{array}{l} g \text{ est un invariant} \\ \text{inductif pour } \mathtt{w} \end{array} \quad \Leftrightarrow \quad \gamma(g) \ \subseteq \ B^{\nu}[\![c]\!]\ \gamma(g)}$$

Démonstration.
Soit $c \in \mathbb{P}$, $b \in \mathbb{T}$ et considérons la boucle $\mathtt{w} \equiv$ **while** b **do** c. Nous démontrons séparément les deux sens de l'équivalence.

Sens ($\Rightarrow$)
Supposons que g est invariant inductif pour $\mathtt{w}$. Soit $\sigma \in \gamma(g)$. Il s'agit alors de démontrer que l'état σ est un élément de l'ensemble $\mathrm{B}^{\nu}[\![c]\!]\ \gamma(g)$. Étant donné que la sémantique opérationnelle n'admet pas d'état bloquant, alors soit il existe σ' tel que $\langle c, \sigma\rangle \rightarrow^* \langle \bullet, \sigma'\rangle$ soit $\langle c, \sigma\rangle \rightarrow^{\infty}$. Dans le cas où $\langle c, \sigma\rangle \rightarrow^{\infty}$, en instanciant la propriété 4.2 avec $S = \gamma(g)$, on peut conclure que σ appartient à $\mathrm{B}^{\nu}[\![c]\!]\ \gamma(g)$. Dans le cas où $\langle c, \sigma\rangle \rightarrow^* \langle \bullet, \sigma'\rangle$, le fait que g est un invariant inductif et que $\sigma \in \gamma(g)$ permet d'affirmer que $\sigma' \in \gamma(g)$. On en conclut, par le lemme 4.1, que l'état σ appartient à $\mathrm{B}^{\nu}[\![c]\!]\ \gamma(g)$ ce qui termine la démonstration de ce sens de l'équivalence.

Sens ($\Leftarrow$)
Supposons que $\gamma(g) \subseteq \mathrm{B}^{\nu}[\![c]\!]\ \gamma(g)$ et soit σ un élément de $\gamma(g)$. Soit σ' tel que $\langle c, \sigma\rangle \rightarrow^* \langle \bullet, \sigma'\rangle$. Il s'agit alors de démontrer que l'état σ' appartient à $\gamma(g)$ pour conclure ce cas. Comme $\gamma(g) \subseteq \mathrm{B}^{\nu}[\![c]\!]\ \gamma(g)$, l'état σ de $\gamma(g)$ est aussi dans l'ensemble $\mathrm{B}^{\nu}[\![c]\!]\ \gamma(g)$. Par le lemme 4.2 on obtient soit l'existence de $\sigma_f \in \gamma(g)$ tel que $\langle c, \sigma\rangle \rightarrow^* \langle \bullet, \sigma_f\rangle$ soit le fait que $\langle c, \sigma\rangle \rightarrow^{\infty}$. Ce dernier cas est écarté par le déterminisme de la relation $\rightarrow^*$ (cas **(Det2)** de la propriété 4.1) puisque l'on a supposé que $\langle c, \sigma\rangle \rightarrow^* \langle \bullet, \sigma'\rangle$. Si l'on considère maintenant le cas où $\langle c, \sigma\rangle \rightarrow^* \langle \bullet, \sigma_f\rangle$, le déterminisme de la relation $\rightarrow^*$ (cas **(Det1)** de la propriété 4.1) permet d'affirmer que $\sigma' = \sigma_f$. Ainsi, l'état σ' appartient bien à l'ensemble $\gamma(g)$, ce qui conclut la démonstration de ce sens de l'équivalence. □

La notion d'invariant inductif doit maintenant être traduite en utilisant la sémantique abstraite. Le théorème suivant présente une condition suffisante pour montrer qu'un polynôme g est invariant.

Théorème 4.6 (Hypothèse d'inductivité)**.**
Soit $c \in \mathbb{P}$, $b \in \mathbb{T}$ et $g \in \mathbb{R}[\mathtt{x}_1, \ldots, \mathtt{x}_m]$.
*Considérons la boucle $\mathtt{w} \equiv$ **while** b **do** c. Alors :*

$$[\![c]\!]^{\sharp}{<}g{>} = {<}g{>} \quad \Rightarrow \quad g \text{ est un invariant inductif}$$

Démonstration.
D'après le théorème 4.4, nous avons $\gamma([\![c]\!]^{\sharp}{<}g{>}) \subseteq \mathrm{B}^{\nu}[\![c]\!]\ \gamma{<}g{>}$. L'hypothèse $[\![c]\!]^{\sharp} g = {<}g{>}$ et le lemme 4.6 permettent de conclure la preuve. □

Ce théorème a une conséquence directe sur les calculs de points fixes. Le théorème 4.7 stipule que, sous l'hypothèse d'inductivité, les points fixes abstraits associés à une expression **while** gardée par une déséglalité polynomiale, sont obtenus de manière directe, sans itération.

Théorème 4.7.
*Soit $g \in \mathbb{P}$ et $\mathtt{w} \equiv$ **while** $p \neq 0$ **do** c une boucle de programme polynomial. Supposons que $[\![c]\!]^\sharp <g> = <g>$. Alors :*

$$[\![\mathtt{w}]\!]^\sharp <g> = <g>$$

Insistons sur le fait que ce théorème est un point clé de notre l'analyse. Il montre que l'on peut s'affranchir de calculs itératifs de point fixes. Ceci permet de se débarrasser de tout calcul de bases de Gröbner (nécessaire pour montrer la stabilité des procédés itératifs) et représente, de ce fait, un gain conséquent en temps d'exécution.

Démonstration.
D'après la Remarque 4.2, nous avons $[\![\textbf{while } p \neq 0 \textbf{ do } c]\!]^\sharp <g> = <g, p.([\![c]\!]^\sharp g), \ldots>$. Par hypothèse, $[\![c]\!]^\sharp <g> = <g>$. Ainsi, $p.([\![c]\!]^\sharp <g>) = p<g> \subseteq <g>$, ce qui prouve que l'itération stabilise dès le premier pas et permet de conclure la preuve. □

Remarque 4.3.
*Dans le cas de boucles munies de gardes positives, l'hypothèse d'inductivité à elle seule ne permet pas de réduire le nombre d'itérations. De ce fait, lors de la recherche d'invariants inductifs, nous choisissons de traiter ces gardes de manière non déterministe. Ceci permet de se ramener au cas précédent : si $\mathtt{w} \equiv$ **while** b **do** c est une boucle dont la garde est traitée de manière non déterministe, alors $[\![\mathtt{w}]\!]^\sharp <g> = <g, [\![c]\!]^\sharp g, \ldots>$, ce qui se réduit, sous l'hypothèse d'inductivité à $<g>$. Ainsi, sous l'hypothèse d'inductivité, toutes les boucles seront traitées de la même manière. Il faut toutefois garder à l'esprit que l'approximation est plus fine dans le cas des boucles gardées par des déséglalités polynomiales.*

4.4.2 L'analyse `fastind`

4.4.2.1 Les contraintes d'idéaux

Il s'agit maintenant d'intégrer l'hypothèse d'inductivité à notre procédure de génération d'invariants. Cette hypothèse s'énonce sous la forme d'une contrainte d'égalité d'idéaux du type $[\![c]\!]^\sharp <g> = <g>$ où c représente le corps d'une boucle. Afin de représenter une telle contrainte, nous adopterons la notation suivante.

Notation 1 (Domaine des contraintes d'idéaux)**.**
Si I_1 et I_2 sont deux idéaux a_i-`pl`, la contrainte énonçant l'égalité des idéaux I_1 et I_2 sera notée :

$$I_1 \equiv I_2$$

Le domaine de ces contraintes d'idéaux sera alors noté $\mathcal{C}^{\mathtt{i}}$.

4.4.2.2 Sémantique abstraite

L'utilisation de l'hypothèse d'inductivité donne lieu à une nouvelle analyse. Pour bien comprendre son fonctionnement, nous commençons par décrire une sémantique abstraite permettant d'intégrer les contraintes inductives.

Définition 4.16 (Abstraction des programmes sous hypothèse d'inductivité)**.**

$$[\![c]\!]^{\mathtt{ic}} : \mathcal{I} \times \mathcal{C}^{\mathtt{i}} \to \mathcal{I} \times \mathcal{C}^{\mathtt{i}}$$

$$[\![\bullet]\!]^{\mathtt{ic}}(I, C) \;=\; (I, C)$$

$$[\![\mathtt{x_j} := p]\!]^{\mathtt{ic}}(I, C) \;=\; (\{q[\![\mathtt{x_j} \mapsto p]\!], q \in I\}, C)$$

$$[\![\textbf{\textit{skip}}]\!]^{\mathtt{ic}}(I, C) \;=\; (I, C)$$

$$[\![s_1; s_2]\!]^{\mathtt{ic}}(I, C) \;=\; ([\![s_1]\!]^{\mathtt{ic}}([\![s_2]\!]^{\mathtt{ic}}(I, C))$$

$$\begin{array}{rcl}
[\![\textbf{\textit{if}}\ p \neq 0\ \textbf{\textit{then}}\ c_1\ \textbf{\textit{else}}\ c_2]\!]^{\mathtt{ic}}(I, C) & = & (<p.I_1, \mathtt{Rem}(I_2, p)>, C_1 \cup C_2) \\
[\![\textbf{\textit{if}}\ p = 0\ \textbf{\textit{then}}\ c_1\ \textbf{\textit{else}}\ c_2]\!]^{\mathtt{ic}}(I, C) & = & (<p.I_2, \mathtt{Rem}(I_1, p)>, C_1 \cup C_2) \\
\text{où}\quad [\![c_1]\!]^{\mathtt{ic}}(I, C) & = & (I_1, C_1) \\
\text{et}\quad [\![c_2]\!]^{\mathtt{ic}}(I, C) & = & (I_2, C_2)
\end{array}$$

$$\begin{array}{rcl}
[\![\textbf{\textit{while}}\ p \neq 0\ \textbf{\textit{do}}\ c]\!]^{\mathtt{ic}}(I, C) & = & (I, C_0 \cup C_w) \\
[\![\textbf{\textit{while}}\ p = 0\ \textbf{\textit{do}}\ c]\!]^{\mathtt{ic}}(I, C) & = & (I, C_0 \cup C_w) \\
\text{où}\quad [\![c_1]\!]^{\mathtt{ic}}(I, C) & = & (I_0, C_0) \\
\text{et}\quad C_w & = & (I \equiv I_0)
\end{array}$$

À première vue, cette sémantique abstraite correspond à une décoration de la sémantique $[\![.]\!]^\sharp$ à l'aide de contraintes d'idéaux. En l'observant de plus près, on s'aperçoit que les boucles ne donnent pas lieu au même calcul que celui réalisé par $[\![.]\!]^\sharp$. En fait, cette sémantique réalise l'hypothèse d'inductivité pour chacune des boucles ce qui implique que le calcul abstrait des boucles ne nécessite pas d'itération (théorème 4.7). L'hypothèse d'inductivité est, quant à elle, transportée sous forme de contraintes d'idéaux, comme deuxième élément de la paire abstraite. D'autre part, comme annoncé dans la remarque 4.3, cette analyse abstrait de la même manière des boucles gardées par des égalités polynomiales et celles gardées par des déséglalités polynomiales.

Cette sémantique abstraite permet de décrire l'algorithme 2 dont le but est de calculer des invariants inductifs.

input : $c \in \mathbb{P}$, $d \in \mathbb{N}$ et $\boldsymbol{a} = \{a_i \mid i \in \mathbb{N}\}$ un ensemble de paramètres
output: un ensemble de polynômes $\mathcal{G}_{ind}$

1 **begin**
2 $g :=$ le polynôme a_i-générique de degré d;
3 calcul de la sémantique abstraite $[\![c]\!]^{\mathtt{ic}}{<}g{>}$ $(= (I, C))$;
4 génération de $\mathscr{C}^{\mathtt{i}}_{g,c}$, ensemble des contraintes permettant de montrer : $C \cap (I = <0>)$;
5 calcul de $\mathscr{S}^{\mathtt{i}}_{g,c}$, l'ensemble des solutions de $\mathscr{C}^{\mathtt{i}}_{g,c}$;
6 $\mathcal{G}_{ind} :=$ ensemble de polynômes obtenu par a_i-instanciations complètes de g par des éléments de $\mathscr{S}^{\mathtt{i}}_{g,c}$;
7 **end**

Algorithm 2: Procédure de génération d'invariants sous l'hypothèse d'inductivité

4.4.2.3 Correction de l'analyse `fastind`

Nous montrons maintenant que l'algorithme 2 définit bien une procédure permettant de générer des invariants inductifs.

Théorème 4.8.
Soit $c \in \mathbb{P}$ et $d \in \mathbb{N}$. Les polynômes calculés par l'algorithme 2 sont des invariants à la fin du programme c, dont le degré est inférieur ou égal à d.

La preuve de ce théorème est une conséquence du résultat énoncé par le théorème 4.4.

Démonstration.
Soit $c \in \mathbb{P}$, $d \in \mathbb{N}$ et soit $\boldsymbol{a} = \{a_i \mid i \in \mathbb{N}\}$ un ensemble de paramètres. Notons $\mathcal{G}$ l'ensemble des polynômes calculés par l'algorithme 2. Soit $h \in \mathcal{G}$ et notons $[\![c]\!]^{\mathtt{ic}}{<}h{>} = (I, C)$.
Puisque l'hypothèse d'inductivité est réalisée pour toutes les boucles du programme, on a :

$$[\![c]\!]^{\sharp}{<}h{>} = I$$

D'autre part, par définition de $\mathcal{G}$, h est tel que :

$$[\![c]\!]^{\sharp}{<}h{>} = <0>$$

et par le théorème 4.4 on a :

$$\gamma([\![c]\!]^{\sharp}{<}h{>}) \ \subseteq \ \mathrm{B}^{\nu}[\![c]\!] \ \gamma(h)$$

En associant ces deux résultats, on obtient que le polynôme h est invariant à la fin du programme c, ce qui conclut la démonstration. □

4.4.3 Étude d'un exemple

L'exemple suivant illustre l'utilisation de l'algorithme 2 pour calculer des invariants sous l'hypothèse d'inductivité. Il permet aussi d'expliquer comment sont effectués les calculs de l'opérateur de division `Rem` dans ce cas. On considère le programme `mannadiv` défini par la figure 4.4.3. Ce programme permet de calculer la division entière de la valeur x_1 par la valeur x_2, ces deux valeurs étant données en entrée du programme.

```
1.   y1 := 0; y2 := 0; y3 := x1;
2.   while y3 ≠ 0 do
3.     if x2 = y2 + 1 then
4.       y1 := y1 + 1;   y2 := 0;   y3 := y3 − 1;
5.     else
6.       y2 := y2 + 1;   y3 := y3 − 1;
7.
```

FIGURE 4.8 – Le programme polynomial `mannadiv`.

Invariant inductif pour `mannadiv`.
Notons I_i l'idéal généré en déroulant la sémantique de l'instruction à la ligne i. Notre analyse prend en entrée l'idéal $I_7 = <g>$, qui contient le polynôme quadratique a_i-`pl` le plus générique (noté g), et effectue une série d'instanciation de variables avant d'évaluer l'instruction **if** de la ligne 3. Notons $g_1 = g[{}^{y_3-1}/_{y_3}; {}^{0}/_{y_2}; {}^{y_1+1}/_{y_1}]$ et $g_2 = g[{}^{y_3-1}/_{y_3}; {}^{y_2+1}/_{y_2}]$ les polynômes résultant de ces instanciations. L'idéal I_3 est défini en utilisant l'opération $\texttt{Rem}(g_1, p)$ avec $p = x_2 - y_2 - 1$. Le choix de l'opérateur de division est crucial pour notre analyse et un mauvais choix pourrait mener à un manque de précision. De ce fait, nous ne choisissons pas d'opérateur de division particulier : la division est effectuée de manière paramétrée. Plus précisément, nous introduisons le polynôme q, le polynôme b_i-`pl` le plus générique de degré $deg(g_1) - deg(p) = 1$, et définissons

$$\texttt{Rem}(g_1, p) = g_1 - q.p = g_1 - (b_0 + b_1\, x_1 + b_2\, x_2 + b_3\, y_1 + b_4\, y_2 + b_5\, y_3).(x_2 - y_2 - 1)$$

Les nouveaux paramètres b_i viennent s'ajouter aux paramètres a_i déjà définis. Ces deux types de paramètres seront traités au même niveau par notre analyse. Il faut alors noter que le polynôme $\texttt{Rem}(g_1, p)$ produit est un $\{a_i, b_i\}$-`pl`. Ceci est essentiel pour que les contraintes générées plus loin soient linéaires. Nous insistons sur le fait que le choix de l'opérateur de division n'a pas été réalisé. En fait, ce choix est effectué par l'analyse elle-même : l'analyse produit des contraintes dont la résolution finale fournira la valeur des b_i. Rappelons au passage que la preuve de correction de l'analyse a été effectuée pour tout opérateur de division. La

division paramétrée permet de fournir le « bon » opérateur de division, *c.à.d.* celui qui a le plus de chance de produire des contraintes admettant une solution et par conséquent de générer un invariant. L'idéal I_3 est alors défini par $I_3 = <(x_2 - y_2 - 1).g_2, \texttt{Rem}(g_1, p)>$.

L'instruction `while` de la ligne 2 déclenche l'utilisation de l'hypothèse d'inductivité (théorème 4.6), qui précise que le corps de la boucle maintient l'invariant original, ce qui revient à contraindre I_3 à être égal à I_7. Cette contrainte peut être satisfaite en imposant $\texttt{Rem}(g_1, p) = g$ et $g_2 = g$. Notons $\mathcal{C}_{w_1}$ et $\mathcal{C}_{w_2}$ ces deux contraintes. De plus, par le théorème 4.7 nous avons $I_2 = I_7$. Les substitutions de la ligne 1 opèrent sur I_2 ; enfin, on note $\mathcal{C}_i$ la contrainte énonçant la nullité de I_1. Il est important de noter ici que les contraintes $\mathcal{C}_{w_*}$ et $\mathcal{C}_i$ expriment deux propriétés différentes de l'invariant : la contrainte $\mathcal{C}_{w_*}$ est une contrainte d'inductivité alors que la contrainte $\mathcal{C}_i$ exprime la nullité initiale.

L'étape 4 de l'algorithme 2 consiste à résoudre la conjonction de toutes les contraintes :

$\mathcal{C}_i$	$\mathcal{C}_{w_2}$	$\mathcal{C}_{w_1}$
$a_6 + a_{10} + a_{20} = 0$	$a_{20} = a_{18}$	$a_{10} = a_8$
$a_1 + a_5 = 0$	$a_{19} = 2\,a_{18}$	$a_{20} = a_{15}$
$a_7 + a_{14} = 0$	$a_{17} = a_{16}$	$b_4 + a_{13} = 0$
$a_0 = a_2 = a_{11} = 0$	$a_4 = a_5$	$b_0 = a_{12} - a_{14}$
	$a_{10} = a_9$	$a_5 = a_3 - a_{14} + a_{12}$
	$a_7 + a_{14}$	$a_4 = a_{12} - a_{13} - a_{14}$
	$a_{14} = a_{13}$	$a_{17} = 2\,a_{15}$
		$a_{18} + a_{13} = 0$
		$a_9 = a_{16} = a_{19} = 0$
		$b_1 = b_2 = b_3 = b_5 = 0$

Ces contraintes admettent pour solution $a_{12} = a_5 = a_4 = b_0 = -a_1$, tous les autres paramètres valant 0. Ceci signifie, en particulier, que le « bon » opérateur de division consistait juste à soustraire à g_1 le polynôme $-a_1.(x_2 - y_2 - 1)$.

Finalement, l'instanciation directe du polynôme a_i-`pl` permet d'obtenir l'invariant suivant : $x_1 = y_1\,x_2 + y_2 + y_3$.

4.5 Résultats expérimentaux

L'analyse `fastind` a, dans un premier temps, donné lieu à une implémentation en Maple. Nous présentons, dans le tableau 4.9 les résultats obtenus par notre analyse sur des benchmarks tirés des articles de Rodríguez-Carbonell & Kapur [RCK07b, RCK07a]. Ces résultats ont été obtenus sur une machine Mac OS X avec processeur Intel Core 2 Duo 2.8 GHz dotée de 4 Go de RAM.

Les exemples listés dans ce tableau sont des programmes pour lesquels l'hypothèse d'inductivité est vérifiée à chaque boucle. En fait, l'ensemble de ces exemples est précisément l'ensemble des exemples de Rodríguez-Carbonell & Kapur [RCK07b, RCK07a] pour lesquels l'hypothèse d'inductivité est vérifiée.

Name	d	**Var**	`fastind`
`dijkstra`	2	5	0.043
`divbin`	2	5	0.005
`freire1`	2	3	0.006
`freire2`	2	4	0.007
`cohencu`	2	4	0.009
`fermat`	2	5	0.006
`wensley`	2	5	0.037
`euclidex`	2	8	0.008
`lcm`	2	6	0.006
`prod4`	3	6	0.013
`knuth`	3	9	0.084
`petter1`	2	2	0.003
`petter2`	3	2	0.003
`petter3`	4	2	0.004
`petter4`	5	2	0.004
`petter5`	6	2	0.006
`sqrt`	2	3	0.010
`mannadiv`	2	5	0.005
`freire2`	3	4	0.012
`cohencu`	3	4	0.018
`petter19`	20	2	0.196
`petter30`	31	2	1.423
`mannadiv2`	2	5	0.0 − −
`mannadiv3`	3	5	0.0 − −

FIGURE 4.9 – Performance de l'analyse `fastind` sur les exemples inductifs de la littérature

La première colonne liste les noms donnés aux programmes [RCK07b, RCK07a]. La deuxième présente le degré d de l'invariant généré. La troisième colonne (`Var`) représente le nombre de variables du programmes. Ainsi, chaque exemple du tableau a été testé en partant du polynôme a_i-générique de degré d de $\mathbb{R}[\mathtt{x}_1, \ldots, \mathtt{x}_{\mathtt{Var}}]$. Enfin, la dernière colonne représente le temps de calcul néces-

saire à notre analyse pour générer l'invariant souhaité. Nous avons listé en fin de tableau des exemples non présents tels quels dans la littérature.

Rappelons maintenant que l'analyse `fastind`, du fait de l'hypothèse d'inductivité, permet la génération d'invariants sans nécessiter de calcul itératif. Ceci représente un avantage important par rapport aux analyses existantes dans la littérature [MOS02, MOS04, SSM04b, RCK04a, RCK07a, RCK07b]. Nous commentons maintenant plus avant ces analyses et les comparons à l'analyse `fastind` d'un point de vue sémantique ainsi que du point de vue des résultats expérimentaux.

4.5.1 Comparaison avec le travail de Müller-Olm & Seidl

L'approche la plus semblable à la nôtre est celle de Müller-Olm & Seidl [MOS02, MOS04, MOPS06]. Nous commençons donc par comparer notre approche avec celle-ci.

4.5.1.1 Description de la méthode

Müller-Olm et Seidl ont proposé une analyse de programmes capable de calculer des invariants polynomiaux pour des programmes polynomiaux [MOS04, MOS02]. Les programmes analysés par cette méthode consistent en des systèmes de transitions étiquetés par des affectations polynomiales et des gardes sous forme de déséglités polynomiales. Le fait de ne pas considérer de programmes munis de gardes égalitaires est un choix délibéré des auteurs. Ceci permet à leur méthode d'être complète. Décrivons maintenant leur méthode. Elle consiste en une interprétation abstraite de la sémantique de traces $\mathbf{R}$ (ensemble des exécutions possibles du programme) du programme.
Les auteurs proposent une méthode de vérification : partant de la relation $p = 0$, ils montrent que la plus faible pré-condition de validité de $p = 0$ après une exécution r peut être représentée sous forme d'un polynôme noté $[\![r]\!]^T p$. Plus précisément, ils montrent qu'une relation p est valide après une exécution r si et seulement si $[\![r]\!]^T p = 0$. Une relation $p = 0$ est valide à la fin d'un programme si elle l'est à la fin de toutes les exécutions de $\mathbf{R}$. Les auteurs cherchent donc à calculer l'ensemble $S_p = \{[\![r]\!]^T p \mid r \in \mathbf{R}\}$. Cet ensemble étant possiblement infini, la méthode consiste à calculer $<S_p>$, l'idéal généré par les polynômes dans S_p. Montrer que cet idéal est l'idéal nul $<0>$, prouve que la relation $p = 0$ est valide à la fin du programme. L'idéal $<S_p>$ est calculé par des mécanismes d'itérations de points fixes. Ces itérations sont terminantes du fait du théorème 4.1 qui énonce que la condition de chaîne ascendante est vérifiée dans le monde des idéaux.
Cette méthode de vérification peut être adaptée en une méthode de génération. Pour ce faire, on part d'un polynôme g paramétré. L'idéal $<S_g>$ calculé est donc

un idéal composé de polynômes paramétrés. Cet idéal coïncide avec l'idéal nul $<0>$ si et seulement si tous les polynômes qu'il contient sont nuls. Or un polynôme est nul si et seulement si tous ces coefficients sont nuls. Ainsi, la relation paramétrée $g = 0$ est valide à la fin du programme si et seulement si le système de contraintes composé des coefficients des polynômes générant $<S_g>$ peut être résolu. La forme des fonctions abstraites utilisées pour le calcul $<S_g>$ permet de montrer que ces coefficients sont des combinaisons linéaires des paramètres. En conséquence, une fois l'idéal $<S_g>$ calculé, la validité de $p = 0$ revient à la résolution d'un système linéaire.

4.5.1.2 Comparaison sémantique

Notre approche est très similaire à celle de Müller-Olm & Seidl. Il existe cependant un certain nombre de différences. Commençons par lister les différences non fondamentales. Tout d'abord, la syntaxe de nos programmes est donnée sous forme structurée et non sous forme de système de transitions étiqueté. Nous avons aussi choisi d'intégrer directement à la syntaxe de nos programmes la possibilité d'utiliser de gardes égalitaires. Cette possibilité n'est pas présente dans l'article originel de Müller-Olm & Seidl [MOS04] mais a quand même été évoquée en tant que pistes par ces deux auteurs [MOPS06]. Plus précisément, ils proposent d'étudier deux pistes. La première est basée sur l'idée simple suivante : la plus faible précondition de validité d'une égalité $q = 0$ après une garde $p = 0$ est donnée par $(p = 0) \Rightarrow (q = 0)$. Cette implication est notamment vérifiée si l'on peut exhiber un réel $\lambda \neq 0$ tel que $q + \lambda p = 0$. Ils proposent donc, dans un premier temps, d'abstraire ce type de garde par l'idéal $<q + \lambda p>$ où λ est une variable. Ceci correspond à notre notion de reste paramétré restreinte à l'utilisation de polynômes quotients constants. La seconde piste évoquée consiste à étendre cette première idée en abstrayant maintenant ces gardes par l'idéal $<q + \lambda p.q_D>$ où q_D est un polynôme générique et où λ est toujours une variable. Notre notion de reste paramétré coïncide avec cette idée lorsque $\lambda = 1$. Mis à part ces différences, notons que la procédure itérative présentée par l'algorithme 1 correspond à une présentation différente du travail de Müller-Olm & Seidl. Au contraire, soulignons que le travail de comparaison des sémantiques concrètes (comparaison de $\mathrm{B}^{\nu}[\![.]\!]$ et `SOPP`) n'a pas, à la connaissance de l'auteur, été déjà présentée. En plus de ce travail rigoureux de comparaison des sémantiques, la contribution fondamentale de ce chapitre réside donc dans l'analyse `fastind` (présentée dans l'algorithme 2). Nous avons montré comment adapter l'analyse de Müller-Olm & Seidl dans le cas où l'on suppose l'hypothèse d'inductivité. De plus, nous avons démontré, qu'à l'aide de cette hypothèse, notre méthode ne nécessitait pas de calcul itératif de points fixes. Ceci nous permet de nous affranchir du test d'arrêt des itérations qui requiert des calculs lourds (généralement implémentés par des

calculs de bases de Gröbner). Ceci est un avantage majeur de la méthode qui permet d'envisager le passage à l'échelle. D'autre part, nous avons montré comment intégrer notre notion de reste paramétré dans l'analyse `fastind` et montré son utilité sur l'exemple du programme `mannadiv`. Nous pensons que ceci représente un pas supplémentaire par rapport au travail de Müller-Olm & Seidl qui avaient seulement évoqué ce type de possibilité sous forme de pistes. Notons que le caractère itératif de leur procédure apporte d'autres questions concernant cet aspect, comme celle du traitement abstrait d'une structure conditionnelle avec garde égalitaire imbriquée dans une boucle. On peut par exemple s'interroger sur la nécessité ou non d'utiliser le même couple (q_D, λ) à chaque itération abstraite.

4.5.1.3 Comparaison des implémentations

L'approche de Müller-Olm & Seidl a été testée sur quelques exemples [Pet04, PS05]. Le tableau de la figure 4.10 compare les résultats obtenus par leur méthode avec la nôtre. Les résultats donnés dans la colonne **MOS** sont ceux présents dans les articles cités au-dessus [Pet04, PS05]. Ils ont été obtenus sur une machine Linux avec architecture Intel, avec processeur AMD Athlon XP 3000+ et doté de 1024 MB de RAM. La dernière colonne du tableau présente le quo-

Name	d	**Var**	**MOS**	`fastind`	**Ratio** MOS/`fastind`
`lcm`	2	6	3.5	0.006	583
`petter1`	2	2	0.776	0.003	259
`petter2`	3	2	1.47	0.003	490
`petter3`	4	2	2.71	0.004	678
`petter4`	5	2	10.3	0.004	2575
`petter5`	6	2	787.2	0.006	131200
`cohencu`	3	4	2.6	0.018	144

FIGURE 4.10 – Comparaison des résultats obtenus par l'approche `fastind` et l'approche **MOS** [Pet04, PS05]

tient des temps d'exécution des deux méthodes. Le ratio est largement en faveur de l'analyse `fastind`. Ceci n'est guère étonnant. En effet, comme nous l'avons vu, l'analyse **MOS** nécessite un calcul itératif de points fixes dans le monde des idéaux. D'après le théorème 4.1, ce calcul est effectué en temps fini. Cependant, le nombre d'itérations n'est pas spécifié par ce théorème. Ainsi, ce calcul de point fixe nécessite l'utilisation d'un critère d'arrêt. Il s'agit de décider à quelle étape de l'itération l'idéal calculé arrête de grossir. Ceci est réalisé en utilisant le problème d'appartenance à un idéal. Ce problème est décidable. Cependant il nécessite le

calcul, très coûteux, de bases de Gröbner. Petter & Seidl ont démontré que l'on pouvait s'affranchir des calculs de bases de Gröbner en proposant une méthode de calcul approchée [PS05]. Les résultats du tableau de la figure 4.10 proviennent de cette approche qui n'utilise pas les bases de Gröbner. Ainsi, même si cette optimisation est nécessaire au calcul d'invariants par la méthode **MOS**, elle ne suffit pas à ce que les calculs se fassent en temps raisonnable. La forte croissance des temps de calculs sur les exemples `Petter`*i* démontre que la méthode **MOS** ne passe pas à l'échelle et que son temps d'exécution explose rapidement avec le degré des polynômes.

4.5.2 Comparaison avec le travail de Sankaranarayanan *et al.*

Sankaranarayanan *et al.* ont, dans un premier temps, proposé une approche basée contraintes [CSS03, SSM04a] permettant de générer des invariants sous forme d'inégalités linéaires. Cette approche repose essentiellement sur l'utilisation du lemme de Farkas. Dans cette section, l'analyse de Sankaranarayanan *et al.* qui nous intéresse est celle permettant de générer des invariants sous forme d'égalités polynomiales [SSM04b].

4.5.2.1 Description de la méthode

Sankaranarayanan *et al.*, à la suite de leur travail sur le cas linéaire, ont proposé une méthode permettant de générer des invariants sous forme d'égalités polynomiales. Comme dans le cas linéaire, ils proposent un processus de génération automatique basée sur des contraintes linéaires satisfaisant les conditions *d'initialisation* et de *consécution*. La plus grande différence avec le cas linéaire réside dans les résultats mathématiques que les auteurs utilisent pour convertir les conditions d'invariance en des solutions de problèmes algébriques. Les résultats d'algèbre linéaire ne sont alors plus suffisants et sont remplacés par des calculs de bases de Gröbner : les conditions d'initialisation et de consécution sont codées par des tests d'inclusion d'idéaux qui sont calculés à l'aide de calculs de bases de Gröbner.

4.5.2.2 Comparaison sémantique

Notre approche partage avec celle de Sankaranarayanan *et al.* le fait d'imposer une condition sur le caractère inductif des invariants générés. En effet, l'hypothèse d'inductivité du théorème 4.6, écrite sous la forme $[\![c]\!]^\sharp <g> = <g>$ correspond à la condition de consécution de l'approche proposée par Sankaranarayanan *et al.*. Leur approche requiert comme la nôtre l'utilisation de polynômes paramétrés.

Outre le fait que notre analyse est une méthode de propagation arrière alors que la leur est une analyse avant, une autre différence importante est l'utilisation, dans leur analyse, de calculs de bases de Gröbner.

4.5.2.3 Comparaison des implémentations

À notre connaissance, ce travail n'a pas été implémenté. Cependant, notre analyse n'effectuant pas de calcul de bases de Gröbner (ni même de calculs approchés de bases de Gröbner comme proposé par Petter & Seidl [PS05]) nous pouvons raisonnablement penser qu'elle s'exécute plus rapidement.

4.5.3 Comparaison avec le travail de Rodríguez-Carbonell et Kapur sur les boucles simples

Rodríguez-Carbonell et Kapur ont développé deux approches différentes pour générer automatiquement des invariants polynomiaux sous forme d'égalités polynomiales. Nous commençons par inspecter les différences entre notre méthode et leur méthode complète qui permet de générer des invariants pour une classe réduite de programmes [RCK04b, RCK04c, RCK07b].

4.5.3.1 Description de la méthode

Rodríguez-Carbonell et Kapur [RCK04b, RCK04c, RCK07b] se sont intéressés à la génération d'invariants polynomiaux dans le cas de programmes très simples qui consistent en une boucle simple dont la garde est ignorée. D'autre part, les tests à l'intérieur de la boucle sont traités de manière non-déterministe. Les programmes analysés par cette méthode sont représentés en figure 4.11. Les

```
while ? do
  x := f_1(x) ;
  or
  ...
  or
  x := f_n(x) ;
end while,
```

FIGURE 4.11 – Une boucle simple.

auteurs montrent que l'ensemble des invariants polynomiaux possède une structure d'idéal. Leur but est de calculer cet idéal, noté I_∞. Étant donné un élément x_0 qui satisfait les conditions initiales du programme, un polynôme p est dit

invariant après une itération de la boucle si $p(x_0) = 0$ (p satisfait les conditions initiales) et $p(f_i(x_0)) = 0$ pour toutes les fonctions d'affectation f_i du programme. Les conditions initiales sont énoncées comme conjonction d'égalités polynomiales ($q_0(x) = 0 \ldots q_t(x) = 0$) et peuvent donc être représentées par l'idéal $I_0 =< q_0, \ldots, q_t >$. Le caractère d'invariance après une itération de la boucle peut s'exprimer sous la forme de l'intersection $I_0 \cap (\cap_{i=1}^{n} f_i^{-1}(I_0))$, que l'on peut réécrire en $\cap_{s=0}^{1} \cap_{i=1}^{n} f_i^{-s}(I_0)$. Un polynôme p est invariant pour la boucle s'il est invariant à chaque itération de la boucle. Les auteurs doivent donc vérifier qu'après r itérations, $p(f_{i_r} \circ \cdots \circ f_{i_1}(x_0)) = 0$, où $i_k \in [\![1, n]\!]$ représente le choix non-déterministe réalisé à l'étape k. L'ensemble des invariants polynomiaux de la boucle est donc représenté par l'intersection infinie d'idéaux $I_\infty = \cap_{s \in \mathbb{N}} \cap_{i-1}^{n} (f_i^{-s}(I_0))$. Se pose alors le problème du calcul d'une telle intersection. Les auteurs démontrent qu'en restreignant l'ensemble des fonctions d'affectation aux *fonctions résolvables*, cette intersection devient calculable. Plus précisément, les fonctions résolvables peuvent être vues comme une extension de l'ensemble des fonctions affines et sont obtenues par combinaison de fonctions linéaires. Une remarque importante est que ces fonctions sont inversibles. Les auteurs démontrent que I_∞ peut être calculé en un nombre fini d'itérations (en $m + 1$ itérations où m est le nombre de variables du programme) si toutes les affectations f_i sont des fonctions résolvables. D'autre part, dans le but d'analyser des programmes possédant un grand nombre de variables, mais avec peu d'affectations, les auteurs démontrent que l'analyse requiert au plus $n + 1$ itérations (où n est le nombre d'affectations de la boucle) dans le cas où les affectations f_i commutent.

4.5.3.2 Comparaison sémantique

Les différences entre cette procédure [RCK07b] et l'analyse `fastind` sont nombreuses. Tout d'abord, notons que notre analyse peut analyser des programmes possédant plusieurs niveaux de boucles. D'autre part, notons que notre analyse ne requiert aucune hypothèse sur les fonctions d'affectations. Notons aussi que les gardes des structures conditionnelles sont prises en compte dans notre analyse. Ainsi, un exemple tel que `mannadiv` qui requiert la prise en compte des gardes, ne peut être traité par cette première approche de Rodríguez-Carbonell et Kapur. Notons encore une fois que notre analyse est une procédure de propagation arrière alors que leur analyse se déroule en avant. Ceci n'est pas anodin. En effet, le fait de travailler en arrière implique que l'abstraction des affectations est très simple à calculer. À l'opposé, le fait de procéder en avant implique de faire un calcul d'inverse des fonctions d'affectation, ce qui est résolu par les auteurs en restreignant les affectations à l'ensemble des fonctions résolvables. Notons aussi que leur analyse est un calcul itératif, contrairement à la nôtre. Enfin, il est important de souligner que leur procédure utilise des mécanismes de bases de Gröbner. Ce

point est important pour la comparaison des implémentations.

4.5.3.3 Comparaison des implémentations

La figure 4.12 permet de comparer les temps d'exécution de notre analyse avec l'analyse de Rodríguez-Carbonell et Kapur [RCK04b, RCK04c, RCK07b] sur les boucles simples. Notons que leur implémentation a été réalisée à l'aide du logiciel Maple, avec une machine dotée d'un processeur Pentium 4 cadencé à 3.4 GHz et munie de 2 Gb de RAM. L'analyse `fastind` se compare très favorablement

Name	d	**Var**	**RCK** (boucles simples [RCK07b])	`fastind`	**Ratio**
`dijkstra`	2	5	1.5	0.043	35
`divbin`	2	5	2.1	0.005	420
`freire1`	2	3	0.7	0.006	117
`freire2`	2	4	0.7	0.007	100
`cohencu`	2	4	0.7	0.009	78
`fermat`	2	5	0.8	0.006	133
`wensley`	2	5	1.1	0.037	30
`euclidex`	2	8	1.4	0.008	175
`lcm`	2	6	1.0	0.006	167
`prod4`	3	6	2.1	0.013	162
`knuth`	3	9	55.4	0.084	660
`petter1`	2	2	1.0	0.003	333
`petter2`	3	2	1.1	0.003	367
`petter3`	4	2	1.3	0.004	325
`petter4`	5	2	1.3	0.004	325
`petter5`	6	2	1.4	0.006	233

FIGURE 4.12 – Comparaison de l'analyse `fastind` avec l'approche **RCK** sur les boucles simples

avec l'analyse **RCK** sur les boucles simples puisqu'elle s'exécute plus rapidement de plus de deux ordres de magnitude. Au vu des temps d'exécution sur les programmes `Petter`i signalons tout de même que la méthode **RCK** semble posséder de bonnes propriétés de passage à l'échelle vis à vis du degré.

4.5.4 Comparaison avec la seconde approche Rodríguez-Carbonell et Kapur

Parallèlement à l'approche sur boucles simples, Rodríguez-Carbonell et Kapur ont développé une méthode qui permet d'analyser un plus grand nombre de programmmes [RCK04a, RCK07a]. En effet, ils se débarrassent dans cette approche des restrictions réalisées dans l'approche précédente. De ce fait, cette approche n'est plus complète, contrairement à la précédente.

4.5.4.1 Description de la méthode

Dans cette approche, les auteurs considèrent des programmes représentés par des graphes de flôt finis avec affectations, de fonctions de tests, de point d'entrée et de sortie, et de points de jonction. Les affectations considérées sont polynomiales et les tests sont des égalités ou des déséglités polynomiales. Le but de cette méthode est de calculer, à chaque point de programme l, l'idéal généré par l'ensemble des polynômes invariants au point l. Plus précisément, à chaque point de programme l, les auteurs calculent un ensemble de polynômes qui s'annulent pour tous les états accessibles au point l. Dans ce but, les auteurs associent à leurs programmes une sémantique abstraite donnée en termes d'idéaux. Par exemple, dans le cas d'une affectation inversible $x_i := f(x)$, leur méthode consiste à calculer $I' = f^{-1}(I)$, si I représente l'idéal calculé au point de programme précédent. Dans le cas des affectations non-inversibles, une variable fraîche x'_i est introduite et entre dans le calcul de I'. Cette variable est ensuite éliminée en calculant l'intersection de I' avec $\mathbb{R}[\mathtt{x_1}, \ldots, \mathtt{x_m}]$. Dans le cas des tests polynomiaux égalitaires ($q = 0$), si l'idéal d'entrée vaut $I =< p_1, \ldots, p_s >$, leur méthode consiste à calculer l'idéal des polynômes qui s'annulent sur l'ensemble des zéros de q et sur l'ensemble des zéros de p_i. Cet idéal correspond à $\mathtt{IV}(< p_1, \ldots, p_s, q >)$. Dans le cas d'une déséglité polynomiale $q \neq 0$, l'idéal $I :< q >$ est calculé. Il représente l'idéal des polynômes qui s'annulent sur l'ensemble $\gamma(I)$ mais qui ne s'annulent pas sur $\gamma(q)$. Les nœuds de jonction sont abstraits par l'intersection des idéaux d'entrée $\cap_{i=1}^{l} I_i$. En fait, on réalise ainsi l'intersection des ensembles de polynômes qui s'annulent sur chaque $\gamma(I_i)$. Les nœuds de jonction de boucle nécessitent l'utilisation d'un opérateur d'élargissement ∇_d afin de rendre l'analyse terminante. Cet opérateur ∇_d consiste à ne garder que les polynômes de degré inférieur ou égal à d lors du calcul itératif. Cette méthode, prouvée correcte assure la terminaison. L'utilisation de cet opérateur implique le caractère incomplet de la méthode, mais il est suffisament performant pour que des invariants intéressants soient générés. Enfin, les auteurs montrent que leur méthode est complète dans le cas où les conditions des nœuds de test sont ignorées et si les affectations sont linéaires.

4.5.4.2 Comparaison sémantique

Notons tout d'abord que l'analyse **RCK** n'est pas limitée au calcul d'invariants inductifs. De ce fait, elle permet de générer des invariants pour un ensemble plus large de programmes. Passons maintenant à des observations plus techniques. Remarquons que cette analyse par interprétation abstraite ne se déroule pas dans le même domaine abstrait qe celui que l'on considère. Nous considèrons le treillis des idéaux alors que la méthode **RCK** considère le treillis des idéaux de variétés, *c.à.d.* des idéaux tels que $\alpha \circ \gamma(I) = I$ (égalité nommée « propriété IV »). La fonction de transfert pour les déségalités revient à calculer des quotients d'idéaux de ce domaine. La propriété IV nous semble difficile à maintenir, puisqu'elle est reliée au calcul de radicaux. Notons que, par défaut, leur implémentation omet ces calculs et ignore les gardes déségalitaires. Du fait du caractère avant de leur méthode, les affectations non-inversibles abstraites nécessitent un calcul d'intersection d'idéaux. Les tests égalitaires, plus simples dans ce type d'approche, nécessitent tout de même des calculs d'IV du fait de la nature du domaine abstrait. Notons que ces calculs sont la plupart du temps omis en pratique. Enfin, nous avons vu que leur approche nécessite l'utilisation d'un opérateur d'élargissement pour terminer, ce qui n'est pas le cas de notre analyse.

4.5.4.3 Comparaison des implémentations

Nous avons comparé les temps d'exécution de notre analyse avec ceux fournis par Rodríguez-Carbonell et Kapur [RCK04a, RCK07a] sur les exemples inductifs qu'ils décrivent. La figure 4.13 liste les résultats obtenus. Notons que leur implémentation a été réalisée à l'aide du logiciel Macaulay 2 et qu'ils ont utilisé une machine avec processeur Pentium 4 cadencé à 3.4 GHz et doté d'1 Gb de mémoire. Notre méthode est plus rapide de plus de 2.2 ordres de magnitude. Certains exemples méritent de plus amples commentaires : nous avons ajouté `petter30` qui produit un invariant de degré 31 et exécuté notre analyse sur les exemples `cohencu` et `freire2` avec un ensemble de paramètres différents afin de trouver un invariant de plus haut degré (resp. $\mathtt{x} = \mathtt{n}^3$, et $4\mathtt{r}^3 - 6\mathtt{r}^2 + 3\mathtt{r} + 4\mathtt{x} - 4\mathtt{a} = 1$).

4.5.5 Intégration de notre analyse dans l'analyseur statique `sawja`

L'analyse `fastind` a été intégrée à Sawja, un outil d'analyse statique permettant d'analyser du bytecode Java, développé au sein de l'équipe Celtique. Le développement de cette analyse en OCaml est dû au travail de Florent Kirchner et de Yannick Zakowski.

Name	d	**Var**	**RCK** (incomplète [RCK07a])	`fastind`	**Ratio**
`dijkstra`	2	5	1.31	0.043	30
`divbin`	2	5	0.99	0.005	198
`freire1`	2	3	0.38	0.006	63
`freire2`	2	4	0.85	0.007	121
`cohencu`	2	4	0.94	0.009	104
`fermat`	2	5	0.92	0.006	153
`wensley`	2	5	0.99	0.037	27
`euclidex`	2	8	1.95	0.008	244
`lcm`	2	6	1.22	0.006	203
`prod4`	3	6	4.63	0.013	356
`knuth`	3	9	2.61	0.084	31
`petter1`	2	2	0.5	0.003	167
`petter2`	3	2	0.8	0.003	267
`petter3`	4	2	4.2	0.004	1050
`petter4`	5	2	> 300	0.004	$> 75.10^3$
`petter5`	6	2	> 300	0.006	$> 50.10^3$
`sqrt`	2	3	0.46	0.010	46
`mannadiv`	2	5	1.12	0.005	224

FIGURE 4.13 – Comparaison `fastind` avec **RCK**-incomplète

Chapitre 5

Conclusion

5.1 Bilan sur nos travaux

Dans cette section, nous présentons un bilan des travaux décrits dans ce manuscrit, aussi bien dans le domaine des analyses quantitatives que dans le domaine de la génération d'invariants.

5.1.1 Dans le domaine des analyses quantitatives

Rappel du contexte
Nous avons décrit, dans le Chapitre 2, la théorie de l'interprétation abstraite [CC77]. Cette théorie propose un cadre d'approximation basé sur la notion de treillis, de connexion de Galois et de calculs de points fixes par itération. Ce cadre permet de définir la qualité des approximations effectuées. Il permet ainsi de définir la notion de meilleure approximation. À l'opposé, les notions quantitatives n'apparaissent pas naturellement dans ce cadre. Nous nous sommes donc posés la question de l'inférence de propriétés quantitatives par analyse statique.

Retour sur le travail d'analyse de programmes à coûts
Dans le Chapitre 3, nous avons présenté une analyse dont le but est de surapproximer le coût *long-run* d'un programme, qui caractérise le comportement asymptotique moyen du programme. Dans ce but, nous avons considéré des programmes représentés par des systèmes de transitions étiquetés par des coûts. L'ensemble de ces coûts Q est supposé muni d'un opérateur $\otimes$ permettant d'accumuler les coûts le long d'un chemin ainsi que d'un opérateur $\oplus$ permettant de combiner les coûts de différents chemins. Ces deux opérateurs définissent une structure de dioïde. De cette structure, nous déduisons un moduloïde grâce auquel nous établissons un modèle linéaire. La sémantique des programmes à coûts s'exprime, dans ce modèle, sous forme matricielle. En plus de ces deux opérateurs, nous avons souhaité munir Q d'un opérateur appelé racine `n-ieme`. Cet opéra-

teur sert à calculer le coût moyen d'un chemin et est nécessaire à la définition du coût *long-run*. Nous avons résumé l'utilisation de ces trois opérateurs dans une structure que nous avons nommée *dioïde de coûts*. Le coût *long-run* étant généralement non calculable, nous avons défini un cadre d'approximation à l'aide du modèle linéaire précédemment évoqué. On a ainsi montré comment calculer une sur-approximation du coût *long-run*. Enfin, nous avons étudié la possibilité d'intégrer les abstractions classiques de l'interprétation abstraite à ce cadre d'approximation nouvellement défini.

Contributions de l'auteur

Le travail décrit dans le Chapitre 3 a été initié lors de la thèse de Pascal Sotin [Sot05]. L'idée originelle [SCJ06] ainsi qu'une partie de ce chapitre doit donc être mis à l'actif de Pascal Sotin et de ses encadrants David Cachera et Thomas Jensen et non à celui de l'auteur de ce manuscrit. De ce fait, il convient de préciser quelles sont les contributions directes de l'auteur qui s'est investi plus particulièrement dans deux parties.

La première contribution concerne la notion de dioïde de coûts. Cette structure, obtenue par conjonction des trois opérateurs $\otimes$, $\oplus$ et de l'opérateur racine `n-ieme` était, à l'origine, assez restrictive. J'ai donc fourni un investissement important pour élargir ce cadre. Il s'agissait alors de déterminer les propriétés minimales nécessaires à la définition de dioïdes de coûts. Ce travail s'est poursuivi par une réflexion visant à déterminer les catégories de dioïdes qui sont des dioïdes de coûts. Elle a abouti à la présentation de trois catégories différentes, basées sur les notions de sélectivité, double-idempotence et d'intégrité. Alors qu'il est assez simple de montrer que les dioïdes des deux premières catégories sont des dioïdes de coûts, je me suis encore une fois interrogé sur les propriétés minimales qui garantissent à un dioïde intègre d'être un dioïde de coût. Le résultat de ces travaux a été présenté en section 3.2.

La deuxième contribution de l'auteur concerne le cadre d'approximation défini dans la section 3.4. Rappelons brièvement son fonctionnement. Son point de départ est une fonction $\alpha : \Sigma \to \Sigma^\sharp$ qui à tout état concret $\sigma \in \Sigma$ associe son abstraction $\sigma^\sharp \in \Sigma^\sharp$. Nous ne supposons **aucune** propriété sur cette fonction initiale. Cette fonction α est alors relevée en un opérateur matriciel $\alpha^\uparrow$. Les propriétés de notre modèle linéaire permettent de démontrer que cette fonction admet une fonction résiduée $\gamma^\uparrow$, et qu'ainsi le couple $(\alpha^\uparrow, \gamma^\uparrow)$ forme une connexion de Galois. C'est cette connexion qui nous permet de définir les calculs approchés du coût *long-run*. Dans notre méthode, il n'est donc pas requis que la fonction initiale α soit issue d'une connexion de Galois. Il semble cependant inopportun de se priver de telles fonctions qui sont classiques en interprétation abstraite. J'ai donc étudié comment de telles fonctions α s'intègrent dans notre cadre. L'opérateur $\uparrow$ n'est pas adapté au relèvement de ces fonctions. Si on l'utilise dans ce cas, les ordres définis sur Σ et $\Sigma^\sharp$ sont « oubliés » et « écrasés » par des nouveaux ordres sur $\Sigma^\uparrow$

et $(\Sigma^\sharp)^\uparrow$. Le problème consiste alors à trouver un relèvement tel que les ordres créés par ce relèvement soient compatibles avec les ordres initiaux. La section 3.5 défini le relèvement semi-parfait, qui répond à cette problématique. Elle est à mettre à l'actif de l'auteur.

Notons que la présentation des résultats du Chapitre 3 a été retravaillée et diffère de celle adoptée par Pascal Sotin dans son manuscrit de thèse [Sot05]. Certaines démonstrations de ce chapitre ont notamment été corrigées.

Enfin, il convient de souligner que ce travail a donné lieu à la publication d'articles [CJJS08, CJ10] ainsi qu'à une version revue [CJJS10].

5.1.2 Dans le domaine de la génération d'invariants

Rappel du contexte

L'inférence d'invariants de programmes est un problème essentiel dans le domaine de la vérification de programmes. Ce problème a été largement étudié, depuis les premières analyses basées sur des propriétés d'algèbre linéaire et permettant d'inférer des invariants linéaires en les variables du programme [Kar76, CH78] jusqu'à des analyses plus récentes basées sur des résultats mathématiques plus sophistiqués et permettant d'obtenir des invariants de programmes sous forme d'égalités polynomiales [MOS04, RCK07b, RCK07a, SSM04b]. Plus précisément, ces dernières analyses sont réalisées grâce au domaine des idéaux de polynômes et utilisent toutes des calculs de bases de Gröbner. Ce procédé est connu pour posséder une complexité doublement exponentielle [1] dans le pire cas [MM82]. Nous nous sommes donc interrogés sur la possibilité de proposer une analyse s'affranchissant de ces calculs complexes de bases de Gröbner.

Contributions de l'auteur

Nous avons défini, dans le Chapitre 4, l'analyse `fastind`, permettant de calculer efficacement des invariants polynomiaux de programmes. Cette analyse s'est inspirée des meilleures idées d'analyses déjà existantes [MOS04, RCK07b, RCK07a, SSM04b]. Plus précisément, nous avons pris comme point de départ le travail réalisé par Müller-Olm et Seidl [MOS04] qui ont développé une analyse arrière par interprétation abstraite. Le fait que cette analyse se déroule en arrière est important car cela permet une grande simplicité des calculs de fonctions abstraites. Par exemple, une affectation abstraite se calcule simplement en effectuant une substitution. En comparaison, dans l'analyse avant développée par Rodríguez-Carbonell et Kapur [RCK07a], seules les affectations *inversibles* sont traitées de

1. Plus précisément, on peut exhiber des idéaux dont toute base de Gröbner possède $2^{2^{O(n)}}$ éléments où n est le nombre de variables des polynômes. Cependant ce type d'exemples doit être vu comme un cas pathologique, qui se rencontre rarement en pratique. Pour des idéaux plus classiques, les algorithmes les plus performants actuellement annoncent une complexité simplement exponentielle [Fau02].

manière aussi simple. Dans les autres cas, la fonction abstraite associée nécessite l'introduction d'une variable fraîche puis son élimination à l'aide de mécanismes de calculs de bases de Gröbner. Toutefois, malgré l'élégante simplicité de la proposition de Müller-Olm et Seidl, l'analyse de Rodríguez-Carbonell et Kapur possède un temps d'exécution bien plus performant. Cette différence importante provient des méthodes proposées pour effectuer les calculs de points fixes. Les deux méthodes utilisent un critère d'arrêt par calcul de bases de Gröbner. Cependant la méthode itérative développée par Rodríguez-Carbonell et Kapur utilise un opérateur d'élargissement qui permet d'accélérer la convergence de la méthode. Ajoutons que malgré l'approximation introduite par cet opérateur d'élargissement, les auteurs ont montré qu'ils étaient capables de traiter une grande diversité d'exemples. Notre problématique a alors été de proposer une analyse dont les fonctions abstraites admettent un calcul aussi simple que celles du travail de Müller-Olm et Seidl et dont les calculs itératifs peuvent être calculés aussi efficacement que dans l'analyse proposée par Rodríguez-Carbonell et Kapur. L'analyse `fastind` que nous avons développée satisfait ces exigences. Elle se base sur l'analyse arrière de Müller-Olm et Seidl. Afin d'alléger les calculs itératifs, nous avons choisi de faire ce que nous avons appelé l'hypothèse d'inductivité. Il s'agit en fait de restreindre notre analyse à la recherche d'invariants de boucles c'est à dire des invariants qui restent valides à chaque itération de boucle. Ce point de vue naturel a notamment été adopté dans l'analyse avant développée par Sankaranarayanan *et al.* [SSM04b]. Cette hypothèse d'inductivité s'exprime, pour chaque boucle du programme, sous forme d'une contrainte d'égalité d'idéaux. Nous avons montré que, sous cette hypothèse, la procédure itérative permettant le calcul de points fixes, termine dès le premier pas de calcul. Précisons d'autre part que les contraintes générées par notre méthode s'expriment généralement sous forme de systèmes d'équations linéaires et admettent donc une résolution aisée. Il en résulte une analyse efficace que j'ai implémentée à l'aide du logiciel Maple. Une implémentation plus générale a aussi été réalisée en Ocaml et intégrée dans l'outil d'analyse statique `Sawja`. Elle est à mettre au crédit de Florent Kirchner et de Yannick Zakowski. Enfin, nous avons prouvé la rapidité d'exécution de notre méthode en comparant très favorablement les temps d'exécution de l'implémentation Maple à ceux donnés dans les articles résumant les travaux de Müller-Olm et Seidl [MOS04] et Rodríguez-Carbonell et Kapur [RCK07b, RCK07a].

5.2 Perspectives

Avant de présenter des perspectives spécifiques aux deux travaux menés lors de cette thèse, revenons sur la structure de ce manuscrit. Il semble légitime de s'interroger sur l'unité de ce manuscrit qui traite dans les chapitres 3 et 4 de do-

maines très différents. Revenons donc sur l'origine de ce choix. Après avoir étudié la possibilité d'inférer statiquement des propriétés quantitatives de programme, l'auteur a souhaité étudier la possibilité d'inférer un coût particulier : la complexité du programme. Le travail de Gulwani *et al.* [GMC09] dans ce domaine a alors particulièrement retenu notre attention. Ce travail expose la possibilité d'inférer statiquement la complexité d'un programme par une méthodologie basée sur une instrumentation de code par compteurs multiples. Chaque compteur permet de mesurer des itérations de boucles. Un outil de génération d'invariants linéaires est alors utilisé afin d'obtenir une borne linéaire pour chacun de ces compteurs. En combinant les bornes linéaires obtenues pour chaque compteur, les auteurs montrent qu'il est possible d'encadrer la complexité du programme par des bornes non-linéaires. Cette approche est basée sur un outil de génération d'invariants linéaires. Nous nous sommes alors interrogés sur la possibilité d'inférer la complexité d'un programme au cas où nous disposerions d'un outil calculant des invariants non-linéaires de programme. Notre méthodologie devait s'articuler ainsi :

(1) instrumenter le code du programme à l'aide d'un compteur $\mathtt{c}$ visant à mesurer le nombre d'itérations de la boucle la plus externe du programme,

(2) obtenir, à l'aide d'un outil d'invariants non-linéaires, un invariant non linéaire du programme dépendant de $\mathtt{c}$ et des autres variables $\mathtt{x}_1, \ldots, \mathtt{x}_\mathtt{m}$ du programme. Cet invariant est généré sous la forme d'une égalité polynomiale $g = 0$ avec $g \in \mathbb{R}[\mathtt{x}_1, \ldots, \mathtt{x}_\mathtt{m}, \mathtt{c}]$,

(3) borner les racines du polynôme g vu comme élément de $(\mathbb{R}[\mathtt{x}_1, \ldots, \mathtt{x}_\mathtt{m}])[\mathtt{c}]$ à l'aide de résultats mathématiques.

Les bornes de l'étape **(3)** de cette méthodologie peuvent être obtenues à l'aide de résultats mathématiques comme les bornes de Cauchy [BPR03].

Cette méthodologie nous semble, avec le recul, extrêmement naïve. Tout d'abord, il faut noter que cette méthodologie suppose que l'on peut toujours obtenir un invariant non-linéaire utilisant la variable $\mathtt{c}$, ce qui n'est pas le cas. D'autre part, même si l'on parvenait à obtenir un tel invariant g, le calcul des bornes de Cauchy fournirait un résultat pauvre. En effet, ces bornes ont été définies afin de prouver la compacité de l'ensemble des racines d'un polynôme et sont donc, par nature, très imprécises. Cette idée a donc été peu à peu abandonnée. Il nous est cependant apparu lors de l'étude bibliographique que la génération d'invariants non-linéaires méritait une étude plus approfondie.

En résumé, l'angle « complexité » nous semble prometteur afin de relier directement les chapitres 3 et 4, même si la méthodologie initiale évoquée au-dessus nous semble actuellement non réaliste.

5.2.1 Dans le domaine des analyses quantitatives

Le travail présenté dans le chapitre 3 nous semble relativement mûr. Cependant, la question de savoir si l'on peut intégrer une structure probabiliste à l'intérieur de notre cadre est régulièrement posée concernant ce travail. Elle provient du fait que la communauté analyse quantitative considère, en grande partie, que les coûts portés par les programmes sont des probabilités. Cette question reste ouverte. En effet, les dioïdes de coûts sont, par nature, incompatibles avec les structures probabilistes. La piste de réflexion mènerait plutôt vers la définition d'un produit entre notre structure de dioïde de coûts et une structure probabiliste. Ce produit pourrait alors induire un cadre d'abstraction obtenu comme produit du cadre d'abstraction obtenu par la structure de dioïdes et un cadre d'abstractions issu de la structure probabiliste. Cette piste n'a pas encore été suivie et demande de plus amples réflexions.

5.2.2 Dans le domaine de la génération d'invariants

Concernant l'analyse `fastind`, nous pouvons présenter deux pistes de réflexion. La première concerne l'implémentation qui a été faite dans le cadre de l'outil d'analyse statique Sawja. Cette implémentation n'est pas complètement satisfaisante du fait des forts coûts d'exécution qu'elle induit. Ces coûts ne proviennent pas de la nature de notre analyse mais de l'implémentation de celle-ci. La génération des contraintes qui, pourtant, est une difficulté mineure de notre analyse, représente un temps de calcul non négligeable dans l'implémentation Sawja. Ceci provient en partie de la représentation choisie pour la définition de ces contraintes qui n'est pas adaptée à un transport aisé.

D'autre part, nous avons vu que les analyses arrières étaient particulièrement adaptées à la définition de certaines fonctions abstraites (affectations et tests déségalitaires). Les analyses avant, elles, semblent plus adaptées à la définition de l'abstraction des tests égalitaires. Une piste pourrait donc être d'essayer de définir une analyse qui combinerait ces deux approches, bénéficiant ainsi des points forts de chacune d'elle.

Il serait aussi intéressant de combiner notre analyse `fastind` à d'autres analyses. Nous pensons particulièrement à la possibilité de combiner les invariants obtenus par cette analyse avec des informations exprimées sous formes d'inégalités. Ceci permettrait d'obtenir des invariant polynomiaux sous forme d'inégalités.

Enfin, nous avons vu que l'analyse `fastind` se base sur une analyse par interprétation abstraite dans laquelle la procédure itérative de calcul de point fixe termine dès le premier pas de calcul. Ceci est dû à l'intégration d'un ensemble de contraintes. Une piste pourrait alors être d'étudier ce mécanisme mêlant interprétation abstraite et contraintes permettant le calcul direct des points fixes. Le but

étant alors de déterminer si ce mécanisme peut être adapté à d'autres analyses.

Annexe A

Chapitre 3 : Analyse des programmes à coûts

A.1 Section 3.4 : Abstraire un programme à coûts

Nous revenons dans cette annexe sur la preuve du lemme 3.11. Cette démonstration nécessite le résultat suivant (lemme 3.10), qui stipule que pour toute abstraction linéaire correcte, les matrices itérées $(M^n, (M^\sharp)^n)$ vérifient l'inégalité de correction.

Lemme A.1.
Soit $(T, T^\sharp, \alpha)$ une abstraction linéaire correcte. Alors :

$$\forall n \geq 1, \quad \alpha^\uparrow \circ M^n \leq (M^\sharp)^n \circ \alpha^\uparrow$$

Nous pouvons maintenant démontrer le lemme A.2.

Lemme A.2.
Soit $(T, T^\sharp, \alpha)$ une abstraction linéaire correcte. On a alors :

$$\forall \sigma_0, \sigma_1 \in \Sigma,\ \forall \sigma_0^\sharp, \sigma_1^\sharp \in \Sigma^\sharp, \quad \left.\begin{array}{rcl} \alpha(\sigma_0) & = & \sigma_0^\sharp \\ \alpha(\sigma_1) & = & \sigma_1^\sharp \end{array}\right\} \Rightarrow M_{\sigma_0,\sigma_1} \leq (M^\sharp)_{\sigma_0^\sharp,\sigma_1^\sharp} \tag{A.1}$$

D'autre part, on a :

$$\forall n \geq 1, \quad \bigoplus_{\substack{\sigma_0 \in \Sigma \\ \sigma_1 \in \Sigma}} M^n_{\sigma_0,\sigma_1} \leq \bigoplus_{\substack{\sigma_0^\sharp \in \Sigma^\sharp \\ \sigma_1^\sharp \in \Sigma^\sharp}} (M^\sharp)^n_{\sigma_0^\sharp,\sigma_1^\sharp} \tag{A.2}$$

Ainsi que :

$$\forall n \geq 1, \quad \bigoplus_{\sigma \in \Sigma} M^n_{\sigma,\sigma} \leq \bigoplus_{\sigma^\sharp \in \Sigma^\sharp} (M^\sharp)^n_{\sigma^\sharp,\sigma^\sharp} \tag{A.3}$$

Démonstration.
Soient $\sigma, \sigma_0 \in \Sigma$ et $\sigma_0^\sharp, \sigma_1^\sharp \in \Sigma^\sharp$ tels que $\alpha(\sigma_0) = \sigma_0^\sharp$ et $\alpha(\sigma_1) = \sigma_1^\sharp$. Comme $(T, T^\sharp, \alpha)$ est une abstraction linéaire correcte, on a, d'après le lemme A.1, $\alpha \circ M \leq M^\sharp \circ \alpha$. En particulier, on a : $(\alpha \circ M)_{\sigma_0^\sharp,\sigma_1} \leq (M^\sharp \circ \alpha)_{\sigma_0^\sharp,\sigma_1}$ ce que l'on peut réécrire sous la forme :

$$\bigoplus_{\sigma_i \in \Sigma} (\alpha_{\sigma_0^\sharp,\sigma_i} \otimes M_{\sigma_i,\sigma_1}) \leq \bigoplus_{\sigma_i^\sharp \in \Sigma^\sharp} (M^\sharp_{\sigma_0^\sharp,\sigma_i^\sharp} \otimes \alpha_{\sigma_i^\sharp,\sigma_1})$$

En décomposant ces deux sommes, on obtient :

$$\bigoplus_{\sigma_i \in \alpha^{-1}(\{\sigma_0^\sharp\})} (\alpha_{\sigma_0^\sharp,\sigma_i} \otimes M_{\sigma_i,\sigma_1}) \oplus \bigoplus_{\sigma_i \notin \alpha^{-1}(\{\sigma_0^\sharp\})} (\alpha_{\sigma_0^\sharp,\sigma_i} \otimes M_{\sigma_i,\sigma_1})$$
$$\leq (M^\sharp_{\sigma_0^\sharp,\sigma_1^\sharp} \otimes \alpha_{\sigma_1^\sharp,\sigma_1}) \oplus \bigoplus_{\sigma_i^\sharp \neq \sigma_1^\sharp} (M^\sharp_{\sigma_0^\sharp,\sigma_i^\sharp} \otimes \alpha_{\sigma_i^\sharp,\sigma_1})$$

Si $\sigma_i \notin \alpha^{-1}(\{\sigma_0^\sharp\})$ est vérifié, alors on a $\alpha(\sigma_i) \neq \sigma_0^\sharp$. De ce fait, $\alpha_{\sigma_0^\sharp,\sigma_i} = \bot$. D'autre part, comme on a supposé $\alpha(\sigma_0) = \sigma_0^\sharp$, on a évidemment $\alpha(\sigma_0) \neq \sigma_i^\sharp$ pour tout $\sigma_i^\sharp \neq \sigma_0^\sharp$. Dans ce cas, $\alpha_{\sigma_i^\sharp,\sigma_0} = \bot$. Ainsi, l'inégalité précédente se simplifie en $\bigoplus_{\sigma_i \in \alpha^{-1}(\{\sigma_0^\sharp\})} M_{\sigma_i,\sigma_1} \leq M^\sharp_{\sigma_0^\sharp,\sigma_1^\sharp}$. Comme σ_0 appartient à $\alpha^{-1}(\{\sigma_0^\sharp\})$, l'inégalité A.1 est vérifiée :

$$M_{\sigma_0,\sigma_1} \leq \bigoplus_{\sigma_i \in \alpha^{-1}(\{\sigma_0^\sharp\})} M_{\sigma_i,\sigma_1} \leq M^\sharp_{\sigma_0^\sharp,\sigma_1^\sharp}$$

Afin de démontrer l'inégalité suivante, on commence par le cas $n = 1$. L'inégalité souhaitée est obtenue en sommant chaque membre de l'inégalité A.1 sur l'ensemble $E = \{(\sigma_0, \sigma_1, \sigma_0^\sharp, \sigma_1^\sharp) \in \Sigma^2 \times (\Sigma^\sharp)^2 \mid \alpha(\sigma_0) = \sigma_0^\sharp,\ \alpha(\sigma_1) = \sigma_1^\sharp\}$:

$$\begin{array}{ccc}
\displaystyle\bigoplus_{(\sigma_0,\sigma_1,\sigma_0^\sharp,\sigma_1^\sharp) \in E} M_{\sigma_0,\sigma_1} & \leq & \displaystyle\bigoplus_{(\sigma_0,\sigma_1,\sigma_0^\sharp,\sigma_1^\sharp) \in E} M^\sharp_{\sigma_0^\sharp,\sigma_1^\sharp} \\
\| & & |\wedge \\
\displaystyle\bigoplus_{(\sigma_0,\sigma_1) \in \Sigma^2} M_{\sigma_0,\sigma_1} & & \displaystyle\bigoplus_{(\sigma_0^\sharp,\sigma_1^\sharp) \in (\Sigma^\sharp)^2} M^\sharp_{\sigma_0^\sharp,\sigma_1^\sharp}
\end{array}$$

L'égalité (verticale) des membres gauches est obtenue en remarquant que les ensembles E et $\Sigma \times \Sigma$ sont en bijection. L'inégalité (verticale) des membres droits est, quant à elle, obtenue par idempotence. Ceci conclut la démonstration pour le cas $n = 1$.

Pour démontrer l'inégalité pour les cas $n > 1$, il suffit de remarquer que la démonstration que l'on vient d'effectuer ne dépend que de propriétés de la fonction

α et reste donc valable pour tout couple $(M, M^\sharp)$ qui vérifie l'inégalité : $\alpha \circ M \leq M^\sharp \circ \alpha$. Or par le lemme A.1, on a, pour tout $n \geq 1$, $\alpha \circ M^n \leq (M^\sharp)^n \circ \alpha$. Finalement, en appliquant la démonstration précédente au couple $(M^n, (M^\sharp)^n)$, on obtient l'ensemble d'inégalités A.2.

Nous démontrons enfin l'ensemble des inégalités A.3. La démonstration est analogue à celle des inégalités A.2. Afin de montrer le cas $n = 1$, on considère maintenant l'inégalité A.1 avec $\sigma_0 = \sigma_1 = \sigma$ et $\alpha(\sigma) = \sigma^\sharp$:

$$M_{\sigma,\sigma} \leq M^\sharp_{\sigma^\sharp,\sigma^\sharp}$$

En sommant alors chaque membre sur l'ensemble $F = \{(\sigma, \sigma^\sharp) \in \Sigma \times \Sigma^\sharp \mid \alpha(\sigma) = \sigma^\sharp\}$, on obtient :

$$\bigoplus_{\sigma\in\Sigma} M_{\sigma,\sigma} = \bigoplus_{(\sigma,\sigma^\sharp)\in F} M_{\sigma,\sigma} \leq \bigoplus_{(\sigma,\sigma^\sharp)\in F} M^\sharp_{\sigma^\sharp,\sigma^\sharp} \leq \bigoplus_{\sigma^\sharp\in\Sigma^\sharp} M^\sharp_{\sigma^\sharp,\sigma^\sharp}$$

Le cas $n > 1$ est obtenu en appliquant la démonstration au couple $(M^n, (M^\sharp)^n)$. □

A.2 Section 3.6 : Implémentation de la méthode sur un exemple

Nous revenons ici sur la méthode de calcul du coût long-run. Pour illustrer ce procédé, nous considérons le programme présenté en Figure A.1 (écrit en ArraySimple). Ce programme décrit une méthode de relaxation (généralisant la méthode du pivot de Gauss) permettant de calculer une approximation de la solution exacte. Plus précisément, il permet de résoudre de manière itérative le système linéaire `mat` $*$ `x` $=$ `b`, où `mat` est une matrice $n \times n$ inversible, `b` est une donnée et `x` est la valeur à trouver.

Cinq modes d'énergies, de **A** à **E** sont définis, et influencent la consommation d'énergie des opérations de tableaux. Par exemple, la table correspondant à l'opération `copy` est la suivante.

A	5	0.2
C	2	$\lambda n.n$
D	1	$\lambda n.3. * n$
E	0	$\lambda n.5. * n$

Notre but est de calculer un coût long-run (énergie moyenne consommée par cycle) pour un programme dont le code contient des instructions de changements de modes dépendantes de la taille commune n des tableaux. Ces changements

```
var x,n,residue,b,i,iter,ITERMAX,mat,omega,tmp,eps,m,p :int,
dumb:int;
begin

assume length n <= 1000;
setsize x n;
setsize residue n;
setsize b n;
setsize tmp n;
apply copy_to residue,x;

__H1__

m = eps - 1;
iter = 0;

while iter < ITERMAX and m <= eps do
      i = 0;
      apply copy_to b, tmp;
      while i < n do
            __H1__
            apply prod residue, mat; // :-> p //
            dumb = dumb; // tmp[i] = tmp[i] - p;
      i = i+1; done;
      apply mult_scal tmp, omega;
      apply sum residue, tmp;

      apply sub x, residue;
      apply max x; // :-> m //
      apply copy_to residue, x;
iter = iter+1; done;

skip;
setmode A;

apply copy_to b, tmp;
i = 0;
while i < n do
      apply prod x, mat; // :-> p //
      dumb = dumb; // residue[i] = p;

i = i+1; done;

end
```

Mode		Seuil								
m_B	m_H	100	200	300	400	500	600	700	800	900
A	B	3.6	3.6	3.6	3.6	3.6	3.6	3.6	3.6	3.6
A	C	13.7	23.0	32.3	41.6	50.9	60.2	69.5	78.8	88.1
A	D	39.9	77.8	115.7	153.6	191.6	229.5	267.4	305.4	343.3
A	E	81.2	162.2	243.1	324.1	405.1	486.0	567.0	647.9	728.9
B	C	13.7	23.0	32.3	41.6	50.9	60.2	69.5	78.8	88.1
B	D	39.9	77.8	115.7	153.6	191.6	229.5	267.4	305.4	343.3
B	E	81.2	162.2	243.1	324.1	405.1	486.0	567.0	647.9	728.9
C	D	97.5	97.5	115.7	153.6	191.6	229.5	267.4	305.4	343.3
C	E	97.5	162.2	243.1	324.1	405.1	486.0	567.0	647.9	728.9
D	E	381.6	381.6	381.6	381.6	405.1	486.0	567.0	647.9	728.9

FIGURE A.2 – Coût long-run pour différentes valeurs du seuil, pour des valeurs m_H et m_B données.

		Bas			
	250	A	B	C	D
High	E	300	300	300	none
	D	600	600	600	
	C	900	900		
	B	900			

(a) coût long-run maximum=250

		Bas			
	50	A	B	C	D
High	E	none	none	none	none
	D	100	100	none	
	C	400	400		
	B	900			

(b) coût long-run maximum=50

FIGURE A.3 – Seuil : meilleurs choix.

sont codés comme suit : la chaîne de caractères `__H1__` du code initial est successivement remplacé par `assume n <= 1000; if n <` xx `then setmode` m_H ; `else setmode` m_B, où xx est un seuil constant appartenant à l'ensemble $\{100, 200, \ldots, 900\}$, et où m_H et m_B sont respectivement des modes d'énergie haut et bas.

Le coût long-run calculé sur ce programme est donné en Table A.2. Comme attendu, plus le niveau d'énergie est haut, plus ce coût est important. Supposons maintenant que l'on fixe un niveau de consommation d'énergie maximal. Ce niveau correspond à une consommation d'énergie que l'on ne souhaite pas dépasser ou que l'on ne peut pas dépasser du fait d'une énergie disponible limitée. Cette valeur étant fixée, il semble intéressant de calculer le couple (m_B, m_H) optimal, *c.à.d.* celui qui permet d'assurer la meilleure performance en temps d'exécution. La figure A.3 présente les couples optimaux pour une consommation d'énergie ne pouvant dépasser 250mWh et ceux pour une consommation maximale de 50mWh.

Annexe B

Chapitre 4 : Génération rapide d'invariants inductifs sous forme d'égalités polynomiales

B.1 Section 4.2 : Les programmes polynomiaux

Nous revenons ici sur la Section 4.2.2.3. Plus précisément, l'objet de cette appendice est la démonstration du théorème B.1 qui stipule l'équivalence de la sémantique $\mathrm{B}^{\mu}[\![.]\!]$ et de la clôture $\rightarrow^*$.

Théorème B.1 ($\mathrm{B}^{\mu}[\![.]\!]$ vs `SOPP`)**.**
Soit c un programme polynomial, S un sous-ensemble $\mathbb{R}^m$ et σ un élément de $\mathbb{R}^m$. On a :

$$\forall c \in \mathbb{P}, \forall \sigma \in B^{\mu}[\![c]\!]\ S \quad \Leftrightarrow \quad \exists \sigma_f \in S,\ \langle c, \sigma \rangle \rightarrow^* \langle \bullet, \sigma_f \rangle$$

Afin d'améliorer la lisibilité de la démonstration du théorème B.1, nous le découpons en deux lemmes qui énoncent chacun un des côtés de l'équivalence.

Lemme B.1.
Soit c un programme polynomial, S un sous-ensemble $\mathbb{R}^m$ et σ un élément de $\mathbb{R}^m$. On a :

$$\sigma \in B^{\mu}[\![c]\!]\ S \quad \Leftarrow \quad \exists \sigma_f \in S,\ \langle c, \sigma \rangle \rightarrow^* \langle \bullet, \sigma_f \rangle$$

Nous démontrons ce lemme par récurrence. Plus précisément, par récurrence forte sur la longueur k de la dérivation, nous démontrons la propriété suivante pour tout $k \in \mathbb{N}$:

$$\sigma \in \mathrm{B}^{\mu}[\![c]\!]\ S \quad \Leftarrow \quad \exists \sigma_f \in S,\ \langle c, \sigma \rangle \rightarrow^k \langle \bullet, \sigma_f \rangle$$

Démonstration.
Soit $c \in \mathbb{P}$, $S \subseteq \mathbb{R}^m$ et $\sigma \in \mathbb{R}^m$.

Initialisation

Supposons $k = 0$. Alors $\langle c, \sigma \rangle \rightarrow^0 \langle \bullet, \sigma_f \rangle$ implique que $c = \bullet$, $\sigma = \sigma_f$ et ainsi $\sigma \in S$. La propriété $\mathrm{B}^\mu[\![\bullet]\!]\ S = S$ permet de conclure ce cas.

Hérédité Supposons la propriété vérifiée pour $k \leq m$ et démontrons la pour $m + 1$. Supposons alors qu'il existe $\sigma_f \in S$ tel que $\langle c, \sigma \rangle \rightarrow^{m+1} \sigma_f$.

Si $c \equiv$ **skip** alors $m = 0$, $\langle c, \sigma \rangle \rightarrow^1 \langle \bullet, \sigma \rangle$. Donc $\sigma_f = \sigma$. Comme $\mathrm{B}^\mu[\![c]\!]\ S = S$, la propriété est trivialement vérifiée dans ce cas.

Si $c \equiv \mathtt{x_j} := p$ alors $m = 0$, $\langle c, \sigma \rangle \rightarrow^1 \langle \bullet, \sigma[v]_j \rangle$ pour $v = p(\sigma)$. Donc $\sigma_f = \sigma[v]_j$. Comme $\mathrm{B}^\mu[\![c]\!]\ S = \{\sigma \in \mathbb{R}^m \mid \sigma[p(\sigma)]_j \in S\}$, la propriété est trivialement vérifiée pour ce cas.

Si $c \equiv c_1; c_2$ alors $\langle c_1, \sigma \rangle \rightarrow^{k_1} \langle \bullet, \sigma' \rangle$ et $\langle c_2, \sigma' \rangle \rightarrow^{k_2} \langle \bullet, \sigma_f \rangle$ pour k_1 et k_2 tels que $1 \leq k_i \leq m$ et $k_1 + k_2 = m$. Par hypothèse de récurrence, $\sigma' \in \mathrm{B}^\mu[\![c_2]\!]\ S$ et $\sigma \in \mathrm{B}^\mu[\![c_1]\!]\ (\mathrm{B}^\mu[\![c_2]\!]\ S)$. Comme $\mathrm{B}^\mu[\![c]\!]\ S = \mathrm{B}^\mu[\![c_1]\!]\ (\mathrm{B}^\mu[\![c_2]\!]\ S)$, la propriété est trivialement vérifiée pour ce cas.

Si $c \equiv$ **if** $p \bowtie 0$ **then** c_1 **else** c_2

Rappelons tout d'abord que :

$$\mathrm{B}^\mu[\![c]\!]\ S = ([\![p \bowtie 0]\!] \bigcap \mathrm{B}^\mu[\![c_1]\!]\ S) \bigcup ([\![p \not\bowtie 0]\!] \bigcap \mathrm{B}^\mu[\![c_2]\!]\ S)$$

On raisonne alors par étude de cas, suivant que $p(\sigma) = 0$ ou $p(\sigma) \neq 0$.

Supposons $p(\sigma) \bowtie 0 \equiv \mathit{true}$ alors $\langle c, \sigma \rangle \rightarrow^1 \langle c_1, \sigma \rangle \rightarrow^m \langle \bullet, \sigma_f \rangle$. Par hypothèse de récurrence, on a $\sigma \in \mathrm{B}^\mu[\![c_1]\!]\ S$. Ainsi, $\sigma \in [\![p \bowtie 0]\!] \bigcap \mathrm{B}^\mu[\![c_1]\!]\ S$ et, par conséquence, $\sigma \in \mathrm{B}^\mu[\![c]\!]\ S$.

Supposons $p(\sigma) \bowtie 0 \equiv \mathit{false}$ alors $\langle c, \sigma \rangle \rightarrow^1 \langle c_2, \sigma \rangle \rightarrow^m \langle \bullet, \sigma_f \rangle$. Par hypothèse de récurrence, on a $\sigma \in \mathrm{B}^\mu[\![c_2]\!]\ S$. Ainsi, $\sigma \in [\![p \not\bowtie 0]\!] \bigcap \mathrm{B}^\mu[\![c_2]\!]\ S$ et, par conséquence, $\sigma \in \mathrm{B}^\mu[\![c]\!]\ S$.

Ainsi, la propriété est vérifiée dans ce cas.

Si $c \equiv$ **while** $p \bowtie 0$ **do** c_1

Supposons $p(\sigma) \bowtie 0 \equiv true$ alors $\langle c, \sigma\rangle \to^1 \langle c_1; c, \sigma\rangle \to^m \langle \bullet, \sigma_f\rangle$. Par hypothèse de récurrence, on a $\sigma \in \mathrm{B}^\mu[\![c_1; c]\!]\ S = \mathrm{B}^\mu[\![c_1]\!]\ (\mathrm{B}^\mu[\![c]\!]\ S)$. De plus, par la définition de la sémantique arrière de l'instruction **while** sous forme de point fixe, on a :

$$\begin{array}{rcl} \mathrm{B}^\mu[\![c]\!]\ S & = & F_{c,p,S}(\mathrm{B}^\mu[\![c]\!]\ S) \\ & = & ([\![p \not\bowtie 0]\!] \bigcap S) \bigcup ([\![p \bowtie 0]\!] \bigcap \mathrm{B}^\mu[\![c_1]\!]\ (\mathrm{B}^\mu[\![c]\!]\ S)) \end{array}$$

Comme $\sigma \in [\![p \bowtie 0]\!] \bigcap \mathrm{B}^\mu[\![c_1]\!]\ (\mathrm{B}^\mu[\![c]\!]\ S)$, on a $\sigma \in \mathrm{B}^\mu[\![c]\!]\ S$.

Supposons $p(\sigma) \bowtie 0 \equiv false$ alors $\langle c, \sigma\rangle \to^1 \langle \mathbf{skip}, \sigma\rangle \to^m \langle \bullet, \sigma_f\rangle$, $m = 1$ et $\sigma_f = \sigma$. Par hypothèse de récurrence, on obtient $\sigma \in \mathrm{B}^\mu[\![\mathbf{skip}]\!]\ S = S$. De plus, on a :

$$\begin{array}{rcl} \mathrm{B}^\mu[\![c]\!]\ S & = & \mu F_{c_1,p,S} \\ & = & \mu\lambda X.([\![p \not\bowtie 0]\!] \bigcap S) \bigcup ([\![p \bowtie 0]\!] \bigcap \mathrm{B}^\mu[\![c_1]\!]\ X) \\ & = & ([\![p \not\bowtie 0]\!] \bigcap S) \bigcup (\mu\lambda X.[\![p \bowtie 0]\!] \bigcap \mathrm{B}^\mu[\![c_1]\!]\ X) \end{array}$$

Comme $\sigma \in [\![p \not\bowtie 0]\!] \bigcap S$, on a $\sigma \in \mathrm{B}^\mu[\![c]\!]\ S$.

La propriété est vérifiée pour ce dernier cas, ce qui prouve qu'elle est toujours satisfaite.

□

Nous pouvons maintenant démontrer l'autre sens de l'équivalence.

Lemme B.2.
Soit c un programme polynomial, S un sous-ensemble $\mathbb{R}^m$ et σ un élément de $\mathbb{R}^m$. On a :

$$\sigma \in B^\mu[\![c]\!]\ S \quad \Rightarrow \quad \exists \sigma_f \in S,\ \langle c, \sigma\rangle \to^* \langle \bullet, \sigma_f\rangle$$

Démonstration.
Par induction sur la syntaxe des programmes polynomiaux.
Soit $c \in \mathbb{P}$, $S \subseteq \mathbb{R}^m$ et $\sigma \in \mathbb{R}^m$.

Si $c \equiv$ **skip** alors on a $\mathrm{B}^\mu[\![c]\!]\ S = S$, et pour tout $\sigma \in \mathbb{R}^m$,

$$\frac{}{\langle \mathbf{skip}, \sigma\rangle \to \langle \bullet, \sigma\rangle}\ \textbf{\textit{skip}}$$

Ainsi, la propriété est vérifiée en prenant $\sigma_f = \sigma$.

Si $c \equiv \mathtt{x_j} := p$ alors on a $\mathrm{B}^{\mu}[\![c]\!]\ S = \{x \in \mathbb{R}^m \mid x[\![p(x)]\!]_{\mathtt{j}} \in S\}$, et pour tout $\sigma \in \mathbb{R}^m$,

$$\frac{p(\sigma) = v}{\langle \mathtt{x_j} := p, \sigma\rangle \to \langle \bullet, \sigma[\![v]\!]_{\mathtt{j}}\rangle}\ \textbf{\textit{assign}} \quad \textit{où} \quad \sigma[\![v]\!]_{\mathtt{j}} = (\sigma_1, \ldots, \sigma_{j-1}, v, \sigma_{j+1}, \ldots, \sigma_m)$$

Ainsi, la propriété est vérifiée en prenant $\sigma_f = \sigma[\![v]\!]_{\mathtt{j}}$.

Si $c \equiv$ **if** $p \bowtie 0$ **then** c_1 **else** c_2 alors on a :

$$\mathrm{B}^{\mu}[\![c]\!]\ S = (\mathrm{B}^{\mu}[\![c_1]\!]\ S \bigcap [\![p \neq 0]\!]) \bigcup (\mathrm{B}^{\mu}[\![c_2]\!]\ S \bigcap [\![p = 0]\!])$$

Considérons maintenant $\sigma \in \mathrm{B}^{\mu}[\![c]\!]\ S$; deux cas peuvent se produire :

soit $p(\sigma) \bowtie 0 \equiv \textit{true}$ ainsi $\sigma \in \mathrm{B}^{\mu}[\![c_1]\!]\ S$. L'hypothèse d'induction permet d'affirmer que : $\exists \sigma' \in S, \langle c_1, \sigma\rangle \to^* \sigma'$. La sémantique petit pas fournit alors la réduction :

$$\frac{p(\sigma) \bowtie 0 \equiv \textit{true}}{\langle \textbf{if}\ p \bowtie 0\ \textbf{then}\ c_1\ \textbf{else}\ c_2, \sigma\rangle \to \langle c_1, \sigma\rangle}\ \textbf{\textit{if}}$$

soit $p(\sigma) \bowtie 0 \equiv \textit{false}$ qui est le cas miroir du précédent. On a $\sigma \in \mathrm{B}^{\mu}[\![c_2]\!]\ S$ et, par hypothèse de récurrence : $\exists \sigma' \in S, \langle c_2, \sigma\rangle \to^* \sigma'$. De plus :

$$\frac{p(\sigma) \bowtie 0 \equiv \textit{false}}{\langle \textbf{if}\ p \bowtie 0\ \textbf{then}\ c_1\ \textbf{else}\ c_2, \sigma\rangle \to \langle c_2, \sigma\rangle}\ \textbf{\textit{if}}$$

Ainsi, pour ces deux cas, la propriété est vérifiée en prenant $\sigma_f = \sigma'$.

Si $c \equiv$ **while** $p \bowtie 0$ **do** c_1 alors on a, par le théorème d'itération de Kleene (théorème 2.2) :

$$\mathrm{B}^{\mu}[\![c]\!]\ S = \mu F_{c_1,p,S} = \bigcup_{n \geq 0} F^n_{c_1,p,S}(\emptyset)$$

Considérons maintenant $\sigma \in \mathrm{B}^{\mu}[\![c]\!]\ S$; alors, $\exists n_0 \in \mathbb{N}$ tel que $\sigma \in F^{n_0}_{c_1,p,S}(\emptyset)$. Pour conclure ce cas, nous démontrons par récurrence que :

$$\forall n \in \mathbb{N}, \quad \sigma \in F^n_{c_1,p,S}(\emptyset) \ \Rightarrow\ \exists \sigma_f \in S, \langle c, \sigma\rangle \to^* \langle \bullet, \sigma_f\rangle$$

- Cas $n = 0$. Comme $F^0_{c_1,p,S}(\emptyset) = \emptyset$, l'implication est toujours valide car sa prémisse est fausse.
- $n \to n+1$. Soit $n \in \mathbb{N}$ tel que $\sigma \in F^n_{c_1,p,S}(\emptyset) \ \Rightarrow\ \exists \sigma_f \in S, \langle c_1, \sigma\rangle \to^* \langle \bullet, \sigma_f\rangle$. Supposons maintenant que $\sigma \in F^{n+1}_{c_1,p,S}(\emptyset)$. On a :

$$\begin{array}{rcl} F^{n+1}_{c_1,p,S}(\emptyset) & = & F_{c_1,p,S}(F^n_{c_1,p,S}(\emptyset)) \\ & = & ([\![p \not\bowtie 0]\!] \bigcap S) \bigcup ([\![p \bowtie 0]\!] \bigcap \mathrm{B}^{\mu}[\![c_1]\!]\ (F^n_{c_1,p,S}(\emptyset))) \end{array}$$

Deux cas peuvent alors se produire.

soit $p(\sigma) \bowtie 0 \equiv \mathit{false}$ alors, la sémantique petit pas fournit :

$$\frac{p(\sigma) \bowtie 0 \equiv \mathit{false}}{\langle c, \sigma \rangle \to \langle \mathbf{skip}, \sigma \rangle}\ \boldsymbol{if}$$

Ainsi, en prenant $\sigma_f = \sigma$, on a $\langle c, \sigma \rangle \to^* \sigma_f$.

soit $p(\sigma) \bowtie 0 \equiv \mathit{true}$ alors $\sigma \in \mathrm{B}^{\mu}[\![c_1]\!]\ (F^n_{c_1,p,S}(\emptyset))$. Par hypothèse d'induction sur c_1, il existe σ'' tel que $\langle c_1, \sigma \rangle \to^* \sigma''$, et par hypothèse de récurrence sur n, on obtient un élément σ_f tel que $\langle c_1, \sigma'' \rangle \to^* \sigma_f$. De plus :

$$\frac{p(\sigma) \bowtie 0 \equiv \mathit{true}}{\langle c, \sigma \rangle \to \langle c_1; c, \sigma \rangle}\ \boldsymbol{if}$$

Ainsi, $\langle c, \sigma \rangle \to^* \sigma_f$.

Ceci pouve que la propriété est vérifiée dans ce dernier cas, ce qui démontre qu'elle est toujours vérifiée.

□

Bibliographie

[AGG08] Xavier Allamigeon, Stéphane Gaubert, and Eric Goubault. Inferring Min and Max Invariants Using Max-plus Polyhedra. In María Alpuente and Germán Vidal, editors, *Proceedings of the 15th International Static Analysis Symposium (SAS'08)*, volume 5079 of *Lecture Notes in Computer Science*, pages 189–204, Valencia, Spain, July 2008. Springer Verlag. 96

[AGG10] Xavier Allamigeon, Stéphane Gaubert, and Eric Goubault. The Tropical Double Description Method. In *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science (STACS'10)*, Nancy, France, March 2010. To appear. 96

[AGK] Xavier Allamigeon, Stéphane Gaubert, and Ricardo D. Katz. The number of extreme points of tropical polyhedra. *Journal of Combinatorial Theory, series A*. To appear. 96

[AJL] Mathias Argoud, Bertrand Jeannet, and Gaël Lalire. The `concurinterproc analyzer`. http ://pop-art.inrialpes.fr/interproc/concurinterprocweb.cgi. 94, 99, 100

[Alf98] Luca De Alfaro. How to Specify and Verify the Long-Run Average Behavior of Probabilistic Systems. In *13th Symposium on Logic in Computer Science (LICS'98)*, pages 174–183. IEEE Computer Society Press, 1998. 67, 95

[All09] Xavier Allamigeon. *Static analysis of memory manipulations by abstract interpretation — Algorithmics of tropical polyhedra, and application to abstract interpretation*. PhD thesis, École Polytechnique, Palaiseau, France, November 2009. 96

[Azi06] Benjamin Aziz. A Semiring-based Quantitative Analysis of Mobile Systems. *Electronic Notes in Theoretical Computer Science*, 157(1) :3–21, 2006. 96

[Bac00] Roland Carl Backhouse. Galois connections and fixed point calculus. In Roland Carl Backhouse, Roy L. Crole, and Jeremy Gibbons, editors, *ACMMPC*, volume 2297 of *Lecture Notes in Computer Science*, pages 89–148. Springer, 2000. 21, 123

[Bal61] M. L. Balinski. An algorithm for finding all vertices of convex polyhedral sets. *Journal of the Society for Industrial and Applied Mathematics*, 9(1) :72–88, March 1961. 19

[BCOQ92] François Baccelli, Guy Cohen, Geert Jan Olsder, and Jean-Pierre Quadrat. *Synchronization and Linearity*. Wiley, 1992. 42, 57, 59, 60, 63, 75, 81, 95

[BCR02] Stefano Bistarelli, Philippe Codognet, and Francesca Rossi. Abstracting soft constraints : framework, properties, examples. *Artif. Intell.*, 139(2) :175–211, 2002. 96

[BEK05] Tomas Brazdil, Javier Esparza, and Antonin Kucera. Analysis and Prediction of the Long-Run Behavior of Probabilistic Sequential Programs with Recursion (Extended Abstract). In *FOCS '05 : Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, pages 521–530, Washington, DC, USA, 2005. IEEE Computer Society. 67, 95

[Bis04] Stefano Bistarelli. *Semirings for Soft Constraint Solving and Programming*, volume 2962 of *Lecture Notes in Computer Science*. SpringerVerlag, 2004. 96

[BMR97] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Semiring-Based Constraint Satisfaction and Optimization. *Journal of the ACM*, 44(2) :201–236, 1997. 56, 96

[BPR03] Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in Real Algebraic Geometry*. Springer-Verlag, 2003. 155

[CC77] Patrick Cousot and Radhia Cousot. Abstract interpretation : A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, pages 238–252. ACM Press, 1977. 10, 11, 15, 151

[CC79] P Cousot and R Cousot. Systematic Design of Program Analysis Frameworks. In *6th POPL, San Antonio, Texas*, pages 269–282, January 1979. 15, 28, 35, 44, 45

[CC92a] P Cousot and R Cousot. Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation. In *PLILP'92*, volume 631 of *LNCS*, pages 269–295. Springer-Verlag, 1992. 24

[CC92b] Patrick Cousot and Radhia Cousot. Abstract interpretation and application to logic programs. *The Journal of Logic Programming*, 13(2 & 3) :103–179, 1992. 15, 21

[CC92c] Patrick Cousot and Radhia Cousot. Abstract interpretation frameworks. *Journal of Logic Computation*, 2(4) :511–547, 1992. 1, 38, 46, 47, 48

[CH78] Patrick Cousot and Nicolas Halbwachs. Automatic discovery of linear restraints among variables of a program. In *POPL*, pages 84–96. ACM Press, 1978. 9, 15, 18, 19, 27, 28, 29, 31, 46, 48, 99, 100, 153

[CJ10] David Cachera and Arnaud Jobin. Injecting abstract interpretations into linear cost models. In Alessandra Di Pierro and Gethin Norman, editors, *Proceedings Eighth Workshop on Quantitative Aspects of Programming Languages*, volume 28 of *EPTCS*, pages 64–81, 2010. 8, 97, 153

[CJJK11] David Cachera, Thomas Jensen, Arnaud Jobin, and Florent Kirchner. Fast inference of polynomial invariants for imperative programs, May 25 2011. 9, 101

[CJJS08] David Cachera, Thomas P. Jensen, Arnaud Jobin, and Pascal Sotin. Long-run cost analysis by approximation of linear operators over dioids. In José Meseguer and Grigore Rosu, editors, *Algebraic Methodology and Software Technology, 12th International Conference, AMAST 2008, Urbana, IL, USA, July 28-31, 2008, Proceedings*, volume 5140 of *Lecture Notes in Computer Science*, pages 122–138. Springer, 2008. 8, 97, 153

[CJJS10] David Cachera, Thomas P. Jensen, Arnaud Jobin, and Pascal Sotin. Long-run cost analysis by approximation of linear operators over dioids. *Mathematical Structures in Computer Science*, 20(4) :589–624, 2010. 8, 97, 153

[CLO07] David Cox, John Little, and Donal O'Shea. *Ideals, varieties, and algorithms*. Undergraduate Texts in Mathematics. Springer, third edition, 2007. An introduction to computational algebraic geometry and commutative algebra. 103, 104

[Cou81] P. Cousot. Semantic foundations of program analysis. In S. Muchnick and N. D. Jones, editors, *Program Flow Analysis*. Prentice Hall, 1981. 15, 35

[CSS03] Michael Colón, Sriram Sankaranarayanan, and Henny Sipma. Linear invariant generation using non-linear constraint solving. In Warren A. Hunt Jr. and Fabio Somenzi, editors, *CAV*, volume 2725 of *Lecture Notes in Computer Science*, pages 420–432. Springer, 2003. 9, 100, 143

[CTCG+98] Jean Cochet-Terrasson, Guy Cohen, Stéphane Gaubert, Michael Mc Gettrick, and Jean-Pierre Quadrat. Numerical Computation of Spec-

tral Elements in Max-Plus Algebra. In *Proceedings of the IFAC Conference on System Structure and Control*, 1998. 68, 95

[DFM+05] Rocco De Nicola, Gianluigi Ferrari, Ugo Montanari, Rosario Pugliese, and Emilio Tuosto. A Basic Calculus for Modelling Service Level Agreements. In *International Conference on Coordination Models and Languages*, volume 3454 of *Lecture Notes in Computer Science*, pages 33 – 48, Namur (Belgium), April 2005. Springer Verlag. 96

[DHW05] Alessandra Di Pierro, Chris Hankin, and Herbert Wiklicky. Probabilistic λ-calculus and Quantitative Program Analysis. *J. Logic and Computation*, 15(2) :159–179, 2005. 95

[Dij76] Edsger W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976. 31, 35

[DP90] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 1990. 21, 39, 84, 87

[DS87] P S Dudnikov and S N Samborskii. Endomorphisms of semimodules over semirings with an idempotent operation, preprint of the mathematical institute of ukrainian academy of sciences. In *Dudnikov, P.S. and Samborskii, S.N. : Endomorphisms of Finitely Generated Free Semimodules, in : Maslov, V.P. and Samborskii, S.N. (eds), Idempotent Analysis (Advances in Soviet Mathematics*, pages 65–85. American Mathematical Society, 1987. 60, 63

[DS92] P Dudnikov and S Samborskii. Endomorphisms of finitely generated free semimodules. In *Idempotent analysis, volume 13 of Adv. in Sov. Math. AMS, RI*, pages 65–85. American Mathematical Society, 1992. 60, 63

[DW00] Alessandra Di Pierro and Herbert Wiklicky. Concurrent Constraint Programming : Towards Probabilistic Abstract Interpretation. In *Principles and Practice of Declarative Programming*, pages 127–138, 2000. 95

[EKMS92] M. Erne, J. Koslowski, A. Melton, and G.E. Strecker. A primer on galois connections. In *York Academy of Science*, 1992. 76

[Fau02] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In *ISSAC*, pages 75–83. ACM Press, 2002. 153

[Flo67] R. W. Floyd. Assigning meaning to programs. In J. T. Schwartz, editor, *Proc. Symp. in Applied Mathematics*, pages 19–32. AMS, 1967. 30

[GM01] Michel Gondran and Michel Minoux. *Graphes, dioïdes et semi-anneaux*. Tec & Doc, Paris, 2001. 42, 56, 76

[GMC09] Sumit Gulwani, Krishna K. Mehra, and Trishul Chilimbi. Speed : precise and efficient static estimation of program computational complexity. *SIGPLAN Not.*, 44(1) :127–139, 2009. 155

[Hoa69] Hoare. An axiomatic basis for computer programming. *CACM : Communications of the ACM*, 12, 1969. 31, 35

[HT05] Dan Hirsch and Emilio Tuosto. SHReQ : Coordinating Application Level QoS. In *SEFM '05 : Proceedings of the Third IEEE International Conference on Software Engineering and Formal Methods*, pages 425–434, Washington, DC, USA, 2005. IEEE Computer Society. 96

[Kar76] Michael Karr. Affine relationships among variables of a program. *Acta Informatica*, 6 :133–151, 1976. 9, 100, 153

[KU77] Kam and Ullman. Monotone data flow analysis frameworks. *ACTAINF : Acta Informatica*, 7, 1977. 36

[Lan66] E Lanery. Recherche d'un système générateur minimal d'un polyhedre convexe. *Thèse de troisième cycle, Caen, France*, 1966. 19

[LG08] Xavier Leroy and Hervé Grall. Coinductive big-step operational semantics. *Information and Computation*, 2008. 130

[Min04] Antoine Miné. *Weakly Relational Numerical Abstract Domains*. PhD thesis, École Polytechnique, Palaiseau, France, December 2004. 47, 90, 92

[Min06] A. Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19(1) :31–100, 2006. `http://www.di.ens.fr/~mine/publi/article-mine-HOSC06.pdf`. 47

[MM82] Ernst W. Mayr and Albert R. Meyer. The complexity of the word problem for commutative semigroups and polynomial ideals. *Advances in Mathematics*, 46 :305–329, 1982. 101, 153

[MOPS06] Markus Müller-Olm, Michael Petter, and Helmut Seidl. Interprocedurally analyzing polynomial identities. In Bruno Durand and Wolfgang Thomas, editors, *STACS 2006, 23rd Annual Symposium on Theoretical Aspects of Computer Science, Marseille, France, February 23-25, 2006, Proceedings*, volume 3884 of *Lecture Notes in Computer Science*, pages 50–67. Springer, 2006. 100, 140, 141

[MOS02] Markus Müller-Olm and Helmut Seidl. Polynomial constants are decidable. In *SAS*, pages 4–19. Springer, 2002. 9, 100, 140

[MOS04] Markus Müller-Olm and Helmut Seidl. Computing polynomial program invariants. *Information Processing Letters*, 91(5) :233–244, 2004. 9, 100, 101, 106, 120, 140, 141, 153, 154

[Pet04] Michael Petter. Berechnung von polynomiellen Invarianten. Master's thesis, Technische Universität München, 2004. 142

[PS05] Michael Petter and Helmut Seidl. Inferring polynomial program invariants with polyinvar. Short paper, NSAD, 2005. 142, 143, 144

[RCK04a] Enric Rodríguez-Carbonell and Deepak Kapur. An abstract interpretation approach for automatic generation of polynomial invariants. In Roberto Giacobazzi, editor, *SAS*, volume 3148 of *Lecture Notes in Computer Science*, pages 280–295. Springer, 2004. 9, 100, 140, 147, 148

[RCK04b] Enric Rodríguez-Carbonell and Deepak Kapur. Automatic generation of polynomial loop invariants : algebraic foundations. In *ISSAC*, pages 266–273. ACM Press, 2004. 9, 144, 146

[RCK04c] Enric Rodríguez-Carbonell and Deepak Kapur. Program verification using automatic generation of invariants. In *ICTAC*, volume 3407 of *Lecture Notes in Computer Science*, pages 325–340. Springer, 2004. 9, 144, 146

[RCK07a] Enric Rodríguez-Carbonell and Deepak Kapur. Automatic generation of polynomial invariants of bounded degree using abstract interpretation. *Science of Computer Programming*, 64(1) :54–75, 2007. 9, 100, 101, 138, 139, 140, 147, 148, 149, 153, 154

[RCK07b] Enric Rodríguez-Carbonell and Deepak Kapur. Generating all polynomial invariants in simple loops. *Journal of Symbolic Computation*, 42(4) :443–476, 2007. 9, 100, 101, 138, 139, 140, 144, 145, 146, 153, 154

[Ric53] H. Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical Society*, 83, 1953. 6, 11

[San08] Francesco Santini. Managing quality of service with soft constraints. In *AAAI'08 : Proceedings of the 23rd national conference on Artificial intelligence*, pages 1869–1870. AAAI Press, 2008. 96

[SCJ06] Pascal Sotin, David Cachera, and Thomas Jensen. Quantitative Static Analysis over Semirings : Analysing Cache Behaviour for Java Card. In Alessandra Di Pierro and Herbert Wiklicky, editors, *QAPL06, Quantitative Aspects of Programming Languages*, 2006. 67, 152

[Sot05] Pascal Sotin. Analyse Statique Quantitative. *Thèse de troisième cycle, Université de Rennes*, 2005. www.irisa.fr/lande/sotin/MasterThesis.pdf. 152, 153

[SSM04a] Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna. Constraint-based linear-relations analysis. In Roberto Giacobazzi, editor, *SAS*, volume 3148 of *Lecture Notes in Computer Science*, pages 53–68. Springer, 2004. 100, 143

[SSM04b] Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna. Non-linear loop invariant generation using Gröbner bases. In *POPL*, pages 318–329. ACM Press, 2004. 9, 100, 101, 140, 143, 153, 154

[Tar55] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Mathematics*, 5(2) :285–309, June 1955. 21

[Win93] Glynn Winskel. *The Formal Semantics of Programming Languages*. MIT Press, 1993. 105