THESE

présentée à

Université Scientifique et Médicale de Grenoble

Institut National Polytechnique de Grenoble

pour obtenir le grade de

DOCTEUR ES SCIENCES MATHEMATIQUES

par

Patrick COUSOT

METHODES ITERATIVES DE CONSTRUCTION ET D'APPROXIMATION DE POINTS FIXES D'OPERATEURS MONOTONES SUR UN TREILLIS, ANALYSE SEMANTIQUE DES PROGRAMMES.

Thèse soutenue le 21 mars 1978 devant la Commission d'Examen :

Président	L. BOILiET
Examinateurs:	C. BENZAKEN
	Ph. JORRAND
	B. IORHO
	C. PAIR
	F. ROBERT
	M. SINTZOFF

1. INTRODUCTION

L'analyse sémantique d'un programme consiste à déterminer les conditions dans lesquelles les exécutions de ce programme se terminent, ne se terminent pas ou conduisent à une erreur (que ce soit parceque les règles de bonne utilisation du langage de programmation n'ont pas été respectées ou parceque le programme ne correspond pas à sa spécification). L'analyse sémantique d'un programme doit également permettre de déterminer en chaque point du programme les propriétés des objets manipulés par le programme.

Nous proposons une théorie de l'analyse sémantique des programmes qui fournit un cadre unifié pour effectuer des analyses depuis les plus fines, comme celles mises en oeuvre dans la justification de la correction totale des programmes, jusqu'aux plus grossières comme celles utilisées à la compilation. A notre avis, il n'y a pas de discontinuité entre ces deux extrêmes et la théorie proposée permet de construire un spectre continu d'applications depuis l'analyse exacte jusqu'aux analyses les plus approximatives. Soucieux des applications pratiques, nous avons consacré une partie de nos efforts à construire un modèle qui apporte des solutions automatisables, (certaines automatisations ayant d'ailleurs effectivement été réalisées), à des problèmes économiquement significatifs:

- Dans la plupart des systèmes de vérification de programmes, l'analyse sémantique du programme à justifier doit être faite par le programmeur, qui doit fournir une documentation du programme souvent lourde par son niveau de détail. Or, mise à part la spécification de sortie qui décrit le problème à résoudre, une bonne partie de cette documentation peut être construite d'après le texte de programme (avec l'assurance que cette documentation et le programme concordent).

- Les techniques de mise au point encore largement utilisées dans l'industrie informatique du logiciel peuvent être en partie évitées (du moins pour les fautes de programmation si ce n'est pour les fautes de conception), en utilisant nos propositions d'analyse sémantique automatique des programmes, et ce,

sans attendre les dix ans (ou plus) qui seront nécessaires pour que les techniques de vérification des programmes utilisant des démonstrateurs de théorèmes soient opérationnelles. Il est d'ailleurs certain que les méthodes que nous proposons sont complémentaires et offrent pour certains types d'analyses un rapport coût/bénéfice très rentable.

- Dans les langages de haut niveau, le programmeur est encouragé à formuler ses algorithmes en termes abstraits appropriés au problème à traiter. Pour faire un choix automatique d'une implémentation efficace des programmes, il faut en faire une analyse sémantique assez précise.

- Presque toutes les définitions des langages de programmation classiques contiennent diverses restrictions qui sont nécessaires pour que les programmes soient corrects mais qui ne peuvent pas, en toute généralité, être testées syntaxiquement. Il faudrait en effet connaître le domaine de valeurs des variables. La solution classique des tests à l'exécution est généralement jugée inacceptable à cause de son coût. Seule une analyse sémantique automatique des programmes peut apporter une solution économiquement rentable.

- La plupart des techniques d'optimisation utilisées dans la compilation des programmes, ne peuvent être mises en oeuvre que lorsque les conditions assurant l'équivalence du programme transformé et du programme original, et, une réelle amélioration des performances sont réunies. Quand il y a doute, l'option classique est de considérer l'hypothèse la plus pessimiste, mais une analyse sémantique plus approfondie du programme permet de l'éviter.

D'une manière générale, le développement d'une théorie de l'analyse sémantique des programmes conduisant à des applications automatisables est motivé par la résolution technique des problèmes de fiabilité et d'efficacité du logiciel et nous semble complémentaire des efforts qui sont faits actuellement pour faire passer l'art de la programmation à l'état de science.

Donnons maintenant un très bref résumé du contenu de cette thèse:

Nous ramènerons le problème de la détermination de propriétés sémantiques d'un programme, au problème de construire les points fixes extrêmes d'opérateurs monotones sur un treillis complet. Après cette introduction, le deuxième chapitre est donc consacré à l'étude mathématique de théorèmes

de points fixes dans les treillis complets. Nous donnons une démonstration constructive du théorème de Tarski qui montre que l'ensemble des points fixes d'un opérateur monotone F sur un treillis complet L est l'image de L par des préfermetures définies au moyen de limites d'itérations transfinies. Il s'agit de montrer, comment les méthodes itératives classiques peuvent être adaptées pour converger à partir de n'importe quel point de départ et également, pour atteindre des points fixes autres que le plus petit et le plus grand. Ceci nous permet également de définir l'union et l'intersection dans le treillis des points fixes de F de manière constructive, c'est-à-dire, par des récurrences utilisant F. Nous obtenons alors comme cas particulier, le théorème de construction du plus petit point fixe d'un opérateur continu. Nous considérons ensuite des systèmes d'équations monotones de points fixes dans un treillis complet. Après avoir rappelé la méthode de résolution formelle par élimination de variables, nous démontrons un résultat de convergence des méthodes itératives chaotiques, asynchrones et asynchrones avec mémoire. Ceci ouvre la voie à la résolution des systèmes d'équations monotones sur un treillis, en utilisant plusieurs processeurs calculant en parallèle sans qu'aucune synchronisation ne soit nécessaire.

Dans le chapitre trois, le problème de l'analyse sémantique des programmes est étudié, indépendamment des problèmes de définition de langage, dans le cadre très général de l'étude du comportement d'un système dynamique discret. Un programme est un système dynamique discret dans la mesure où il définit une relation de transition (ou une fonction de transition s'il est déterministe), entre les états de mémoire précédant et suivant l'exécution d'une instruction élémentaire quelconque. Pour étudier le comportement d'un système dynamique discret, il faut caractériser l'ensemble des états descendant des états satisfaisant à une spécification d'entrée donnée, ou bien, caractériser l'ensemble des ascendants des états satisfaisant à une spécification de sortie donnée. Autrement dit, il faut déterminer la plus faible pré-condition, portant sur les états d'entrée, pour que le système évolue vers un état satisfaisant à une post-condition donnée, ou bien, la plus forte post-condition caractérisant les états vers lesquels le système évolue à partir d'un état d'entrée quelconque qui satisfait à une pré-condition donnée. Nous montrons que ces conditions s'obtiennent comme solutions d'équations de points fixes ou de systèmes d'équations, quand l'ensemble des états du système dynamique discret est partitionné. Nous formalisons ensuite la sémantique opérationnelle d'un langage de programmation simple, correspon-

dant à des programmes séquentiels itératifs, nous montrons comment un programme définit un système dynamique discret, puis nous appliquons les résultats obtenus sur l'analyse du comportement des systèmes dynamiques discrets à l'analyse sémantique des programmes. Ceci nous conduit à définir des sémantiques déductives en avant et en arrière des programmes, qui généralisent les méthodes classiques de vérification de programmes "en avant" à la Floyd-Naur ou "en arrière" à la Hoare-Dijkstra, à des techniques de formalisation de la sémantique des langages de programmation. En effet, les sémantiques déductives en avant et en arrière définissent les conditions dans lesquelles l'exécution d'un programme se termine correctement, ne se termine pas ou conduit à une erreur comme solutions de systèmes d'équations sémantiques associées au programme. Chacune des deux sémantiques peut être utilisée pour caractériser en chaque point du programme, l'ensemble des descendants des états d'entrée et l'ensemble des ascendants des états de sortie et par conséquent, elles sont équivalentes entre elles et permettent toutes les deux d'effectuer l'analyse sémantique exacte des programmes.

Ayant montré que l'analyse sémantique exacte des programmes consiste à résoudre des systèmes d'équations, sachant que les solutions de ces équations ne sont pas automatiquement calculables et désirant cependant trouver des techniques automatiques d'analyse sémantique des programmes, nous sommes contraints de nous borner à des analyses automatiques approchées des programmes. Nous étudions donc dans le chapitre quatre des méthodes de calcul d'approximations de points fixes d'opérateurs monotones sur un treillis. Pour calculer effectivement des approximations inférieures et supérieures des solutions d'un système d'équations, nous proposons essentiellement deux méthodes complémentaires. Elles consistent d'une part, à simplifier les équations à résoudre et d'autre part, à accélérer la convergence des méthodes itératives de construction de points fixes. Pour accélérer la convergence d'une itération qui ne se stabilise par naturellement en un nombre fini de pas, nous proposons d'extrapoler en cours de calcul les termes de la suite des itérés pour obtenir, en un nombre fini de pas, une approximation de sa limite. De même que l'accélération de la convergence de méthodes itératives, la simplification d'équations est très utilisée en analyse numérique mais, pour le besoin de nos problèmes, nous devons les étudier dans un cadre purement algébrique. Pour simplifier les systèmes d'équations sémantiques associées aux programmes, nous proposons, pour chaque problème particulier d'analyse sémantique des programmes, d'ignorer a priori certaines propriétés

pour ne retenir que les propriétés des programmes qui sont significatives pour cette application spécifique. D'un point de vue algébrique, cette approximation se formalise par une fermeture sur le domaine des équations à résoudre, fermeture que nous définirons de façon équivalente au moyen de parties de Moore , de familles d'idéaux principaux, de relations de congruence ou de paires de fonctions adjointes. Diverses analyses approchées peuvent être combinées en combinant les fermetures correspondantes et en particulier, le treillis des fermetures formalise la hiérarchisation des approximations selon leur précision.

Nous développons au chapitre cinq, des méthodes d'analyse sémantique automatique et donc nécessairement approchée des programmes. Pour procéder à l'analyse sémantique approchée d'un programme, nous proposons de calculer une solution approchée des systèmes d'équations sémantiques en avant et en arrière associés à ce programme. Nous montrons comment, ayant choisi une classe particulière de propriétés des programmes apportant des réponses utiles à un problème donné, les résultats du chapitre quatre permettent de concevoir un algorithme réalisant automatiquement l'analyse d'un programme quelconque pour cette classe de propriétés. La conception de cet algorithme est basée sur le choix d'une fermeture qui permet de définir un espace de propriétés approchées ainsi que les règles de construction des systèmes d'équations simplifiés associés à un programme. Pour résoudre ces équations, nous utiliserons une méthode itérative et s'il est nécessaire d'accélérer la convergence, il reste donc à imaginer des opérations d'extrapolation. Nous illustrons notre approche par quelques exemples de conception d'une technique d'analyse sémantique approchée des programmes. Après avoir brièvement examiné plusieurs exemples classiques en optimisation des programmes, nous traitons quelques applications à la découverte des propriétés des pointeurs, à la détermination du type des variables dans les langages de haut niveau sans déclarations, à l'analyse de l'intervalle de valeurs des variables numériques et à la découverte de relations linéaires d'égalité ou d'inégalité entre variables d'un programme.

Le chapitre six traite des procédures récursives dont l'analyse est plus complexe que celle des programmes séquentiels itératifs puisqu'il faut considérer des équations fonctionnelles de la forme $f(x)=F(f)(x)$ et non plus des équations de type $x=f(x)$. Nous utilisons la même approche que pour les programmes séquentiels itératifs en définissant une sémantique déductive

puis, en introduisant des méthodes d'approximation qui en fait, généralisent l'étude du chapitre quatre au cas de systèmes d'équations fonctionnelles.

2. THEOREMES DE POINTS FIXES DANS LES TREILLIS COMPLETS

Nous ramènerons le problème de la détermination de propriétés sémantiques d'un programme au problème de résoudre un système d'équations X=F(X) dont la solution (ou point fixe de F) caractérise les propriétés du programme. Les chapitres 3, 5 et 6 montreront qu'il est judicieux de choisir un treillis complet L comme modèle des propriétés à découvrir et d'exprimer le système d'équations X=F(X) à l'aide d'un opérateur monotone F de L.

Ceci étant admis, l'objectif de ce chapitre est de rappeler, d'améliorer et d'établir un certain nombre de résultats mathématiques qui nous seront utiles pour résoudre un système d'équations monotones dans un treillis complet.

Les trois premiers paragraphes (2.1, 2.2, 2.3) comportent de brefs rappels sur les treillis complets, entre autres, sur l'image d'un treillis complet par une fermeture.

Nous montrons, dans le paragraphe 2.4, que l'ensemble des opérateurs monotones sur un treillis complet est l'image de l'ensemble des opérateurs sur ce treillis par une fonctionnelle qui est une fermeture.

Le résultat fondamental, pour résoudre une équation monotone à point fixe dans un treillis complet, est celui de Tarski[1955] qui assure l'existence de solutions, notamment d'une plus petite et d'une plus grande solution. On sait également construire ces solutions extrêmes par récurrence transfinie (ou finie en utilisant une hyptohèse supplémentaire de continuité). Nous donnons, au paragraphe 2.5, une démonstration constructive du théorème de Tarski dont l'originalité est de définir l'ensemble des points fixes d'un opérateur monotone F sur un treillis complet L comme image de L par des préfermetures définies au moyen de limites d'itérations transfinies. Il s'agit de montrer comment les méthodes itératives classiques peuvent être adaptées pour converger à partir de n'importe quel point de départ et également pour atteindre des points fixes autres que le plus petit et le plus grand. Ceci nous permet également de définir l'union et l'intersection dans

le treillis des points fixes de F de manière constructive, c'est-à-dire par des récurrences utilisant F.
Dans les paragraphes 2.6 et 2.7, nous montrons comment l'hypothèse classique de continuité consiste en fait à ne considérer que des opérateurs monotones dont les points fixes sont limites d'itérations "finies".

Par la suite, nous nous intéressons à la résolution d'un système d'équations monotones à points fixes dans un treillis complet: après avoir rapelé la méthode de résolution formelle par élimination de variables (2.8), nous démontrons un résultat de convergence des méthodes itératives chaotiques, asynchrones et asynchrones avec mémoire (2.9). Ceci ouvre la voie en particulier, à l'utilisation de plusieurs processeurs calculant en parallèle, (sans aucune synchronisation), pour résoudre de tels systèmes d'équations.

2.1 TREILLIS COMPLETS

Comme ouvrages fondamentaux sur les ensembles ordonnés et les treillis, on peut consulter Birkhoff[1967], Bourbaki[1967], Grätzer[1971] et Szàsz[1971]. Les quelques rappels donnés dans les paragraphes 2.1, 2.2, 2.3 ont pour but de fixer la terminologie et les notations.

Soit $L(\sqsubseteq,\bot,\top,\bigsqcup,\bigsqcap)$ un *treillis* (synonymes *ensemble ordonné, réticulé, réseau ordonné, lattis) complet* (synonyme *achevé)* pour *l'ordre partiel* $\sqsubseteq$. Par définition, toute partie S de L admet une *borne supérieure* (notée $\bigsqcup S$) et une *borne inférieure* (notée $\bigsqcap S$) dans L, ce qui entraîne en particulier que L admet un plus petit élément (*l'infimum* $\bot=\bigsqcap L$) et un plus grand élément (le *supremum* $\top=\bigsqcup L$). Si $x,y\in L$ nous noterons $\bigsqcup\{x,y\}$ par $x\sqcup y$ et $\bigsqcap\{x,y\}$ par $x\sqcap y$ et de même par convention $\{\{x\sqsubseteq y\}\Leftrightarrow\{x\sqsubseteq y \text{ et } x\neq y\}\}$, $\{\{x\sqsupseteq y\}\Leftrightarrow\{y\sqsubseteq x\}\}$ et $\{\{x\sqsupseteq y\}\Leftrightarrow\{y\sqsubseteq x\}\}$.

Soient μ un ordinal quelconque et $\langle x^\delta:\delta\in\mu\rangle$ une famille d'éléments de L. On dit que $\langle x^\delta:\delta\in\mu\rangle$ est une chaîne *ascendante* si $\{\forall\delta,\eta\in\mu,\ \{\{\delta\leq\eta\}\Rightarrow\{x^\delta\sqsubseteq x^\eta\}\}\}$ et que $\langle x^\delta:\delta\in\mu\rangle$ est une *chaîne strictement ascendante* si et seulement si $\{\forall\delta,\eta\in\mu,\ \{\{\delta<\eta\}\Rightarrow\{x^\delta\sqsubset x^\eta\}\}$. Les notions duales sont celles de *chaîne descendante* et *chaîne strictement descendante*.

On dit qu'un treillis satisfait à la *condition de chaîne ascendante* (ou encore à la *condition maximale*) si toute chaîne strictement ascendante est finie. La notion duale est celle de *condition de chaîne descendante*

(ou encore *condition minimale*).

2.2 TREILLIS COMPLET DES OPERATEURS SUR UN TREILLIS COMPLET

Nous noterons E→E' l'ensemble des applications de l'ensemble E dans l'ensemble E' partout définies sur E (la notation classique typographiquement trop encombrante est E'^{E}).

Soit L(⊑,⊥,⊤,⊔,⊓) un treillis complet, les éléments de L→L seront appelés *opérateurs* sur L. L'ensemble des opérateurs sur L est un treillis complet (L→L)(⊑',⊥',⊤',⊔',⊓') pour l'ordre partiel ⊑' défini par {∀f,g∈(L→L), {f ⊑' g} <=> {∀x∈L, f(x) ⊑ g(x)}}. En utilisant la notation lambda de Church[1951] nous avons ⊥'=λx.⊥, ⊤'=λx.⊤, ⊔'=λS.(λx.⊔{f(x):f∈S}) et ⊓'=λS.(λx.⊓{f(x):f∈S}). Pour alléger les notations nous omettrons les primes et la distinction entre (⊑,⊥,⊤,⊔,⊓) et (⊑',⊥',⊤',⊔',⊓') sera contextuelle.

2.3 IMAGE D'UN TREILLIS COMPLET PAR UNE FERMETURE

Pour faire l'étude d'une partie R d'un treillis complet L il est souvent utile de la représenter comme image f(L)=R de L par un opérateur f sur L. En effet, l'étude des propriétés de f apporte très fréquemment des informations intéressantes sur R. Ainsi nous rencontrerons souvent (§2.4, 2.5.3, 2.6, 5, 6) le cas particulier important où f est une fermeture et dans cette circonstance la partie R de L est un treillis complet dont on sait construire l'union et l'intersection.

Rappelons qu'un opérateur ρ sur un ensemble ordonné L(⊑) est une *fermeture supérieure* de L si et seulement si il est *monotone*(synonymes *isotone*, *croissant*,c'est-à-dire {∀x,y∈L, {x ⊑ y} => {ρ(x) ⊑ ρ(y)}}), *extensif* (λx.x ⊑ ρ) et *idempotent* (ρ=ρ∘ρ).

THEOREME 2.3.0.1 Ward[1942, th.4.1]

Soient L(⊑,⊥,⊤,⊔,⊓) un treillis complet et ρ une fermeture supérieure de L alors l'image ρ(L) de L par ρ est un treillis complet ρ(L)(⊑,ρ(⊥),⊤,λS.ρ(⊔S),⊓).

DEFINITION 2.3.0.2 Szàsz[1971, p.50]

Un opérateur f du treillis complet $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ est un *morphisme complet pour l'union* si et seulement si $\{\forall S\subseteq L,\ f(\sqcup S)=\sqcup f(S)\}$. La notion duale est celle de *morphisme complet pour l'intersection*. Un opérateur est un *morphisme complet* si et seulement si c'est un morphisme complet pour l'union et l'intersection.

Soient $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ et $M(\sqsubseteq,\bot',\top',\sqcup',\sqcap')$ deux treillis complets tels que $M\subseteq L$. Nous dirons que M est un *sous-sup-demi-treillis* de L si $\sqcup'=\sqcup$ (dualement *sous-inf-demi-treillis*) et que M est un *sous-treillis* de L si $\sqcup'=\sqcup$ et $\sqcap'=\sqcap$.

PROPOSITION 2.3.0.3 Ward[1942, p.193]

L'image $\rho(L)$ d'un treillis complet $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ par une fermeture supérieure ρ de L est un sous-treillis complet de L si et seulement si ρ est un morphisme complet pour l'union $\sqcup$.

PROPOSITION 2.3.0.4 Monteiro & Ribeiro[1942]

(a) - Soit ρ une fermeture supérieure d'un ensemble ordonné $L(\sqsubseteq)$. Pour tout $x\in L$ l'ensemble $\{y\in\rho(L) : x\sqsubseteq y\}$ n'est pas vide et admet un plus petit élément égal à $\rho(x)$.

(b) - Réciproquement si R est une partie de L telle que pour tout $x\in L$ l'ensemble $\{y\in R : x\sqsubseteq y\}$ admette un plus petit élément $\rho(x)$ alors ρ est une fermeture supérieure et $R=\rho(L)$.

A partir de ces résultats connus nous pouvons démontrer par dualité des propriétés analogues:

Un opérateur ρ d'un ensemble ordonné $L(\sqsubseteq)$ est une *fermeture inférieure* de L si et seulement si il est monotone, idempotent et *réductif* ($\rho\sqsubseteq\lambda x.x$).

COROLLAIRE 2.3.0.5

Soient $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet et ρ une fermeture inférieure L alors l'image $\rho(L)$ de L par ρ est un treillis complet $\rho(L)(\sqsubseteq,\bot,\rho(\top),\sqcup,\lambda S.\rho(\sqcap S))$.

C'est un sous-treillis complet de L si et seulement si ρ est un morphisme complet pour l'intersection $\sqcap$.

COROLLAIRE 2.3.0.6

Pour qu'une partie R d'un treillis complet $L(\sqsubseteq,\perp,\top,\sqcup,\sqcap)$ soit l'image $\rho(L)$ de L par une fermeture inférieure ρ de L il faut et il suffit que pour tout $x\in L$ l'ensemble $\{y\in R : y \sqsubseteq x\}$ ne soit pas vide et admette un plus grand élément égal à $\rho(x)$.

Ces quelques propriétés des fermetures sont les seules que nous utiliserons dans le chapitre 2. Pour les besoins des chapitres 5 et 6 l'étude sera complétée au chapitre 4.

2.4 TREILLIS COMPLET DES OPERATEURS MONOTONES SUR UN TREILLIS COMPLET

Un opérateur f sur $L(\sqsubseteq)$ est *monotone* si et seulement si $\{\forall x\in L, \{x \sqsubseteq y\} \Rightarrow \{f(x) \sqsubseteq f(y)\}\}$. Quand L est un treillis complet cette définition est équivalente à (Bourbaki[1967, ex.10, p.102]):

- Un opérateur f sur le treillis complet $L(\sqsubseteq,\perp,\top,\sqcup,\sqcap)$ est monotone si et seulement si $\{\forall S\subseteq L, \sqcup f(S) \sqsubseteq f(\sqcup S)\}$
- Un opérateur f sur le treillis complet $L(\sqsubseteq,\perp,\top,\sqcup,\sqcap)$ est monotone si et seulement si $\{\forall S\subseteq L, f(\sqcap S) \sqsubseteq \sqcap f(S)\}$.

Il est bien connu que les opérateurs monotones sur un treillis complet L forment un sous-treillis complet de $(L\to L)$ (par exemple Bourbaki[1967, ex.11.d, p.103]). L'intérêt de la démonstration que nous donnons est d'illustrer une démarche que nous utiliserons souvent par la suite: soit à étudier l'ensemble R des éléments d'un treillis complet L satisfaisant une propriété P. Si on peut trouver un opérateur ρ sur L tel que pour tout x de L, l'ensemble des éléments de L supérieurs à x et satisfaisant P est non vide et admet un plus petit élément égal à $\rho(x)$ alors on sait que ρ est une fermeture supérieure (2.3.0.4) et $R=\rho(L)$ est un treillis complet (2.3.0.1). De plus R est un sous-treillis de L quand ρ est un morphisme complet pour l'union (2.3.0.3).

Ainsi l'ensemble des opérateurs monotones sur un treillis complet L est un sous-treillis du treillis $(L\to L)$ des opérateurs sur L dont il est l'image par une fermeture supérieure *mon* sur $(L\to L)$ que nous savons construire:

DEFINITION 2.4.0.1

Soit $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet. Nous noterons *mon* l'opérateur sur $(L \to L)$ défini par:

$mon = \lambda f.(\lambda x.\sqcup\{f(y) : (y \in L) \text{ et } (y \sqsubseteq x)\})$

THEOREME 2.4.0.2

Soit f un opérateur sur L alors *mon*(f) est le plus petit opérateur monotone sur L supérieur ou égal à f.

Preuve: Soient a et b appartenant à L tels que $a \sqsubseteq b$. Pour tout y de L $\{y \sqsubseteq a\}$ implique $\{y \sqsubseteq b\}$ donc $\sqcup\{f(y):y \sqsubseteq a\} \sqsubseteq \sqcup\{f(y):y \sqsubseteq b\}$ ce qui prouve que *mon*(f)(a) $\sqsubseteq$ *mon*(f)(b) c'est-à-dire que *mon*(f) est un opérateur monotone de L.

Pour tout a de L nous avons $f(a) \sqsubseteq \sqcup\{f(y):y \sqsubseteq a\}$ car $\sqsubseteq$ est réflexive. En conséquence *mon*(f) est supérieur ou égal à f.

Soit g un opérateur monotone sur L tel que $f \sqsubseteq g$. Alors pour tout y de L, $f(y) \sqsubseteq g(y)$ ce qui implique pour tout a de L *mon*(f)(a) = $\sqcup\{f(y):y \sqsubseteq a\}$ $\sqsubseteq \sqcup\{g(y):y \sqsubseteq a\} \sqsubseteq \sqcup\{g(y):g(y) \sqsubseteq g(a)\} \sqsubseteq g(a)$ ce qui montre que *mon*(f) est le plus petit opérateur monotone sur L supérieur ou égal à f.
Fin de la preuve.

Appliquant le théorème 2.3.0.4.(b) nous obtenons:

COROLLAIRE 2.4.0.3

mon est une fermeture supérieure sur $(L \to L)$ et *mon*$(L \to L)$ est l'ensemble des opérateurs monotones sur L.

Les théorèmes 2.3.0.1 et 2.3.0.3 permettent alors de retrouver le résultat connu:

THEOREME 2.4.0.4 Bourbaki[1967,p.163]

L'ensemble des opérateurs monotones sur un treillis complet L est un sous-treillis complet $(\sqsubseteq,\lambda x.\bot,\lambda x.\top,\sqcup,\sqcap)$ de $(L \to L)$.

Appliquant le principe de dualité nous obtenons:

COROLLAIRE 2.4.0.5

Soit f un opérateur sur L. Alors $\lambda x.\sqcap\{f(y):(y\in L)\underline{et}(x\sqsubseteq y)\}$ est le plus grand des opérateurs monotones sur L plus petits ou égaux à f. De plus $\lambda f.(\lambda x.\sqcap\{f(y):(y\in L)\underline{et}(x\sqsubseteq y)\}$ est une fermeture inférieure de $(L\to L)$ et l'ensemble des opérateurs monotones sur L est un sous-treillis complet de $(L\to L)$ dont il est l'image par cette fermeture inférieure.

2.5 VERSION CONSTRUCTIVE DU THEOREME DE POINT FIXE DE TARSKI

Le théorème fondamental de Tarski[1955] indique que l'ensemble $fp(f)$ des *points fixes* d'un opérateur f du treillis complet non vide $L(\sqsubseteq,\perp,\top,\sqcup,\sqcap)$ (c'est-à-dire $fp(f) = \{x\in L : f(x)=x\}$) est un treillis complet non vide pour l'ordre $\sqsubseteq$ (et n'est pas nécessairement un sous-treillis de L). La preuve de Tarski est basée sur la définition du plus petit point fixe $lfp(f)$ de f par $lfp(f) = \sqcap\{x\in L:f(x)\sqsubseteq x\}$. La borne supérieure de $S\subseteq fp(f)$ dans $fp(f)$ est alors $lfp(f|\{x\in L:(\sqcup S)\sqsubseteq x\})$ et la preuve se termine en appliquant le principe de dualité.

Les utilisateurs aussi bien informaticiens qu'analystes numériciens (Amann[1976]) du théorème de Tarski préfèrent définir $lfp(f)$ comme $\bigsqcup_{i<\omega} f^i(\perp)$ où ω est le premier ordinal infini. Ce schéma itératif remonte au premier théorème de récursion de Kleene[1952] et fut utilisé par Tarski[1955] pour les morphismes complets. Plus généralement il exige une hypothèse de continuité (voir par exemple Kolodner[1968]) qui est plus forte que la monotonie (il faut en l'occurrence que $f(\bigsqcup_{i<\omega} f^i(\perp)) = \bigsqcup_{i<\omega} f^{i+1}(\perp)$). D'un point de vue abstrait l'hypothèse de continuité n'est pas nécessaire et nous aurons recours à des itérations transfinies comme le font par exemple Devidé[1964], Hitchcock & Park[1973] et Pasini[1974]. Pour donner une version constructive du théorème de Tarski nous utiliserons également l'idée de Cousot & Cousot[1977a] et Manna & Shamir[1977] de calculer les points fixes de f en deux temps: une itération croissante à partir d'un point de départ arbitraire d converge vers un post-point-fixe p de f (i.e. $f(p)\sqsubseteq p$), puis une itération décroissante partant de p converge vers un point fixe q. Posant $q=f^*(d)$, nous montrons que l'ensemble $fp(f)$ des points fixes de f est l'image du treillis complet L par la fonction f^* qui est une préfermeture, ce qui implique que $fp(f) = f^*(L)$ est un treillis complet.

2.5.1 Définition d'une itération transfinie

La classe Ord des ordinaux est bien ordonnée par $\leq=\{(x,y):(x,y \in Ord)$ et $((x \in y)$ ou $(x=y))\}$. Nous noterons 0 le type d'ordre de l'ensemble vide et $\cup_{i \in I} \delta_i$ l'ordinal qui est la borne supérieure de la famille d'ordinaux $\{\delta_i : i \in I\}$. Rappelons que si δ est un *ordinal limite* alors $\cup_{\alpha<\delta} \alpha=\delta$ sinon δ est un *ordinal successeur* et $\cup_{\alpha<\delta} \alpha = (\delta-1)$ où le prédécesseur de δ est noté $\delta-1$.

DEFINITION 2.5.1.0.1 *Itération croissante*

Soient $L(\sqsubseteq,\perp,\top,\sqcup,\sqcap)$ un treillis complet, $\mu(L)$ le plus petit ordinal tel que la cardinalité de la classe $\{\delta:\delta \in \mu(L)\}$ soit strictement supérieure à la cardinalité *card*(L) de L. Soit $f \in mon(L \to L)$.

L'*itération croissante partant de* $d \in L$ *et définie par* f est une séquence $\langle x^\delta : \delta \in \mu(L)\rangle$ d'éléments de L définie par récurrence transfinie comme suit:

(a) - $x^0 = d$

(b) - $x^\delta = f(x^{\delta-1})$ pour tout ordinal successeur $\delta \in \mu(L)$.

(c) - $x^\delta = \bigsqcup_{\alpha<\delta} x^\alpha$ pour tout ordinal limite $\delta \in \mu(L)$.

En général, les éléments d'une "itération croissante" ne sont pas comparables mais nous utilisons le qualificatif "croissant" car nous choisirons toujours des points de départ assurant qu'une itération croissante est une chaîne ascendante.

DEFINITION 2.5.1.0.2 *Itération décroissante*

L'*itération décroissante partant de* $d \in L(\sqsubseteq,\perp,\top,\sqcup,\sqcap)$ *et définie par* $f \in mon(L \to L)$ est une séquence $\langle x^\delta : \delta \in \mu(L)\rangle$ d'éléments de L définie par récurrence transfinie comme suit:

(a) - $x^0 = d$

(b) - $x^\delta = f(x^{\delta-1})$ pour tout ordinal successeur $\delta \in \mu(L)$.

(c) - $x^\delta = \bigsqcap_{\alpha<\delta} x^\alpha$ pour tout ordinal limite $\delta \in \mu(L)$.

DEFINITION 2.5.1.0.3 *Limite d'une séquence transfinie stationnaire*

Nous dirons que la séquence $<x^{\delta}:\delta\epsilon\mu>$ d'éléments de $L(\sqsubseteq,\perp,\top,\sqcup,\sqcap)$ est *stationnaire* si et seulement si $\{\exists\varepsilon\epsilon\mu : \{\forall\beta\epsilon\mu, \{\beta\geq\varepsilon\} \Rightarrow \{x^{\varepsilon}=x^{\beta}\}\}\}$ auquel cas la *limite* de cette séquence est x^{ε}. Nous noterons *luis*(f)(d) (respectivement *llis*(f)(d)) la limite d'une itération croissante (respectivement décroissante) stationnaire partant de d et définie par $f \in mon(L\rightarrow L)$.

LEMME 2.5.1.0.4

Soient $L(\sqsubseteq,\perp,\top,\sqcup,\sqcap)$ un treillis complet et $<x^{\delta}:\delta\epsilon\mu(L)>$ une itération croissante (respectivement décroissante) partant de $d\epsilon L$ et définie par $f \in mon(L\rightarrow L)$. Si $<x^{\delta}:\delta\epsilon\mu(L)>$ est une chaîne ascendante (respectivement descendante) alors elle est stationnaire et sa limite est le plus petit des points fixes de f plus grands ou égaux à d (respectivement le plus grand des points fixes de f plus petits ou égaux à d).

Preuve: Plaçons nous dans le cas d'une itération croissante et supposons que $\{\forall\varepsilon,\ \varepsilon\epsilon\mu(L)\ \underline{\text{et}}\ (\varepsilon+1)\epsilon\mu(L)\} \Rightarrow \{x^{\varepsilon}\neq x^{\varepsilon+1}\}\}$. Alors comme par hypothèse $<x^{\delta}:\delta\epsilon\mu(L)>$ est une chaîne ascendante il s'en suit que c'est une chaîne strictement ascendante et les classes $\{x^{\delta}:\delta\epsilon\mu(L)\}$ et $\{\delta:\delta\epsilon\mu(L)\}$ sont équipotentes de sorte que par définition de $\mu(L)$ $card(\{x^{\delta}:\delta\epsilon\mu(L)\}) > card(L)$. Mais comme f est partout définie sur L nous savons que $\{\forall\delta\epsilon\mu(L),\ x^{\delta}\epsilon L\}$ ce qui permet de déduire la contradiction $card(\{x^{\delta}:\delta\epsilon\mu(L)\}) \leq card(L)$. Par l'absurde nous avons montré que $\{\exists\varepsilon\epsilon\mu(L) : (\varepsilon+1)\epsilon\mu(L)\ \underline{\text{et}}\ x^{\varepsilon}= x^{\varepsilon+1}\}$.

Supposons que $\{\forall\beta,\ \{\varepsilon\leq\beta<\gamma<\mu(L)\} \Rightarrow \{x^{\varepsilon}=x^{\beta}\}\}$. Alors si γ est un ordinal successeur $x^{\varepsilon} = f(x^{\varepsilon}) = x^{\varepsilon+1} = x^{\gamma-1} = f(x^{\gamma-1}) = x^{\gamma}$ par la définition 2.5.1.0.1.(b) et l'hypothèse de récurrence. Autrement γ est un ordinal limite et $x^{\gamma} = \bigsqcup_{\alpha<\gamma} x^{\alpha} = (\bigsqcup_{\alpha<\varepsilon} x^{\alpha}) \sqcup (\bigsqcup_{\varepsilon\leq\beta<\gamma} x^{\beta}) \sqsubseteq x^{\varepsilon}$. Aussi comme $<x^{\delta}:\delta\epsilon\mu(L)>$ est une chaîne descendante et $\varepsilon<\gamma$ nous avons $d = x^{0} \sqsubseteq x^{\varepsilon} \sqsubseteq x^{\gamma}$ et par antisymétrie $d \sqsubseteq x^{\varepsilon} = x^{\gamma}$. Par récurrence transfinie nous concluons que $<x^{\delta}:\delta\epsilon\mu(L)>$ est stationnaire et $d \sqsubseteq x^{\varepsilon} = x^{\varepsilon+1} = f(x^{\varepsilon})$.

Supposons qu'il existe $p \in fp(f)$ tel que $d \sqsubseteq p$. Montrons que $\forall\delta\epsilon\mu(L)$, $x^{\delta} \sqsubseteq p$. Pour $\delta=0$ nous avons $x^{0} = d \sqsubseteq p$. Supposons que pour tout β tel que $\beta<\delta<\mu(L)$ on ait $x^{\beta} \sqsubseteq p$. Si δ est un ordinal successeur alors par monotonie $x^{\delta} = f(x^{\delta-1}) \sqsubseteq f(p) = p$. Si δ est un ordinal limite alors $x^{\delta} = \bigsqcup_{\beta<\delta} x^{\beta} \sqsubseteq p$ car p est un majorant de tous les β tels que $\beta<\delta$. Par récurrence transfinie $\forall\delta\epsilon\mu(L)$, $x^{\delta} \sqsubseteq p$ et ceci est vrai en particulier pour la limite.
Fin de la preuve.

2.5.2 Itération croissante partant d'un pré-point fixe

DEFINITION 2.5.2.0.1 *Pré-points fixes et post-points fixes d'un opérateur*

Soient $L(\sqsubseteq)$ un ensemble ordonné et $f \in (L \to L)$, alors l'ensemble des *pré-points fixes* de f est $prefp(f) = \{x \in L : x \sqsubseteq f(x)\}$. Dualement l'ensemble des *post-points fixes* de f est $postfp(f) = \{x \in L : f(x) \sqsubseteq x\}$.

THEOREME 2.5.2.0.2

Soit $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet. Une itération croissante $\langle x^\delta : \delta \in \mu(L)\rangle$ partant de $d \in prefp(f)$ et définie par $f \in mon(L \to L)$ est une chaîne ascendante stationnaire et sa limite $luis(f)(d)$ est le plus petit des points fixes de f plus grands ou égaux à d.

Preuve: Comme $x^0 = d \sqsubseteq f(d) = x^1$ et f est monotone, il est facile de montrer par récurrence transfinie que $\langle x^\delta : \delta \in \mu(L)\rangle$ est une chaîne ascendante et donc stationnaire. D'après le lemme 2.5.1.0.4., sa limite $luis(f)(d)$ est le plus petit des points fixes de f plus grands ou égaux à d.
Fin de la preuve.

Appliquant le principe de dualité, nous obtenons:

COROLLAIRE 2.5.2.0.3

Soit $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet. Une itération décroissante $\langle x^\delta : \delta \in \mu(L)\rangle$ partant de $d \in postfp(f)$ et définie par $f \in mon(L \to L)$ est une chaîne descendante stationnaire et sa limite $llis(f)(d)$ est le plus grand d points fixes de f plus petits ou égaux à d.

Remarque 2.5.2.0.4 *Points fixes extrêmes d'un opérateur monotone*

Comme $\bot \in prefp(f)$ et $\top \in postfp(f)$ pour tout f, $f \in mon(L \to L)$ admet au moins deux points fixes (pas nécessairement distincts). De plus comme tout point fixe de f est compris entre $\bot$ et $\top$, f admet un plus petit point fixe $lfp(f) = luis(f)(\bot)$ et un plus grand point fixe $gfp(f) = llis(f)(\top)$.
Fin de la remarque.

PROPOSITION 2.5.2.0.5

Soient $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet et $f \in mon(L \to L)$ alors la restriction $(luis(f)|prefp(f))$ de $luis(f)$ à $prefp(f)$ est une fermeture supérieure de $prefp(f)$ et $fp(f) = luis(f)(prefp(f))$.

Preuve: $fp(f)$ est une partie de $prefp(f)$ qui d'après le théorème 2.5.2.0.2. est telle que pour tout $x \in prefp(f)$ l'ensemble $\{y \in fp(f) : x \sqsubseteq y\}$ admet un plus petit élément $luis(f)(x)$ et par conséquent le théorème 2.3.0.4.(b) implique que $(luis(f)|prefp(f))$ est une fermeture supérieure de $prefp(f)$ et $fp(f) = luis(f)(prefp(f))$.
Fin de la preuve.

Appliquant le principe de dualité, nous obtenons:

COROLLAIRE 2.5.2.0.6

Soient $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet et $f \in mon(L \to L)$ alors la restriction de $llis(f)$ à $postfp(f)$ est une fermeture inférieure de $postfp(f)$ et $fp(f) = llis(postfp(f))$.

PROPOSITION 2.5.2.0.7

Soient $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet et $f, g \in mon(L \to L)$ tels que $f \sqsubseteq g$. Alors $\{\forall d \in prefp(f),\ luis(f)(d) \sqsubseteq luis(g)(d)\}$.

Preuve: Comme $f \sqsubseteq g$ on a $prefp(f) \subseteq prefp(g)$ et la démonstration se fait par récurrence transfinie sur les itérations croissantes pour f et g.
Fin de la preuve.

PROPOSITION 2.5.2.0.8

Soient $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ et $L'(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ des treillis complets, $f \in mon(L \to L)$ $g \in mon(L' \to L')$ et h un morphisme complet pour l'union de L dans L' tels que $h \circ f = g \circ h$ alors $\{\forall d \in prefp(f),\ h(luis(f)(d)) = luis(g)(h(d))\}$.

Preuve: Soient $\mu = \max(\mu(L),\mu(L'))$ et $\langle X^{\delta} : \delta \in \mu \rangle$ l'itération croissante pour f partant de d. Comme h est un morphisme complet de L dans L', il est monotone et par conséquent $\forall d \in prefp(f)$ nous avons $d \sqsubseteq f(d)$ qui implique $h(d) \sqsubseteq h(f(d)) = g(h(d))$ soit $h(d) \in prefp(g)$. Soit $\langle Y^{\delta} : \delta \in \mu \rangle$ l'itération croissante

définie par g et partant de h(d). Montrons que $\forall\delta$, $h(X^\delta)=Y^\delta$. Nous avons $h(X^0) = h(d) = Y^0$. Supposons que par hypothèse d'induction $\forall\alpha<\delta$, $h(X^\alpha)=Y^\alpha$. Si δ est un ordinal successeur alors en particulier $h(X^{\delta-1}) = Y^{\delta-1}$ et $h(X^\delta) = h(f(X^{\delta-1})) = g(h(X^{\delta-1})) = g(Y^{\delta-1}) = Y^\delta$. Si δ est un ordinal limite alors $h(X^\delta) = h(\bigsqcup_{\alpha<\delta} X^\alpha) = \bigsqcup_{\alpha<\delta} h(X^\alpha) = \bigsqcup_{\alpha<\delta} Y^\alpha = Y^\delta$ car h est un morphisme complet et par hypothèse d'induction. Par induction transfinie nous concluons $\forall\delta\epsilon\mu$, $h(X^\delta)=Y^\delta$. Soient X^ε et $Y^{\varepsilon'}$ les limites respectives de $\langle X^\delta:\delta\epsilon\mu\rangle$ et $\langle Y^\delta:\delta\epsilon\mu\rangle$ Il vient $h(\mathit{luis}(f)(d)) = h(X^\varepsilon) = h(X^{max(\varepsilon,\varepsilon')}) = Y^{max(\varepsilon,\varepsilon')} = Y^{\varepsilon'} = \mathit{luis}(g)(h(d))$.

Fin de la preuve.

2.5.3 Caractérisation constructive des ensembles des pré- et post-points fixes d'un opérateur monotone sur un treillis complet

PROPOSITION 2.5.3.0.1

Soient $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet et $f\in mon(L\to L)$. Pour tout $d\in L$ les itérations croissantes partant de d et définies par $\lambda x.x\sqcup f(x)$ et $\lambda x.d\sqcup f(x)$ sont des chaînes ascendantes stationnaires. Leurs limites $\mathit{luis}(\lambda x.x\sqcup f(x))(d)$ et $\mathit{luis}(\lambda x.d\sqcup f(x))(d)$ sont égales au plus petit des post-points fixes de f plus grands ou égaux à d.

Preuve: Tout élément d de L est un pré-point fixe de $\lambda x.x\sqcup f(x)$ et de $\lambda x.d\sqcup f(x)$ qui d'après l'hypothèse $f\in mon(L\to L)$ sont des opérateurs monotones de L. Alors le théorème 2.5.2.0.2. implique que les itérations croissantes $\langle x^\delta:\delta\epsilon\mu(L)\rangle$ et $\langle y^\delta:\delta\epsilon\mu(L)\rangle$ partant de d et définies respectivement par $\lambda x.x\sqcup f(x)$ et $\lambda x.d\sqcup f(x)$ sont des chaînes ascendantes stationnaires de limites respectives x^{ε_1} et y^{ε_2}.

Comme x^{ε_1} est le plus petit des points fixes de $\lambda x.x\sqcup f(x)$ supérieurs à d et que $\{\forall p\epsilon L,\ \{p=p\sqcup f(p)\} \Longleftrightarrow \{f(p)\sqsubseteq p\}\}$ nous concluons que $x^{\varepsilon_1} = \mathit{luis}(\lambda x.x\sqcup f(x))(d)$ est le plus petit des post-points fixes de f plus grands ou égaux à d.

Comme $y^{\varepsilon_2} = d\sqcup f(y^{\varepsilon_2})$ nous avons $f(y^{\varepsilon_2})\sqsubseteq y^{\varepsilon_2}$ et donc $d\sqsubseteq y^{\varepsilon_2}$ implique $x^{\varepsilon_1}\sqsubseteq y^{\varepsilon_2}$. Il est facile de montrer par récurrence transfinie sur δ que $\{\forall\delta\epsilon\mu(L),\ y^\delta\sqsubseteq x^\delta\}$. Alors $y^{\varepsilon_2} = y^{max(\varepsilon_1,\varepsilon_2)}\sqsubseteq x^{max(\varepsilon_1,\varepsilon_2)} = x^{\varepsilon_1}$ et par antisymétrie $x^{\varepsilon_1} = y^{\varepsilon_2}$ soit $\mathit{luis}(\lambda x.x\sqcup f(x))(d) = \mathit{luis}(\lambda x.d\sqcup f(x))(d)$.

Fin de la preuve.

THEOREME 2.5.3.0.2 *Treillis des post-points fixes d'un opérateur monotone*

Soient $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet et $f \in mon(L \to L)$. L'ensemble des post-points fixes de f est un treillis complet non vide:

$postfp(f)(\sqsubseteq, lfp(f), \top, \lambda S.luis(\lambda z.z \sqcup f(z))(\sqcup S), \sqcap)$

où le plus petit point fixe de f est $lfp(f) = \sqcap\{x \in L : f(x) \sqsubseteq x\} = luis(f)(d)$ pour tout $d \in L$ tel que $d \sqsubseteq lfp(f)$.

Preuve: D'après 2.5.3.0.1, 2.3.0.4 et 2.3.0.1 $postfp(f)$ est un treillis complet $(\sqsubseteq, luis(\lambda z.z \sqcup f(z))(\bot), \top, \lambda S.luis(\lambda z.z \sqcup f(z))(\sqcup S), \sqcap)$.

Le théorème 2.5.3.0.1. implique que $luis(\lambda z.z \sqcup f(z))(\bot) = luis(\lambda z.\bot \sqcup f(z))(\bot) = luis(f)(\bot) = lfp(f)$ (remarque 2.5.2.0.4). Comme $lfp(f)$ est l'infimum de $postfp(f)$ nous avons $lfp(f) = \sqcap postfp(f)$.

Finalement, soit $d \in L$ tel que $d \sqsubseteq lfp(f)$ et $<x^\delta : \delta \in \mu(L)>$, $<y^\delta : \delta \in \mu(L)>$ et $<z^\delta : \delta \in \mu(L)>$ les itérations croissantes définies par f et partant respectivement de $\bot$, d et $lfp(f)$. Par récurrence transfinie il est immédiat que $\{\forall \delta \in \mu(L),\ x^\delta \sqsubseteq y^\delta \sqsubseteq z^\delta = lfp(f)\}$. Comme $<x^\delta : \delta \in \mu(L)>$ est stationnaire de limite $lfp(f)$, il s'en suit immédiatement que $<y^\delta : \delta \in \mu(L)>$ est stationnaire de limite $lfp(f)$. (On notera que $<y^\delta : \delta \in \mu(L)>$ n'est pas nécessairement une chaîne ascendante).

Fin de la preuve.

Par application du principe de dualité, il vient:

COROLLAIRE 2.5.3.0.3 *Treillis des pré-points fixes d'un opérateur monotone*

Soient $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet et $f \in mon(L \to L)$. L'ensemble des pré-points fixes de f est un treillis complet non vide:

$prefp(f)(\sqsubseteq, \bot, gfp(f), \sqcup, \lambda S.llis(\lambda z.z \sqcap f(z))(\sqcap S))$

où le plus grand point fixe de f est $gfp(f) = \sqcup\{x \in L : x \sqsubseteq f(x)\} = llis(f)(d)$ pour tout $d \in L$ tel que $gfp(f) \sqsubseteq d$.

2.5.4 Polynômes unaires monotones d'un treillis complet définis par une famille d'opérateurs monotones

Ce paragraphe est en aparté dans nos préliminaires de la version constructive du théorème de Tarski. Il s'agit de répondre à la question suivante

posée au chapitre 5: soient $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet et $\{f_i : i\in I\}$ une famille d'opérateurs monotones sur L. Quelle est la plus petite et la plus grande valeur de L que l'on peut calculer à partir d'un ensemble $S\subseteq L$ de valeurs initiales en appliquant $\sqcup$, $\sqcap$ et les f_i? La notion de calcul étant formalisée par un polynôme unaire de l'algèbre $\langle L;\sqcup,\sqcap,\{f_i : i\in I\}\rangle$ il s'agit de déterminer le plus petit et le plus grand polynôme de cette algèbre.

DEFINITION 2.5.4.0.1 *Polynômes unaires*

Soient $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet et $\{f_i : i\in I\}$ une famille d'éléments de $mon(L\to L)$. L'ensemble $\mathbb{P}$ des polynômes unaires de l'algèbre $\langle L;\sqcup,\sqcap,\{f_i : i\in I\}\rangle$ est le plus petit sous-ensemble de $(L\to L)$ tel que:

(a) - $(\lambda x.x)\in\mathbb{P}$

(b) - Si $P\in\mathbb{P}$ et $i\in I$ alors $(f_i\circ P)\in\mathbb{P}$

(c) - Si $\{P_\gamma : \gamma\in J\}\subseteq\mathbb{P}$ alors $(\bigsqcup_{\gamma\in J} P_\gamma)\in\mathbb{P}$ et $(\bigsqcap_{\gamma\in J} P_\gamma)\in\mathbb{P}$

Il découle immédiatement de cette définition que les polynômes unaires de $\langle L;\sqcup,\sqcap,\{f_i : i\in I\}\rangle$ sont des opérateurs monotones sur L.

THEOREME 2.5.4.0.2

Tout polynôme unaire de $\langle L;\sqcup,\sqcap,\{f_i : i\in I\}\rangle$ est inférieur ou égal à $\mathit{luis}(\lambda z.z\sqcup(\bigsqcup_{i\in I} f_i(z)))$ et supérieur ou égal à $\mathit{llis}(\lambda z.z\sqcap(\bigsqcap_{i\in I} f_i(z)))$.

Preuve: Posons $\overline{F} = \lambda z.z\sqcup(\bigsqcup_{i\in I} f_i(z))$ et $\underline{F} = \lambda z.z\sqcap(\bigsqcap_{i\in I} f_i(z))$. Nous savons que $\overline{F}$, $\underline{F}\in mon(L\to L)$. D'après le théorème 2.5.3.0.1 et son dual nous en déduisons que $\mathit{luis}(\overline{F})(x)$ et $\mathit{llis}(\underline{F})(x)$ sont définis pour tout x de L. La preuve du théorème se fait par induction structurale:

(a) - $\mathit{luis}(\overline{F})$ est extensive (th.2.5.2.0.5) et $\mathit{llis}(\underline{F})$ est réductive (th.2.5.2.0.6). Par conséquent pour tout x de L il vient $\mathit{llis}(\underline{F})(x)\sqsubseteq x\sqsubseteq\mathit{luis}(\overline{F})(x)$.

(b) - Soit P un polynôme unaire tel que $\mathit{llis}(\underline{F})\sqsubseteq P\sqsubseteq\mathit{luis}(\overline{F})$. Alors pour tout $i\in I$ il vient par monotonie $f_i\circ\mathit{llis}(\underline{F})\sqsubseteq f_i\circ P\sqsubseteq f_i\circ\mathit{luis}(\overline{F})$. Mais $\mathit{llis}(\underline{F}) = \underline{F}\circ\mathit{llis}(\underline{F})\sqsubseteq f_i\circ\mathit{llis}(\underline{F})\sqsubseteq f_i\circ P\sqsubseteq f_i\circ\mathit{luis}(\overline{F})\sqsubseteq\overline{F}\circ\mathit{luis}(\overline{F}) = \mathit{luis}(\overline{F})$.

(c) - Soit $\{P_\gamma : \gamma\in J\}$ une famille de polynômes unaires tels que $\mathit{llis}(\underline{F})\sqsubseteq P_\gamma\sqsubseteq\mathit{luis}(\overline{F})$ pour tout $\gamma\in J$. Par définition d'une borne supérieure nous avons $\mathit{llis}(\underline{F})\sqsubseteq\bigsqcup_{\gamma\in J} P_\gamma\sqsubseteq\mathit{luis}(\overline{F})$ et de même $\mathit{llis}(\underline{F})\sqsubseteq\bigsqcap_{\gamma\in J} P_\gamma\sqsubseteq\mathit{luis}(\overline{F})$.

Fin de la preuve.

2.5.5 Caractérisation constructive de l'ensemble des points fixes d'un opérateur monotone sur un treillis complet

THEOREME 2.5.5.0.1 *Version constructive du théorème de* Tarski[1955]

Soient $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet et $f \in mon(L \to L)$. L'ensemble $fp(f)$ des points fixes de f est un treillis complet:

$fp(f)(\sqsubseteq,\ lfp(f),\ gfp(f),\ \lambda S.luis(f)(\sqcup S),\ \lambda S.llis(f)(\sqcap S))$

Preuve: D'après les théorèmes 2.5.2.0.2 et 2.5.2.0.5 $fp(f)$ est l'image de $prefp(f)$ par la fermeture supérieure $luis(f)$ et comme $prefp(f)$ est un treillis complet $(\sqsubseteq,\bot,\ gfp(f),\ \sqcup,\ \lambda S.llis(\lambda z.z\sqcap f(z))(\sqcap S))$ (théorème 2.5.3.0.3) le théorème 2.3.0.1 implique que $fp(f)$ est un treillis complet pour l'ordre partiel $\sqsubseteq$ dont l'infimum est $luis(f)(\bot) = lfp(f)$ et dont l'opération de borne supérieure est $\lambda S.luis(f)(\sqcup S)$. Par dualité, $fp(f)$ est l'image du treillis complet $postfp(f)(\sqsubseteq,\ lfp(f),\ \top,\ \lambda S.luis(\lambda z.z\sqcup f(z))(\sqcup S),\ \sqcap)$ (théorème 2.5.3.0.2) par la fermeture inférieure $llis(f)$ (théorème 2.5.2.0.6) et par conséquent (théorème 2.3.0.3) c'est un treillis complet pour l'ordre partiel $\sqsubseteq$ dont le supremum est $llis(f)(\bot) = gfp(f)$ est dont l'opération de borne inférieure est $\lambda S.llis(f)(\sqcap S)$.

Fin de la preuve.

COROLLAIRE 2.5.5.0.2

Soient $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet, $d \in L$ et $f \in mon(L \to L)$. $luis(f)\circ llis(\lambda z.z\sqcap f(z))(d)$ et $llis(f)\circ luis(\lambda z.z\sqcup f(z))(d)$ sont des points fixes de f plus grands ou égaux à n'importe quel point fixe de f qui est plus petit ou égal à d et plus petits ou égaux à n'importe quel point fixe de f qui est plus grand ou égal à d. De plus $luis(f)\circ llis(\lambda z.z\sqcap f(z))(d) \sqsubseteq llis(f)\circ luis(\lambda z.z\sqcup f(z))(d)$.

Preuve: Soient $a, b \in L$ tels que $f(a) = a \sqsubseteq d \sqsubseteq b = f(b)$. Alors par monotonie $a = luis(f)\circ llis(\lambda z.z\sqcap f(z))(a) \sqsubseteq luis(f)\circ llis(\lambda z.z\sqcap f(z))(d) \sqsubseteq luis(f)\circ llis(\lambda z.z\sqcap f(z))(b) = b$. De même $a \sqsubseteq llis(f)\circ luis(\lambda z.z\sqcup f(z))(d) \sqsubseteq b$.

Posons $p = llis(\lambda z.z\sqcap f(z))(d)$ et $q = luis(\lambda z.z\sqcup f(z))(d)$. Soit $S = \{x \in L : p \sqsubseteq x \sqsubseteq q\}$ c'est un sous-treillis complet de L avec infimum p et supremum q. Par le théorème 2.5.3.0.1 et son dual $p \sqsubseteq f(p)$ et $f(q) \sqsubseteq q$ et donc $f(S) \subseteq S$. Le théorème 2.5.5.0.1 implique que le plus petit point fixe de la

restriction de f à S est $luis(f)(p)$ tandis que le plus grand point fixe de la restriction de f à S est $llis(f)(q)$ ce qui montre que $luis(f)\circ llis(\lambda z.z\sqcap f(z))(d) \leq llis(f)\circ luis(\lambda z.z\sqcup f(z))(d)$.
Fin de la preuve.

DEFINITION 2.5.5.0.3 *Préfermeture d'un treillis complet* (Ladegaillerie[1973])

Une *préfermeture inférieure* $\underline{\rho}$ d'un treillis complet $L(\leq,\perp,\top,\sqcup,\sqcap)$ est un opérateur de L, monotone, idempotent et satisfaisant *l'axiome de connectivité inférieure* $\{\forall x\in L,\ \underline{\rho}(x\sqcap\underline{\rho}(x)) = \underline{\rho}(x)\}$.

Dualement une *préfermeture supérieure* $\overline{\rho}$ d'un treillis complet L est un opérateur de L, monotone, idempotent et satisfaisant *l'axiome de connectivité supérieure* $\{\forall x\in L,\ \overline{\rho}(x\sqcup\overline{\rho}(x)) = \overline{\rho}(x)\}$.

COROLLAIRE 2.5.5.0.4

Soient $L(\leq,\perp,\top,\sqcup,\sqcap)$ et $f\in mon(L\to L)$. L'ensemble $fp(f)$ des points fixes de F est l'image de L par la préfermeture inférieure $luis(f)\circ llis(\lambda z.z\sqcap f(z))$ et l'image de L par la préfermeture supérieure $llis(f)\circ luis(\lambda z.z\sqcup f(z))$.

Preuve: $luis(F)\circ llis(\lambda z.z\sqcap f(z))$ est une préfermeture inférieure de L car c'est la composition d'une fermeture supérieure $luis(f)$ et d'une fermeture inférieure $llis(\lambda z.z\sqcap f(z))$. (2.5.2.0.5, 2.5.3.0.1, 2.5.2.0.6 et Ladegaillerie [1973, prop.5, p.41].
Fin de la preuve.

PROPOSITION 2.5.5.0.5 Park[1969]

Soit $L(\leq,\perp,\top,\sqcup,\sqcap,\neg)$ un treillis booléen complet et $f\in mon(L\to L)$. Alors $\lambda x.\neg f(\neg x)\in mon(L\to L)$, $gfp(f) = \neg lfp(\lambda x.\neg f(\neg x))$, $lfp(\lambda x.\neg f(\neg x))\sqcap lfp(f) = \perp$, $\{\{lfp(\lambda x.\neg f(\neg x))\sqcup lfp(f)=\top\} \Longleftrightarrow \{lfp(f)=gfp(f)\}\}$.

2.5.6 Non calculabilité des points fixes d'un opérateur monotone sur un treillis complet

Nous montrons qu'en général les points fixes extrêmes d'un opérateur monotone arbitraire sur un treillis complet arbitraire ne sont pas calculables. Le résultat indique une difficulté et non une impossibilité. Il n'exclut pas les cas particuliers où les points fixes sont calculables (par exemple quand le treillis satisfait aux conditions de chaîne). Il indique toutefois qu'en général nous devrons nous contenter de calculer des approximations des points fixes extrêmes (que ce soit parcequ'ils ne sont pas calculables ou plus simplement parce que le calcul est trop complexe).

PROPOSITION 2.5.6.0.1

Soient $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet arbitraire et f un opérateur monotone arbitraire de L. Le problème du calcul de $lfp(f)$ et $gfp(f)$ est indécidable.

Preuve: Nous montrons que si l'on pouvait calculer $lfp(f)$ pour tous L et $f \in mon(L \to L)$ arbitraires, nous pourrions résoudre le problème dit "problème de correspondance de Post modifié" qui est indécidable (Hopcroft & Ullman[1969]). Ce problème est le suivant: "étant données deux suites arbitraires $A=A_1,\ldots,A_k$ et $B=B_1,\ldots,B_k$ de k chaînes non vides sur un alphabet comportant au moins deux caractères distincts, est-ce qu'il existe une suite $i_1,\ldots,i_n$ d'entiers telle que $A_1,A_{i_1},\ldots,A_{i_n} = B_1,B_{i_1},\ldots,B_{i_n}$ ".

Soient $C=\{1,\ldots,k\}$ et C^* l'ensemble des chaînes éventuellement vides de l'alphabet C. Soit $C^{1*} = \{1s : s\in C^*\}$. Soit $P(C^{1*})$ l'ensemble de toutes les parties de C^{1*}. Soit $F = \lambda E.(\{\{1\}\} \cup \{\bigcup_{i=1}^{k} \{si : s\in x\} : x\in E\})$. C'est un opérateur monotone de $P(C^{1*})$. Considérons la séquence $E^0 = \{1\}$, $E^\delta = F(E^{\delta-1})$ si δ est un ordinal successeur, et $E^\delta = \bigcup_{\alpha<\delta} E^\alpha$ si δ est un ordinal limite. C'est une chaîne ascendante pour l'ordre $\sqsubseteq$ dont la limite est $L=lfp(F)$ car $\{1\} \sqsubseteq lfp(F)$. Définissons un ordre partiel sur L par $\{\forall x, y\in L, \{x \sqsubseteq y\} \iff \{\{x=y\}$ ou $\{\exists\delta\in\mu(P(C^{1*})) : (x\in E^\delta)$ et $(y \notin E^\delta)\}\}\}$. La relation $\sqsubseteq$ est trivialement réflexive. Supposons qu'il existe $x, y\in L$ tels que $x \sqsubseteq y$, $y \sqsubseteq x$ et $x\neq y$. Alors $\exists\delta,\eta$: $x\in E^\delta$, $y \notin E^\delta$ et de plus $y\in E^\eta$, $x \notin E^\eta$. Il est impossible que $\eta=\delta$. Si $\delta<\eta$ alors $x\in E^\delta \sqsubseteq E^\eta$ et $x \notin E^\eta$ contradictoire, sinon $\eta<\delta$ et $y\in E^\eta \sqsubseteq E^\delta$ contradictoire avec $y \notin E^\delta$. Par l'absurde $\sqsubseteq$ est antisymétrique. Supposons $x \sqsubseteq y$ et $y \sqsubseteq z$. Si $x=y$

ou y=z, $\subseteq$ est trivialement transitive sinon $\exists\delta,\eta$ tels que $x\in E^{\delta}$, $y\notin E^{\delta}$, $y\in E^{\eta}$ et $z\notin E^{\eta}$. On a forcément $\delta\neq\eta$. Si $\delta<\eta$ alors $E^{\delta}\subseteq E^{\eta}$ et $x\in E^{\eta}$ et $z\notin E^{\eta}$ donc $x\subseteq z$ sinon $\eta<\delta$ et $y\in E^{\eta}\subseteq E^{\delta}$ contradictoire avec $y\notin E^{\delta}$. Nous concluons que $\subseteq$ est un ordre partiel pour L. Supposons $x\neq y$ et $x\not\subseteq y$ et $y\not\subseteq x$. Alors $\forall\delta$, $x\notin E^{\delta}$ ou $y\in E^{\delta}$ et $\forall\eta$, $y\notin E^{\eta}$ et $x\in E^{\eta}$ en particulier pour $\delta=\eta$ nous obtenons une contradiction qui montre que $\subseteq$ est un ordre total. L est une chaîne avec infimum {1}. De plus L a un supremum τ car sinon la chaîne $\langle E^{\delta}:\delta\in\mu(\mathcal{P}(C^{1*}))\rangle$ ne serait pas stationnaire. En effet, supposons $\forall y\in L$, $\exists x\in L$, $x\not\subseteq y$. Alors $x\neq y$ et $\forall\delta$ on a $x\notin E^{\delta}$ et $y\in E^{\delta}$. En particulier pour la limite $L=E^{\varepsilon}$ on a $x\notin L$ ce qui est contradictoire. Par l'absurde $\exists\tau\in L$: $\forall x\in L$, $x\subseteq\tau$. Donc L est un treillis complet.

Considérons maintenant l'opérateur f de L défini par $\lambda x.$ si $\{\exists s\in x$: $s=1i_1\ldots i_n$ et $A_1,A_{i_1},\ldots A_{i_n}=B_1,B_{i_1},\ldots,B_{i_n}\}$ alors x sinon $\bigcup_{i=1}^{k}\{si:s\in x\}$ finsi.

f est monotone donc admet un plus petit point fixe. De plus le problème de correspondance de Post modifié a une solution si et seulement si le plus petit point fixe de f est différent de τ. Si $lfp(f)$ était calculable, ce problème ne serait pas indécidable.

Fin de la preuve.

2.6 TREILLIS COMPLET DES OPERATEURS CONTINUS SUPERIEUREMENT D'UN TREILLIS COMPLET

DEFINITION 2.6.0.1 *Continuité*

Soit ω le premier ordinal limite infini.

Nous dirons qu'un opérateur f de $L(\subseteq,\perp,\tau,\sqcup,\sqcap)$ est *continu supérieurement* si et seulement si pour toute chaîne ascendante $\langle x^{\delta}:\delta\in\omega\rangle$ nous avons $f(\bigsqcup_{\delta\in\omega}x^{\delta})=\bigsqcup_{\delta\in\omega}f(x^{\delta})$.

Nous dirons qu'un opérateur f de $L(\subseteq,\perp,\tau,\sqcup,\sqcap)$ est *continu inférieurement* si et seulement si pour toute chaîne descendante $\langle x^{\delta}:\delta\in\omega\rangle$ nous avons $f(\bigsqcap_{\delta\in\omega}x^{\delta})=\bigsqcap_{\delta\in\omega}f(x^{\delta})$.

Un opérateur f de L est *continu* s'il est continu supérieurement et inférieurement.

Le terme continuité est justifié par le fait que la définition 2.6.0.1.

est équivalente à la notion topologique de continuité pour une topologie de L convenablement choisie (il faut pour cela supposer des hypothèses supplémentaires sur L, voir Scott[1972], Cousot & Cousot[1977d]).

Noter que si f est un opérateur monotone de L alors $f(\bigsqcup_{\delta\in\mu} x^\delta) = \bigsqcup_{\delta\in\mu} f(x^\delta)$ et $f(\bigsqcap_{\delta\in\mu} x^\delta) = \bigsqcap_{\delta\in\mu} f(x^\delta)$ est vrai pour des chaînes $\langle x^\delta : \delta\in\mu\rangle$ respectivement ascendantes et descendantes finies (c'est-à-dire quand $\mu<\omega$). Dans un treillis satisfaisant la condition de chaîne ascendante (respectivement descendante) toute chaîne infinie est stationnaire et par conséquent tout opérateur monotone est continu supérieurement (respectivement inférieurement). Noter également que la continuité supérieure ou inférieure entraîne la monotonie. Enfin si un opérateur est un morphisme complet pour l'union (définition 2.3.0.2) (respectivement pour l'intersection) il est continu supérieurement (respectivement inférieurement).

Nous caractérisons maintenant l'ensemble des opérateurs continus supérieurement d'un treillis complet L comme image de $(L\to L)$ par une fermeture inférieure *conts*. Contrairement à ce qui a été fait pour les opérateurs monotones (paragraphe 2.4) nous n'avons pas pu trouver de définition de *conts* qui soit suffisamment simple pour être utile en pratique. Nous nous contentons donc de montrer l'existence de *conts*.

THEOREME 2.6.0.2

Soit $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet alors il existe une fermeture inférieure *conts* de $(L\to L)$ telle que $conts(L\to L)$ soit l'ensemble des opérateurs continus supérieurement de L. De plus, pour tout f de $(L\to L)$, $conts(f)$ est le plus grand opérateur continu supérieurement de L plus petit ou égal à f.

Preuve : Soit C la partie du treillis complet $(L\to L)$ correspondant aux opérateurs continus supérieurement de L. Alors $(\lambda.\bot)\in C$ et pour toute partie non vide S de C nous avons $(\bigsqcup S)\in C$. En effet, soit $\langle x^\delta : \delta\in\omega\rangle$ une chaîne ascendante de L. Pour tout f de S nous avons $f(\bigsqcup_{\delta\in\omega} x^\delta) = \bigsqcup_{\delta\in\omega} f(x^\delta)$. Donc $\bigsqcup_{f\in S} f(\bigsqcup_{\delta\in\omega} x^\delta) = \bigsqcup_{f\in S}\bigsqcup_{\delta\in\omega} f(x^\delta) = \bigsqcup_{\delta\in\omega}\bigsqcup_{f\in S} f(x^\delta)$ c'est-à-dire $(\bigsqcup S)(\bigsqcup_{\delta\in\omega} x^\delta) = \bigsqcup_{\delta\in\omega}(\bigsqcup S)(x^\delta)$.

Soit *conts* l'application de $(L\to L)$ dans lui-même définie par :

$$conts = \lambda f.\bigsqcup\{g\in(L\to L) : f\sqsupseteq g\}$$

alors *conts* est une fermeture inférieure dont les éléments fermés sont les éléments de C. Pour tout $f \in (L \to L)$, *conts*(f) est le plus grand élément de C plus petit ou égal à f car si $g \in conts(L \to L)$ et $f \sqsupseteq g$ alors $conts(f) \sqsupseteq conts(g) = g$.

Fin de la preuve.

2.7 THEOREME DE POINTS FIXES POUR LES OPERATEURS CONTINUS SUPERIEUREMENT SUR UN TREILLIS COMPLET

Soit $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet tel que $\mu(L)>\omega$. Une condition nécessaire et suffisante pour qu'une itération croissante $<x^\delta:\delta \in \mu(L)>$ définie par $f \in (L \to L)$ et partant de $d \in L$ qui est une chaîne ascendante soit stationnaire après ω termes est:

$$\{\forall \delta \in \mu(L),\ \{\delta \geq \omega\} \Rightarrow \{x^\omega = x^\delta\}\}$$

$$\Leftrightarrow \{x^\omega = x^{\omega+1}\} \qquad \text{(lemme 2.5.1.0.4)}$$

$$\Leftrightarrow \{x^\omega = f(x^\omega)\} \qquad \text{(définition 2.5.1.0.1(b))}$$

$$\Leftrightarrow \{\bigsqcup_{\alpha<\omega} x^\alpha = f(\bigsqcup_{\alpha<\omega} x^\alpha)\} \qquad \text{(définition 2.5.1.0.1(c))}$$

$$\Leftrightarrow \{\bigsqcup_{\alpha<\omega} x^{\alpha+1} = f(\bigsqcup_{\alpha<\omega} x^\alpha)\} \qquad \text{(car } \{\alpha<\omega\} \Leftrightarrow \{(\alpha+1)<\omega\}\text{)}$$

$$\Leftrightarrow \{\bigsqcup_{\alpha<\omega} f(x^\alpha) = f(\bigsqcup_{\alpha<\omega} x^\alpha)\} \qquad \text{(définition 2.5.1.0.1.(b))}$$

On voit donc que l'hypothèse de continuité supérieure est "faite pour éviter d'avoir à considérer des itérations transfinies (du moins comportant strictement plus de ω termes). En fait, elle est même un peu forte puisqu'il n'est pas exigé que la chaîne $<x^\delta:\delta\epsilon\omega>$ satisfasse à la définition 2.5.1.0.1. En appliquant le théorème 2.5.2.0.2 il vient:

PROPOSITION 2.7.0.1

Soient $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet et $f \in conts(L \to L)$. Une itération croissante $<x^\delta:\delta \in min(\mu(L),\omega)>$ partant de $d \in prefp(f)$ et définie par est une chaîne ascendante stationnaire, sa limite *luis*(f)(d) est le plus petit des points fixes de f plus grands ou égaux à d.

2.8 RESOLUTION FORMELLE D'UN SYSTEME D'EQUATIONS MONOTONES DE POINTS FIXES PAR ELIMINATION DE VARIABLES

Soit $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet alors l'ensemble L^n des n-uplets d'éléments de L est un treillis complet pour l'ordre:

$$\{(X_1,\ldots,X_n) \sqsubseteq (Y_1,\ldots,Y_n)\} \Longleftrightarrow \{\forall i\in[1,n],\ X_i \sqsubseteq Y_i\}$$

Alors, il vient:

$$\bigsqcup_{i\in\Delta}(X_1^i,\ldots,X_n^i) = (\bigsqcup_{i\in\Delta}X_1^i,\ldots,\bigsqcup_{i\in\Delta}X_n^i)$$

$$\bigsqcap_{i\in\Delta}(X_1^i,\ldots,X_n^i) = (\bigsqcap_{i\in\Delta}X_1^i,\ldots,\bigsqcap_{i\in\Delta}X_n^i)$$

de sorte que l'infimum de L^n est $(\bot,\ldots,\bot)$ et le supremum est $(\top,\ldots,\top)$. Par abus de notation, nous écrirons $L^n(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$.

Soit un système d'équations à n inconnues $(X_1,\ldots,X_n)\in L^n$ de la forme:

$$\begin{cases} X_1 = F_1(X_1,\ldots,X_n) \\ \ldots \\ X_n = F_n(X_1,\ldots,X_n) \end{cases}$$

où $\{\forall i\in[1,n],\ F_i \in (L^n \to L)$ que nous abrégerons par $X=F(X)$ où $F\in(L^n\to L^n)$. Si les F_i $(i=1,\ldots,n)$ sont monotones alors F est monotone et le théorème 2.5.5.0.1 donne une définition constructive des solutions de ce système d'équations.

Plus généralement à une équation $X=F(X)$ sur un treillis $\bar{L}$ nous pouvons faire correspondre un système d'équations sur une décomposition directe de L ayant mêmes solutions à un isomorphisme près:

PROPOSITION 2.8.0.1

Soient $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ et $\forall i\in[1,n]$, $M_i(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ des treillis complets tels que $\prod_{i=1}^{n} M_i$ est une *décomposition directe* de L (c'est-à-dire qu'il existe des morphismes de décomposition $\sigma_i \in (L\to M_i)$, tels que $\sigma=\lambda x.(\sigma_1(x),\ldots,\sigma_n(x))$ est un isomorphisme complet de L sur $\prod_{i=1}^{n} M_i$). Soit $F\in mon(L\to L)$ alors l'ensemble $fx(F)$ des points fixes de F et l'ensemble des solutions du système d'équations:

$$\begin{cases} X_i = \sigma_i\circ F\circ\sigma^{-1}(X_1,\ldots,X_n) \\ i = 1,\ldots,n \end{cases}$$

sont complètement isomorphes par σ.

Outre la méthode itérative, un calcul formel procédant par élimination de variables permet également de résoudre le système d'équations. La variable X_i est éliminée en résolvant l'équation $X_i=(\lambda Y.F_i(X_1,\ldots,Y,\ldots,X_n))(X_i)$ en terme des X_j $(j\neq i)$ considérés comme des variables libres, puis en substituant la solution à toutes les occurrences de X_i dans les équations restantes. Le processus est répété jusqu'à l'élimination de toutes les variables.

PROPOSITION 2.8.0.2 Bekič[1969], Leszczylowski[1971]

Soient $F \in mon(L^{n+m} \to L^n)$ et $G \in mon(L^{n+m} \to L^m)$. Nous noterons $(X,Y) = (X_1,\ldots,X_n,Y_1,\ldots,Y_m)$ quand $X \in L^n$ et $Y \in L^m$. Considérons le système d'équations,

$$(1)\begin{cases} X = F(X,Y) \\ Y = G(X,Y) \end{cases}$$

le *résolvant* $R = \lambda Y.\mathit{lfp}(\lambda X.F(X,Y))$ et le système d'équations,

$$(2)\begin{cases} X = R(Y) \\ Y = G(R(Y),Y) \end{cases}$$

Nous notons $\mathit{fp}(i)$ et $\mathit{lfp}(i)$, $(i=1,2)$ respectivement l'ensemble des solutions du système d'équations (i) et sa plus petite solution. Alors $\mathit{fp}(2) \subseteq \mathit{fp}(1)$ et $\mathit{lfp}(1) = \mathit{lfp}(2)$.

Preuve: Si $Y, Z \in L^m$ alors $\{Y \subseteq Z\} \Rightarrow \{(X,Y) \subseteq (X,Z)\} \Rightarrow \{\lambda X.F(X,Y) \subseteq \lambda X.F(X,Z)\}$ $\Rightarrow \{\mathit{lfp}(\lambda X.F(X,Y)) \subseteq \mathit{lfp}(\lambda X.F(X,Z))\} \Rightarrow \{R(Y) \subseteq R(Z)\} \Rightarrow \{R \in mon(L^m \to L^n)\}$ donc $\mathit{fp}(2)$ n'est pas vide.
Soit $(A_2,B_2) \in \mathit{fp}(2)$ un point fixe de (2). Alors $A_2=R(B_2)$ c'est-à-dire $\mathit{lfp}(\lambda X.F(X,B_2))=A_2$ donc $F(A_2,B_2)=A_2$ et $B_2=G(R(B_2),B_2)$ soit $B_2=G(A_2,B_2)$ ce qui implique que (A_2,B_2) est solution du système d'équations (1) c'est-à-dire $\mathit{fp}(2) \subseteq \mathit{fp}(1)$.

Pour montrer qu'en général $\mathit{fp}(2) \neq \mathit{fp}(1)$, considérons $L=\{\bot,\top\}$, $F=\lambda(X,Y).[X \sqcap Y]$ et $G=\lambda(X,Y).[X \sqcup Y]$. Le résolvant est $R=\lambda Y.\mathit{lfp}(\lambda X.F(X,Y)) = \lambda Y.\mathit{lfp}(\lambda X.[X \sqcap Y]) = \lambda Y.\bot$. Le système d'équations (1) admet la solution $(\top,\top)$ qui n'est pas une solution de (2).

Comme $\mathit{lfp}(2) \in \mathit{fp}(1)$ on a $\mathit{lfp}(1) \subseteq \mathit{lfp}(2)$. Soit (A_1,B_1) un point fixe de (1), on a $F(A_1,B_1)=A_1$ donc $F(A_1,B_1) \subseteq A_1$ donc A_1 est un post point fixe de $\lambda X.F(X,B_1)$ ce qui implique que $\mathit{lfp}(\lambda X.F(X,B_1)) \subseteq A_1$ soit $R(B_1) \subseteq A_1$. Comme $(R(B_1),B_1) \subseteq (A_1,B_1)$ et G est monotone $G(R(B_1),B_1) \subseteq G(A_1,B_1) \subseteq B_1$ car (A_1,B_1) est post point fixe de (1). On en déduit que (A_1,B_1) est post point fixe de (2) ce qui implique $\mathit{lfp}(2) \subseteq (A_1,B_1)$ soit en particulier $\mathit{lfp}(2) \subseteq \mathit{lfp}(1)$ et par antisymétrie $\mathit{lfp}(2) = \mathit{lfp}(1)$.
Fin de la preuve.

2.9 METHODES ITERATIVES CHAOTIQUES, ASYNCHRONES ET ASYNCHRONES AVEC MEMOIRE POUR RESOUDRE UN SYSTEME D'EQUATIONS MONOTONES DE POINTS FIXES SUR UN TREILLIS COMPLET

L'itération croissante partant de $D \in L^n$ et définie par $F \in mon(L^n \to L^n)$ (définition 2.5.1.0.1) se détaille en :

- $X^0 = D$
- $\begin{cases} X_i^\delta = F_i(X_1^{\delta-1},\ldots,X_n^{\delta-1}) & \text{si } \delta \text{ est un ordinal successeur} \\ i = 1,\ldots,n \end{cases}$
- $X^\delta = \bigsqcup_{\alpha<\delta} X^\alpha$ si δ est un ordinal limite

qui est en fait la méthode des approximations successives. On peut penser que la méthode de Gauss-Seidel :

$$\begin{cases} X_1^\delta = F_1(X_1^{\delta-1},\ldots,X_n^{\delta-1}) \\ \ldots \\ X_i^\delta = F_i(X_1^\delta,\ldots,X_{i-1}^\delta,X_i^{\delta-1},\ldots,X_n^{\delta-1}) \\ \ldots \\ X_n^\delta = F_n(X_1^\delta,\ldots,X_{n-1}^\delta,X_n^{\delta-1}) \end{cases}$$

qui consiste à reinjecter constamment dans le calcul les derniers résultats du calcul lui-même, renforce la convergence tout en réduisant l'encombrement mémoire. Robert[1976a] montre qu'en général ces méthodes ne sont pas équivalentes. Sans hypothèses suffisamment fortes sur L^n et F il arrive que l'une des méthodes des approximations successives et Gauss-Seidel converge tandis que l'autre diverge.

Nous allons montrer que lorsque L est un treillis complet et $F \in mon(L^n \to L^n)$ ce phénomène est impossible, c'est-à-dire que les deux stratégies conduisent au même résultat. D'une manière plus générale, nous montrons au paragraphe 2.9.1, que l'hypothèse de monotonie permet d'utiliser n'importe quelle *stratégie chaotique*, c'est-à-dire que dans l'itération on peut aléatoirement déterminer à chaque pas quelles sont les composantes qui sont calculées en fonction de l'itéré précédent (à condition de ne jamais en oublier une définitivement) et obtenir la convergence vers le même point fixe qu'avec la méthode des approximations successives.

Au paragraphe 2.9.2 ce résultat est généralisé au cas de *méthodes itératives asynchrones* qui offrent une possibilité de retard dans l'accès aux

itérés précédemment calculés et correspondent donc à l'utilisation de plusieurs processeurs qui travaillent en parallèle sans (nécessairement) être synchronisés.

En fait, les itérations chaotiques et asynchrones ne sont que des cas particuliers des *méthodes itératives asynchrones avec mémoire* considérées paragraphe 2.9.3. En plus de la décomposition des calculs par groupes de composantes évoluant en parallèle, les méthodes itératives asynchrones avec mémoire permettent de décomposer les calculs selon la structure de chacune des composantes puisque différentes valeurs de la même variable peuvent être utilisées dans le calcul d'une composante.

En pratique, on considère des méthodes itératives qui convergent en un nombre fini de pas et pourtant nous démontrons tous nos résultats de convergence pour des itérations transfinies. D'un point de vue abstrait, ceci permet de séparer le problème de terminaison des itérations (nombre d'itérés considérés) de celui de la convergence (stabilisation des itérés). (Par exemple, pour trouver X tel que $X \in \mathit{lfp}(F)$, on peut calculer l'itération croissante $\langle X^\delta : \delta \in \mu(L) \rangle$ partant de $\perp$ et définie par F en arrêtant le calcul après un nombre de pas fixé à l'avance ou quand la précision est suffisante. L'algorithme est correct car d'un point de vue abstrait tout terme X^δ est inférieur à la limite $\mathit{luis}(F)(\perp) = \mathit{lfp}(F)$ de l'itération transfinie dont seulement les premiers termes ont été effectivement calculés).

2.9.1 Convergence d'itérations chaotiques

DEFINITION 2.9.1.0.1 *Itération chaotique croissante*

. Soient $L(\sqsubseteq, \perp, \top, \sqcup, \sqcap)$ un treillis complet, n un entier strictement positif et $F \in \mathit{mon}(L^n \to L^n)$.

. Soit $\langle J^\delta : \delta \in \mathrm{Ord} \rangle$ une séquence de parties de $\{1,\ldots,n\}$ telle que:

(a) - $\{\forall \delta \in \mathrm{Ord},\ \forall i \in \{1,\ldots,n\},\ \exists \alpha \geq \delta : i \in J^\alpha\}$

. *L'itération chaotique croissante partant de* $D \in L^n$ *et définie par* F *et* $\langle J^\delta : \delta \in \mathrm{Ord} \rangle$ est une séquence $\langle X^\delta : \delta \in \mu(L) \rangle$ d'éléments de L^n définie par récurrence transfinie comme suit:

(b) - $X^0 = D$

(c) - $X_i^\delta = X_i^{\delta-1}$ pour tout ordinal successeur δ et tout $i \notin J^\delta$

(d) - $X_i^{\delta} = F_i(X^{\delta-1})$ pour tout ordinal successeur δ et tout $i \in J^{\delta}$

(e) - $X_i^{\delta} = \bigsqcup_{\alpha<\delta} X_i^{\alpha}$ pour tout ordinal limite δ

Cette définition s'interprète en considérant qu'au pas δ du calcul seules les composantes $i \in J^{\delta}$ évoluent en fonction des valeurs obtenues au pas précédent. La condition 2.9.1.0.1(a) impose qu'aucune composante ne soit définitivement oubliée. Le cas abstrait 2.9.1.0.1.(e) a été ajouté à la définition classique pour que la définition soit compatible avec 2.5.1.0.1.

Par exemple, la méthode des approximations successives consiste à prendre $\{\forall \delta \in Ord,\ J^{\delta}=\{1,\ldots,n\}\}$ tandis que la méthode de Gauss-Seidel correspond au choix $J^{\delta}=\{1\}$ si $\delta=1$ où δ est successeur d'un ordinal limite et $J^{\delta}=\{1+(j \underline{\text{modulo}}\ n)\}$ si δ est un ordinal successeur et $J^{\delta-1}=\{j\}$.

THEOREME 2.9.1.0.2

Une itération chaotique croissante partant de $D \in prefp(F)$ et définie par $F \in mon(L^n \to L^n)$ et $\langle J^{\delta}:\delta \in Ord\rangle$ est une chaîne ascendante stationnaire dont la limite est $luis(F)(D)$.

Preuve: C'est un cas particulier du théorème 2.9.2.0.2. Pour montrer en plus que c'est une chaîne ascendante voir Cousot & Cousot[1977e].
Fin de la preuve.

Comme tous les composants $X_i^{\delta+1}$ (tels que $i \in J^{\delta+1}$) sont évalués en fonction de X^{δ}, la définition 2.9.1.0.1 suppose implicitement que les calculs sont faits par un seul processus de calcul séquentiel ou par plusieurs processus parallèles (par exemple un pour chaque $i \in J^{\delta+1}$) synchronisés. La définition des itérations asynchrones évite cette contrainte de synchronisation en offrant une possibilité de retard dans l'accès aux itérés précédemment calculés.

2.9.2 Convergence d'itérations asynchrones

DEFINITION 2.9.2.0.1 *Itération asynchrone croissante*

. Soit $\langle J^{\delta}:\delta \in Ord\rangle$ une séquence d'éléments de $\{1,\ldots,n\}$ telle que:
(a) - $\{\forall \delta \in Ord,\ \forall i \in \{1,\ldots,n\},\ \exists \alpha \geq \delta : i=J^{\alpha}\}$

. Soit $<S^{\delta}:\delta \in Ord>$ une séquence d'éléments de Ord^n telle que:

(b) - $\{\forall i\in\{1,\ldots,n\}, \forall\delta \in Ord, S_i^{\delta}<\delta\}$

(c) - $\{\forall\delta \in Ord, \forall i\in\{1,\ldots,n\}, \exists\beta\geq\delta : \{\forall\alpha\geq\beta, \delta\leq S_i^{\alpha}\}\}$

(d) - $\{\forall\beta,\delta \in Ord, \{\beta$ est un ordinal limite <u>et</u> $\beta<\delta\}\} \Rightarrow \{\forall i\in\{1,\ldots,n\}, \beta$

. Soit F un opérateur monotone du treillis complet L^n. *Une itération asynchrone croissante pour* F *partant de* $D\in L^n$ *et définie par* $<J^{\delta}:\delta\in Ord>$ *et* $<S^{\delta}:\delta \in Ord>$ est une séquence $<X^{\delta}:\delta \in Ord>$ d'éléments de L^n définie par currence transfinie comme suit:

(e) - $X^0 = D$

(f) - $X_i^{\delta} = X_i^{\delta-1}$ pour tout ordinal successeur δ et tout $i\neq J^{\delta}$

(g) - $X_i^{\delta} = F_i(X_1^{S_1^{\delta}},\ldots,X_n^{S_n^{\delta}})$ pour tout ordinal successeur δ et $i=J^{\delta}$

(h) - $X_i^{\delta} = \bigcup_{\alpha<\delta} X^{\alpha}$ pour tout ordinal limite δ

La définition des itérations asynchrones est due aux analystes numé ciens Chazan & Miranker[1969] mais la forme donnée ici est plus proche d version de Baudet[1976]. Nous avons cependant simplifié cette dernière nition en supposant que $\forall\delta \in Ord$, $J^{\delta}\in\{1,\ldots,n\}$ et non pas $J^{\delta}\subseteq\{1,\ldots,n\}$. n'apporte aucune restriction, en effet, quand plusieurs composants sont à jour en même temps, nous considérons d'un point de vue *abstrait* qu'ils été calculés de la même façon mais mis à jour à des instants différents. (Par exemple la méthode des approximations successives correspond au cho $J^{\alpha+jn+i}=i$ et $S_k^{\alpha+jn+i}=\alpha+jn$ pour tous ordinal limite α et entiers $j\geq 0$, $1\leq$ et $1\leq k\leq n$. Plus généralement les itérations chaotiques sont un cas parti lier des itérations asynchrones). Nous avons également rajouté la règle 2.9.2.0.1.(h) ainsi que la condition 2.9.2.0.1.(d) pour que la notion d' ration asynchrone soit compatible avec les itérations transfinies 2.5.1.

Les itérations asynchrones offrent un modèle des possibilités de ca sur un multiprocesseur. Une mémoire globale initialisée par D est acces ble par chacun des processeurs qui peut lire et écrire chaque composant de la mémoire globale X. Ces opérations sont mutuellement exclusives dan le temps et peuvent donc être considérées comme instantanées. (Selon la taille des mémoires X_i $(i=1,\ldots,n)$ cette exclusion mutuelle est assurée p le matériel ou le logiciel, c'est la seule synchronisation élémentaire qu soit nécessaire entre les différents processeurs).

Interpréter $<\delta:\delta \in Ord>$ comme une séquence croissante d'instants δ où prennent place la lecture ou l'écriture d'un composant X_i. Quand un

processeur est oisif il est affecté à l'évaluation d'un composant quelconque du système d'équations. La définition 2.9.2.0.1. indique qu'au temps δ un processeur termine l'évaluation du ième composant tel que $i=J^{\delta}$. La valeur correspondante X_i^{δ} est instantanément écrite dans la mémoire X_i. L'évaluation de X_i^{δ} consiste à lire la valeur $X_1^{S_1^{\delta}}$ de la mémoire X_1 au temps S_1^{δ}, ..., à lire X_n au temps S_n^{δ}, à appliquer l'opération F_i aux arguments $X_1^{S_1^{\delta}},\ldots,X_n^{S_n^{\delta}}$ et à écrire la valeur correspondante $X_i^{\delta}=F_i(X_1^{S_1^{\delta}},\ldots,X_n^{S_n^{\delta}})$ au temps δ dans X_i. Tous les composants X_j de X pour lesquels $j\neq J^{\delta}$ ne sont pas modifiés au temps δ.

Noter qu'aucune synchronisation n'est nécessaire entre les processeurs contribuant au calcul et que la répartition des tâches des divers processeurs est libre. En particulier, un processeur qui tombe en panne peut être éliminé du pool des processeurs disponibles et remplacé dans sa tâche par les processeurs valides. La répartition des tâches des divers processeurs doit toutefois respecter la condition 2.9.2.0.1.(a) qui stipule qu'aucune composante ne peut être abandonnée définitivement, (cette hypothèse est un peu forte puisque par exemple une composante constante est toujours stable. Nous ne tenons pas compte de telles situations pour alléger le formalisme).

Il est également naturel de supposer que l'évaluation de chaque $F_i(X_1^{S_1^{\delta}},\ldots,X_n^{S_n^{\delta}})$ dure un temps fini (mais pas nécessairement uniformément borné). Donc pour tout δ, la durée du calcul $\max_{i=1}^{n}(\delta-S_i^{\delta})$ doit être finie. C'est ce que la condition 2.9.2.0.1.(c) exprime sous une forme un peu différente: pour tout δ il existe $\beta\geq\delta$ tel que après le temps β aucun processeur ne peut terminer un calcul commencé au temps δ.

L'hypothèse que l'évaluation élémentaire d'un composant dure un temps fini doit rester valable bien que nous considérions des itérations transfinies. La condition 2.9.2.0.1.(d) est alors nécessaire pour tenir compte de ce fait. Par l'absurde, supposons que le calcul de X_i^{δ} commencé au temps $\alpha=\max_{i=1}^{n}(S_i^{\delta})$ prend r unités de temps (où $r<\omega$ car la durée du calcul peut être arbitrairement longue mais finie). Nous avons $(\alpha+r)=\delta$. Supposons qu'il existe un ordinal limite β et $j\in\{1,\ldots,n\}$ tels que $S_j^{\delta}<\beta<\delta$. Alors $\alpha<\beta<\delta$ et comme β est un ordinal limite $(\alpha+1)<\beta$ puis par récurrence finie $(\alpha+r)<\beta<\delta$ en contradiction avec $(\alpha+r)=\delta$.

THEOREME 2.9.2.0.2

Soient $L(\leq,\perp,\top,\sqcup,\sqcap)$ un treillis complet, n un entier naturel et

$F \in mon(L^n \to L^n)$. Une itération asynchrone croissante pour F partant de $D \in prefp(F)$ et définie par $\langle J^\delta : \delta \in Ord \rangle$, $\langle S^\delta : \delta \in Ord \rangle$ est une séquence stationnaire de limite $luis(F)(D)$.

Preuve: Cas particulier du théorème 2.9.3.0.9. quand m=1.
Fin de la preuve.

2.9.3. Convergence d'itérations asynchrones avec mémoire.

DEFINITION 2.9.3.0.1 *Itération asynchrone croissante avec mémoire*

. Soit $\langle J^\delta : \delta \in Ord \rangle$ une séquence d'éléments de $\{1,\ldots,n\}$ telle que:

(a) - $\{\forall \delta \in Ord,\ \forall i \in \{1,\ldots,n\},\ \exists \alpha \geq \delta : i = J^\alpha\}$

. Soit $\langle S^\delta : \delta \in Ord \rangle$ une séquence d'éléments de $(Ord^n)^m$ telle que:

(b) - $\{\forall i \in \{1,\ldots,n\},\ \forall j \in \{1,\ldots,m\},\ \forall \delta \in Ord,\ (S^\delta_j)_i < \delta\}$

(c) - $\{\forall \delta \in Ord,\ \forall i \in \{1,\ldots,n\},\ \forall j \in \{1,\ldots,m\},\ \exists \beta \geq \delta : \{\forall \alpha \geq \beta : \delta \leq (S^\alpha_j)_i\}\}$

(d) - $\{\forall \beta, \delta \in Ord,\ \{\beta$ est un ordinal limite et $\beta < \delta\}$
$\Rightarrow \{\forall i \in \{1,\ldots,n\},\ \forall j \in \{1,\ldots,m\},\ \beta \leq (S^\delta_j)_i\}\}$

. Soient $L(\leq, \perp, \top, \sqcup, \sqcap)$ un treillis complet et F une application monotone de $(L^n)^m$ dans L^n. *Une itération asynchrone croissante avec* m *mémoires par de* $D \in L^n$ *et définie par* $\langle J^\delta : \delta \in Ord \rangle$ *et* $\langle S^\delta : \delta \in Ord \rangle$ est une séquence $\langle X^\delta : \delta \in Ord \rangle$ d'éléments de L^n définie par récurrence transfinie comme suit

(e) - $X^0 = D$

(f) - $X^\delta_i = X^{\delta-1}_i$ pour tout ordinal successeur δ et tout $i \neq J^\delta$

(g) - $X^\delta_i = F_i(Z^1,\ldots,Z^m)$ pour tout ordinal successeur δ et tout $i = J^\delta$
où $\forall j \in \{1,\ldots,m\},\ \forall i \in \{1,\ldots,n\},\ Z^j_i = X_i^{(S^\delta_j)_i}$

(h) - $X^\delta = \bigsqcup_{\alpha<\delta} X^\alpha$ pour tout ordinal limite δ

On notera que lorsque m=1 cette définition est équivalente à la définition 2.9.2.0.1.

Soit σ l'application de L^n dans $(L^n)^m$ telle que $\{\forall X \in L^n,\ \forall i \in \{1,\ldots,n$ $(\sigma(X))_i = X\}$. Soit F une application monotone de $(L^n)^m$ dans L^n. Nous définissons un point fixe de F comme étant un élément X de L^n tel que X=F(X, c'est-à-dire X=F(σ(X)). Comme $F \circ \sigma \in mon(L^n \to L^n)$ les résultats du paragraphe 2.5 peuvent être appliqués à F$\circ\sigma$.

Pour montrer une application pratique de la définition 2.9.3.0.1., supposons que l'on ait à calculer $luis(f)(D)$ où $f \in mon(L^n \to L^n)$ et $D \in prefp(f)$ et que l'on puisse trouver un entier naturel m et $F \in ((L^n)^m \to L^n)$ tel que $luis(f)(D) = luis(F\circ\sigma)(D)$. On peut alors utiliser n'importe quelle itération asynchrone croissante avec mémoire pour ce calcul car nous allons montrer que c'est une séquence stationnaire de limite $luis(F\circ\sigma)(D)$.

Soit par exemple $f=\lambda X(g(X) \sqcup h(X))$. Alors $luis(f)(D)=luis(F\circ\sigma)(D)$ où $F=\lambda(X,Y).(g(X) \sqcup h(Y))$. Une itération possible avec deux mémoires pour F est:

$$\begin{cases} X^0 = \bot \\ X^1 = \bot \\ X^{\delta+2} = F(X^{\delta+1}, X^{\delta}) \end{cases}$$

qui est équivalente aux deux itérations collatérales:

$$\begin{cases} X^0 = \bot \\ X^{\delta+1} = g(X^{\delta}) \sqcup Y^{\delta} \end{cases} \qquad \begin{cases} Y^0 = \bot \\ Y^{\delta+1} = h(X^{\delta}) \end{cases}$$

qui sont une décomposition naturelle des calculs qu'il n'est pas possible de décrire par la définition 2.9.2.0.1.

Dans le reste de ce paragraphe, nous considérons une itération asynchrone croissante avec m mémoires $\langle X^{\delta}:\delta \in Ord\rangle$ pour une application monotone F de $(L^n)^m$ dans L^n partant d'un pré-point fixe D de F ($D \sqsubseteq F\circ\sigma(D)$) et définie par $\langle J^{\delta}:\delta \in Ord\rangle$ et $\langle S^{\delta}:\delta \in Ord\rangle$ comme dans la définition 2.9.3.0.1.

LEMME 2.9.3.0.2

$\{\forall\delta \in Ord,\ D \sqsubseteq X^{\delta} \sqsubseteq luis(F\circ\sigma)(D)\}$

Preuve: Par récurrence transfinie sur δ en utilisant la propriété $D \sqsubseteq F\circ\sigma(D) \sqsubseteq luis(F\circ\sigma)(D) = F\circ\sigma(luis(F\circ\sigma)(D))$ (théorème 2.5.2.0.2). Si nécessaire les détails sont dans Cousot[1977d].
Fin de la preuve.

DEFINITION 2.9.3.0.3

La condition 2.9.3.0.1.(a) implique que pour tout $\delta \in Ord$, il y a un ordinal $\Pi(\delta)$ défini par:

$\Pi(\delta) = min\{\alpha \in Ord:\ (\delta \le \alpha)$ et $(\{1,\ldots,n\}=\{J^{\beta}:\delta \le \beta \le \alpha\})\}$

Intuitivement entre l'instant δ et l'instant $\Pi(\delta)+1$ toutes les

DEFINITION 2.9.3.0.7

Nous notons $\langle B^\delta : \delta \in Ord \rangle$ la séquence d'éléments de L^n définie par récurrence transfinie comme suit:

(a) - $B^0 = \emptyset$

(b) - $B^\delta = F(\sigma(B^{\delta-1}))$ si δ est un ordinal successeur

(c) - $B^\delta = \bigsqcup_{\alpha<\delta} B^\alpha$ si δ est un ordinal limite

LEMME 2.9.3.0.8

$\{\forall \beta,\ \delta \in Ord,\ \{\eta^\delta \leq \beta\} \Rightarrow \{B^\delta \subseteq X^\beta\}\}$

Preuve: Par récurrence transfinie sur δ (cas 1, 2, 3).

Cas 1: Si $\delta=0$ alors $\forall \beta \geq \eta^0 = 0$ nous avons $B^0 = \emptyset \subseteq X^\beta$ (lemme 2.9.3.0.2).

Cas 2: Supposant que δ est un ordinal successeur et que le lemme est vrai pour tout $\delta'<\delta$ nous montrons par récurrence transfinie sur β qu'il est également vrai pour δ (cas 2.1, 2.2, 2.3).

Cas 2.1: Si $\beta=\eta^\delta$ alors d'après 2.9.3.0.5.(b) $\eta^\delta = \Pi(\lambda(\eta^{\delta-1}))+1$. Alors pour tout $i=1,\ldots,n$ il y a un plus grand ordinal ε tel que $\lambda(\eta^{\delta-1}) \leq \varepsilon \leq \beta$ et $i=J^\varepsilon$. D'après 2.9.3.0.1 nous avons donc $X_i^\varepsilon = X_i^{\varepsilon+1} = \ldots = X_i^\beta$ avec $X_i^\varepsilon = F_i(Z^1,\ldots,Z^m)$. Comme $\varepsilon \geq \lambda(\eta^{\delta-1})$ on a $\{\forall j \in \{1,\ldots,m\},\ \forall k \in \{1,\ldots,n\},\ \eta^{\delta-1} \leq (S_j^\varepsilon)_k < \varepsilon\}$. Par hypothèse d'induction il suit que $B_k^{\delta-1} \subseteq X_k^{(S_j^\varepsilon)_k} = Z_k^j$. Nous en déduisons $\{\forall j \in \{1,\ldots,m\},\ B^{\delta-1} \subseteq Z^j\}$ donc par 2.9.3.0.7.(b) et monotonie, nous obtenons $B_i^\delta = F_i(\sigma(B^{\delta-1})) \subseteq F_i(Z^1,\ldots,Z^m) = X_i^\varepsilon = X_i^\beta$, soit $B_i^\delta \subseteq X_i^{\eta^\delta}$.

Cas 2.2: Supposons que β soit un ordinal successeur et que pour tout α tel que $\eta^\delta \leq \alpha < \beta$ nous ayons $B^\delta \subseteq X^\alpha$. Nous montrons que $\{\forall i \in \{1,\ldots,n\},\ B_i^\delta \subseteq X_i^\beta\}$. Si $i \neq J^\beta$ alors $B_i^\beta \subseteq X_i^{\beta-1} = X_i^\beta$ sinon $i=J^\beta$ et $X_i^\beta = F_i(Z^1,\ldots,Z^m)$. Comme $\beta > \eta^\delta > \Pi(\lambda(\eta^{\delta-1})) \geq \lambda(\eta^{\delta-1})$ nous savons que $\{\forall j \in \{1,\ldots,m\},\ \forall k \in \{1,\ldots,n\},\ \eta^{\delta-1} \leq (S_j^\delta)_k\}$ ce qui implique $B_k^{\delta-1} \subseteq X_k^{(S_j^\beta)_k}$ par hypothèse d'induction. Donc $\forall j \in \{1,\ldots,m\}$ nous avons $B^{\delta-1} \subseteq Z^j$ et par monotonie $B_i^\delta = F_i(\sigma(B_i^{\delta-1})) \subseteq F_i(Z^1,\ldots,Z^m) = X_i^\beta$.

Cas 2.3: Supposons que β soit un ordinal limite et que pour tout ordinal α tel que $\eta^\delta \leq \alpha < \beta$ nous ayons $B^\delta \subseteq X^\alpha$. Alors $B^\delta \subseteq \bigsqcup_{\eta^\alpha \leq \alpha < \beta} X^\alpha \subseteq \bigsqcup_{\alpha<\beta} X^\alpha = X^\beta$

Cas 3: Supposant que δ est un ordinal limite et que le lemme est vrai tout $\delta'<\delta$ nous montrons par récurrence transfinie sur β qu'il est égal vrai pour δ (cas 3.1, 3.2, 3.3).

Cas 3.1: Si $\beta=\eta^{\delta}$ alors η^{δ} est aussi un nombre limite (lemme 2.9.3. Par hypothèse d'induction sur δ nous avons $\forall\gamma<\delta$, $B^{\gamma}\subseteq X^{\eta^{\gamma}}$ et donc B^{δ} $\bigcup_{\gamma<\delta}B^{\gamma}\subseteq\bigcup_{\gamma<\delta}X^{\eta^{\gamma}}$. Comme $<\eta^{\gamma}:\gamma\in Ord>$ est strictement croissante $\{\{\gamma<\delta$ $\{\eta^{\gamma}<\eta^{\delta}\}\}$ et donc $\bigcup_{\gamma<\delta}X^{\eta^{\gamma}}\subseteq\bigcup_{\eta^{\gamma}<\eta^{\delta}}X^{\eta^{\gamma}}\subseteq\bigcup_{\alpha<\eta^{\delta}}X^{\alpha}=X^{\eta^{\delta}}$. Par transitivit $B^{\delta}\subseteq X^{\beta}$.

Cas 3.2: Supposant que β est un ordinal successeur et que pour tou nal α tel que $\eta^{\delta}\leq\alpha<\beta$ nous avons $B^{\delta}\subseteq X^{\alpha}$, nous montrons que $\forall i\in\{1,..$ $B_i^{\delta}\subseteq X_i^{\beta}\}$. Si $i\neq J^{\beta}$ alors $B_i^{\delta}\subseteq X_i^{\beta-1}=X_i^{\beta}$ sinon $i=J^{\beta}$ et $X_i^{\beta}=F_i(Z^1,..$ Comme $\eta^{\delta}<\beta$ et η^{δ} est un ordinal limite 2.9.3.0.1.(b) et 2.9.3.0.1.(pliquent que $\{\forall j\in\{1,...,m\}$, $\forall k\in\{1,...,n\}$, $\eta^{\delta}\leq(S_j^{\beta})_k<\beta\}$. Donc pa thèse d'induction $\{\forall j\in\{1,...,m\}$, $B_k^{\delta}\subseteq X_k^{(S_j^{\beta})_k}=Z^j\}$. Comme $<B^{\delta}:\delta\in O$ est une chaîne ascendante (théorème 2.5.2.0.2) nous obtenons par mo $B_i^{\delta}\subseteq F_i(\sigma(B^{\delta}))\subseteq F_i(Z^1,...,Z^m)=X_i^{\beta}$

Cas 3.3: Si β est un ordinal limite et pour tout ordinal α tel que $\eta^{\delta}\leq\alpha<\beta$ nous avons $B^{\delta}\subseteq X^{\alpha}$ alors $B^{\delta}\subseteq\bigcup_{\eta^{\delta}\leq\alpha<\beta}X^{\alpha}\subseteq\bigcup_{\alpha<\beta}X^{\alpha}=X^{\beta}$.

Fin de la preuve.

THEOREME 2.9.3.0.9

Une itération asynchrone croissante avec m mémoires pour une appli monotone F de $(L^n)^m$ dans L^n (où $L(\subseteq,\perp,\top,\sqcup,\sqcap)$ est un treillis complet) p d'un pré-point fixe D de F est une séquence stationnaire de limite *luis*

Preuve: La séquence $<B^{\delta}:\delta\in\mu(L)>$ est une chaîne ascendante stationnaire limite $luis(F\circ\sigma)(D)$ (théorème 2.5.2.0.2). Donc il existe un ordinal ε tel que $\forall\gamma\geq\varepsilon$, $luis(F\circ\sigma)(D)=B^{\gamma}$. Donc $\forall\beta\geq\eta^{\varepsilon}$ nous avons $luis(F\circ\sigma)(D)=X^{\beta}\subseteq luis(F\circ\sigma)(D)$ (lemmes 2.9.3.0.9 et 2.9.3.0.2). Noter que $<X^{\delta}:\delta\in Or$ n'est pas nécessairement une chaîne ascendante.

Fin de la preuve.

2.10 NOTES BIBLIOGRAPHIQUES

Comme ouvrages fondamentaux sur les ensembles ordonnés et les treillis on peut consulter Birkhoff[1967], Bourbaki[1967], Grätzer[1971] et Szász[1971].

Nous reviendrons plus longuement sur les fermetures d'un treillis au paragraphe 4.2 et la bibliographie correspondante se trouve au paragraphe 4.4.

De nombreux auteurs ont remarqué que le théorème de point fixe de Knaster[1928] et Tarski[1955] peut être démontré avec des hypothèses plus faibles que la complétude (voir parmi d'autres Abian & Brown[1961], Höft & Höft[1976], Pasini[1974], Pelczar[1961], Markowsky[1976], Ward[1967], Wolk[1957]. De même, dans notre version constructive du théorème de Tarski, nous avons besoin de l'hypothèse d'existence de bornes de certaines chaînes et non de l'hypothèse moins générale que le treillis est complet. Une généralisation est donc possible pour affaiblir l'hypothèse de complétude mais nous n'en aurons pas besoin. Tarski[1955] (suivi par De Marr[1964], Markowsky[1976], Pelczar[1971], Smithson[1973], Wong[1967],...) démontre également un théorème de points fixes communs à une famille d'opérateurs monotones d'un treillis complet qui commutent. Il est également possible d'en donner une version constructive (Cousot & Cousot[1977d]).

Le théorème 2.5.6.0.1. sur l'indécidabilité du problème de calcul de points fixes d'un opérateur monotone d'un treillis complet généralise le théorème [7] de Kam & Ullman[1975] sur la non existence d'un algorithme pour faire une analyse optimale du flot des données d'un programme.

La notion d'itération chaotique vient de l'analyse numérique et remonte à Southwell[1955] et Ostrowski[1955]. Chazan & Miranker[1969] ont introduit la notion d'itération chaotique à retards pour tenir compte des possibilités de calcul parallèle sur des multiprocesseurs. Ils ont montré qu'une itération chaotique à retard converge vers la solution de l'équation $x=Ax+b$ (où A est une matrice nxn de réels et b un vecteur colonne de réels) si et seulement si $\rho(|A|)<1$ (où $|A|$ est la matrice nxn obtenue en remplaçant chaque élément de A par sa valeur absolue et $\rho(|A|)$ est le rayon spectral de $|A|$). Ces résultats ont été étendus pour des problèmes non linéaires et des études de convergence ont été menées pour des hypothèses autres que l'hypothèse de contraction. Signalons entre autres les travaux des équipes d'analyse numérique de Grenoble (Abtroun[1977], Charnay[1975], Mahjoub[1977],

Robert[1974], de Besançon (Comte[1976], El Tarazi[1976], Jacquemard[1977
Luong[1975], Miellou [1975a, 1975b]) et de l'Université Carnegie-Mellon
(Baudet[1976], Traub[1964]). En particulier, le terme "itérations asyn
nes" vient de Baudet[1976] qui généralise la notion d'itérations chaoti
à retards en éliminant l'hypothèse que les retards sont bornés par un r
maximum fixe (Miranker[1977]). Miellou[1977] a obtenu indépendamment l
théorème 2.9.3.0.9 avec des hypothèses plus restrictives que les nôtres
Notamment L est un treillis normal de Banach et F est semi-continue ce
est une hypothèse proche de la continuité supérieure (2.6.0.1) et non n
saire pour donner une version constructive du théorème de Tarski. Auss
rapport à la définition 2.9.3.0.1 Miellou[1971] ajoute l'hypothèse (en
sant nos notations) $\{\forall i\in\{1,\ldots,n\}, \forall j\in\{1,\ldots,m\}, \forall\delta,\delta' \in Ord, \{\delta\leq\delta'\} \Rightarrow \{(S_j^{\delta})_i \leq (S_j^{\delta'})_i\}\}$. Cette hypothèse spécifique à l'étude d'algorithmes m
nes facilite la preuve de convergence car elle implique que l'itération
chrone avec mémoire est une chaîne ascendante mais elle a l'inconvénient
nécessiter une synchronisation entre les processeurs participant au calc
Comme le montre Miranker[1977], un algorithme asynchrone peut être supér
à un algorithme interdisant de régresser dans les calculs à cause des te
d'attente qui résultent de la synchronisation des processeurs.

Nous n'avons pas donné de critère d'arrêt des itérations asynchrone
c'est un problème de programmation traité dans un cadre très général par
Dijkstra[1977].

Le problème du choix d'une stratégie chaotique optimale remonte en
lyse numérique à Stein & Rosenberg[1948]. Bien que d'un intérêt pratiqu
certain, ce problème n'a pas reçu en analyse numérique de réponse complè
ment satisfaisante, sauf dans quelques cas particuliers (par exemple,
Abtroun[1977] ou par des méthodes interactives (Mahjoub[1977]). Pour le
cas des systèmes d'équations que nous considérons en analyse sémantique
programmes, le problème est à l'étude par R.Cousot (voir remarque 4.1.2.

CHAPITRE 3

ETUDE DU COMPORTEMENT D'UN SYSTEME DYNAMIQUE DISCRET, ANALYSE SEMANTIQUE EXACTE DES PROGRAMMES ET APPLICATIONS

3. ETUDE DU COMPORTEMENT D'UN SYSTEME DYNAMIQUE DISCRET, ANALYSE SEMANTIQUE EXACTE DES PROGRAMMES ET APPLICATIONS

3. ETUDE DU COMPORTEMENT D'UN SYSTEME DYNAMIQUE DISCRET, ANALYSE SEMANTIQUE EXACTE DES PROGRAMMES ET APPLICATIONS

Nous consacrons le paragraphe 3.1 à la définition et à l'étude du comportement de systèmes dynamiques discrets. Ces systèmes dynamiques discrets offrent un cadre très général pour formuler et résoudre les problèmes d'analyse sémantique des programmes comme la vérification de la correction totale ou partielle d'un programme ou la découverte de l'invariant le plus général en chaque point d'un programme. Un programme est un système dynamique discret dans la mesure où il définit une relation de transition (ou une fonction de transition s'il est déterministe) entre les états de mémoire précédant et suivant l'exécution d'une instruction élémentaire quelconque. Pour étudier le comportement d'un système dynamique discret il faut caractériser l'ensemble des descendants des états satisfaisant à une spécification d'entrée donnée ou bien caractériser l'ensemble des ascendants des états satisfaisant à une spécification de sortie. Nous montrons que ces ensembles s'obtiennent comme solutions d'équations de point fixe ou de systèmes d'équations quand l'ensemble des états du système dynamique discret est partitionné.

Au paragraphe 3.2 nous définissons la sémantique opérationnelle d'un langage de programmation simple correspondant à des programmes séquentiels itératifs avec instructions d'affectation et de branchement conditionnel ou inconditionnel. (Pour séparer les difficultés, les programmes récursifs seront traités au chapitre 6).

Les résultats du paragraphe 3.1 sont appliqués au paragraphe 3.3 pour définir la sémantique déductive en avant du langage considéré. Un système d'équations sémantiques est associé au programme en exprimant l'ensemble des états possibles des variables en un point a_j du programme en termes des états possibles aux points a_i du programme qui précèdent a_j lors d'une exécution de ce programme. La plus petite solution de ce système d'équations définit l'ensemble des états des variables en un point quelconque du programme, lors d'une exécution quelconque de ce programme partant d'un état initial satisfaisant à une spécification d'entrée donnée.

Nous montrons alors dans la section 3.4 comment la sémantique déductive en avant permet de justifier la méthode de Floyd-Naur de vérification de correction partielle d'un programme. Cette méthode est ensuite étendue aux preuves de correction totale. Enfin, nous montrons que l'exécution symbolique consiste en fait à résoudre le système d'équations sémantiques en avant en utilisant une stratégie itérative chaotique.

Au paragraphe 3.5 nous appliquons les résultats du paragraphe 3.1 pour définir la sémantique déductive en arrière du langage considéré. Un système d'équations sémantiques est associé au programme en exprimant l'ensemble des états des variables en un point a_i du programme en termes des états possibles des variables aux points a_j du programme qui suivent a_i lors d'une exécution de ce programme. Les solutions de ce système d'équations sémantiques en arrière caractérisent l'ensemble des états m des variables pour lesquels une exécution du programme partant de a_i dans un état m des variables se termine en satisfaisant à une spécification de sortie donnée, se termine sans satisfaire cette spécification de sortie, ne se termine jamais ou conduit à une erreur sémantique.

Nous pouvons alors justifier au paragraphe 3.5 la méthode de preuve de correction partielle des programmes de Hoare et son extension par Dijkstra à des preuves de correction totale. Nous remarquons alors que les méthodes de raisonnement en avant et en arrière sur les programmes sont en fait équivalentes.

Enfin, nous établissons au paragraphe 3.6 quelques résultats sur la combinaison des analyses sémantiques en avant et en arrière qui seront utilisés au chapitre 5.

3.1 SYSTEMES DYNAMIQUES DISCRETS

3.1.1 Notion de système dynamique discret

DEFINITION 3.1.1.0.1 *Système dynamique discret*

Un *système dynamique discret* est un quintuplet $(S,\tau,\nu_\varepsilon,\nu_\sigma,\nu_\xi)$ défini comme suit:

(a) - S est *l'ensemble des états* du système,

(b) - $\tau \in ((S \times S) \to B)$ où $B=\{\underline{vrai},\underline{faux}\}$ est une *relation de transition* entre chaque état et ses successeurs,

(c) - $\nu_\varepsilon \in (S \to B)$ caractérise les *états d'entrée*,

(d) - $\nu_\sigma \in (S \to B)$ caractérise les *états de sortie*,

(e) - $\nu_\xi \in (S \to B)$ caractérise les *états erronés*,

et satisfaisant aux conditions suivantes:

(f) - l'ensemble des états n'est pas vide: $\{S \neq \emptyset\}$,

(g) - les états d'entrée sont exogènes:
$\{\forall e_1, e_2 \in S,\ \{\tau(e_1,e_2)\} \Rightarrow \{\underline{non}(\nu_\varepsilon(e_2))\}\}$,

(h) - les états de sortie sont stables:
$\{\forall e_1, e_2 \in S,\ \{\nu_\sigma(e_1)\ \underline{et}\ \tau(e_1,e_2)\} \Rightarrow \{e_1 = e_2\}\}$

(i) - les ensembles des états d'entrée, de sortie et erronés sont deux à deux disjoints:
$\{\forall i,j \in \{\varepsilon,\delta,\xi\},\ \{i \neq j\} \Rightarrow \{\forall e \in S,\ \underline{non}(\nu_i(e)\ \underline{et}\ \nu_j(e))\}\}$.

DEFINITION 3.1.1.0.2

Soit $\tau \in ((S \times S) \to B)$, *l'inverse* de τ est $\tau^{-1} = \lambda(e_1,e_2).\tau(e_2,e_1)$. Une relation de transition (et par extension un système dynamique discret $\pi(S,\tau,\nu_\varepsilon,\nu_\sigma,\nu_\xi)$) est:

(a) - *totale* si et seulement si $\{\forall e_1 \in S,\ \exists e_2 \in S : \tau(e_1,e_2)\}$,

(b) - *partielle* si et seulement si elle n'est pas totale,

(c) - *fonctionnelle* ou *déterministe* si et seulement si:
$\{\forall e_1, e_2, e_3 \in S,\ \{\tau(e_1,e_2)\ \underline{et}\ \tau(e_1,e_3)\} \Rightarrow \{e_2 = e_3\}\}$,

(d) - *injective* si et seulement si τ^{-1} est déterministe,

(e) - *inversible* si et seulement si elle est totale et injective,

(f) - *sans reprise d'erreurs* si et seulement si:
$\{\forall e_1, e_2 \in S,\ \{\nu_\xi(e_1)\ \underline{et}\ \tau(e_1,e_2)\} \Rightarrow \{\nu_\xi(e_2)\}\}$

Le concept de système dynamique discret est évidemment très général. Il s'applique aussi bien aux systèmes informatiques qu'économiques ou biologiques, à condition que le modèle du système étudié soit à évolution discrète dans le temps. En particulier, les systèmes dynamiques discrets sont des modèles des programmes aussi bien séquentiels que parallèles.

Exemple 3.1.1.0.3 : *Programmes séquentiels*

Un programme séquentiel π à la FORTRAN comportant n variables globales

$X_1,\ldots,X_n$ à valeurs respectives dans $U_1,\ldots,U_n$ et α étiquettes ou points-programmes $a_1,\ldots,a_\alpha$ est un système dynamique discret $(S,\tau,\nu_\varepsilon,\nu_\sigma,\nu_\xi)$. Tout état $e\in S$ est un couple $<m,c>$ où $m\in(U_1\times\ldots\times U_n)$ est un état de mémoire qui associe à chaque variable X_i du programme une valeur de U_i et $c\in\{a_1,\ldots,a_\alpha,\underline{\text{erreur}}\}$ est un état de contrôle qui est l'étiquette de la prochaine instruction à exécuter dans le programme ou indique qu'une erreur s'est produite au cours de l'exécution du programme. La relation de transition définit pour tout état $<m,c>$ un état successeur unique $<m',c'> = \tau(<m,c>)$ résultant de l'exécution de l'instruction élémentaire étiquetée c dans le programme π. τ étant une fonction, le système est déterministe. (On observera que le système n'est pas, en général, injectif. Par exemple, pour le programme défini pour tout entier x par $\tau(<a_\varepsilon,x>)=<a_\sigma,x^2>$ nous avons $\tau(<a_\varepsilon,2>) = <a_\sigma,4>$ et $\tau(<a_\varepsilon,-2>) = <a_\sigma,4>$ mais $<a_\varepsilon,2> \neq <a_\varepsilon,-2>$). Quand l'exécution de l'instruction étiquetée c dans le programme n'est pas définie pour l'état mémoire m (c'est le cas par exemple d'une erreur à l'exécution causée par une division par zéro), nous choisirons $\tau(<m,c>) = <m,\underline{\text{erreur}}>$. La conséquence de ce choix est que le système est total. En pratique la détection d'une erreur à l'exécution conduit à l'arrêt de l'exécution du programme. La définition $\tau(<m,\underline{\text{erreur}}>) = <m,\underline{\text{erreur}}>$ est un bon modèle de cette situation car le système étudié est sans reprise d'erreurs. Enfin, supposant que le programme possède un seul point d'entrée a_ε et un seul point de sortie a_σ nous définirons $\nu_\varepsilon=\lambda<m,c>.(c=a_\varepsilon)$, $\nu_\sigma=\lambda<m,c>.(c=a_\sigma)$ et $\nu_\xi=\lambda<m,c>.(c=\underline{\text{erreur}})$.
Fin de l'exemple.

3.1.2 Spécification du comportement d'un système dynamique discret

A partir d'un état $e_0\in S$ un système $\pi(S,\tau,\nu_\varepsilon,\nu_\sigma,\nu_\xi)$ évolue vers des états $e_1,e_2,\ldots,e_i,e_{i+1},\ldots$ tels que $\tau(e_i,e_{i+1})$. L'ensemble des états descendants de e_0 est donné par la fermeture transitive de τ:

DEFINITION 3.1.2.0.1 *Fermeture transitive d'une relation*

Soient $\alpha,\beta\in(S\times S\to B)$ deux relations entre éléments de S. Nous noterons $\alpha\circ\beta$ le *produit* de α par β défini par:

$\alpha\circ\beta = \lambda(e_1,e_2).[\exists e_3\in S : \alpha(e_1,e_3)\ \underline{\text{et}}\ \beta(e_3,e_2)]$

Pour tout entier $n\in\omega$ nous définissons α^n par récurrence finie comme suit:

$\alpha^0 = eq = \lambda(e_1,e_2).(e_1=e_2)$

$\alpha^n = \alpha^{n-1}\circ\alpha = \alpha\circ\alpha^{n-1}$ pour tout $n\geq 1$

La *fermeture transitive reflexive* de α est $\alpha^* = \underset{n\in\omega}{OU}\,\alpha^n$

La *fermeture transitive stricte* de α est $\alpha^+ = \alpha\circ\alpha^* = \alpha^*\circ\alpha = \underset{n\in\omega}{OU}\,\alpha^{n+1}$

La spécification d'un système dynamique discret $\pi(S,\tau,\nu_\varepsilon,\nu_\sigma,\nu_\xi)$ comporte la donnée d'une spécification d'entrée $\phi\in(S\to\mathcal{B})$ et d'une spécification de de sortie $\psi\in(S\to\mathcal{B})$. Cette spécification exprime l'intention que tout état d'entrée satisfaisant à la spécification d'entrée ϕ provoque l'évolution du système π vers un état de sortie satisfaisant à la spécification de sortie ψ. Dans le cas d'un système dynamique discret déterministe, on appelle :

. *Vérification de comportement totalement correct* de π pour ϕ et ψ, la preuve que :

$$\{\forall e_1\in S,\ [\nu_\varepsilon(e_1)\ \underline{et}\ \phi(e_1)]\ \Rightarrow\ \{\exists e_2\in S\ :\ \tau^*(e_1,e_2)\ \underline{et}\ \nu_\sigma(e_2)\ \underline{et}\ \psi(e_2)\}\}$$

. *Vérification de terminaison*, la preuve que tout état d'entrée satisfaisant à la spécification d'entrée ϕ provoque l'évolution du système vers un état de sortie :

$$\{\forall e_1\in S,\ [\nu_\varepsilon(e_1)\ \underline{et}\ \phi(e_1)]\ \Rightarrow\ \{\exists e_2\in S\ :\ \tau^*(e_1,e_2)\ \underline{et}\ \nu_\sigma(e_2)\}\}$$

. *Vérification de comportement partiellement correct* de π pour ϕ et ψ, la preuve que si un état d'entrée satisfaisant à ϕ provoque l'évolution du système vers un état de sortie alors cet état de sortie satisfait à la spécification ψ :

$$\{\forall e_2\in S,\ \{\exists e_1\in S\ :\ \nu_\varepsilon(e_1)\ \underline{et}\ \phi(e_1)\ \underline{et}\ \tau^*(e_1,e_2)\ \underline{et}\ \nu_\sigma(e_2)\}\ \Rightarrow\ \{\psi(e_2)\}\}$$

. *Vérification de l'invariance de* $\beta\in(S\to\mathcal{B})$, la preuve que tout état satisfaisant à β provoque l'évolution du système vers un état satisfaisant à β :

$$\{\forall e_1,e_2\in S,\ \{\beta(e_1)\ \underline{et}\ \tau^*(e_1,e_2)\}\ \Rightarrow\ \{\beta(e_2)\}\}$$

3.1.3 L'approche du point fixe à l'étude du comportement d'un système dynamique discret

Pour faire l'étude du comportement d'un système dynamique discret, nous chercherons à caractériser l'*ensemble des ascendants des états satisfaisant à une condition* $\beta\epsilon(S\to\mathbb{B})$ c'est-à-dire à déterminer :

$$\lambda e_1.\{\exists e_2\epsilon S : \tau^*(e_1,e_2) \underline{\text{ et }} \beta(e_2)\} = wp(\tau^*)(\beta)$$

en notant :

DEFINITION 3.1.3.0.1

$$wp \epsilon (((S\times S)\to\mathbb{B})\to((S\to\mathbb{B})\to(S\to\mathbb{B})))$$
$$= \lambda\theta.\{\lambda\beta.[\lambda e_1.(\exists e_2\epsilon S : \theta(e_1,e_2) \underline{\text{ et }} \beta(e_2))]\}$$

Par exemple une vérification de comportement totalement correct de $\pi(S,\tau,\nu_\varepsilon,\nu_\sigma,\nu_\xi)$ pour ϕ et ψ consiste à montrer que :

$$\{\nu_\varepsilon \underline{\text{ et }} \phi\} \Rightarrow \{wp(\tau^*)(\nu_\sigma \underline{\text{ et }} \psi)\}$$

De même nous chercherons à caractériser l'*ensemble des descendants des états satisfaisant à une condition* $\beta\epsilon(S\to\mathbb{B})$ c'est-à-dire à déterminer :

$$\lambda e_2.\{\exists e_1\epsilon S : \beta(e_1) \underline{\text{ et }} \tau^*(e_1,e_2)\} = sp(\tau^*)(\beta)$$

en notant :

DEFINITION 3.1.3.0.2

$$sp \epsilon (((S\times S)\to\mathbb{B})\to((S\to\mathbb{B})\to(S\to\mathbb{B})))$$
$$= \lambda\theta.\{\lambda\beta.[\lambda e_2.(\exists e_1\epsilon S : \beta(e_1) \underline{\text{ et }} \theta(e_1,e_2))]\}$$

Par exemple une vérification de comportement partiellement correct de π pour ϕ et ψ consiste à montrer que $\{sp(\tau^*)(\nu_\varepsilon \underline{\text{ et }} \phi) \underline{\text{ et }} \nu_\sigma\} \Rightarrow \{\psi\}$. De même la vérification de l'invariance de β consiste à montrer que $\beta \Rightarrow \underline{\text{non}}(wp(\tau^*)(\underline{\text{non}}(\beta)))$ ou encore $sp(\tau^*)(\beta) \Rightarrow \beta$.

Partant du fait que $\tau^* = eq \underline{\text{ ou }} \tau^*\circ\tau = eq \underline{\text{ ou }} \tau\circ\tau^*$, nous obtiendrons $wp(\tau^*)$ et $sp(\tau^*)$ comme points fixes d'une équation.

THEOREME 3.1.3.0.3

(a) - $((S\times S)\to\mathbb{B})(\Longrightarrow, \lambda(e_1,e_2).\underline{faux}, \lambda(e_1,e_2).\underline{vrai}, \underline{OU}, \underline{ET}, \underline{non})$ est un treillis booléen complet,

(b) - Soient $a, b \in ((S\times S)\to\mathbb{B})$ alors $\lambda\alpha.[a \ \underline{ou}\ b\circ\alpha]$ et $\lambda\alpha.[a \ \underline{ou}\ \alpha\circ b]$ sont des morphismes complets pour la disjonction,

(c) - Soient $\tau \in ((S\times S)\to\mathbb{B})$ et *eq* la relation d'égalité alors $\tau^* = \mathit{lfp}(\lambda\alpha.[eq \ \underline{ou}\ \alpha\circ\tau]) = \mathit{lfp}(\lambda\alpha.[eq \ \underline{ou}\ \tau\circ\alpha])$.

La proposition suivante indique que l'étude peut être indifféremment menée pour *sp* ou *wp* en considérant une relation de transition ou son inverse:

PROPOSITION 3.1.3.0.4

(a) - $\lambda\theta.\theta^{-1}$ est un automorphisme complet de $((S\times S)\to\mathbb{B})$,

(b) - $\forall\alpha,\beta \in ((S\times S)\to\mathbb{B})$, $(\alpha\circ\beta)^{-1} = \beta^{-1}\circ\alpha^{-1}$,

(c) - $\forall\tau \in ((S\times S)\to\mathbb{B})$, $\forall n\in\omega$, $(\tau^n)^{-1} = (\tau^{-1})^n$,

(d) - $\forall\tau \in ((S\times S)\to\mathbb{B})$, $(\tau^*)^{-1}=(\tau^{-1})^*$ et $(\tau^+)^{-1}=(\tau^{-1})^+$,

(e) - $\forall\theta \in ((S\times S)\to\mathbb{B})$, $sp(\theta)=wp(\theta^{-1})$ et $wp(\theta)=sp(\theta^{-1})$.

LEMME 3.1.3.0.5

Quels que soient $\theta \in ((S\times S)\to\mathbb{B})$ et $\beta \in (S\to\mathbb{B})$ nous avons:

(a) - $wp(\theta)$, $\lambda\theta.(wp(\theta)(\beta))$, $sp(\theta)$ et $\lambda\theta.(wp(\theta)(\beta))$ sont inférieurement stricts (c.a.d. $f(\perp)=\perp$). Ce sont des morphismes complets pour la disjonction,

(b) - Si θ est déterministe alors $wp(\theta)$ est un morphisme complet pour la conjonction. Si θ est injective alors $sp(\theta)$ est un morphisme complet pour la conjonction.

Preuve:

(a) résulte du fait que pour toute famille $\{\beta_i : i\in I\}$ d'éléments de $(S\to\mathbb{B})$ et tout $\beta\in(S\to\mathbb{B})$ on a $\{\exists e\in S : \beta(e) \ \underline{et}\ (\underline{OU}_{i\in I}\ \beta_i(e))\} = \underline{OU}_{i\in I}\{\exists e\in S : \beta(e) \ \underline{et}\ \beta_i(e)\}$.

(b) Quand θ est déterministe il existe une application f de S dans S telle que $\{\forall e_1,e_2\in S,\ \{\theta(e_1,e_2)\} \Longleftrightarrow \{f(e_1)=e_2\}\}$. Pour e_1 fixé et quand $\{e\in S : \theta(e_1,e)\}$ n'est pas vide, nous avons $wp(\theta)(\underline{ET}_{i\in I}\ \beta_i)(e_1) = \{\exists e_2\in S : \theta(e_1,e_2) \ \underline{et}\ (\underline{ET}_{i\in I}\ \beta_i(e_2))\} = \underline{ET}_{i\in I}\ (\beta_i(f(e_1)) = \underline{ET}_{i\in I}\{\exists e_2\in S : \theta(e_1,e_2) \ \underline{et}\ \beta_i(e_2)\} = \underline{ET}_{i\in I}\ wp(\theta)(\beta_i)(e_1)$. D'après 3.1.1.0.2.(d) et 3.1.3.0.4 si θ est injective

alors θ^{-1} est déterministe et $sp(\theta)=wp(\theta^{-1})$ est un morphisme complet pour la conjonction.

Fin de la preuve.

THEOREME 3.1.3.0.6.

Quels que soient a, b $\in ((S\times S)\to\mathcal{B})$ et $\beta \in (S\to\mathcal{B})$ nous avons:

- $wp(lfp(\lambda\alpha.[a \underline{ou} b\circ\alpha]))(\beta)$
  $= lfp(\lambda\alpha.[wp(a)(\beta) \underline{ou} wp(b)(\alpha)])$
  $= \underset{n\in\omega}{\underline{OU}}\ wp(b^n)(wp(a)(\beta))$
- $sp(lfp(\lambda\alpha.[a \underline{ou} \alpha\circ b]))(\beta)$
  $= lfp(\lambda\alpha.[sp(a)(\beta) \underline{ou} sp(b)(\alpha)])$
  $= \underset{n\in\omega}{\underline{OU}}\ sp(b^n)(sp(a)(\beta))$

Preuve: Posons $h=\lambda\theta.[wp(\theta)(\beta)]$, $f=\lambda\alpha.[a \underline{ou} b\circ\alpha]$ et $g = \lambda\alpha.[wp(a)(\beta) \underline{ou} wp(b)(\alpha)]$ et montrons que $h\circ f = g\circ h$. En effet soit $\alpha\in((S\times S)\to\mathcal{B})$ alors:

$h\circ f(\alpha) = wp(a \underline{ou} b\circ\alpha)(\beta)$
$= \lambda e_1.[\exists e_2\in S : (a(e_1,e_2) \underline{ou} (b\circ\alpha)(e_1,e_2)) \underline{et} \beta(e_2)]$
$= \lambda e_1.[\exists e_2\in S : a(e_1,e_2) \underline{ou} \beta(e_2)] \underline{ou}$
$\lambda e_1.[\exists e_2,e_3\in S : b(e_1,e_3) \underline{et} \alpha(e_3,e_2) \underline{et} \beta(e_2)]$
$= wp(a)(\beta) \underline{ou} \lambda e_1.[\exists e_3\in S : b(e_1,e_3) \underline{ou} wp(\alpha)(\beta)(e_3)]$
$= wp(a)(\beta) \underline{ou} wp(b)(wp(\alpha)(\beta))$
$= g\circ h(\alpha)$

f, g et h sont des morphismes complets pour l'union et par conséquant la proposition 2.5.2.0.8 implique que $h(lfp(f)) = h(luis(f)(\lambda(e_1,e_2).[\underline{faux}]))$ $= luis(g)(h(\lambda(e_1,e_2).[\underline{faux}])) = luis(g)(\lambda(e_1,e_2).[\underline{faux}]) = lfp(g)$.

Soit $\langle X^n : n\in\omega\rangle$ l'itération croissante définie par g et partant de l'infimum $\lambda e.\underline{faux}$. Nous avons $X^0=\lambda e.\underline{faux}$ et $X^1=wp(a)(\beta)$ car $wp(b)$ est inférieurement strict. Supposons que $X^n = \underset{i=0}{\overset{n-1}{\underline{OU}}}\ wp(b^i)(wp(a)(\beta))$. Alors $X^{n+1} = g(X^n)$ $= wp(a)(\beta) \underline{ou} \underset{i=0}{\overset{n-1}{\underline{OU}}}\ wp(b)(wp(b^i)(wp(a)(\beta))) = \underset{i=0}{\overset{n}{\underline{OU}}}\ wp(b^i)(wp(a)(\beta))$ car $wp(b)(wp(a)(\beta)) = wp(b\circ a)(\beta)$. Par récurrence finie nous avons trouvé le terme général de l'itération et passant à la limite nous obtenons $lfp(g) = X^\omega = \underset{n\in\omega}{\underline{OU}}\ (\underset{i=0}{\overset{n-1}{\underline{OU}}}\ wp(b^i)(wp(a)(\beta))) = \underset{n\in\omega}{\underline{OU}}\ wp(b^i)(wp(a)(\mathcal{E}))$.)

D'après la proposition 3.1.3.0.4 une preuve symétrique s'applique à sp.
Fin de la preuve.

Les théorèmes 3.1.3.0.3 et 3.1.3.0.6 impliquent le:

COROLLAIRE 3.1.3.0.7

Soit $\tau \in ((S\times S)\to\mathcal{B})$, alors

. $wp(\tau^*) = \lambda\beta.lfp(\lambda\alpha.[\beta \underline{ou}\ wp(\tau)(\alpha)]) = \underset{n\in\omega}{\underline{OU}}\ wp(\tau^n)$

. $sp(\tau^*) = \lambda\beta.lfp(\lambda\alpha.[\beta \underline{ou}\ sp(\tau)(\alpha)]) = \underset{n\in\omega}{\underline{OU}}\ sp(\tau^n)$

S'il est facile de calculer $wp(lfp(\lambda\alpha.[a \underline{ou}\ b\circ\alpha]))(\beta)$ en utilisant la proposition 2.5.2.0.8 il n'en va pas de même de $wp(lfp(\lambda\alpha.[a \underline{ou}\ \alpha\circ b]))(\beta)$ car $\lambda\theta.[wp(\theta)(\beta)]\circ\lambda\alpha.[a \underline{ou}\ \alpha\circ b]=\lambda\alpha.[wp(a)(\beta) \underline{ou}\ wp(\alpha)(wp(b)(\beta))]$ n'est pas la composition d'une fonction avec $\lambda\alpha.wp(\alpha)(\beta)$. Dans ces conditions il est plus facile d'exprimer wp en utilisant sp pour lequel nous appliquerons le théorème 3.1.3.0.7. En effet:

$$\begin{aligned} wp &= \lambda\theta.\{\lambda\beta.[\lambda\bar{e}.(\exists e_2\in S : \theta(\bar{e},e_2) \underline{et}\ \beta(e_2))]\} \\ &= \lambda\theta.\{\lambda\beta.[\lambda\bar{e}.(\exists e_2\in S : [\exists e\in S : (e=\bar{e}) \underline{et}\ \theta(e,e_2)] \underline{et}\ \beta(e_2))])\} \\ &= \lambda\theta.\{\lambda\beta.[\lambda\bar{e}.(\exists e_2\in S : sp(\theta)(\lambda e.[e=\bar{e}])(e_2) \underline{et}\ \beta(e_2))]\} \end{aligned}$$

de sorte que 3.1.3.0.7. et 3.1.3.0.4. impliquent le :

COROLLAIRE 3.1.3.0.8

Soit $\tau\in((S\times S)\to\mathcal{B})$, alors

. $wp(\tau^*) = \lambda\beta.\{\lambda\bar{e}.[\exists e_2\in S : \beta(e_2) \underline{et}\ lfp(\lambda\alpha.[\lambda e.(e=\bar{e}) \underline{ou}\ sp(\tau)(\alpha)])(e_2)]\}$

. $sp(\tau^*) = \lambda\beta.\{\lambda\bar{e}.[\exists e_1\in S : \beta(e_1) \underline{et}\ lfp(\lambda\alpha.[\lambda e.(e=\bar{e}) \underline{ou}\ wp(\tau)(\alpha)])(e_1)]\}$

Ce résultat montre que nous pouvons indifféremment mener l'étude d'un système dynamique discret en résolvant une équation "en avant" de type $\alpha = \beta \underline{ou}\ sp(\tau)(\alpha)$ ou bien une équation "en arrière" de type $\alpha = \beta \underline{ou}\ wp(\tau)(\alpha)$

3.1.4 Systèmes dynamiques discrets à ensemble d'états partitionné

Nous avons ramené l'étude d'un système dynamique discret $\pi(S,\tau,\nu_\varepsilon,\nu_\sigma,\nu_\xi)$ au problème de résoudre une équation associée au système. Quand l'ensemble des états S est partitionné, la partition de S induit une décomposition directe de $(S\to\mathcal{B})$ et la proposition 2.8.0.1. indique alors comment passer d'une équation à un système d'équations ayant des ensembles de solutions isomorphes. Cette décomposition nous permettra d'utiliser diverses techniques de résolution de systèmes d'équations monotones dans un treillis complet étudiées aux paragraphes 2.8 et 2.9.

DEFINITION 3.1.4.0.1 *Système dynamique discret partitionné*

Un système dynamique discret partitionné est un tuple $(S,\tau,\nu_1,\ldots,\nu_n,\varepsilon,\sigma,\xi)$ tel que :

(a) - $\{n\geq 1\}$ et $\{\forall i\in[1,n],\ \nu_i\in(S\to\mathcal{B})\}$

(b) - $\{\varepsilon,\sigma,\xi\in[1,n]\}$ et $\{(S,\tau,\nu_\varepsilon,\nu_\sigma,\nu_\xi)$ est un système dynamique discret$\}$

(c) - $\{\forall e\in S,\ \exists i\in[1,n]:\ \nu_i(e)\}$

(d) - $\{\forall i,j\in[1,n],\ \forall e\in S,\ \{\nu_i(e)$ et $\nu_j(e)\}\Rightarrow\{i=j\}\}$

Pour tout $i\in[1,n]$ définissons :

$$\sigma_i \in ((S\to\mathcal{B})\to(S\to\mathcal{B})) = \lambda\beta.[\nu_i \text{ et } \beta]$$

$$\sigma \in ((S\to\mathcal{B})\to(\sigma_1(S\to\mathcal{B})\times\ldots\times\sigma_n(S\to\mathcal{B}))) = \lambda\beta.[\sigma_1(\beta),\ldots,\sigma_n(\beta)] = \lambda\beta.\prod_{i=1}^{n}\sigma_i(\beta)$$

$$\sigma^{-1} \in ((\sigma_1(S\to\mathcal{B})\times\ldots\times\sigma_n(S\to\mathcal{B}))\to(S\to\mathcal{B})) = \lambda\beta.[\bigcup_{i=1}^{n}\beta_i]$$

D'après 3.1.4.0.1.(c) $\sigma^{-1}\circ\sigma=\lambda\beta.[\bigcup_{i=1}^{n}\nu_i \text{ et } \beta]=\lambda\beta.\beta$ et d'après 3.1.4.0.1.(d)

$$\sigma_i\circ\sigma^{-1}=\lambda\beta.\sigma_i\circ\sigma^{-1}(\beta_1 \text{ et } \nu_1,\ldots,\beta_n \text{ et } \nu_n)=\lambda\beta.[\nu_i \text{ et } (\bigcup_{j=1}^{n}\beta_j \text{ et } \nu_j)]=\lambda\beta.\beta_i$$

de sorte que σ est une bijection. Les propriétés de distributivité β et $(\bigcup_{j\in J}\beta_j)=\bigcup_{j\in J}(\beta \text{ et } \beta_j)$ et β et $(\bigcap_{j\in J}\beta_j)=\bigcap_{j\in J}(\beta \text{ et } \beta_j)$ impliquent que σ est un morphisme complet de $(S\to\mathcal{B})$ sur $\prod_{i=1}^{n}\sigma_i(S\to\mathcal{B})$. Dans ces conditions

les propositions 2.5.2.0.8, 2.5.3.0.3, 2.5.3.0.2 et 2.5.5.0.1 indiquent que les ensembles des pré-, post- et points fixes de $F \in ((S \to \mathcal{B}) \to (S \to \mathcal{B}))$ sont complètement isomorphes par σ aux ensembles des pré-, post- et solutions du système d'équations :

$$\begin{cases} x_i = \sigma_i \circ F \circ \sigma^{-1}(x_1,\ldots,x_n) \\ i = 1,\ldots,n \end{cases}$$

En particulier pour $F=\lambda\alpha.[\beta \underline{\text{ou}} sp(\tau)(\alpha)]$ et $F=\lambda\alpha.[\beta \underline{\text{ou}} wp(\tau)(\alpha)]$ nous obtenons :

PROPOSITION 3.1.4.0.2

Soit $\pi(S,\tau,\nu_1,\ldots,\nu_n,\varepsilon,\sigma,\xi)$ un système dynamique discret partitionné. Alors $\forall i \in [1,n]$, $\sigma_i \circ \lambda\alpha.[\beta \underline{\text{ou}} sp(\tau)(\alpha)] \circ \sigma^{-1}$ est égal à :

$$\lambda(\alpha_1,\ldots,\alpha_n).[(\nu_i \underline{\text{et}} \beta) \underline{\text{ou}} (\underset{j \in \underline{\text{pred}}_\tau(i)}{\underline{\text{OU}}} sp(\tau_{ji})(\alpha_j))]$$

en notant

$$\forall i,j \in [1,n],\ \forall \theta \in ((S\times S) \to \mathcal{B}),\ \theta_{ij} = \lambda(e_1,e_2).[\nu_i(e_1) \underline{\text{et}} \theta(e_1,e_2) \underline{\text{et}} \nu_j(e_2)]$$

$$\underline{\text{pred}}_\tau \in [1,n] \to 2^{[1,n]} = \lambda i.\{j \in [1,n] : (\exists e_1,e_2 \in S : \tau_{ji}(e_1,e_2))\}$$

Preuve :

$\sigma_i(\beta \underline{\text{ou}} sp(\tau)(\sigma^{-1}(\alpha_1,\ldots,\alpha_n)))$ où $(\forall i \in [1,n],\ \alpha_i \in \sigma_i(S \to \mathcal{B}))$

$= \sigma_i(\beta) \underline{\text{ou}} \sigma_i(sp(\tau)(\sigma^{-1}(\nu_1 \underline{\text{et}} \alpha_1,\ldots,\nu_n \underline{\text{et}} \alpha_n)))$

$= \sigma_i(\beta) \underline{\text{ou}} \lambda e_2.\{\nu_i(e_2) \underline{\text{et}} sp(\tau)(\underset{j=1}{\overset{n}{\underline{\text{OU}}}}(\nu_j \underline{\text{et}} \alpha_j))(e_2)\}$

$= \sigma_i(\beta) \underline{\text{ou}} \lambda e_2.\{\nu_i(e_2) \underline{\text{et}} \underset{j=1}{\overset{n}{\underline{\text{OU}}}} (sp(\tau)(\nu_j \underline{\text{et}} \alpha_j)(e_2))\}$ (d'après 3.1.3.0.5.(a))

$= \sigma_i(\beta) \underline{\text{ou}} \underset{j=1}{\overset{n}{\underline{\text{OU}}}} \lambda e_2.\{\exists e_1 \in S : \alpha_j(e_1) \underline{\text{et}} \nu_j(e_1) \underline{\text{et}} \tau(e_1,e_2) \underline{\text{et}} \nu_i(e_2)\}$ (d'après 3.1.2.0.2)

$= \sigma_i(\beta) \underline{\text{ou}} \underset{j=1}{\overset{n}{\underline{\text{OU}}}} \lambda e_2.\{\exists e_1 \in S : \alpha_j(e_1) \underline{\text{et}} \tau_{ji}(e_1,e_2)\}$

$= (\nu_i \underline{\text{et}} \beta) \underline{\text{ou}} \underset{j \in \underline{\text{pred}}_\tau(i)}{\underline{\text{OU}}} sp(\tau_{ji})(\alpha_j)$ car $\{j \notin \underline{\text{pred}}_\tau(i)\} \Rightarrow \{\underline{\text{non}}(\tau_{ji}(e_1,e_2))\}$

Fin de la preuve.

En remarquant que $(\tau_{ij})^{-1} = (\tau^{-1})_{ji}$ et en utilisant la proposition 3.1.3.0.4 nous obtenons directement la proposition :

PROPOSITION 3.1.4.0.3

Soit $\pi(S,\tau,\nu_1,\ldots,\nu_n,\varepsilon,\sigma,\xi)$ un système dynamique discret partitionné. Alors $\forall i\in[1,n]$, $\sigma_i\circ\lambda\alpha.[\beta \underline{\text{ou}} \mathit{wp}(\tau)(\alpha)]\circ\sigma^{-1}$ est égal à :

$$\lambda(\alpha_1,\ldots,\alpha_n).[(\nu_i \underline{\text{et}} \beta) \underline{\text{ou}} (\underset{j\in \underset{\tau}{\underline{\text{succ}}}(i)}{\underline{\text{OU}}} \mathit{wp}(\tau_{ij})(\alpha_j))]$$

en notant

$$\underset{\tau}{\underline{\text{succ}}} \in [1,n] \to 2^{[1,n]}$$
$$= \lambda i.\{j\in[1,n] : (\exists e_1,e_2\in S : \tau_{ij}(e_1,e_2))\}$$

3.1.5 Propriétés des systèmes dynamiques discrets déterministes

D'après les définitions 3.1.1.0.2.(c) et (d) et la proposition 3.1.3.0.4, l'étude de $\mathit{wp}(\tau^*)$ quand τ est déterministe et celle de $\mathit{sp}(\tau^*)$ quand τ est injective sont symétriques. Cependant, nous n'énoncerons les résultats que pour les systèmes déterministes car les programmes injectifs sont fort rares. Nous avons vu que $\mathit{wp}(\tau^*) = \lambda\beta.\mathit{lfp}(\lambda\alpha.[\beta \underline{\text{ou}} \mathit{wp}(\tau)(\alpha)])$. Nous nous intéressons maintenant aux propriétés des autres points fixes de $\lambda\alpha.[\beta \underline{\text{ou}} \mathit{wp}(\tau)(\alpha)]$.

PROPOSITION 3.1.5.0.1

Soient $\pi(S,\tau,\nu_\varepsilon,\nu_\sigma,\nu_\xi)$ un système dynamique discret déterministe, $\beta\in(S\to B)$ tel que $\{\forall e_1,e_2\in S, \{\beta(e_1) \underline{\text{et}} \tau(e_1,e_2)\} \Rightarrow \{e_1=e_2\}\}$ et $\alpha\in(S\to B)$ un pré-point fixe de $\lambda x.[\beta \underline{\text{ou}} \mathit{wp}(\tau)(x)]$ alors $\mathit{sp}(\tau^*)(\alpha) \Rightarrow \alpha$.

Preuve : Montrons tout d'abord que $\{\mathit{sp}(\tau)(\alpha) \Rightarrow \alpha\}$. Pour tout e de S nous avons $\mathit{sp}(\tau)(\alpha)(e) = \{\exists e_1\in S : \alpha(e_1) \underline{\text{et}} \tau(e_1,e)\} \Rightarrow \{\exists e_1\in S : \alpha(e_1) \underline{\text{et}} [\beta(e_1) \underline{\text{ou}} \mathit{wp}(\tau)(\alpha)(e_1)] \underline{\text{et}} \tau(e_1,e)\}$.
Comme d'une part $\{\{\beta(e_1) \underline{\text{et}} \tau(e_1,e)\} \Rightarrow \{e_1=e\}\}$ et d'autre part π étant déterministe $\{\{\tau(e_1,e_2) \underline{\text{et}} \tau(e_1,e)\} \Rightarrow \{e_2=e\}\}$ il vient $\mathit{sp}(\tau)(\alpha)(e) \Rightarrow \{\exists e_1\in S : [\alpha(e_1) \underline{\text{et}} (e_1=e)] \underline{\text{ou}} [\alpha(e_2) \underline{\text{et}} (e_2=e)]\} \Rightarrow \alpha(e)$ et donc $(\alpha \underline{\text{ou}} \mathit{sp}(\tau)(\alpha)) \Rightarrow \alpha$. Comme α est un post-point fixe de $\lambda x.[\alpha \underline{\text{ou}} \mathit{sp}(\tau)(x)]$ les théorèmes 3.1.3.0.5 et 2.5.3.0.2 impliquent que $\mathit{sp}(\tau^*)(\alpha) = \mathit{lfp}(\lambda x.[\alpha \underline{\text{ou}} \mathit{sp}(\tau)(x)]) \Rightarrow \alpha$.
Fin de la preuve.

THEOREME 3.1.5.0.2

Soient $\tau \in ((S\times S)\to \mathbb{B})$ une relation de transition totale et déterministe et $\beta \in (S\to\mathbb{B})$, alors :

$$\underline{non}[wp(\tau^*)(\beta)] = gfp(\lambda\alpha.[\underline{non}(\beta)\ \underline{et}\ wp(\tau)(\alpha)])$$

Preuve : D'après le théorème 3.1.3.0.7, nous savons que $\underline{non}[wp(\tau^*)(\beta)] = \underline{non}[lfp(\lambda\alpha.[\beta\ \underline{ou}\ wp(\tau)(\alpha)])] = \underline{non}[lfp(\lambda\alpha.[\underline{non}(\underline{non}(\beta)\ \underline{et}\ \underline{non}(wp(\tau)(\underline{non}(\underline{non}(\alpha))))])])]$ ce qui d'après la proposition 2.5.5.0.5 est égal à $gfp(\lambda\alpha.[\underline{non}(\beta)\ \underline{et}\ \underline{non}(wp(\tau)(\underline{non}(\alpha)))])$. Comme τ est totale et déterministe il existe une fonction totale $\overline{\tau}\in(S\to\mathbb{B})$ telle que $\{\forall e_1,e_2\in S,\ \{\tau(e_1,e_2)\}\} \Leftrightarrow \{\overline{\tau}(e_1)=e_2\}\}$. Dans ces conditions $\underline{non}(wp(\tau)(\underline{non}(\alpha)))(e_1) = \underline{non}(\underline{non}(\alpha(\overline{\tau}(e_1)))) = \alpha(\overline{\tau}(e_1)) = wp(\tau)(\alpha)(e_1)$ et donc $\underline{non}[wp(\tau^*)(\beta)] = gfp(\lambda\alpha.[\underline{non}(\beta)\ \underline{et}\ wp(\tau)(\alpha)])$.
Fin de la preuve.

PROPOSITION 3.1.5.0.3

Soient $\pi(S,\tau,\nu_\varepsilon,\nu_\sigma,\nu_\xi)$ un système dynamique discret déterministe et $\phi,\ \psi \in (S\to\mathbb{B})$, alors :

(a) - $[sp(\tau^*)(\phi)\ \underline{et}\ wp(\tau^*)(\nu_\sigma\ \underline{et}\ \psi)] = sp[\tau^*][wp(\tau^*)(\nu_\sigma\ \underline{et}\ \psi)\ \underline{et}\ \phi]$

(b) - $[wp(\tau)(\psi)\ \underline{et}\ sp(\tau^*)(\phi)] \Rightarrow wp[\tau][\psi\ \underline{et}\ sp(\tau^*)(\phi)]$

(c) - $sp(\tau^*)(\phi)\ \underline{et}\ wp(\tau^*)(\psi) = lfp\{\lambda\alpha.[sp(\tau^*)(\phi)\ \underline{et}\ (\psi\ \underline{ou}\ wp(\tau)(\alpha))]\}$

(d) - $= lfp\{\lambda\alpha.[wp(\tau^*)(\phi)\ \underline{et}\ (\phi\ \underline{ou}\ sp(\tau)(\alpha))]\}$

(e) - $= lfp\{\lambda\alpha.[sp(\tau^*)(\phi)\ \underline{et}\ wp(\tau^*)(\psi)\ \underline{et}\ (\psi\ \underline{ou}\ wp(\tau)(\alpha))]\}$

(f) - $= lfp\{\lambda\alpha.[sp(\tau^*)(\phi)\ \underline{et}\ wp(\tau^*)(\psi)\ \underline{et}\ (\phi\ \underline{ou}\ sp(\tau)(\alpha))]\}$

Preuve:

(a) - Montrons tout d'abord que $sp(\tau^*)[wp(\tau^*)(\nu_\sigma\ \underline{et}\ \psi)] \Rightarrow wp(\tau^*)(\nu_\sigma\ \underline{et}\ \psi)$.
$sp(\tau^*)(wp(\tau^*)(\nu_\sigma\ \underline{et}\ \psi))(e)$

$= \{\exists e_1,e_2\in S,\ \exists k,\ell\in\omega : \tau^k(e_1,e_2)\ \underline{et}\ \nu_\sigma(e_2)\ \underline{et}\ \psi(e_2)\ \underline{et}\ \tau^\ell(e_1,e)\}$

. Si $k>\ell$ alors $\tau^k = \tau^\ell\circ\tau^{k-\ell}$

$= \{\exists e_1,e_2,e_3\in S,\ \exists k,\ell\in\omega : \tau^\ell(e_1,e_3)\ \underline{et}\ \tau^{k-\ell}(e_3,e_2)\ \underline{et}\ \nu_\sigma(e_2)\ \underline{et}\ \psi(e_2)\ \underline{et}\ \tau^\ell(e_1,e)\}$

Comme τ est déterministe, τ^ℓ l'est également et par conséquent $\{e_3=e\}$,

$\Rightarrow \{\exists e_2\in S,\ \exists k,\ell\in\omega : \tau^{k-\ell}(e,e_2)\ \underline{et}\ \nu_\sigma(e_2)\ \underline{et}\ \psi(e_2)\} = wp(\tau^*)(\nu_\sigma\ \underline{et}\ \psi)(e)$

. Si $k\le\ell$ alors $\tau^\ell = \tau^k\circ\tau^{\ell-k}$

$= \{\exists e_1,e_2,e_3\in S,\ \exists k,\ell\in\omega : \tau^k(e_1,e_2)\ \underline{et}\ \nu_\sigma(e_2)\ \underline{et}\ \psi(e_2)\ \underline{et}\ \tau^k(e_1,e_3)\ \underline{et}\ \tau^{\ell-k}(e_3,e)\}$

Comme τ est déterministe, τ^k l'est également et par conséquent $\{e_3=e_2\}$.

Mais d'après 3.1.1.0.1.(h) $\{\nu_\sigma(e_2)$ et $\tau^{\ell-k}(e_2,e)\} \Rightarrow \{e_2=e\}$.

$\Rightarrow \{\exists e_2 \in S,\ \exists k,\ell\in\omega : \tau^{k-\ell}(e,e_2)$ et $\nu_\sigma(e_2)$ et $\psi(e_2)\} = wp(\tau^*)(\nu_\sigma$ et $\psi)(e)$

Maintenant,

$sp(\tau^*)(wp(\tau^*)(\nu_\sigma$ et $\psi)$ et $\phi)$

$\Rightarrow sp(\tau^*)(wp(\tau^*)(\nu_\sigma$ et $\psi))$ et $sp(\tau^*)(\phi)$ par monotonie

$\Rightarrow wp(\tau^*)(\nu_\sigma$ et $\psi)$ et $sp(\tau^*)(\phi)$

Réciproquement,

$wp(\tau^*)(\nu_\sigma$ et $\psi)$ et $sp(\tau^*)(\phi)$

$= \lambda e.\{(\exists e_2\in S : \tau^*(e,e_2)$ et $\nu_\sigma(e_2)$ et $\psi(e_2))$ et $(\exists e_1\in S : \phi(e_1)$ et $\tau^*(e_1,e))\}$

Comme $\{\exists e_3\in S : \tau^*(e_1,e_3)$ et $\tau^*(e_3,e_2)\} = \tau^*\circ\tau^*(e_1,e_2) = \tau^*(e_1,e_2)$, il vient:

$\Rightarrow \lambda e.\{\exists e_1\in S : (\exists e_2\in S: \tau^*(e_1,e_2)$ et $\nu_\sigma(e_2)$ et $\psi(e_2))$ et $\phi(e_1)$ et $\tau^*(e_1,e)\}$

$= sp(\tau^*)(wp(\tau^*)(\nu_\sigma$ et $\psi)$ et $\phi)$

(b) - Montrons que $[wp(\tau)(\psi)$ et $sp(\tau^*)(\phi) \Rightarrow wp[\tau][\psi$ et $sp(\tau^*)(\phi)]$.

$wp(\tau)(\psi)$ et $sp(\tau^*)(\phi)$

$= \lambda e.\{(\exists e_2\in S: \tau(e,e_2)$ et $\psi(e_2))$ et $(\exists e_1\in S: \phi(e_1)$ et $\tau^*(e_1,e))\}$

Comme $\{\exists e_3\in S: \tau^*(e_1,e_3)$ et $\tau(e_3,e_2)\} = \tau^*\circ\tau(e_1,e_2) \Rightarrow \tau^*(e_1,e_2)$, il vient:

$\Rightarrow \lambda e.\{\exists e_2\in S: \psi(e_2)$ et $(\exists e_1\in S: \phi(e_1)$ et $\tau^*(e_1,e_2))$ et $\tau(e,e_2)\}$

$= wp(\tau)(\psi$ et $sp(\tau^*)(\phi))$

(c) - Soient $\langle X^\delta : \delta\le\omega\rangle$ et $\langle Y^\delta : \delta\le\omega\rangle$ les itérations croissantes partant de l'infimum $\lambda\beta.$faux de $(S\to B)$ et respectivement définies par $\lambda\alpha.[\psi$ ou $wp(\tau)(\alpha)]$ et $\lambda\alpha.[sp(\tau^*)(\phi)$ et $(\psi$ ou $wp(\tau)(\alpha))]$. Nous avons $(X^0$ et $sp(\tau^*)(\phi)) = \lambda\beta.$faux $= Y^0$. Supposons que $(X^{\delta-1}$ et $sp(\tau^*)(\phi)) \Rightarrow Y^{\delta-1}$, alors :

X^δ et $sp(\tau^*)(\phi)$

$= (\psi$ ou $wp(\tau)(X^{\delta-1}))$ et $sp(\tau^*)(\phi)$

$\Rightarrow (\psi$ ou $(wp(\tau)(X^{\delta-1})$ et $sp(\tau^*)(\phi)))$ et $sp(\tau^*)(\phi)$

$\Rightarrow (\psi$ ou $wp(\tau)(X^{\delta-1}$ et $sp(\tau^*)(\phi)))$ et $sp(\tau^*)(\phi)$ d'après (b)

$\Rightarrow (\psi$ ou $wp(\tau)(Y^{\delta-1}))$ et $sp(\tau^*)(\phi)$ par hypothèse de récurrence et monotonie

$= Y^\delta$

Alors $(X^\omega$ et $sp(\tau^*)(\phi)) = (($ ou$_{\alpha<\omega}$ $X^\alpha)$ et $sp(\tau^*)(\phi)) = ($ ou$_{\alpha<\omega}$ $(X^\alpha$ et $sp(\tau^*)(\phi)))$

$\Rightarrow$ ou$_{\alpha<\omega}$ $Y^\alpha = Y^\omega$. Il vient :

$sp(\tau^*)(\phi)$ et $wp(\tau^*)(\psi)$

$= sp(\tau^*)(\phi)$ et X^ω d'après 3.1.3.0.7, 3.1.3.0.5 et 2.7.0.1.

$\Rightarrow Y^\omega = lfp(\lambda\alpha.[sp(\tau^*)(\phi)$ et $(\psi$ ou $wp(\tau)(\alpha))])$

$\Rightarrow$ $\mathit{lfp}(\lambda\alpha.[sp(\tau^*)(\phi)])$ et $\mathit{lfp}(\lambda\alpha.[\psi$ ou $wp(\tau)(\alpha)])$ car lfp est monotone,

$= sp(\tau^*)(\phi)$ et $wp(\tau^*)(\psi)$

Par antisymétrie $(sp(\tau^*)(\phi)$ et $wp(\tau^*)(\psi)) = \mathit{lfp}(\lambda\alpha.[sp(\tau^*)(\phi)$ et $(\psi$ ou $wp(\tau)(\alpha))])$. (d), (e) et (f) s'obtiennent par des démonstrations analogues.

Fin de la preuve.

3.2 DEFINITION DE LA SEMANTIQUE OPERATIONNELLE D'UN LANGAGE DE PROGRAMMATION

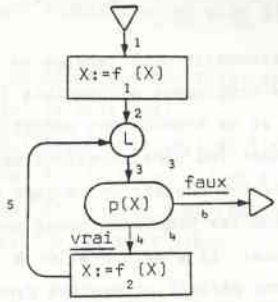
Nous définissons la sémantique opérationnelle d'un langage de programmation simple qui permet de construire des programmes séquentiels itératifs comportant des instructions d'affectation et de branchement conditionnel ou inconditionnel. Nous ignorons volontairement les considérations de méthodologie de la programmation (qui conduiraient à choisir un langage de plus haut niveau que les organigrammes) ainsi que les problèmes posés par la définition de structures d'information complexes. Il s'agit d'aller à l'essentiel qui est de montrer comment un programme définit un système dynamique discret.

3.2.1 Syntaxe abstraite des programmes

Un programme à n variables $X=X_1,\ldots,X_n$ sera représenté par un organigramme connexe fini comportant un seul noeud d'entrée, un seul noeud de sortie et des noeuds d'affectation, de test et de jonction; ces derniers correspondant aux étiquettes du programme. Par exemple, le programme suivant :

```
{1}
        X:=f1(X);
{2}
    L:
{3}
        si p(X) alors
{4}
            X:=f2(X);
{5}
            allera L;
        finsi;
{6}
```

sera représenté par le graphe de programme suivant :



Nous adoptons une présentation intuitive étant admis que les méthodes de définition formelle de la syntaxe sous contexte des langages de programation ainsi que les méthodes de transformation d'une représentation concrète d'un programme (sous forme de chaîne de caractères conforme à une syntaxe concrète) en une représentation abstraite (comme les arbres abstraits ou les organigrammes) sont connues (voir par exemple Lorho[1974]).

3.2.2 Sémantique opérationnelle du langage

3.2.2.1 Ensemble des états *S*

Soit π un programme comportant n variables $X_1,\ldots,X_n$ à valeurs dans U et α points de programmes (ou arcs de l'organigramme correspondant)

$a_1,\ldots,a_\alpha$. Un état $e \in S$ est un couple <m,c> où $m \in U^n$ est l'état de mémoire et $c \in \{a_1,\ldots,a_\alpha,\underline{\text{erreur}}\}$ est l'état de contrôle.

3.2.2.2 Fonction de transition $\bar{\tau}$

Considérant des programmes déterministes, la relation de transition $\tau \in ((S \times S) \to \mathbb{B})$ est définie par une fonction de transition $\bar{\tau}$ par $\tau = \lambda(e_1,e_2).[\bar{\tau}(e_1)=e_2]$.

En notant *dom*(f) le domaine de définition d'une fonction partielle f, la fonction de transition $\bar{\tau}$ induite par le programme π est définie pour tout $m \in U^n$ par les schémas de règles suivants :

- $\bar{\tau}$(<m,erreur>) = <m,erreur>
- Si a_1 est point d'entrée d'une instruction d'affectation X:=f(X) dont le point de sortie est a_2, alors en confondant l'expression syntaxique f à n variables $X_1,\ldots,X_n$ avec la fonction partielle de U^n dans U^n qu'elle dénote, nous avons :
  $\bar{\tau}$(<m,a_1>) = si $m \in dom(f)$ alors <f(m),a_2> sinon <m,erreur> finsi
- Si a est point d'entrée d'une instruction de test p(X) dont les points de sortie vrai et faux sont respectivement a_t et a_f, alors en confondant l'expression syntaxique booléenne p à n variables $X_1,\ldots,X_n$ avec la fonction partielle de U^n dans $\mathbb{B}$={vrai,faux} qu'elle dénote, nous avons :
  $\bar{\tau}$(<m,a>) = si $m \in dom(p)$ alors
  si p(m) alors <m,a_t> sinon <m,a_f> finsi
  sinon
  <m,erreur>
  finsi
- Si a_1 est un point du programme précédant une étiquette L (ou arc d'entrée d'un noeud de jonction L) et a_2 est le point du programme suivant cette étiquette (ou arc de sortie du noeud de jonction L) c'est-à-dire :

$$\begin{array}{l}\{a_1\}\\ \quad L:\\ \{a_2\}\end{array} \quad \text{ou} \quad \begin{array}{l}\{a_1\}\\ \quad \underline{\text{allera}}\ L;\\ \quad \ldots\\ L:\\ \{a_2\}\end{array} \quad \text{ou} \quad \begin{array}{l}\quad\quad L:\\ \{a_2\}\\ \quad \ldots\\ \{a_1\}\\ \quad \underline{\text{allera}}\ L;\end{array}$$

alors :

$\bar{\tau}(<m,a_1>) = <m,a_2>$

- Si a_σ est le point de sortie du programme, alors :

$\bar{\tau}(<m,a_\sigma>) = <m,a_\sigma>$

3.2.2.3 Système dynamique discret total, déterministe et partitionné défini par un programme séquentiel

Un programme séquentiel π comportant n variables $X_1,\ldots,X_n$ à valeurs dans U et α points de programme $a_1,\ldots,a_\alpha$ (où a_ε est le point d'entrée et a_σ le point de sortie) définit un système dynamique discret total, déterministe et partitionné $\pi(S,\tau,\nu_1,\ldots,\nu_\alpha,\nu_\xi,\varepsilon,\sigma,\xi)$ en choisissant S et τ comme précédemment, $\nu_\xi=\lambda<m,c>.(c=\underline{\text{erreur}})$ et pour tout $i=1,\ldots,\alpha$, $\nu_i=\lambda<m,c>.(c=a_i)$.

Nous dirons que l'exécution du programme π pour l'état de mémoire initial m_1

- conduit à une *erreur sémantique* si et seulement si $\{\exists m_2\in U^n : \tau^*(<m_1,a_\varepsilon>,<m_2,\underline{\text{erreur}}>)\}$
- *se termine* si et seulement si $\{\exists m_2\in U^n : \tau^*(<m_1,a_\varepsilon>,<m_2,a_\sigma>)\}$

3.3 SEMANTIQUE DEDUCTIVE EN AVANT

Pour faire l'analyse sémantique d'un programme π, c'est-à-dire étudier le comportement du système dynamique discret défini par π, nous caractérisons l'ensemble des descendants des états d'entrée satisfaisant à une condition d'entrée $\phi\in P_n$ où $P_n=(U^n\to B)$. Il s'agit donc de déterminer $sp(\tau^*)(\nu_\varepsilon \underline{\text{et}} \bar{\phi})$ en notant $\bar{\phi}=\lambda<m,c>.\phi(m)$. L'ensemble S des états étant partitionné, les propositions 3.1.3.0.7 et 3.1.4.0.2 indiquent que $sp(\tau^*)(\nu_\varepsilon \underline{\text{et}} \bar{\phi})$ est isomorphe à la plus petite solution d'un *système d'équations sémantiques en avant* associé au programme π de la forme :

$$\left\{ P_i = (\nu_i \underline{\text{et}} \nu_\varepsilon \underline{\text{et}} \bar{\phi}) \underline{\text{ou}} \left(\underset{j\in \underline{\text{pred}}(i)}{\underline{\text{OU}}} sp(\tau_{ji})(P_j) \right) \right\}_{i=1,\ldots,\alpha,\xi}$$

Pour tout i=1,...,α nous choisirons $P_i \in (U^n \to B)$ ce qui est plus simple que $P_i \in \sigma_i(S \to B) = \sigma_i([\{a_1,\ldots,a_\alpha,\underline{erreur}\} \times U^n] \to B)$ mais équivalent puisque $\sigma_i(S \to B)$ et $(U^n \to B)$ sont isomorphes par l'isomorphisme complet $\iota_i = \lambda\beta.\{\lambda m.[\beta(<m,a_i>)]\}$ dont l'inverse est $\iota_i^{-1} = \lambda\beta.\{\lambda<m,c>.[\beta(m)]\}$.

Les états d'entrée étant exogènes (3.1.1.0.1.(g)), l'ensemble $\underline{pred}(\varepsilon)$ est vide et par conséquent $P_\varepsilon = \phi$. Pour tout $i \neq \varepsilon$, le prédicat (ν_i et ν_ε) est faux (3.1.4.0.1.(d)) et donc

$$P_i = \underset{j \in \underline{pred}(i)}{\underline{OU}} \iota_i[sp(\tau_{ji})(\iota_j^{-1}[P_j])]$$

Il nous faut calculer :

$$\begin{aligned}
&\iota_i[sp(\tau_{ji})(\iota_j^{-1}[P_j])] \\
&= \iota_i[sp(\tau_{ji})(\lambda<m,c>.[P_j(m)])] \\
&= \iota_i[\lambda<m_2,c_2>.\{\exists<m_1,c_1>\in S : P_j(m_1) \ \underline{et}\ \tau_{ji}(<m_1,c_1>,<m_2,c_2>)\}] \\
&= \lambda m_2.\{\exists<m_1,c_1>\in S : P_j(m_1) \ \underline{et}\ \tau_{ji}(<m_1,c_1>,<m_2,a_i>)\} \\
&= \lambda m_2.\{\exists<m_1,c_1>\in S : P_j(m_1) \ \underline{et}\ \nu_j(<m_1,c_1>) \ \underline{et}\ \tau(<m_1,c_1>,<m_2,a_i>) \\
&\qquad\qquad \underline{et}\ \nu_i(<m_2,a_i>)\} \\
&= \lambda m_2.\{\exists<m_1,c_1>\in S : P_j(m_1) \ \underline{et}\ (c_1 = a_j) \ \underline{et}\ \tau(<m_1,c_1>,<m_2,a_i>)\} \\
&= \lambda m_2.\{\exists m_1 \in S : P_j(m_1) \ \underline{et}\ \overline{\tau}(<m_1,a_j>) = <m_2,a_i>\}
\end{aligned}$$

. Si a_i est point de sortie d'une instruction d'affectation X:=f(X) alors $\underline{pred}(i)=\{j\}$ où a_j est le point d'entrée de cette instruction et donc

$$\begin{aligned}
P_i &= \lambda m_2.\{\exists m_1 \in S : P_j(m_1) \ \underline{et}\ [(m_1 \in \mathit{dom}(f)) \ \underline{et}\ (<f(m_1),a_i> = <m_2,a_i>)]\} \\
&= \lambda m_2.\{\exists m_1 \in S : P_j(m_1) \ \underline{et}\ m_1 \in \mathit{dom}(f) \ \underline{et}\ f(m_1) = m_2\} \\
&= \underline{affectation}(f)(P_j) \quad \text{(par définition de } \underline{affectation})
\end{aligned}$$

. Si a_i est point de sortie <u>vrai</u> d'une instruction de test p(X) alors $\underline{pred}(i)=\{j\}$ où a_j est le point d'entrée de cette instruction et donc:

$$\begin{aligned}
P_i &= \lambda m_2.\{\exists m_1 \in S : P_j(m_1) \ \underline{et}\ \overline{\tau}(<m_1,a_j>) = <m_2,a_i>\} \\
&= \lambda m_2.\{\exists m_1 \in S : P_j(m_1) \ \underline{et}\ (m_1 \in \mathit{dom}(f)) \ \underline{et}\ p(m_1) \ \underline{et}\ (<m_1,a_i> = <m_2,a_i>)\} \\
&= \lambda m_2.\{P_j(m_2) \ \underline{et}\ (m_2 \in \mathit{dom}(p)) \ \underline{et}\ p(m_2)\} \\
&= \underline{test}(p)(P_j) \quad \text{(par définition de } \underline{test})
\end{aligned}$$

. De même, si a_i est point de sortie <u>faux</u> d'une instruction de test p(X) et a_j le point d'entrée de cette instruction alors:

$$P_i = \underline{test}(\underline{non}(p))(P_j)$$

. Si a_i est le point de programme suivant une étiquette précédée par les points de programme a_j tels que $j \in \underline{pred}(i)$ alors:

$$P_i = \underset{j \in \underline{pred}(i)}{\underline{OU}} \iota_i[sp(\tau_{ji})(\iota_j^{-1}[P_j])]$$

$$= \underset{j \in \underline{pred}(i)}{\underline{OU}} \lambda m_2.\{\exists m_1 \in S : P_j(m_1) \ \underline{et}\ \overline{\tau}(<m_1,a_j>)=<m_2,a_i>\}$$

$$= \underset{j \in \underline{pred}(i)}{\underline{OU}} \lambda m_2.\{\exists m_1 \in S : P_j(m_1) \ \underline{et}\ (<m_1,a_i>=<m_2,a_i>)\}$$

$$= \underset{j \in \underline{pred}(i)}{\underline{OU}} P_j$$

. Comme les erreurs à l'exécution correspondent à un arrêt du programme, le programme est sans reprise d'erreur et les états erronés sont stables. Par conséquent, pour tout i=1,...,α, P_i est indépendant de P_ξ. Dans la résolution du système d'équations sémantiques en avant associé au programme, nous pourrons donc ignorer l'équation définissant P_ξ qui, éventuellement, sera évalué en dernier, si nécessaire, pour l'application considérée. Dans ce cas:

$$P_\xi = \underset{j \in \underline{pred}(\xi)}{\underline{OU}} \iota_\xi[sp(\tau_{j\xi})(\iota_j^{-1}[P_j])]$$

$$= \underset{j \in \underline{pred}(\xi)}{\underline{OU}} \lambda m_2.\{\exists m_1 \in U^n : P_j(m_1) \ \underline{et}\ \overline{\tau}(<m_1,a_j>)=<m_2,\underline{erreur}>\}$$

$$= P_\xi \ \underline{ou} \underset{j \in \underline{at}(\pi)}{\underline{OU}} \lambda m_2.[P_j(m_2) \ \underline{et}\ m_2 \notin dom(\underline{expr}(j))]$$

en notant $\underline{at}(\pi)$ l'ensemble des $j \in [1,\alpha]$ tels que a_j est le point d'entrée d'une instruction d'affectation ou d'une instruction de test dans le programme π et $\underline{expr}(j)$, la fonction f de l'instruction d'affectation X:=f(X) où le prédicat p de l'instruction de test p(X) dont le point d'entrée est a_j.

En résumé:

DEFINITION 3.3.0.1. *Système d'équations sémantiques en avant associé à un programme et à une spécification d'entrée.*

Soient π un programme comportant n variables à valeurs dans U et α points de programme $a_1,\ldots,a_\alpha$ (où a_ε est le point d'entrée) et une spécification d'entrée $\phi \in \mathcal{P}^n = (U^n \to \mathcal{B})$. Soient $P_1,\ldots,P_\alpha$ des variables à valeurs dans $\mathcal{P}_n$ alors le *système d'équations sémantiques en avant* $P=F_\pi(\phi)(P)$ *associé à* (π,ϕ)

est défini par les règles suivantes:

- Si a_i est le point d'entrée du programme alors $P_i = \phi$
- Si a_j est le point d'entrée d'une instruction d'affectation X:=f(X) dont le point de sortie est a_i alors $P_i = \underline{affectation}(f)(P_j)$ où
  $\underline{affectation} = \lambda f.\{\lambda P.[\lambda m_2.(\exists m_1 \epsilon U^n : P(m_1) \ \underline{et} \ (m \epsilon dom(f)) \ \underline{et} \ (m_2 = f(m_1)))]\}$
- Si a_j est le point d'entrée d'une instruction de test p(X) et a_i le point de sortie $\underline{vrai}$ (respectivement $\underline{faux}$) de cette instruction alors $P_i = \underline{test}(p)(P_j)$ (respectivement $P_i = \underline{test}(\underline{non}(p))(P_j)$) où
  $\underline{test} = \lambda p.\{\lambda P.[\lambda m.(P(m) \ \underline{et} \ (m \epsilon dom(p)) \ \underline{et} \ p(m))]\}$
- Si a_i est un point de programme suivant une étiquette précédée par les points de programme $a_{j_1},\ldots,a_{j_k}$, alors $P_i = \underline{\underline{OU}}_{\ell=1}^{k} P_{j_\ell}$

Nous déduisons de 3.1.3.0.5.(a), 3.1.3.0.7 et 3.1.4.0.2 la

PROPOSITION 3.3.0.2 *Propriété du plus petit point fixe du système d'équations sémantiques en avant.*

L'opérateur $F_\pi(\phi)$ sur $(P_n)^\alpha$ est un morphisme complet pour la disjonction. Le plus petit point fixe $(P_1,\ldots,P_\alpha)$ de $F_\pi(\phi)$ est tel que pout tout $i \epsilon [1,\alpha]$ nous avons

$$P_i = \lambda m_2.\{\exists m_1 \epsilon U^n : \phi(m_1) \ \underline{et} \ \tau^*(<m_1,a_\varepsilon>,<m_2,a_i>)\}$$

PROPOSITION 3.3.0.3 *Conjonction et disjonction de spécifications d'entrée*

$\lambda\phi.[\mathcal{Ifp}(F_\pi(\phi))]$ est un morphisme complet pour la disjonction. Si π est injectif, c'est un morphisme complet pour la conjonction.

3.4 TECHNIQUES D'ANALYSE DE PROGRAMMES BASEES SUR LA SEMANTIQUE DEDUCTIVE EN AVANT

Nous utilisons les résultats du paragraphe 3.3 pour justifier la méthode de Floyd et Naur de vérification de la correction partielle d'un programme, puis, pour étendre cette méthode à la vérification de la correction totale d'un programme. Nous justifions ensuite le critère de terminaison des programmes de Katz et Manna. Nous illustrons les techniques d'analyse des

conditions de terminaison sans fautes, de non-terminaison et d'éxécution incorrecte d'un programme basées sur la sémantique déductive en avant. Nous montrons également comment utiliser la sémantique déductive en avant pour caractériser en chaque point d'un programme l'ensemble des descendants des états initiaux satisfaisant à une condition d'entrée. Enfin nous montrons que l'exécution symbolique d'un programme consiste à résoudre le système d'équations sémantiques en avant associé au programme, en utilisant une stratégie itérative chaotique.

3.4.1 Justification de la méthode de Floyd et Naur de vérification de la correction partielle d'un programme

Floyd[1967] et Naur[1966] ont justifié leur méthode de vérification de correction partielle des programmes, par un raisonnement basé sur la sémantique opérationnelle. La sémantique déductive en avant conduit à une preuve plus élégante.

Soient π un programme comportant n variables à valeurs dans U, α points de programme $a_1,\ldots,a_\alpha$ (les points d'entrée et de sortie étant respectivement a_ε et a_σ) et $P_n=(U^n\to\mathcal{B})$. Une preuve de correction partielle du programme π pour les spécifications d'entrée ϕ et de sortie ψ, consiste à montrer que:

$$\{\forall m_2\in U^n,\ [\exists m_1\in U^n : \phi(m_1) \underline{\text{ et }} \tau^*(<m_1,a_\varepsilon>,<m_2,a_\sigma>)] \Rightarrow [\psi(m_2)]\}$$

Si on peut *inventer* $P\in(P_n)^\alpha$ qui soit un post-point fixe de $F_\pi(\psi)$ et tel que $\{P_\sigma \Rightarrow \psi\}$, alors le théorème 2.5.3.0.2 montre que $\{lfp(F_\pi(\phi)) \Rightarrow P\}$ et donc $\{[lfp(F_\pi(\phi))]_\sigma \Rightarrow\psi\}$ et la proposition 3.3.0.2 implique que le programme π est partiellement correct pour (ϕ,ψ) puisque nous avons:

$$\lambda m_2.\{\exists m_1\in U^n : \phi(m_1) \underline{\text{ et }} \tau^*(<m_1,a_\varepsilon>,<m_2,a_\sigma>)\} \Rightarrow \psi$$

En pratique P est spécifié en donnant seulement les invariants de boucle puisque les autres s'en déduisent en remplaçant les invariants de boucle par leur valeur dans le système d'équations.

Nous avons montré que la méthode de Floyd-Naur est valide mais en même temps qu'elle est complète, c'est-à-dire (de Bakker & Meertens[1975]), que si π est partiellement correct pour (ϕ,ψ), il existe toujours $P\in(P_n)^\alpha$ permettant de le prouver avec cette méthode, c'est-à-dire de trouver P tel que $\{[F_\pi(\phi)(P) \Rightarrow P] \underline{\text{ et }} [P_\sigma \Rightarrow \psi]\}$. D'après ce qui précède, il suffit en effet de choisir $P=lfp(F_\pi(\phi))$.

3.4.2. Extension de la méthode de Floyd et Naur à la vérification de la correction totale d'un programme.

Une preuve de correction totale d'un programme π pour une spécification d'entrée ϕ et de sortie ψ, consiste à montrer que:

$$\{\nu_\varepsilon \underline{\text{ et }} \phi\} \Rightarrow \{wp(\tau^*)(\nu_\sigma \underline{\text{ et }} \psi)\}$$

La proposition 3.1.3.0.8 nous permet d'utiliser un système d'équations sémantiques en avant pour faire cette preuve puisque de manière équivalente, il s'agit de démontrer que:

$$\{\nu_\varepsilon \underline{\text{ et }} \phi\} \Rightarrow \{\lambda\overline{e}.[\exists e_1 \in S : \nu_\sigma(e_1) \underline{\text{ et }} \psi(e_1) \underline{\text{ et }} lfp(\lambda\alpha.[\lambda e.(e=\overline{e}) \underline{\text{ ou }} sp(\tau)(\alpha)])(e_1)]\}$$

c'est-à-dire à un isomorphisme près:

$$\phi \Rightarrow \lambda\overline{m}.\{\exists m_1 \in \mathcal{U}^n : [lfp(F_\pi(\lambda m.(m=\overline{m})))]_\sigma(m_1) \underline{\text{ et }} \psi(m_1)\}$$

En particulier, la plus faible spécification d'entrée garantissant la terminaison du programme π est

$$\lambda\overline{m}.\{\exists m_1 \in \mathcal{U}^n : [lfp(F_\pi(\lambda m.(m=\overline{m})))]_\sigma(m_1)\}$$

3.4.3. Justification du critère de terminaison des programmes dû à Katz et Manna.

En utilisant nos notations, le critère de terminaison des programmes donné par Katz & Manna[1976] est le suivant:
"si pour tout $P \in (\mathcal{P}_n)^\alpha$ tel que P est un ensemble d'invariants du programme π, on peut montrer que $\{\forall\overline{m} \in \mathcal{U}^n, \phi(\overline{m}) \Rightarrow P_\sigma(\overline{m})\}$, alors l'exécution du programme π se termine pour toute donnée satisfaisant ϕ". Dans cette formulation, un invariant P_i au point a_i du programme π est informellement défini comme "une assertion sur les variables qui est vraie des valeurs courantes des variables chaque fois que le point i est atteint durant une exécution commençant dans l'état initial $\overline{m}$". La méthode utilisée pour vérifier qu'une assertion P est un invariant est celle de Floyd-Naur, c'est-à-dire d'après le paragraphe 3.4.1, on vérifie que $P \Leftarrow F_\pi(\lambda x.(x=\overline{m}))(P)$. Formellement le critère de terminaison de Katz et Manna est donc:

$$\phi \Rightarrow \lambda\overline{m}.[\exists m_1 \in \mathcal{U}^n : \underline{ET}\{P_\sigma(m_1) : P \Leftarrow F_\pi(\lambda x.(x=\overline{m}))(P)\}]$$

D'après le théorème 2.5.3.0.2, nous obtenons:

$$\underline{ET}\{P : P <= F_\pi(\lambda x.(\overline{x=m}))(P)\} = lfp(F_\pi(\lambda x.(x=\overline{m})))$$

et nous avons retrouvé notre critère.

Katz et Manna ont remarqué que leur critère n'est pas utilisable en pratique puisqu'en général il y a une infinité d'invariants associés au programme. Il est en effet rarement possible de découvrir la forme générale des post-points fixes de $F_\pi(\lambda x.(x=\overline{m}))$.

Au contraire,notre critère repose sur $lfp(F_\pi(\lambda x.(x=\overline{m})))$ qui peut être calculé par approximations successives en inventant le terme général de la suite, en montrant par récurrence que la forme de ce terme général est bien choisie et en passant à la limite pour le ω-ème terme (théorème 2.7.0.1). Ce procédé est évidemment réservé aux calculs à la main et dans bien des cas le calcul exact n'est pas faisable car l'invention du terme général de la suite est très difficile. Nous donnerons dans les paragraphes 3.4 et 3.6 quelques exemples de résolution exacte des équations sémantiques. Nous étudierons ensuite au chapitre 4 des méthodes constructives d'approximation de ces solutions exactes.

3.4.4 Détermination des descendants des états d'entrée et des conditions de terminaison, non-terminaison et d'exécution erronée d'un programme

Considérons le programme suivant, comportant une variable x à valeurs entières comprises dans l'intervalle [-b+1,b] de définition des entiers sur une machine:

```
{1}
    tantque x≥1000 faire
{2}
        x:=x+α;
{3}
    refaire;
{4}
```

Nous supposerons que α est une constante entière positive. Par définition de l'instruction d'itération "tantque", ce programme est équivalent au programme:

```
{1}
   L :
        si x≥1000 alors
{2}
            x:=x+α;
{3}
            allera L ;
        finsi;
{4}
```

Le système d'équations sémantiques en avant associé à ce dernier programme et à la spécification d'entrée $\lambda x.(x=\overline{x})$ est:

$$\begin{cases} P_1 = \lambda x.[x=\overline{x}] \\ P_2 = \underline{\text{test}}(\lambda x.[x\geq 1000])(P_1 \ \underline{\text{ou}} \ P_3) \\ P_3 = \underline{\text{affectation}}(\lambda x.[x+\alpha])(P_2) \\ P_4 = \underline{\text{test}}(\lambda x.[x<1000])(P_1 \ \underline{\text{ou}} \ P_3) \end{cases}$$

L'équation définissant P_2 est plus simplement:

$$\begin{aligned} P_2 &= \underline{\text{test}}(\lambda x.[x\geq 1000])(\lambda x.(x=\overline{x}) \ \underline{\text{ou}} \ \underline{\text{affectation}}(\lambda x.[x+\alpha])(P_2)) \\ &= \lambda x.\{(-b+1\leq x\leq b) \ \underline{\text{et}} \ (1000\leq x) \ \underline{\text{et}} \ [(x=\overline{x}) \ \underline{\text{ou}} \\ &\qquad (\exists y : (-b+1\leq y+\alpha\leq b) \ \underline{\text{et}} \ (x=y+\alpha) \ \underline{\text{et}} \ P_2(y))]\} \\ &= \lambda x.\{(1000\leq x\leq b) \ \underline{\text{et}} \ [(x=\overline{x}) \ \underline{\text{ou}} \ P_2(x-\alpha)]\} \end{aligned}$$

Le plus petit point fixe est obtenu par approximations successives en partant de l'infimum $\lambda x.[\underline{\text{faux}}]$ (théorème 2.5.3.0.2):

$$\begin{aligned} P_2^0 &= \lambda x.[\underline{\text{faux}}] \\ P_2^1 &= \lambda x.\{(1000\leq x\leq b) \ \underline{\text{et}} \ (x=\overline{x})\} \\ P_2^2 &= \lambda x.\{(1000\leq x\leq b) \ \underline{\text{et}} \ [(x=\overline{x}) \ \underline{\text{ou}} \ ((1000\leq x-\alpha\leq b) \ \underline{\text{et}} \ (x-\alpha=\overline{x}))]\} \\ &= \lambda x.\{(1000\leq \overline{x}) \ \underline{\text{et}} \ (x\leq b) \ \underline{\text{et}} \ [(x=\overline{x}) \ \underline{\text{ou}} \ (x=\overline{x}+\alpha)]\} \end{aligned}$$

Supposons pour le pas d'induction que:

$$P_2^k = \lambda x.\{(1000\leq \overline{x}) \ \underline{\text{et}} \ (x\leq b) \ \underline{\text{et}} \ \underline{\underset{j=0}{\overset{k-1}{\text{OU}}}} \ (x=\overline{x}+j\alpha)\}$$

et vérifions que P_2^{k+1} est de la même forme:

$$\begin{aligned} P_2^{k+1} &= \lambda x.\{(1000\leq x\leq b) \ \underline{\text{et}} \ [(x=\overline{x}) \ \underline{\text{ou}} \ P_2^k(x-\alpha)]\} \\ &= \lambda x.\{(1000\leq x\leq b) \ \underline{\text{et}} \ [(x=\overline{x}) \ \underline{\text{ou}} \ ((1000\leq \overline{x}) \ \underline{\text{et}} \ (x\leq b+\alpha) \ \underline{\text{et}} \\ &\qquad\qquad \underline{\underset{j=0}{\overset{k-1}{\text{OU}}}} \ (x=\overline{x}+(j+1)\alpha))]\} \end{aligned}$$

$$= \lambda x.\{(1000 \leq \bar{x}) \text{ et } (x \leq b) \text{ et } \underset{j=0}{\overset{k}{OU}} (x = \bar{x} + j\alpha)\}$$

Par induction sur k nous avons trouvé le terme général de la séquence, de sorte que le plus petit point fixe s'obtient en passant à la limite (théorème 2.7.0.1 et proposition 3.3.0.2):

$$P_2^{\omega} = \underset{k \in \omega}{OU} P_2^k = \lambda x.\{(1000 \leq \bar{x}) \text{ et } (x \leq b) \text{ et } (\exists j \geq 0 : x = \bar{x} + j\alpha)\}$$

Les autres composantes s'obtiennent en remplaçant P_2 par P_2^{ω} dans le système d'équations original ce qui donne:

$$\begin{cases} P_1^{\omega} = \lambda x.\{x = \bar{x}\} \\ P_2^{\omega} = \lambda x.\{(1000 \leq \bar{x}) \text{ et } (x \leq b) \text{ et } (\exists j \geq 0 : x = \bar{x} + j\alpha)\} \\ P_3^{\omega} = \lambda x.\{(1000 \leq \bar{x}) \text{ et } (x \leq b) \text{ et } (\exists j \geq 1 : x = \bar{x} + j\alpha)\} \\ P_4^{\omega} = \lambda x.\{(x = \bar{x}) \text{ et } (-b+1 \leq x < 1000)\} \end{cases}$$

D'après la proposition 3.3.0.2 nous avons obtenu en chaque point du programme une caractérisation de l'ensemble des valeurs possibles de x, au cours d'une exécution quelconque du programme partant d'une valeur initiale de x satisfaisant à la spécification d'entrée $\phi = \lambda x.(x = \bar{x})$.

Nous avons vu au paragraphe 3.4.2 que la plus faible spécification d'entrée garantissant la terminaison du programme sans conduire à une erreur sémantique est:

$$\lambda \bar{x}.\{\exists x_1 : P_4^{\omega}(x_1)\}$$
$$= \lambda \bar{x}.\{-b+1 \leq \bar{x} < 1000\}$$

L'ensemble des états d'entrée conduisant à une erreur sémantique est caractérisé par:

$$\lambda \bar{x}.\{\exists x_1 : P_{\xi}^{\omega}(x_1)\}$$
$$= \lambda \bar{x}.\{\exists x : [(P_1^{\omega}(x) \text{ ou } P_3^{\omega}(x)) \text{ et } \text{non}(-b+1 \leq x \leq b)] \text{ ou } [P_2^{\omega}(x) \text{ et } \text{non}(-b+1 \leq x+\alpha \leq b)]\}$$
$$= \lambda \bar{x}.\{[(\bar{x} < -b+1) \text{ ou } (b < \bar{x}) \text{ ou } [(\alpha \neq 0) \text{ et } (1000 \leq \bar{x} \leq b)]]\}$$

L'ensemble des états d'entrée pour lesquels le programme ne se termine pas est caractérisé par:

$$\lambda\overline{x}.\{\exists x_1 : \underline{non}(P^{\omega}_{\xi}(x_1)\ \underline{ou}\ P^{\omega}_{4}(x_1))\}$$
$$= \lambda\overline{x}.\{(1000\leq x\leq b)\ \underline{et}\ (\alpha=0)\}$$

3.4.5 Exécution symbolique

Soient π un programme comportant n variables $x=(x_1,\ldots,x_n)$ et $\overline{x}=(\overline{x}_1,\ldots,\overline{x}_n)$ des constantes de Skolem associées à des éléments indéterminés mais fixes de U^n. Nous montrons que l'exécution symbolique du programme π consiste à calculer la plus petite solution du système d'équations sémantiques en avant $P=F_{\pi}(\lambda x.(\phi(\overline{x})\ \underline{et}\ (x=\overline{x})))(P)$ associé à π, en utilisant une stratégie chaotique quelconque (Cousot & Cousot[1977e]).

Soit $P^0,\ldots,P^n,\ldots,P^{\omega}$ une itération chaotique croissante partant de l'infimum P^0 de $(P_n)^{\alpha}$ et définie par $F_{\pi}(\lambda x.(\phi(\overline{x})\ \underline{et}\ (x=\overline{x})))$. Chaque terme P^n_i s'écrit sous la forme :

$$\lambda x.\{\underset{j\in\Delta}{\underline{OU}}\ [Q_j(\overline{x})\ \underline{et}\ x=E_j(\overline{x})]\}$$

où Q_j et E_j sont des expressions formelles dépendant de la valeur initiale $\overline{x}$ des variables et où aucune des variables $x_1,\ldots,x_n$ n'apparaît comme variable libre. Par convention, cette expression se réduit à $\lambda x.\underline{faux}$ quand $\Delta=\emptyset$.

Pour démontrer que P^n_i s'écrit sous cette forme, il suffit de remarquer que $\underline{affectation}(f)$ et $\underline{test}(p)$ sont stricts et que :

- $\underline{affectation}(f)(\lambda x.\{\underset{j\in\Delta}{\underline{OU}}\ [(Q_j(\overline{x})\ \underline{et}\ x=E_j(\overline{x})]\})$
$$= \lambda x.\{\underset{j\in\Delta}{\underline{OU}}\ [(Q_j(\overline{x})\ \underline{et}\ E_j(\overline{x})\in dom(f))\ \underline{et}\ x=f(E_j(\overline{x}))]\}$$
- $\underline{test}(p)(\lambda x.\{\underset{j\in\Delta}{\underline{OU}}\ [Q_j(\overline{x})\ \underline{et}\ x=E_j(\overline{x})]\})$
$$= \lambda x.\{\underset{j\in\Delta}{\underline{OU}}\ [(Q_j(\overline{x})\ \underline{et}\ E_j(\overline{x})\in dom(f)\ \underline{et}\ p(E_j(\overline{x})))\ \underline{et}\ x=E_j(\overline{x})]\}$$
- $\underset{k=1}{\overset{\ell}{\underline{OU}}}(\lambda x.\{\underset{j_k\in\Delta_k}{\underline{OU}}\ [Q_{j_k}(\overline{x})\ \underline{et}\ x=E_{j_k}(\overline{x})]\})$
$$= \lambda x.(\underline{OU}\{[Q_{j_k}(\overline{x})\ \underline{et}\ x=E_{j_k}(\overline{x})] : j_k\in\overset{\ell}{\underset{k=1}{\cup}}\Delta_k\})$$

Dans l'expression :

$$P = \underset{j\in\Delta}{\text{OU}}\ C_j \quad \text{avec} \quad C_j = \lambda x.\{Q_j(\bar{x}) \ \underline{\text{et}}\ x=E_j(\bar{x})\}$$

on peut interpréter chaque C_j comme décrivant un chemin d'exécution de π où $Q_j(\bar{x})$ décrit les conditions qui doivent être vérifiées pour qu'une exécution de π partant de a_ε avec l'état initial $\bar{x}$ des variables passe par le point de programme a_i avec des valeurs des variables x égales à $E_j(\bar{x})$.

De façon équivalente, nous noterons P sous la forme d'un *contexte symbolique* $\{<Q_j(\bar{x}),(E_j)_1(\bar{x}),\ldots,(E_j)_n(\bar{x})> : j\in\Delta\}$ où $\emptyset$ correspond à $\lambda x.[\underline{\text{faux}}]$.

Considérons par exemple, le programme suivant :

```
{1}
    tantque x≥y faire
{2}
        x:=x-y;
{3}
    refaire;
{4}
```

Le système sémantique en avant correspondant est :

$$\begin{cases} P_1 = \{<\underline{\text{vrai}},\bar{x},\bar{y}>\} \\ P_2 = \underline{\text{test}}(\lambda(x,y).[x\geq y])(P_1 \ \underline{\text{ou}}\ P_3) \\ P_3 = \underline{\text{affectation}}(\lambda(x,y).[x-y,y])(P_2) \\ P_4 = \underline{\text{test}}(\lambda(x,y).[x<y])(P_1 \ \underline{\text{ou}}\ P_3) \end{cases}$$

Nous supposerons que le programme porte sur des entiers, (ce qui apporte la simplification que x-y est toujours défini).

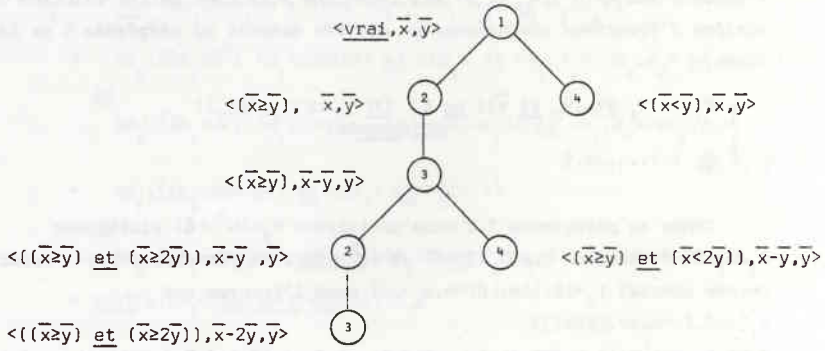
Choisissant une statégie chaotique correspondant à l'ordre d'exécution du programme, nous obtenons les premiers termes de l'itération :

$$\left[P_i^0 = \emptyset \quad (i=1,\ldots,4) \right.$$

$$\begin{cases} P_1^1 = \{<\underline{\text{vrai}},\bar{x},\bar{y}>\} \\ P_2^1 = \underline{\text{test}}(\lambda(x,y).[x\geq y])(P_1^1 \ \underline{\text{ou}}\ P_3^0) = \{<(\bar{x}\geq\bar{y}),\bar{x},\bar{y}>\} \\ P_3^1 = \underline{\text{affectation}}(\lambda(x,y).[x-y,y])(P_2^1) = \{<(\bar{x}\geq\bar{y}),\bar{x}-\bar{y},\bar{y}>\} \\ P_4^1 = \underline{\text{test}}(\lambda(x,y).[x<y])(P_1^1 \ \underline{\text{ou}}\ P_3^0) = \{<(\bar{x}<\bar{y}),\bar{x},\bar{y}>\} \end{cases}$$

$$\begin{cases} P_1^2 = \{<\underline{vrai},\overline{x},\overline{y}>\} \\ P_2^2 = \{<(\overline{x}\geq\overline{y}),\overline{x},\overline{y}> \ , \ <((\overline{x}\geq\overline{y}) \ \underline{et} \ (\overline{x}\geq 2\overline{y})),\overline{x}-\overline{y},\overline{y}>\} \\ P_3^2 = \{<(\overline{x}\geq\overline{y}),\overline{x}-\overline{y},\overline{y}> \ , \ <((\overline{x}\geq\overline{y}) \ \underline{et} \ (\overline{x}\geq 2\overline{y})),\overline{x}-2\overline{y},\overline{y}>\} \\ P_4^2 = \{<(\overline{x}<\overline{y}),\overline{x},\overline{y}> \ , \ <(\overline{x}<2\overline{y}),\overline{x}-\overline{y},\overline{y}>\} \end{cases}$$

Après deux itérations, nous avons construit l'*arbre d'exécution symbolique* (Hantler & King[1976]) suivant :



Nous avons représenté le contexte symbolique P_i associé au point a_i du programme π, par l'ensemble des chemins d'exécution associés aux noeuds étiquetés i dans l'arbre d'exécution symbolique. Nous aurions pu également, représenter le contexte symbolique P_i par le sous-arbre maximal de l'arbre ci-dessus dont les feuilles sont étiquetées i.

Il est clair que sans hypothèses particulières sur $\overline{x}$ et $\overline{y}$, les termes suivants de l'itération chaotique correspondraient à des arbres symboliques de hauteur de plus en plus grande jusqu'à obtenir l'arbre de hauteur infinie correspondant à $lfp(F_\pi(\lambda x.(x=\overline{x})))$.

3.5 SEMANTIQUE DEDUCTIVE EN ARRIERE

Pour faire l'analyse sémantique d'un programme π c'est-à-dire étudier le comportement du système dynamique discret défini par π nous avons vu au paragraphe 3.1 qu'il faut caractériser l'ensemble des ascendants des états de sortie satisfaisant à une condition de sortie $\psi \epsilon P_n$ où $P_n=(U^n \to B)$. Il s'agit donc de déterminer $wp(\tau^*)(\nu_\sigma \underline{\text{ et }} \overline{\psi})$ en notant $\overline{\psi}=\lambda<m,c>.\psi(m)$. L'ensemble S des états étant partitionné, les propositions 3.1.3.0.7 et 3.1.4.0.3 indiquent que $wp(\tau^*)(\nu_\sigma \underline{\text{ et }} \overline{\psi})$ est isomorphe à la plus petite solution d'un *système d'équations sémantiques en arrière* associé au programme π de la forme :

$$\begin{cases} P_i = (\nu_i \underline{\text{ et }} (\nu_\sigma \underline{\text{ et }} \overline{\psi})) \underline{\text{ ou }} (\underset{j \epsilon \underline{succ}(i)}{\underline{OU}} wp(\tau_{ij})(P_j)) \\ i = 1,\ldots,\alpha,\xi \end{cases}$$

Comme au paragraphe 3.3 nous choisirons $P_i \epsilon (U^n \to B)$ plutôt que $P_i \epsilon \sigma_i(S \to B)$ sachant que $\sigma_i(S \to B)$ et $(U^n \to B)$ sont isomorphes par l'isomorphisme complet $\iota_i=\lambda\beta.\{\lambda m.[\beta(<m,a_i>)]\}$ dont l'inverse est $\iota_i^{-1}=\lambda\beta.\{\lambda<m,c>.[\beta(m)]\}$.

Pour tout $i \neq \sigma$ le prédicat $(\nu_i \underline{\text{ et }} \nu_\sigma)$ est faux (3.1.4.0.1.(d)) et donc :

$$P_i = \underset{j \epsilon \underline{succ}(i)}{\underline{OU}} \iota_i[wp(\tau_{ij})(\iota_j^{-1}[P_j])]$$

avec

$$\begin{aligned} &\iota_i[wp(\tau_{ij})(\iota_j^{-1}[P_j])] \\ &= \iota_i[wp(\tau_{ij})(\lambda<m,c>.[P_j(m)])] \\ &= \iota_i[\lambda<m_1,c_1>.\{\exists<m_2,c_2>\epsilon S : \tau_{ij}(<m_1,c_1>,<m_2,c_2>) \underline{\text{ et }} P_j(m_2)\}] \\ &= \lambda m_1.\{\exists<m_2,c_2>\epsilon S : \tau_{ij}(<m_1,a_i>,<m_2,c_2>) \underline{\text{ et }} P_j(m_2)\} \\ &= \lambda m_1.\{\exists<m_2,c_2>\epsilon S : \nu_i(<m_1,a_i>) \underline{\text{ et }} \tau(<m_1,a_i>,<m_2,c_2>) \underline{\text{ et }} \\ &\qquad\qquad \nu_j(<m_2,c_2>) \underline{\text{ et }} P_j(m_2)\} \\ &= \lambda m_1.\{\exists<m_2,c_2>\epsilon S : \tau(<m_1,a_i>,<m_2,c_2>) \underline{\text{ et }} (c_2=a_j) \underline{\text{ et }} P_j(m_2)\} \\ &= \lambda m_1.\{\exists m_2 \epsilon U^n : \overline{\tau}(<m_1,a_i>)=<m_2,a_j> \underline{\text{ et }} P_j(m_2)\} \end{aligned}$$

. Si a_i est le point d'entrée d'une instruction d'affectation X:=f(X) alors $\underline{succ}(i)=\{j\}$ où a_j est le point de sortie de cette instruction et donc :

$$P_i = \lambda m_1.\{\exists m_2 \in U^n : \overline{\tau}(\langle m_1, a_i\rangle) = \langle m_2, a_j\rangle \text{ et } P_j(m_2)\}$$
$$= \lambda m_1.\{\exists m_2 \in U^n : (m_1 \in dom(f)) \text{ et } (f(m_1) = m_2) \text{ et } P_j(m_2)\}$$
$$= \lambda m_1.\{(m_1 \in dom(f)) \text{ et } P_j(f(m_1))$$
$$= \text{affectation-1}(f)(P_j) \quad \text{(par définition de affectation-1)}$$

. Si a_i est le point d'entrée d'une instruction de test p(X) alors succ(i)={t,f} où a_t et a_f sont respectivement les points de sortie vrai et faux du test et donc :

$$P_i = \underset{j \in \{t,f\}}{OU} \lambda m_1.\{\exists m_2 \in U^n : \overline{\tau}(\langle m_1, a_i\rangle) = \langle m_2, a_j\rangle \text{ et } P_j(m_2)\}$$
$$= \lambda m_1.\{\exists m_2 \in U^n : (m_1 \in dom(p)) \text{ et } p(m_1) \text{ et } \langle m_1, a_t\rangle = \langle m_2, a_t\rangle \text{ et } P_t(m_2)\}$$
$$\text{ou}$$
$$\lambda m_1.\{\exists m_2 \in U^n : (m_1 \in dom(p)) \text{ et non}(p(m_1)) \text{ et } \langle m_1, a_f\rangle = \langle m_2, a_f\rangle \text{ et } P_f(m_2)\}$$
$$= \lambda m_1.\{(m_1 \in dom(p)) \text{ et } p(m_1) \text{ et } P_t(m_1)\}$$
$$\text{ou}$$
$$\lambda m_1.\{(m_1 \in dom(p)) \text{ et non}(p(m_1)) \text{ et } P_f(m_1)\}$$
$$= \text{test}(p)(P_t) \text{ ou test}(\text{non}(p))(P_f)$$

. Si a_i est un point du programme précédant une étiquette L (séquentiellement ou par un branchement inconditionnel), alors succ(i)={j} où a_j est le point du programme suivant L et donc :

$$P_i = \lambda m_1.\{\exists m_2 \in U^n : \overline{\tau}(\langle m_1, a_i\rangle) = \langle m_2, a_j\rangle \text{ et } P_j(m_2)\}$$
$$= \lambda m_1.\{\exists m_2 \in U^n : \langle m_1, a_j\rangle = \langle m_2, a_j\rangle \text{ et } P_j(m_2)\}$$
$$= P_j$$

. Les états de sortie étant stables nous avons :

$$\text{succ}(\sigma) = \{j : (\exists e_1, e_2 \in S : \tau_{\sigma j}(e_1, e_2))\}$$
$$= \{j : (\exists e_1, e_2 \in S : \nu_\sigma(e_1) \text{ et } \tau(e_1, e_2) \text{ et } \nu_j(e_2))\}$$
$$= \{j : (\exists e_1, e_2 \in S : \nu_\sigma(e_1) \text{ et } \tau(e_1, e_2) \text{ et } (e_1 = e_2) \text{ et } \nu_j(e_2))\}$$
$$= \{j : (\exists e_1 \in S : \nu_\sigma(e_1) \text{ et } \tau(e_1, e_1) \text{ et } \nu_j(e_1))\}$$
$$= \{\sigma\} \text{ (d'après 3.1.4.0.1.(d))}$$

et il vient :

$P_\sigma = \iota_\sigma[\nu_\sigma \text{ et } \iota_\sigma^{-1}(\psi)] \text{ ou } \iota_\sigma[wp(\tau_{\sigma\sigma})(\iota_\sigma^{-1}[P_\sigma])]$

$= (\iota_\sigma[\nu_\sigma] \text{ et } \psi) \text{ ou } \lambda m_1.\{\exists m_2 \epsilon U^n : \overline{\tau}(<m_1,a_\sigma>) = <m_2,a_\sigma> \text{ et } P_\sigma(m_2)\}$

$= (\lambda m.(\nu_\sigma(<m,a_\sigma>)) \text{ et } \psi) \text{ ou } \lambda m_1.\{\exists m_2 \epsilon U^n : (m_1 = m_2) \text{ et } P_\sigma(m_2)\}$

$= \psi \text{ ou } P_\sigma$

- Enfin les états erronés étant stables nous aurons $P_\xi = P_\xi$ que nous pourrons donc ignorer.

En résumé :

DEFINITION 3.5.0.1 *Système d'équations sémantiques en arrière associé à un programme et une spécification de sortie*

Soient π un programme comportant n variables à valeurs dans U, α points de programme $a_1,\ldots,a_\alpha$ (a_σ est le point de sortie) et une spécification de sortie $\psi \epsilon P_n = (U^n \to \mathcal{B})$. Soient $P_1,\ldots,P_\alpha$ des variables à valeurs dans P_n, alors le *système d'équations sémantiques en arrière* $P = B_\pi(\psi)(P)$ *associé à* (π,ψ) est défini par les règles suivantes :

- Si a_i est le point de sortie du programme alors $P_i = \psi$
- Si a_i est le point d'entrée d'une instruction d'affectation X:=f(X) dont le point de sortie est a_j, alors $P_i = \underline{\text{affectation-1}}(f)(P_j)$ où $\underline{\text{affectation-1}} = \lambda f.\{\lambda P.[\lambda m.((m \epsilon dom(f)) \text{ et } P(f(m)))]\}$
- Si a_i est le point d'entrée d'une instruction de test p(X) dont les points de sortie vrai et faux sont respectivement a_t et a_f, alors $P_i = \underline{\text{test}}(p)(P_t) \text{ ou } \underline{\text{test}}(\underline{\text{non}}(p))(P_f)$
- Si a_i est un point de programme précédant une étiquette suivie du point de programme a_j alors $P_i = P_j$.

Nous déduisons de 3.1.3.0.5, 3.1.3.0.7, 3.1.4.0.3, 2.8.0.2 et de $lfp(\lambda P_\sigma.[\psi \text{ ou } P_\sigma]) = lfp(\lambda P_\sigma.[\psi]) = \psi$ la

PROPOSITION 3.5.0.2 *Propriété du plus petit point fixe du système d'équations sémantiques en arrière*

L'opérateur $B_\pi(\psi)$ sur $(P_n)^\alpha$ est un morphisme complet pour la conjonction et la disjonction. Le plus petit point fixe $(P_1,\ldots,P_\alpha)$ de $B_\pi(\psi)$ est tel que pour tout $i \epsilon [1,\alpha]$ nous avons :

$P_i = \lambda m_1.\{\exists m_2 \epsilon U^n : \tau^*(<m_1,a_i>,<m_2,a_\sigma>) \text{ et } \psi(m_2)\}$

PROPOSITION 3.5.0.3 *Propriété d'un pré-point fixe quelconque du système d'équations sémantiques en arrière*

Soient π un programme et $P=B_\pi(\psi)(P)$ le système d'équations sémantiques en arrière associé à (π,ψ). Pour tout $i=1,\ldots,\alpha$ et tout pré-point fixe P de $B_\pi(\psi)$ nous avons :

$$\lambda m_2.\{]m_1\epsilon U^n : P_i(m_1) \underline{\text{ et }} \tau^*(<m_1,a_i>,<m_2,a_\sigma>)\} \Longrightarrow \psi$$

Preuve : La proposition résulte de 3.1.3.0.7, 3.1.4.0.3, 3.1.5.0.1 avec $\beta=(\nu_\sigma \underline{\text{ et }} \iota_\sigma^{-1}(\psi))$ en notant que tout pré-point fixe de $\lambda P_\sigma.[\psi]$ est également pré-point fixe de $\lambda P_\sigma.[P_\sigma \underline{\text{ ou }} \psi]$ de sorte que $\iota_\sigma[sp((\tau^*)_{i\sigma})(\iota_i^{-1}[P_i])] \Longrightarrow P_\sigma \Longrightarrow \psi$.
Fin de la preuve.

PROPOSITION 3.5.0.4 *Propriété du plus grand point fixe du système d'équations sémantiques en arrière*

Soient π un programme comportant n variables et α points de programme $a_1,\ldots,a_\alpha$ et $(Q_1,\ldots,Q_\alpha) = gfp(B_\pi(\lambda X.\underline{\text{vrai}}))$. Alors, $\forall i\epsilon[1,\alpha]$, nous avons :

$$Q_i = \lambda m_1.\{\forall m_2 \epsilon U^n, \underline{\text{non}}(\tau^*(<m_1,a_i>,<m_2,\underline{\text{erreur}}>))\}$$

Preuve : Nous appliquons les théorèmes 3.1.3.0.7, 3.1.4.0.3 et 3.1.5.0.2 avec $\beta=\nu_\xi$. Dans ces conditions $\underline{\text{non}}(\beta) = \underline{\bigvee_{i=1}^{\alpha}} \nu_i$ et la décomposition directe de $\lambda\alpha.[\underline{\text{non}}(\nu_\xi) \underline{\text{ et }} wp(\tau)(\alpha)]$ est précisément $B_\pi(\lambda X.\underline{\text{vrai}})$ sauf pour l'équation définissant P_σ ce qui n'importe pas puisque les équations $P_\sigma=\lambda X.\underline{\text{vrai}}$ et $P_\sigma=(\lambda X.\underline{\text{vrai}} \underline{\text{ et }} P_\sigma)$ ont même plus grand point fixe. Il vient :

$$\begin{aligned} Q_i &= \iota_i[\underline{\text{non}}(wp(\tau^*)(\nu_\xi))] \\ &= \lambda m_1.\{\underline{\text{non}}(wp(\tau^*)(\nu_\xi)(m_1,a_i))\} \\ &= \lambda m_1.\{\underline{\text{non}}[\exists (m_2,c_2)\epsilon S : \tau^*(<m_1,a_i>,<m_2,c_2>) \underline{\text{ et }} \nu_\xi(<m_2,c_2>)]\} \\ &= \lambda m_1.\{\forall m_2 \epsilon U^n, \underline{\text{non}}(\tau^*(<m_1,a_i>,<m_2,\underline{\text{erreur}}>))\} \end{aligned}$$

Fin de la preuve.

En accord avec les définitions des paragraphes 3.1.2 et 3.2.2.3 nous dérivons des propositions précédentes la

PROPOSITION 3.5.0.5 *Terminaison, non-terminaison et erreurs sémantiques d'un programme*

Soient π un programme comportant n variables à valeurs dans U et α points de programme $a_1,\ldots a_\alpha$. Une exécution de π partant du point de programme a_i avec l'état initial $m \in U^n$ des variables

(a) - se termine sans conduire à une erreur sémantique si et seulement si $[lfp(B_\pi(\lambda m_i.\underline{vrai}))]_i(m)$

(b) - se termine dans un état de mémoire satisfaisant à la condition de sortie $\psi \in (U^n \to B)$ si et seulement si $[lfp(B_\pi(\psi))]_i(m)$,

(c) - ne se termine pas si et seulement si $\{[gfp(B_\pi(\lambda m_i.\underline{vrai}))]_i(m)$ et non $([lfp(B_\pi(\lambda m_i.\underline{vrai}))]_i)(m)\}$,

(d) - conduit à une erreur sémantique si et seulement si $\underline{non}([gfp(B_\pi(\lambda m_i.\underline{vrai}))]_i)(m)$.

Les propositions 3.1.3.0.5 (pour $\theta=\tau^*$), 3.1.3.0.7, 3.1.4.0.3 et le principe de dualité impliquent la

PROPOSITION 3.5.0.6 *Conjonction et disjonction de spécifications de sortie*

Soient π un programme comportant n variables à valeurs dans U et α points de programme, $P_n=(U^n \to B)$ et $P=B_\pi(\psi)(P)$ le système d'équations sémantique en arrière associé à (π,ψ). Les fonctions $\lambda\psi.[lfp(B_\pi(\psi))]$ et $\lambda\psi.[gfp(B_\pi(\psi))]$ de P_n dans $(P_n)^\alpha$ sont des morphismes complets pour la conjonction et la disjonction.

3.6 TECHNIQUES D'ANALYSE DE PROGRAMMES BASEES SUR LA SEMANTIQUE DEDUCTIVE EN ARRIERE

Nous utilisons les résultats du paragraphe 3.5 pour justifier la méthode de Hoare[1969] de vérification de correction partielle des programmes ainsi que l'extension de cette méthode par Dijkstra[1976] à des preuves de correction totale. Nous illustrons ensuite l'utilisation de la sémantique déductive en arrière pour analyser les conditions dans lesquelles l'exécution d'un programme se termine, ne se termine pas ou conduit à une erreur sémantique. Enfin nous montrons que la sémantique déductive en arrière

peut être utilisée pour déterminer en chaque point d'un programme l'ensemble des états possibles des variables au cours d'une exécution quelconque du programme partant d'un état initial satisfaisant à une spécification d'entrée donnée.

3.6.1 Justification de la méthode de Hoare de vérification de la correction partielle d'un programme

Soient π un programme comportant n variables à valeurs dans U et α points de programme $a_1,\ldots,a_\alpha$ et $P_n=(U^n \to B)$. Une preuve de correction partielle d'un programme π pour les spécifications d'entrée ϕ et de sortie ψ consiste à montrer que :

$$\{\forall m_2 \in U^n,\ [\exists m_1 \in U^n : \phi(m_1) \text{ et } \tau^*(<m_1,a_\varepsilon>,<m_2,a_\sigma>)] \Rightarrow [\psi(m_2)]\}$$

Si on peut *inventer* $P \in (P_n)^\alpha$ qui soit un pré-point fixe de $B_\pi(\psi)$ et tel que $\{\phi \Rightarrow P_\varepsilon\}$ alors la proposition 3.5.0.3 montre que π est partiellement correct pour (ϕ,ψ). La méthode de Hoare est donc valide. Elle est également complète (Cook[1975], Gorelick[1975], Clarke[1977]). En effet, si π est partiellement correct pour (ϕ,ψ) il existe toujours $P \in (P_n)^\alpha$ permettant de faire la preuve c'est-à-dire tel que $\{\{P \Rightarrow B_\pi(\psi)(P)\}$ et $\{\phi \Rightarrow P_\varepsilon\}\}$. Il suffit en fait de choisir $P=lfp(B_\pi(\psi))$ puisque la proposition 3.5.0.5.(b) montre que si ce choix ne convenait pas le programme serait erroné.

3.6.2 Justification de la méthode de Dijkstra de vérification de la correction totale d'un programme

Une preuve de correction totale d'un programme π pour les spécifications d'entrée ϕ et de sortie ψ consiste à montrer que :

$$\{\forall m_1 \in U^n,\ [\phi(m_1)] \Rightarrow [\exists m_2 \in S : \tau^*(<m_1,a_\varepsilon>,<m_2,a_\sigma>) \text{ et } \psi(m_2)]\}$$

La proposition 3.5.0.2 indique qu'il s'agit donc de montrer que :

$$\{\phi \Rightarrow [lfp(B_\pi(\psi))]_\sigma\}$$

La sémantique d'un programme ou d'une instruction est donc entièrement définie si l'on sait déterminer $\lambda\psi.\{[lfp(\Gamma_\pi(\psi))]_\sigma\}$. Par exemple, le schéma

de programme à n variables $X=X_1,\ldots,X_n$ suivant :

```
tantque p(X) faire
    X:=f(X);
refaire;
```

est par définition équivalent au schéma de programme

```
{1}
    L:
{2}
        si p(X) alors
{3}
            X:=f(X);
{4}
            allera L;
        finsi;
{5}
```

Le système d'équations sémantiques en arrière associé à ce dernier schéma de programme et à une spécification de sortie ψ est :

$$\begin{cases} P_1 = P_2 \\ P_2 = \underline{test}(p)(P_3)\ \underline{ou}\ \underline{test}(\underline{non}(p))(P_5) \\ P_3 = \underline{affectation\text{-}1}(f)(P_4) \\ P_4 = P_2 \\ P_5 = \psi \end{cases}$$

La proposition 2.8.0.2 nous permet la simplification en :

$$P_1 = \underline{test}(p)(\underline{affectation\text{-}1}(f)(P_1))\ \underline{ou}\ \underline{test}(\underline{non}(p))(\psi)$$

dont le plus petit point fixe est :

$$P_1^{\omega} = \underset{k\in\omega}{\underline{OU}}\left((\underline{test}(p)\circ\underline{affectation\text{-}1}(f))^k \circ \underline{test}(\underline{non}(p))\right)(\psi)$$

Nous avons retrouvé la règle de transformation de prédicats énoncée par Dijkstra[1976] pour l'instruction d'itération "tantque". Bien sûr Dijkstra ne parle pas d'équations de point fixe ni de résolution par approximation successives mais il définit à priori la sémantique de la boucle "tantque" comme suit :

$$P_1^{\omega} = \{\exists k \in \omega : I_k\}$$

où

$$I_0 = (\underline{non}(p) \ \underline{et} \ \psi)$$

$$I_k = (p \ \underline{et} \ \underline{affectation\text{-}1}(f)(I_{k-1})$$

ce qui est équivalent au résultat que nous avons trouvé quand p est défini pour tout $X \in \mathcal{U}^n$, puisque dans ce cas :

$$I_0 = \underline{test}(\underline{non}(p))(\psi)$$

$$I_k = \underline{test}(p) \circ \underline{affectation\text{-}1}(f)(I_{k-1})$$

$$= ((\underline{test}(p) \circ \underline{affectation\text{-}1}(f))^k \circ \underline{test}(\underline{non}(p)))(\psi)$$

3.6.3. Analyse des conditions de terminaison, non terminaison et d'exécution incorrecte d'un programme basées sur la sémantique déductive en arrière.

Très peu d'articles sont consacrés à l'étude de méthodes permettant d'étudier les conditions dans lesquelles un programme est incorrect. La proposition 3.5.0.5 est la base d'une telle étude. Considérons par exemple le programme :

```
{1}
    tantque x≥1000 faire
{2}
        x:=x+α;
{3}
    refaire;
{4}
```

où x est une variable simple à valeurs entières comprises entre b et (-b+1). Nous supposerons que b et (-b+1) notent le plus grand et le plus petit entier qu'on peut représenter sur une machine et que b>1000. Pour simplifier, nous supposerons que α est une constante entière comprise entre 0 et b.

Le système d'équations sémantiques en arrière à quatre inconnues $X_1,\ldots,X_4$ associé à ce programme est le suivant :

$$\begin{cases} X_1 = \underline{test}(\lambda x.[x \geq 1000])(X_2)\ \underline{ou}\ \underline{test}(\lambda x.[x<1000])(X_4) \\ X_2 = \underline{affectation\text{-}1}(\lambda x.[x+\alpha])(X_3) \\ X_3 = \underline{test}(\lambda x.[x \geq 1000])(X_2)\ \underline{ou}\ \underline{test}(\lambda x.[x<1000])(X_4) \\ X_4 = \lambda x.[\underline{vrai}] \end{cases}$$

Nous pouvons simplifier ce système d'équations comme suit :

$$\begin{aligned} X_1 &= \underline{test}(\lambda x.[x \geq 1000])(X_2)\ \underline{ou}\ \underline{test}(\lambda x.[x<1000])(\lambda x.[\underline{vrai}]) \\ &= \lambda x.\{[X_2(x)\ \underline{et}\ (-b+1 \leq x \leq b)\ \underline{et}\ (x \geq 1000)] \\ &\qquad \underline{ou} \\ &\qquad [\underline{vrai}\ \underline{et}\ (-b+1 \leq x \leq b)\ \underline{et}\ (x<1000)]\} \\ &= \lambda x.\{(-b+1 \leq x \leq b)\ \underline{et}\ (X_2(x)\ \underline{ou}\ (x<1000))\} \\ X_2 &= \underline{affectation\text{-}1}(\lambda x.[x+\alpha])(X_1) \\ &= \lambda x.\{(-b+1 \leq x+\alpha \leq b)\ \underline{et}\ (X_2(x+\alpha)\ \underline{ou}\ (x+\alpha<1000))\ \underline{et}\ (-b+1 \leq x \leq b)\} \\ &= \lambda x.\{(-b+1 \leq x \leq b-\alpha)\ \underline{et}\ (X_2(x+\alpha)\ \underline{ou}\ (x<1000-\alpha))\} \end{aligned}$$

Appliquant le théorème 2.8.0.2, nous calculons la plus petite solution de l'équation définissant X_2 par approximations successives :

$$\begin{aligned} P_2^0 &= \lambda x.\{\underline{faux}\} \\ P_2^1 &= \lambda x.\{(-b+1 \leq x \leq b-\alpha)\ \underline{et}\ (\underline{faux}\ \underline{ou}\ (x<1000-\alpha))\} \\ &= \lambda x.\{(-b+1 \leq x \leq 1000-\alpha)\} \quad \text{car } (b>1000) \\ P_2^2 &= \lambda x.\{(-b+1 \leq x \leq b-\alpha)\ \underline{et}\ ((-b+1 \leq x+\alpha<1000-\alpha)\ \underline{ou}\ (x<1000-\alpha))\} \\ &= \lambda x.\{(-b+1 \leq x<1000-2\alpha)\ \underline{ou}\ (-b+1 \leq x<1000-\alpha)\} \\ &= \lambda x.\{(-b+1 \leq x<1000-\alpha)\} \end{aligned}$$

les itérations ayant convergé, nous avons calculé le résolvant et nous obtenons :

$$\begin{aligned} P_1 &= \lambda x.\{(-b+1 \leq x \leq b)\ \underline{et}\ ((-b+1 \leq x<1000-\alpha)\ \underline{ou}\ (x<1000))\} \\ &= \lambda x.\{-b+1 \leq x<1000\} \end{aligned}$$

Calculons maintenant par approximations successives la plus grande solution de l'équation définissant X_2 :

$Q_2^0 = \lambda x.\{\underline{vrai}\}$

$Q_2^1 = \lambda x.\{(-b+1 \leq x \leq b-\alpha)$ et $(\underline{vrai}$ ou $x<1000-\alpha)\}$

$= \lambda x.\{-b+1 \leq x \leq b-\alpha\}$

$Q_2^2 = \lambda x.\{(-b+1 \leq x \leq b-\alpha)$ et $((-b+1 \leq x+\alpha \leq b-\alpha)$ ou $(x<1000-\alpha))\}$

$= \lambda x.\{(-b+1 \leq x \leq b-2\alpha)$ ou $(-b+1 \leq x \leq 999-\alpha)\}$

$= \lambda x.\{-b+1 \leq x \leq max(b-2\alpha, 999-\alpha)\}$

Supposons par hypothèse d'induction que :

$Q_2^k = \lambda x.\{-b+1 \leq x \leq max(b-k\alpha, 999-\alpha)\}$

alors il vient :

$Q_2^{k+1} = \lambda x.\{(-b+1 \leq x \leq b-\alpha)$ et

$((-b+1 \leq x+\alpha \leq max(b-k\alpha, 999-\alpha))$ ou $(x<1000-\alpha))\}$

$= \lambda x.\{-b+1 \leq x \leq max(b-(k+1)\alpha, 999-\alpha)\}$

passant à la limite nous obtenons le plus grand point fixe

$Q_2 = \lambda x.\{\forall k \geq 1, (-b+1 \leq x \leq max(b-k\alpha, 999-\alpha))\}$

$= \lambda x.\{(-b+1 \leq x < 1000-\alpha)$ ou $[\forall k \geq 1, (-b+1 \leq x \leq b-k\alpha)]\}$

$= \lambda x.\{(-b+1 \leq x < 1000-\alpha)$ ou $((-b+1 \leq x \leq b)$ et $(\alpha=0))\}$

ayant calculé le résolvant nous obtenons

$Q_1 = \lambda x.\{[-b+1 \leq x \leq b]$ et $[(-b+1 \leq x+\alpha < 1000-\alpha)$ ou

$((-b+1 \leq x+\alpha \leq b)$ et $(\alpha=0))$ ou $(x<1000)]\}$

$= \lambda x.\{(-b+1 \leq x < 1000)$ ou $((-b+1 \leq x \leq b)$ et $(\alpha=0))\}$

$= P_1$ ou $\lambda x.\{(-b+1 \leq x \leq b)$ et $(\alpha=0)\}$

Finalement la proposition 3.5.0.5 nous permet de déduire que pour une valeur initiale m de la variable x, le programme considéré

- se termine sans conduire à une erreur sémantique si et seulement si $P_1(m)$ est vrai c'est-à-dire $(-b+1 \leq m < 1000)$,
- ne se termine pas si et seulement si $(Q_1(m)$ et non$(P_1(m)))$ est vrai soit $\{[(-b+1 \leq m \leq b)$ et $(\alpha=0)]$ et $[(m<-b+1)$ ou $(m \geq 1000)]\} = \{(1000 \leq m \leq b)$ et $(\alpha=0)\}$,
- conduit à une erreur sémantique si et seulement si non$(Q_1(m))$ est vrai soit $\{(m<-b+1)$ ou $(b<m)$ ou $((1000 \leq m)$ et $(\alpha \neq 0))\}$.

Ce résultat correspond bien à l'intuition parce que le programme n'est défini que si $-b+1 \leq m \leq b$. Alors si $m<1000$ la boucle n'est pas exécutée et le

programme se termine. Par contre, si m≥1000 la boucle est exécutée indéfiniment quand α=0 et se termine par une erreur par débordement supérieur au cours de l'addition x+α quand α≠0.

3.6.4 Utilisation de la sémantique déductive en arrière pour caractériser en chaque point d'un programme l'ensemble des descendants des états initiaux satisfaisant à une spécification d'entrée.

Soient π un programme comportant n variables à valeurs dans U et α points de programme $a_1,\ldots,a_\alpha$ dont le point d'entrée est a_ε. Soit $\phi\in P_n$ une spécification d'entrée. Il s'agit de déterminer pour tout $i\in[1,\alpha]$,

$$\lambda m_2.\{\exists m_1\in U^n : \phi(m_1) \text{ et } \tau^*(\langle m_1,a_\varepsilon\rangle,\langle m_2,a_i\rangle)\}$$

c'est-à-dire :

$$\lambda e_2.\{\exists e_1\in S : \nu_\varepsilon(e_1) \text{ et } \bar{\phi}(e_1) \text{ et } \tau^*(e_1,e_2) \text{ et } \nu_i(e_2)\}$$

en notant,

$$\bar{\phi} = \iota^{-1}(\phi)$$

Les propositions 3.1.3.0.8 et 3.1.4.0.3 indiquent que nous pouvons utiliser un système d'équations sémantiques en arrière en calculant :

$$\lambda\bar{e}.\{\exists e_1\in S: \nu_\varepsilon(e_1) \text{ et } \bar{\phi}(e_1) \text{ et } lfp(\lambda\theta.[\lambda e.(e=\bar{e}) \text{ ou } wp(\tau)(\theta)])(e_1) \text{ et } \nu_i(\bar{e})\}$$

c'est-à-dire :

$$\lambda\bar{m}_i.\{\exists(\bar{m}_1,\ldots,\bar{m}_{i-1},\bar{m}_{i+1},\ldots,\bar{m}_\alpha,m_\varepsilon)\in(U^n)^\alpha : \phi(m_\varepsilon) \text{ et } [lfp(\lambda X.[\lambda(m_1,\ldots,m_\alpha).(m_1=\bar{m}_1,\ldots,m_\alpha=\bar{m}_\alpha) \text{ ou } B_\pi(\lambda x.\underline{vrai})(X)])]_\varepsilon(m_\varepsilon)\}$$

Pour illustrer notre propos sur un exemple très simple considérons le programme suivant (où a est une constante entière) :

```
{1}
      x:=a;
{2}
   L:
{3}
      x:=x+1;
{4}
      allera L;
```

Considérant que les valeurs de x sont des entiers, nous devons résoudre le système d'équations en arrière :

$$\begin{cases} P_1 = \lambda x.[(x=\bar{m}_1) \underline{\text{ou}} P_2(a)] \\ P_2 = \lambda x.[(x=\bar{m}_2) \underline{\text{ou}} P_3(x)] \\ P_3 = \lambda x.[(x=\bar{m}_3) \underline{\text{ou}} P_4(x+1)] \\ P_4 = \lambda x.[(x=\bar{m}_4) \underline{\text{ou}} P_3(x)] \end{cases}$$

la plus petite solution est :

$$\begin{cases} P_1 = \lambda x.[(x=\bar{m}_1) \underline{\text{ou}} (a=\bar{m}_2) \underline{\text{ou}} (\bar{m}_3 \geq a) \underline{\text{ou}} (\bar{m}_4 \geq a+1)] \\ P_2 = \lambda x.[(x=\bar{m}_2) \underline{\text{ou}} (\bar{m}_3 \geq x) \underline{\text{ou}} (\bar{m}_4 \geq x+1)] \\ P_3 = \lambda x.[(\bar{m}_3 \geq x) \underline{\text{ou}} (\bar{m}_4 \geq x+1)] \\ P_4 = \lambda x.[(\bar{m}_3 \geq x) \underline{\text{ou}} (\bar{m}_4 \geq x)] \end{cases}$$

Donc l'ensemble des descendants des états d'entrée, satisfaisant à la condition d'entrée ϕ au point {i} du programme, est caractérisé par :

$$\lambda \bar{m}_i.\{(\bar{m}_1,\ldots,\bar{m}_{i-1},\bar{m}_{i+1},\ldots,\bar{m}_4,m_\varepsilon) \in N : \phi(m_\varepsilon) \underline{\text{et}}$$
$$[(m_\varepsilon=\bar{m}_1) \underline{\text{ou}} (a=\bar{m}_2) \underline{\text{ou}} (\bar{m}_3 \geq a) \underline{\text{ou}} (\bar{m}_4 \geq a+1)]\}$$

soit pour $\phi=\lambda x.\underline{\text{vrai}}$

$$\begin{cases} \lambda \bar{m}_1.\{\underline{\text{vrai}}\} \\ \lambda \bar{m}_2.\{\bar{m}_2=a\} \\ \lambda \bar{m}_3.\{\bar{m}_3 \geq a\} \\ \lambda \bar{m}_4.\{\bar{m}_4 \geq a+1\} \end{cases}$$

3.7 COMBINAISON DE L'ANALYSE SEMANTIQUE EN AVANT ET EN ARRIERE D'UN PROGRAMME

Soient $F_\pi(\phi)$ et $B_\pi(\psi)$ les systèmes d'équations sémantiques en avant et en arrière associés à un programme π et des spécifications d'entrée ϕ

et de sortie ψ. Au chapitre 5, nous chercherons à caractériser en chaque point a_i du programme π, l'ensemble des descendants des états d'entrée satisfaisant à la condition d'entrée ϕ, qui sont les ascendants des états de sortie satisfaisant à la condition de sortie ψ, c'est-à-dire qu'il nous faudra déterminer :

$$\lambda e.\{]e_1,e_2 \in S : \nu_\varepsilon(e_1) \text{ et } \overline{\phi}(e_1) \text{ et } \tau^*(e_1,e) \text{ et } \nu_i(e) \text{ et } \tau^*(e,e_2] \text{ et } \nu_\sigma(e_2) \text{ et } \overline{\psi}(e_2)\}$$

D'après les propositions 3.3.0.2 et 3.5.0.2 il s'agit donc, à un isomorphisme près, de déterminer :

$$[lfp(F_\pi(\phi)) \text{ et } lfp(B_\pi(\psi))]_i$$

Comme les plus petits points fixes des systèmes d'équations sémantiques ne sont pas automatiquement calculables (théorème 2.5.6.0.1), nous chercherons plus simplement à approcher ces points fixes en utilisant les méthodes constructives d'approximation de points fixes développées au chapitre 4. Pour les mettre en oeuvre, au chapitre 5, nous aurons besoin de la proposition suivante, qui énonce quelques propriétés de $\{lfp(F_\pi(\phi)) \text{ et } lfp(B_\pi(\psi))\}$. Ces propriétés sont la conséquence immédiate des propositions 3.1.5.0.3, 3.1.5.0.7, 3.1.4.0.2 et 3.1.4.0.3.

PROPOSITION 3.7.0.1

Soit π un programme comportant n variables à valeurs dans Π et α points de programme. Soient $P_n \in (\Pi^n \to \mathbb{B})$ et $\phi, \psi \in P_n$.

(a) - $\{\forall P \in P_n,\ \{B_\pi(\psi)(P) \text{ et } lfp(F_\pi(\phi))\} \Rightarrow B_\pi(\psi)(P \text{ et } lfp(F_\pi(\phi)))\}$

$\{lfp(F_\pi(\phi)) \text{ et } lfp(B_\pi(\phi))\}$

(b) - $= lfp(F_\pi\{[lfp(B_\pi(\psi))]_\varepsilon \text{ et } \phi\})$

(c) - $= lfp(\lambda X.[lfp(F_\pi(\phi)) \text{ et } B_\pi(\psi)(X)])$

(d) - $= lfp(\lambda X.[lfp(B_\pi(\psi)) \text{ et } F_\pi(\phi)(X)])$

(e) - $= lfp(\lambda X.[lfp(F_\pi(\phi)) \text{ et } lfp(B_\pi(\psi)) \text{ et } F_\pi(\phi)(X)])$

(f) - $= lfp(\lambda X.[lfp(F_\pi(\phi)) \text{ et } lfp(B_\pi(\psi)) \text{ et } B_\pi(\psi)(X)])$

3.8 NOTES BIBLIOGRAPHIQUES

De nombreux algorithmes pour analyser des propriétés simples des programmes sont basés sur une sémantique opérationnelle (par exemple Kildall[1973], Wegbreit[1975], Cousot & Cousot[1975b]). Il est intéressant de comprendre que ces algorithmes consistent en fait, à résoudre un système d'équations associé au programme, ces équations étant obtenues par simplification des équations sémantiques (Cousot & Cousot[1977a]). Le passage d'une sémantique opérationnelle à une sémantique déductive a l'avantage de permettre de raisonner non pas sur un programme, mais au contraire sur les systèmes d'équations sémantiques associés, ce qui ramène le problème de l'analyse sémantique des programmes, à celui, classique en mathématiques, de résoudre ou d'approcher les solutions d'un système d'équations. Par exemple, il existe très peu d'algorithmes d'analyse de propriétés de procédures récursives. Une raison possible est que la définition d'une sémantique des procédures récursives par recopie du corps de procédure à chaque appel (Wegbreit[1975]) ou par transformation en un programme itératif avec pile de récursivité (Karr[1975]), ne soit pas un bon guide pour l'intuition. Au contraire, les raisonnements sur des systèmes d'équations (Cousot & Cousot[1977d]), nous ont permis d'obtenir de meilleurs résultats. Les difficultés pour étendre la sémantique axiomatique de Hoare[1969] à des instructions dont la syntaxe est sous-contexte (Clint & Hoare[1972]) offrent un autre exemple. Dans la sémantique déductive le processus syntaxique, pour construire le système d'équations est sous-contexte, mais la justification de la méthode de Hoare (3.4.1) ne dépend pas de problèmes syntaxiques puisqu'elle repose uniquement sur les propriétés du système d'équations.

Le langage que nous considérons pour illustrer nos méthodes d'analyse sémantique des programmes est simple, mais de complexité suffisante, pour illustrer notre propos. En fait, le paragraphe 3.1 montre bien que les méthodes d'analyse sémantique des programmes peuvent être étudiées indépendamment d'un langage de programmation particulier. C.Pair nous a donné l'idée d'utiliser la notion de système dynamique discret. Elle est également utilisée par Pnueli[1977] pour formaliser les raisonnements temporels intervenant dans la vérification de programmes parallèles.

Pour étendre notre étude à des langages de programmation plus riches il faudrait considérer des structures de données plus complexes (voir par exemple De Bakker[1977a], Finance[1976], Luckham & Suzuki[1976], Pair[1974], Rémy[1974]).

Considérant un langage plus riche, la définition de la sémantique de ce langage serait plus difficile et la méthode opérationnelle serait probablement trop lourde pour permettre de déduire simplement une sémantique déductive. Les méthodes qui pourraient remplacer la sémantique opérationnelle sont trop nombreuses pour donner une liste exhaustive de références (voir par exemple Bjørner[1977a],[1977b]). La sémantique calculatoire serait certainement bien adaptée (Finance[1976]), la sémantique dénotationnelle également (Tennent[1977] donne une introduction qui peut être complétée par Stoy[1977]. Milne & Strachey[1977] est l'ouvrage de référence tandis que Scott[1976] donne une bibliographie complète). Noter toutefois pour la sémantique dénotationnelle qu'il n'est pas nécessaire d'utiliser la technique des "continuations" (Strachey & Wadsworth[1977], Milne[1977]) pour les branchements inconditionnels tant que le langage ne comporte pas de variables étiquettes (ni de passage de procédures en paramètre).

A la suite de Kleene[1952], cette école d'Oxford a introduit l'utilisation de point fixes pour définir la sémantique dénotationnelle des langages de programmation. L'application aux preuves de procédures récursives a été immédiate (par exemple Manna, Ness & Vuillemin[1973]. Bien que ce ne soit pas nécessaire (Cousot & Cousot[1977a], Milne[1977]), les techniques de vérification de programmes itératifs sont le plus souvent justifiées en considérant les programmes récursifs équivalents obtenus par la transformation de MacCarthy (Bird[1976], Clarke[1977], Manna[1974], Vuillemin[1973]) qui est quelquefois appliquée implicitement (de Bakker[1977a]). Il nous semble que l'emploi de la sémantique déductive est la mieux adaptée à la fois pour justifier les méthodes de vérification de correction partielle des programmes, les étendre d'une part à la correction totale (ce qui a déjà été fait pour la méthode de Hoare par Dijkstra[1976] et ses suivants Basu & Yeh[1975], de Bakker[1976], Hehner[1976]) mais aussi à l'analyse de programmes incorrects (en particulier pour le domaine de non terminaison, problème assez peu étudié (Katz & Manna[1976], Sintzoff[1976a], van Lamsweerde[1977])) et d'autre part pour justifier ou découvrir des méthodes d'analyse sémantique approchée des programmes (chapitre 5).

CHAPITRE 4

METHODES CONSTRUCTIVES D'APPROXIMATION DE POINTS FIXES D'OPERATEURS MONOTONES D'UN TREILLIS COMPLET

4. METHODES CONSTRUCTIVES D'APPROXIMATION DE POINTS FIXES D'OPERATEURS MONOTONES D'UN TREILLIS COMPLET

4. METHODES CONSTRUCTIVES D'APPROXIMATION DE POINTS FIXES D'OPERATEURS MONOTONES D'UN TREILLIS COMPLET

Soient $L(\sqsubseteq)$ un ensemble partiellement ordonné par $\sqsubseteq$ et $x,y \in L$. Nous dirons que x est une *approximation inférieure* de y si et seulement si $x \sqsubseteq y$ et dualement que x est une *approximation supérieure* de y si et seulement si $y \sqsubseteq x$.

Le chapitre précédent a montré que l'analyse sémantique d'un programme revient à calculer les points fixes extrêmes de systèmes d'équations associés au programme. Le calcul exact de ces points fixes n'étant pas automatisable, nous chercherons plus simplement à les encadrer par des approximations inférieures et supérieures. C'est pourquoi nous développons dans ce chapitre des méthodes de calcul effectif d'approximations inférieures et supérieures de points fixes extrêmes d'opérateurs monotones sur un treillis complet. Dans les paragraphes 4.1 et 4.3 nous présentons deux types de méthodes d'approximation qui, en pratique, seront employées conjointement. Les méthodes d'approximation de points fixes d'un système d'équations présentées au paragraphe 4.3, reposent sur l'idée de simplification des équations à résoudre. Les termes à négliger ne peuvent pas être simplifiés sur des critères numériques mais uniquement sur des critères algébriques que nous ferons reposer sur la notion de fermeture, étudiée au paragraphe 4.2. Les méthodes d'approximation de points fixes présentées dans le paragraphe 4.1 sont basées sur l'idée d'accélérer la convergence des méthodes itératives exactes du chapitre 2. Il s'agit en somme d'extrapoler les termes de la suite des itérés pour donner en un nombre fini de pas une approximation de sa limite.

4.1 ALGORITHMES ITERATIFS D'APPROXIMATION DE POINTS FIXES BASES SUR UNE ACCELERATION DE LA CONVERGENCE PAR EXTRAPOLATION

Nous considérons dans le paragraphe 4.1.1 des algorithmes d'approximation de points fixes extrêmes d'un opérateur monotone sur un treillis complet. Ensuite le paragraphe 4.1.2 raffine sur le cas particulier d'un système d'équations monotones.

4.1.1 Approximation de points fixes d'un opérateur monotone

Pour encadrer un point fixe P d'un opérateur f sur un treillis complet L nous reprenons les méthodes itératives du chapitre 2 en accélerant la convergence par extrapolation des termes de la suite des itérés. Pour éviter d'itérer indéfiniment le long d'un cycle d'éléments non comparables nous reprenons l'idée centrale du chapitre 2, de construire une suite d'itérés qui soit une chaîne croissante ou décroissante. De plus, nous considérons une extrapolation supérieure ou inférieure des termes de la suite, ce qui donne quatre méthodes itératives approchées. Nous montrons ensuite comment elles peuvent être utilisées pour encadrer les points fixes extrêmes d'un opérateur monotone sur un treillis complet.

Itération croissante approchée supérieurement

DEFINITION 4.1.1.0.1

Soient $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet, $f \in mon(L \to L)$, alors une *itération croissante partant de* $d \in L$ *et approchée supérieurement pour* f est une séquence $\langle x^{\delta} : \delta \in \mu(L) \rangle$ d'éléments de L telle que :

(a) - $x^0 = d$

(b) - $x^{\delta} \sqsupseteq x^{\delta-1} \sqcup f(x^{\delta-1})$ si δ est un ordinal successeur et $x^{\delta-1} \notin postfp(f)$

(c) - $x^{\delta} = x^{\delta-1}$ si δ est un ordinal successeur et $x^{\delta-1} \in postfp(f)$

(d) - $x^{\delta} \sqsupseteq \bigsqcup_{\alpha<\delta} x^{\alpha}$ si δ est un ordinal limite

THEOREME 4.1.1.0.2

Soient $d \in L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ et $f \in mon(L \to L)$ alors une itération croissante partant de d et approchée supérieurement pour f est une chaîne ascendante stationnaire dont la limite est un post point fixe de f approchant supérieurement $luis(\lambda x.x \sqcup f(x))(d)$.

Preuve : Soit $\langle x^\delta : \delta \in \mu(L)\rangle$ une itération croissante partant de d et approchée supérieurement pour f. C'est une chaîne ascendante, c'est-à-dire $\{\forall \delta \in \mu(L), \forall \beta \in \mu(L), \{\delta \leq \beta\} \Rightarrow \{x^\delta \sqsubseteq x^\beta\}\}$. Soit $\delta \in \mu(L)$ quelconque fixé. Quand $\beta=\delta$ le lemme est vrai car $\sqsubseteq$ est reflexive. Supposons que le lemme soit vrai pour tout β tel que $\delta \leq \beta < \gamma < \mu(L)$. Si γ est un ordinal successeur on a $x^\delta \sqsubseteq x^{\gamma-1}$ par hypothèse d'induction. Si $x^{\gamma-1} \in postfp(f)$ alors $x^\delta \sqsubseteq x^{\gamma-1} = x^\gamma$ sinon $x^\delta \sqsubseteq x^{\gamma-1} \sqsubseteq x^{\gamma-1} \sqcup f(x^{\gamma-1}) \sqsubseteq x^\gamma$. Si γ est un ordinal limite alors $x^\delta \sqsubseteq \bigsqcup_{\beta<\gamma} x^\beta \sqsubseteq x^\gamma$. Par induction transfinie nous avons montré que $\langle x^\delta : \delta \in \mu(L)\rangle$ est une chaîne ascendante qui, par définition de $\mu(L)$, ne peut pas être strictement ascendante : $\{\exists \varepsilon \in \mu(L) : (\varepsilon+1) \in \mu(L) \text{ et } x^\varepsilon = x^{\varepsilon+1}\}$. Comme $\varepsilon+1$ est un ordinal successeur $x^\varepsilon = x^{\varepsilon+1}$ est un post point fixe de f car $x^{\varepsilon+1}$ ne peut pas être calculé en utilisant la règle 4.1.1.0.1.(b) puisque $x^\varepsilon = x^{\varepsilon+1} = x^\varepsilon \sqcup f(x^\varepsilon)$ implique $x^\varepsilon = f(x^\varepsilon)$ en contradiction avec $x^\varepsilon \notin postfp(f)$. La règle 4.1.1.0.1.(c) implique alors que $\langle x^\delta : \delta \in \mu(L)\rangle$ est stationnaire à partir du rang ε. Le théorème 2.5.3.0.1 implique que $luis(\lambda x.x \sqcup f(x))(d) \sqsubseteq x^\varepsilon$ car x^ε est un post point fixe de f plus grand que d. *Fin de la preuve.*

Itération décroissante approchée inférieurement

En appliquant le principe de dualité nous obtenons :

DEFINITION 4.1.1.0.3

Soient $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet et $f \in mon(L \to L)$, alors *une itération décroissante partant de* $d \in L$ *et approchée inférieurement pour* f est une séquence $\langle x^\delta : \delta \in \mu(L)\rangle$ d'éléments de L telle que :

(a) - $x^0 = d$

(b) - $x^\delta \sqsubseteq x^{\delta-1} \sqcap f(x^{\delta-1})$ si δ est un ordinal successeur et $x^{\delta-1} \notin prefp(f)$

(c) - $x^\delta = x^{\delta-1}$ si δ est un ordinal successeur et $x^{\delta-1} \in prefp(f)$

(d) - $x^{\delta} \sqsubseteq \bigsqcap_{\alpha<\delta} x^{\alpha}$ si δ est un ordinal limite

THEOREME 4.1.1.0.4

Soient $d \in L(\sqsubseteq,\bot,\top,\bigsqcup,\bigsqcap)$ et $f \in mon(L \to L)$ alors une itération décroissante partant de $d \in L$ et approchée inférieurement pour f est une chaîne descendante stationnaire dont la limite est un pré-point-fixe de f approchant inférieurement $llis(\lambda x.x \sqcap f(x))(d)$.

Itération croissante approchée inférieurement

DEFINITION 4.1.1.0.5

Soient $L(\sqsubseteq,\bot,\top,\bigsqcup,\bigsqcap)$ un treillis complet et $f \in mon(L \to L)$ alors *une itération croissante partant de* $d \in L$ *et approchée inférieurement pour* f est une séquence $\langle x^{\delta} : \delta \in \mu(L)\rangle$ d'éléments de L telle que :

(a) - $x^{0} = d$

(b) - $x^{\delta-1} \sqsubseteq x^{\delta} \sqsubseteq f(x^{\delta-1}) \sqcup x^{\delta-1}$ si δ est un ordinal successeur

(c) - $x^{\delta} = \bigsqcup_{\alpha<\delta} x^{\alpha}$ si δ est un ordinal limite

THEOREME 4.1.1.0.6

Soient $d \in L(\sqsubseteq,\bot,\top,\bigsqcup,\bigsqcap)$ et $f \in mon(L \to L)$ alors une itération croissante partant de d et approchée inférieurement pour f est une chaîne ascendante dont chaque terme approche $luis(\lambda x.x \sqcup f(x))(d)$ inférieurement.

Preuve : Soit $\langle x^{\delta} : \delta \in \mu(L)\rangle$ une itération croissante partant de d et approchée inférieurement pour f. Montrons que $\{\forall \delta \in \mu(L), \forall \beta \in \mu(L), \{\delta \leq \beta\} \Rightarrow \{x^{\delta} \sqsubseteq x^{\beta}\}\}$. Supposant δ fixé, la preuve se fait par induction transfinie sur β. Si $\beta=\delta$ le lemme est vrai car $\sqsubseteq$ est réflexive. Supposons que le lemme soit vrai pour tout β tel que $\delta \leq \beta < \gamma < \mu(L)$. Si γ est un ordinal successeur alors $x^{\delta} \sqsubseteq x^{\gamma-1}$ par hypothèse d'induction et $x^{\gamma-1} \sqsubseteq x^{\gamma}$ par 4.1.1.0.5.(b). De même, si γ est un ordinal limite alors $x^{\delta} \sqsubseteq \bigsqcup_{\beta<\gamma} x^{\beta} = x^{\gamma}$.

Nous avons montré par induction transfinie que $\langle x^{\delta} : \delta \in \mu(L)\rangle$ est une chaîne ascendante.

Soit $\langle y^\delta : \delta \in \mu(L)\rangle$ l'itération croissante partant de d et définie par $\lambda x.x \sqcup f(x)$. Nous savons que $x^0 = y^0 = d$. Supposons que pour tout $\gamma < \delta$ on ait $x^\gamma \sqsubseteq y^\gamma$. Si δ est un ordinal successeur alors en particulier $x^{\delta-1} \sqsubseteq y^{\delta-1}$ donc par 4.1.1.0.5.(b) et monotonie $x^\delta \sqsubseteq f(x^{\delta-1}) \sqcup x^{\delta-1} \sqsubseteq f(y^{\delta-1}) \sqcup y^{\delta-1} = y^\delta$. Si δ est un ordinal limite on a $x^\delta = \bigsqcup_{\gamma<\delta} x^\gamma \sqsubseteq \bigsqcup_{\gamma<\delta} y^\gamma = y^\delta$. Par induction transfinie et le théorème 2.5.3.0.1 nous avons montré que $\forall \delta \in \mu(L)$, $x^\delta \sqsubseteq y^\delta \sqsubseteq \mathit{luis}(\lambda x.x \sqcup f(x))(d)$.

Fin de la preuve.

Itération décroissante approchée supérieurement

En appliquant le principe de dualité, nous obtenons :

DEFINITION 4.1.1.0.7

Soient $L(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$ un treillis complet et $f \in \mathit{mon}(L \to L)$, alors *une itération décroissante partant de* $d \in L$ *et approchée supérieurement pour* f est une séquence $\langle x^\delta : \delta \in \mu(L)\rangle$ d'éléments de L telle que :

(a) - $x^0 = d$

(b) - $f(x^{\delta-1}) \sqcap x^{\delta-1} \sqsubseteq x^\delta \sqsubseteq x^{\delta-1}$ si δ est un ordinal successeur

(c) - $x^\delta = \bigsqcap_{\alpha<\delta} x^\alpha$ si δ est un ordinal limite

THEOREME 4.1.1.0.8

Soient $d \in L(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$ et $f \in \mathit{mon}(L \to L)$ alors une itération décroissante partant de d et approchée supérieurement pour f est une chaîne descendante dont chaque terme approche $\mathit{llis}(\lambda x.x \sqcap f(x))(d)$ supérieurement.

Le chapitre 2 a montré que les points fixes d'un opérateur monotone f sur un treillis complet s'obtiennent comme limites d'itérations croissantes ou décroissantes pour $\lambda x.x \sqcup f(x)$ et $\lambda x.x \sqcap f(x)$ (théorème 2.5.5.0.2). Ayant défini dans un cadre très général des méthodes itératives à convergence accélérée qui permettent d'encadrer inférieurement et supérieurement ces limites, nous pouvons maintenant présenter des méthodes d'approximation de points fixes d'un opérateur monotone sur un treillis complet.

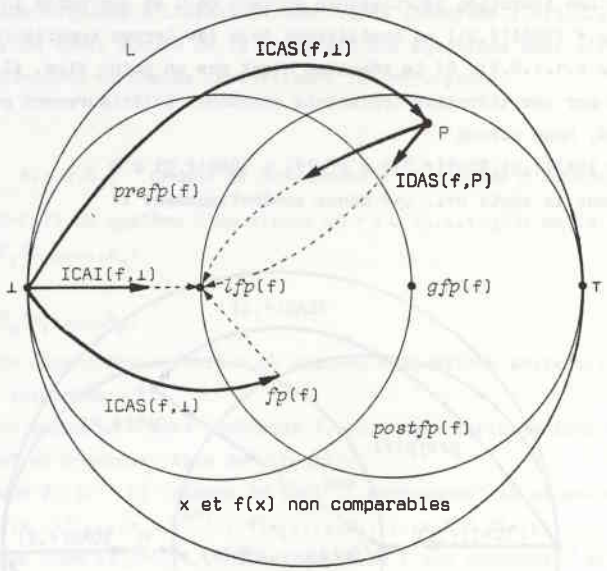
Remarque 4.1.1.0.9 : ***Approximation de points fixes extrêmes d'un opérateur monotone sur un treillis complet***

Soit à encadrer le plus petit point fixe $lfp(f)$ d'un opérateur monotone $f \in mon(L \to L)$ sur le treillis complet $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$. Il est toujours possible de trouver d et $D \in L$ qui encadrent $lfp(F)$ car par exemple $d=\bot \sqsubseteq lfp(f) \sqsubseteq \top=D$ et d'améliorer ce résultat car $d \sqsubseteq luis(\lambda x.x \sqcup f(x))(d) = lfp(f) \sqsubseteq llis(\lambda x.x \sqcap f(x))(D) \sqsubseteq D$. En pratique nous pouvons approcher ces limites par n'importe quel terme d'une itération croissante partant de d et approchée inférieurement pour f (ICAI(f,d)) et d'une itération décroissante partant de D et approchée supérieurement pour f (IDAS(f,D)) puisque les théorèmes 4.1.1.0.6 et 4.1.1.0.8 impliquent $d \sqsubseteq ICAI(f,d) \sqsubseteq luis(\lambda x.x \sqcup f(x))(d) = lfp(f) \sqsubseteq llis(\lambda x.x \sqcap f(x))(D) \sqsubseteq IDAS(f,D) \sqsubseteq D$. En s'arrêtant après un certain rang dans les itérations la convergence de ces algorithmes d'approximation peut toujours être forcée.

Pour améliorer l'approximation supérieure initiale D de $lfp(f)$ par les termes de IDAS(f,D), il ne faut pas franchir les points fixes de f (c'est-à-dire autoriser le choix d'un terme x de IDAS(f,D) tel que $x \sqsubseteq P = f(P) \sqsubseteq D$) pour la bonne raison qu'il serait alors possible de franchir en particulier $lfp(f)$, ce qui conduit à une approximation supérieure incorrecte. Donc si D est plus grand qu'un autre point fixe P de f, tous les termes de IDAS(f,D) sont supérieurs à P ce qui ne conduit pas à une bonne approximation supérieure de $lfp(f)$. Il vaut donc toujours mieux utiliser la limite d'une itération croissante partant de l'approximation inférieure d et approchée supérieurement pour f (ICAS(f,d)). En effet le théorème 4.1.1.0.2 implique $lfp(f) = luis(\lambda x.x \sqcup f(x))(d) \sqsubseteq ICAS(f,d)$ et il est toujours possible de forcer la convergence en choisissant des termes approchés suffisamment grands mais inférieurs à D ce qui garantit que $lfp(f) \sqsubseteq ICAS(f,d) \sqsubseteq D$. Si la limite de ICAS(f,d) n'est pas un point fixe de f, il est possible de l'améliorer en appliquant comme précédemment le théorème 4.1.1.0.8. En résumé, disposant d'approximations inférieure d et supérieure D de $lfp(f)$ nous proposons de les améliorer comme suit :

$$d \sqsubseteq ICAI(f,d) \sqsubseteq lfp(f) \sqsubseteq IDAS(f,D \sqcap ICAS(f,d)) \sqsubseteq D$$

Il est toujours possible de choisir $d=\bot$ et $D=\top$ ce qui donne schématiquement :



Légende :

f	: opérateur monotone sur le treillis complet L(⊑,⊥,⊤,⊔,⊓)
lfp(f)	: plus petit point fixe de f
gfp(f)	: plus grand point fixe de f
prefp(f)	: {x ∈ L : x ⊑ f(x)}
postfp(f)	: {x ∈ L : f(x) ⊑ x}

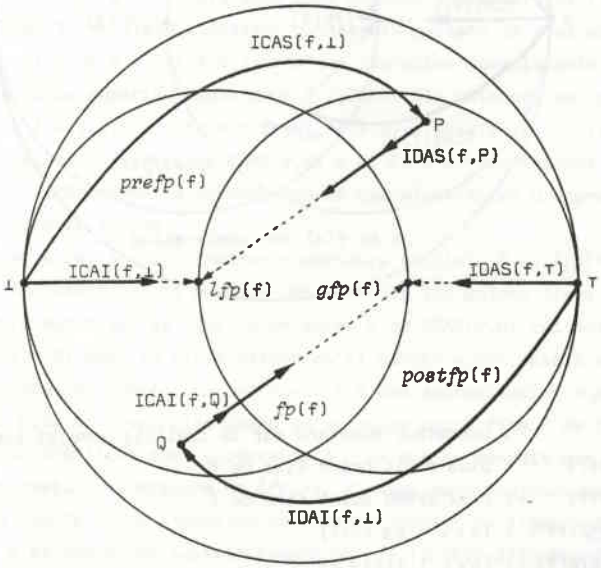
Itération
- Croissante Approchée
  - Supérieurement : ICAS
  - Inférieurement : ICAI
- Décroissante Approchée
  - Supérieurement : IDAS
  - Inférieurement : IDAI

Dualement, connaissant un encadrement d ⊑ *gfp*(f) ⊑ D du plus grand point fixe de f, nous obtiendrons une meilleure approximation supérieure de *gfp*(f) par un terme quelconque d'une itération décroissante partant de D et approchée supérieurement pour f (IDAS(f,D)) (théorème 4.1.1.0.8). Pour obtenir une meilleure approximation inférieure de *gfp*(f), nous calculerons la

limite d'une itération décroissante partant de D et approchée inférieurement pour f (IDAI(f,D)) en choisissant tous les termes supérieurs à d (théorème 4.1.1.0.4). Si le résultat n'est pas un point fixe, il sera amélioré par une itération croissante approchée inférieurement pour f. En résumé, nous aurons :

$$d \sqsubseteq ICAI(f, d \sqcup IDAI(f,D)) \sqsubseteq gfp(f) \sqsubseteq IDAS(f,D) \sqsubseteq D$$

ce qui pour le choix $d=\bot$, $D=\top$ donne schématiquement :



Fin de la remarque.

4.1.2 Approximation de solutions d'un système d'équations

L'étude précédente sur les méthodes itératives d'approximation de point fixes d'un opérateur monotone basées sur une accélération de la convergence par extrapolation est complétée dans le cas d'un système d'équations. Il s'agit de montrer comment l'idée d'extrapolation peut être

appliquée aux méthodes d'itérations chaotiques (théorème 2.9.1.0.2) et en particulier de tenir compte de la structure des équations pour effectuer les extrapolations minimales garantissant la convergence.

DEFINITION 4.1.2.0.1 *Graphe de dépendance d'un système d'équations*

Soit $X=F(X)$ un système d'équations où $F \in L^n(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ de la forme :

$$\begin{cases} X_1 = F_1(X_1,\ldots,X_n) \\ \ldots \\ X_n = F_n(X_1,\ldots,X_n) \end{cases}$$

Un graphe de dépendance associé à ce système d'équations satisfait aux conditions suivantes :

- Le graphe comporte n arcs numérotés 1,...,n dont certains sont étiquetés "simples" et d'autres "tête de circuit".
- On dit que $f \in (L^n \to L)$ "dépend de la $i^{ème}$ composante" si et seulement si $\{\exists x_1,\ldots,x_i,x_i',\ldots,x_n \in L^{n+1} : f(x_1,\ldots,x_i,\ldots,x_n) \neq f(x_1,\ldots,x_i',\ldots,x_n)\}$. Alors pour tous i, j=1,...,n l'extrémité de l'arc numéroté i est l'origine de l'arc numéroté j si F_j dépend de la $i^{ème}$ composante.
- Tout circuit (Berge[1973], p.8) du graphe passe par un arc étiqueté "tête de circuit" et le nombre d'arcs "tête de circuit" est minimal.

Remarque 4.1.2.0.2 : *Choix des têtes de circuits*

Considérons le système d'équations sémantiques en avant associé à un programme π. Alors le graphe du programme π est un graphe de dépendance de ce système d'équations. Comme têtes de circuits nous avons montré qu'on peut choisir n'importe quel membre de la famille des ensembles minimaux pour l'inclusion de la grille de l'hypergraphe (Berge[1973, p.404-405]) dont chaque arête est l'ensemble des arcs de sortie des noeuds de jonction appartenant à un circuit élémentaire du graphe de programme. Le choix n'étant pas unique diverses heuristiques de choix ont été proposées dans Cousot & Cousot[1975, p. 40-46]. En particulier quand le graphe est réductible au sens d'Allen & Cocke[1972] (et d'après Knuth[1971], c'est le cas de 95% des programmes FORTRAN) les arcs étiquetés "tête de circuit" sont alors les arcs de sortie des noeuds de jonction têtes des intervalles du graphe de programme.

Fin de la remarque.

LEMME 4.2.1.0.3 *Condition nécessaire et suffisante de convergence d'une itération chaotique*

Soit $\langle x^{\delta} : \delta \in Ord \rangle$ une itération chaotique partant de 0 et définie par $F \in (L^n \to L^n)$ et $\langle J^{\delta} : \delta \in Ord \rangle$ (satisfaisant à l'hypothèse 2.9.2.0.1.(a)). Soit G un graphe de dépendance du système d'équations X=F(X).

Pour que la suite $\langle x^{\delta} : \delta \in Ord \rangle$ soit stationnaire il faut et il suffit que la suite $\langle x_i^{\delta} : \delta \in Ord \rangle$ soit stationnaire, pour tous les i=1,...,n tels que l'arc numéroté i de G est étiqueté "tête de circuit".

Le lemme 4.2.1.0.3 montre qu'une itération chaotique quelconque pour un opérateur quelconque sur L^n est stationnaire si et seulement si toutes les composantes "tête de circuit" sont stables après un certain rang. On voit donc que pour accélérer la convergence d'une itération chaotique il suffit de forcer la convergence des seules composantes têtes de circuit. Au cours des itérations nous utiliserons une seule technique d'extrapolation qui est formalisée à l'aide d'opérateurs d' "élargissement" et "rétrécissement". A nouveau nous considérons des itérations croissantes ou décroissantes avec extrapolation supérieure ou inférieure.

Itération chaotique croissante avec élargissement supérieur

DEFINITION 4.1.2.0.4 *Elargissement supérieur*

Soit $L(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$ un treillis complet alors $\bar{\nabla} \in (L \times L \to L)^-$ est dite opération d'*élargissement supérieur* si et seulement si elle satisfait aux conditions suivantes :

(a) - $\{\forall x, y \in L,\ x \sqcup y \sqsubseteq x \bar{\nabla} y\}$

(b) - Pour toute chaîne ascendante $\langle x^{\delta} : \delta \in \omega \rangle$ d'éléments de L, la suite $\langle y^{\delta} : \delta \in \omega \rangle$ définie par $y^0 = x^0$, $y^{\delta+1} = y^{\delta} \bar{\nabla} x^{\delta+1}$ est une chaîne non strictement ascendante.

DEFINITION 4.1.2.0.5 *Itération chaotique croissante avec élargissement supérieur*

Soient $L(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$ un treillis complet et $F \in mon(L^n \to L^n)$ alors une *itération chaotique croissante partant de* $0 \in L^n$ *définie par* F, *l'élargissement supérieur* $\bar{\nabla}$, *le graphe de dépendance* G *du système d'équations* X=F(X) *et la suite* $\langle J^{\delta} : \delta \in \omega \rangle$ (satisfaisant à la condition 2.9.2.0.1.(a)) est une

suite $\langle X^\delta : \delta \in \omega \rangle$ d'éléments de L^n définie comme suit :

(a) - $X^0 = D$

(b) - $X_i^\delta = X_i^{\delta-1} \sqcup F_i(X^{\delta-1})$ si $\delta>0$, $i \in J^\delta$, l'arc numéroté i de G est "simple" et $\underline{non}(F_i(X^{\delta-1}) \sqsubseteq X_i^{\delta-1})$

(c) - $X_i^\delta = X_i^{\delta-1} \bar{\nabla} F_i(X^{\delta-1})$ si $\delta>0$, $i \in J^\delta$, l'arc numéroté i de G est "tête de circuit" et $\underline{non}(F_i(X^{\delta-1}) \sqsubseteq X_i^{\delta-1})$

(d) - $X_i^\delta = X_i^{\delta-1}$ si $\delta>0$ et $((i \notin J^\delta)$ $\underline{ou}$ $(F_i(X^{\delta-1}) \sqsubseteq X_i^{\delta-1}))$

THEOREME 4.1.2.0.6 *Convergence d'une itération chaotique croissante avec élargissement supérieur*

Soient $D \in L^n(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ et $F \in mon(L^n \to L^n)$ alors une itération $\langle X^\delta : \delta \in \omega \rangle$ chaotique croissante avec élargissement supérieur pour F et D est une chaîne ascendante stationnaire après un nombre fini de pas et sa limite est un post-point fixe de F approchant supérieurement $luis(\lambda X.X \sqcup F(X))(D)$.

Preuve :

La suite $\langle X^\delta : \delta \in \omega \rangle$ est une chaîne ascendante car pour tout i=1,...,n on a soit $X_i^\delta = X_i^{\delta-1} \sqcup F_i(X_1^{\delta-1},...,X_n^{\delta-1}) \sqsupseteq X_i^{\delta-1}$ soit $X_i^\delta = X_i^{\delta-1} \bar{\nabla} F_i(X_1^{\delta-1},...,X_n^{\delta-1}) \sqsupseteq X_i^{\delta-1} \sqcup F_i(X_1^{\delta-1},...,X_n^{\delta-1}) \sqsupseteq X_i^{\delta-1}$ ou enfin $X_i^\delta = X_i^{\delta-1}$.

Soit i quelconque appartenant à {1,...,n} tel que l'arc numéroté i de G soit étiqueté "tête de circuit". Considérons la suite $\delta^0,...,\delta^k,...$ telle que $\delta^0=0$ et pour tout $k \geq 1$ on ait $i \in J^{\delta^k}$ tandis que pour tout δ tel que $\delta^{k-1}<\delta<\delta^k$ on ait $i \notin J^\delta$. D'après la définition 4.1.2.0.5 nous savons que $X_i^{\delta^{k-1}} = X_i^{\delta^{k-1}+1} = ... = X_i^{\delta^k-1}$ et $X_i^{\delta^k} = X_i^{\delta^k-1} \bar{\nabla} F_i(X^{\delta^k-1})$ ceci implique $X_i^{\delta^k} = X_i^{\delta^{k-1}} \bar{\nabla} F_i(X^{\delta^{k-1}})$. Comme $\langle X^\delta : \delta \in \omega \rangle$ est une chaîne ascendante et F_i est monotone, la définition 4.1.2.0.4.(b) implique que la suite $X_i^{\delta^0},...,X_i^{\delta^k},...$ est une chaîne ascendante stationnaire au bout d'un nombre fini de pas et par conséquent il en va de même de la suite $\langle X_i^\delta : \delta \in \omega \rangle$.

D'après le lemme 4.1.2.0.3 la chaîne ascendante $\langle X^\delta : \delta \in \omega \rangle$ est stationnaire au bout d'un nombre fini de pas.

La chaîne ascendante $\langle X^\delta : \delta \in \omega \rangle$ étant stationnaire après un nombre ε de pas, on sait que pour tout i=1,...,n il existe d'après la définition de

$<J^\delta:\delta\in\omega>$ un $\delta\in\omega$ tel que $\delta>\varepsilon$ et $i\in J^\delta$ avec $X^\varepsilon = X^{\delta-1} = X^\delta$. Dans le calcul de X_i^δ on ne peut pas avoir appliqué les règles 4.1.2.0.5.(b) ni 4.1.2.0.5.(c) car $X_i^\delta = X_i^{\delta-1}\sqcup F_i(X^{\delta-1})$ et $X_i^\delta = X_i^{\delta-1}$ implique $F(X_i^{\delta-1})\subseteq X_i^{\delta-1}$ en contradiction avec $\underline{\text{non}}(F_i(X^{\delta-1})\subseteq X_i^{\delta-1})$. De même $X_i^\delta = X_i^{\delta-1}\bar\nabla F_i(X^{\delta-1})$ et $X^\delta = X^{\delta-1}$ implique $X_i^{\delta-1} = X_i^{\delta-1}\bar\nabla F_i(X^{\delta-1})\supseteq X_i^{\delta-1}\sqcup F_i(X^{\delta-1})$ soit $X_i^{\delta-1} = X_i^{\delta-1}\sqcup F_i(X^{\delta-1})$ en contradiction avec $\underline{\text{non}}(F_i(X^{\delta-1})\subseteq X_i^{\delta-1})$. On a donc appliqué la règle 4.1.2.0.4.(d) pour obtenir X_i^δ et comme $i\in J^\delta$ on avait $F_i(X^{\delta-1})\subseteq X_i^{\delta-1}$. Nous en déduisons que X^ε est post-point fixe de F plus grand que D, c'est donc une approximation supérieure de *luis*$(\lambda X.X\sqcup F(X))(D)$.
Fin de la preuve.

Remarques 4.1.2.0.7 *Sur la rapidité de convergence et la précision des résultats*

(a) - Quand le point de départ D de l'itération chaotique est un pré-point fixe de F alors la règle 4.1.2.0.5.(b) peut être simplifiée en $X_i^\delta = F_i(X^{\delta-1})$.

(b) - La convergence ne dépend pas de l'ordre d'itération (c'est-à-dire du choix de $<J^\delta:\delta\in\omega>$) mais la rapidité de convergence en dépend. De plus quand l'élargissement n'est pas monotone, la précision du résultat dépend également de l'ordre d'itération.

(c) - On peut, dans la règle 4.1.2.0.5.(c), choisir à chaque fois qu'elle est utilisée un élargissement supérieur différent $\bar\nabla^\delta$ tel que $\{\forall x,y\in L,\ \forall\delta\in\omega,\ x\sqcup y\subseteq x\bar\nabla^\delta y\}$ et pour toute suite $<x^\delta:\delta\in\omega>$ d'éléments de L la suite $<y^\delta:\delta\in\omega>$ définie par $y^0 = x^0$, $y^{\delta+1} = y^\delta\bar\nabla^{\delta+1}x^{\delta+1}$ est une chaîne non strictement ascendante.

(d) - Dans la règle 4.1.2.0.5.(c) l'élargissement permet une extrapolation basée sur deux itérés consécutifs, nous avons donc choisi une méthode itérative à pas séparés. Il est également possible de baser l'extrapolation sur tous les itérés de rang strictement inférieur à δ ou de choisir une méthode à pas liés utilisant les itérés de rangs δ-1, ..., δ-p.

(e) - Soit $M\subseteq L$ tel que $F(M^\cap)\subseteq M^\cap$. Quand $D\in M$ et *luis*$(\lambda X.X\sqcup F(X))(D)\notin M$ il est possible de trouver une approximation supérieure de *luis*$(\lambda X.X\sqcup F(X))(D)$ dans M en utilisant une itération chaotique

croissante avec un élargissement supérieur $\bar{\nabla}$ tel que $\forall x,y \in M$, $(x \bar{\nabla} y) \in M$.

Fin des remarques.

Itération chaotique décroissante avec élargissement inférieur

Appliquant le principe de dualité, nous obtenons :

DEFINITION 4.1.2.0.8 *Elargissement inférieur*

Soit $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet alors $\underline{\nabla} \in (L\times L \to L)$ est dite opération d'élargissement inférieur si et seulement si :

(a) - $\{\forall x, y \in L,\ x \underline{\nabla} y \sqsubseteq x \sqcap y\}$.

(b) - Pour toute chaîne descendante $\langle x^\delta : \delta \in \omega\rangle$ d'éléments de L, la suite $\langle y^\delta : \delta \in \omega\rangle$ définie par $y^0 = x^0$, $y^{\delta+1} = y^\delta \underline{\nabla} x^{\delta+1}$ est une chaîne non strictement descendante.

DEFINITION 4.1.2.0.9 *Itération chaotique décroissante avec élargissement inférieur*

Soient $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet et $F \in mon(L^n \to L^n)$, alors *une itération chaotique décroissante partant de* $D \in L^n$ *définie par* F, *l'élargissement inférieur* $\underline{\nabla}$, *le graphe de dépendance* G *du système d'équations* $X=F(X)$ *et la suite* $\langle J^\delta : \delta \in \omega\rangle$ (satisfaisant à la condition 2.9.2.0.1.(a)) est une suite $\langle X^\delta : \delta \in \omega\rangle$ d'éléments de L^n définie comme suit :

(a) - $X^0 = D$

(b) - $X_i^\delta = X_i^{\delta-1} \sqcap F_i(X^{\delta-1})$ si $\delta>0$, $i \in J^\delta$, l'arc numéroté i de G est "simple" et $\underline{non}(X_i^{\delta-1} \sqsubseteq F_i(X^{\delta-1}))$

(c) - $X_i^\delta = X_i^{\delta-1} \underline{\nabla} F_i(X^{\delta-1})$ si $\delta>0$, $i \in J^\delta$, l'arc numéroté i de G est "tête de circuit" et $\underline{non}(X_i^{\delta-1} \sqsubseteq F_i(X^{\delta-1}))$

(d) - $X_i^\delta = X_i^{\delta-1}$ si $\delta>0$ et $((i \notin J^\delta)$ $\underline{ou}$ $(X_i^{\delta-1} \sqsubseteq F_i(X^{\delta-1}))$

THEOREME 4.1.2.0.10 *Convergence d'une itération chaotique décroissante avec élargissement inférieur*

Soient $D \in L^n(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ et $F \in mon(L^n \to L^n)$ alors une itération chaotique décroissante avec élargissement inférieur pour F et D est une chaîne descendante stationnaire après un nombre fini de pas et sa limite est un pré-

point fixe de F approchant inférieurement $lli s(\lambda X.X \sqcap F(X))(D)$.

Itération chaotique croissante avec rétrécissement supérieur

DEFINITION 4.1.2.0.11 *Rétrécissement supérieur*

Soit $L(\sqsubseteq,\bot,\top,\dot{\sqcup},\sqcap)$ un treillis complet alors $\bar{\Delta} \in (L \times L \to L)$ est dite opération de rétrécissement supérieur si et seulement si :

(a) - $\{\forall x,y \in L,\ x \sqsubseteq x \bar{\Delta} y \sqsubseteq x \sqcup y\}$

(b) - Pour toute chaîne ascendante $\langle x^{\delta} : \delta \in \omega \rangle$ d'éléments de L, la suite $\langle y^{\delta} : \delta \in \omega \rangle$ définie par $y^0 = x^0$, $y^{\delta+1} = y^{\delta} \bar{\Delta} x^{\delta+1}$ est une chaîne non strictement ascendante.

DEFINITION 4.1.2.0.12 *Itération chaotique croissante avec rétrécissement supérieur*

Soient $L^n(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet et $F \in mon(L^n \to L^n)$ alors *une itération chaotique croissante partant de* $D \in L^n$ *définie par* F, *le rétrécissement supérieur* $\bar{\Delta}$, *le graphe de dépendance* G *du système d'équations* X=F(X) *et la suite* $\langle J^{\delta} : \delta \in \omega \rangle$ (satisfaisant à la condition 2.9.2.0.1.(a)) est une suite $\langle X^{\delta} : \delta \in \omega \rangle$ d'éléments de L^n définie comme suit :

(a) - $X^0 = D$

(b) - $X_i^{\delta} = X_i^{\delta-1} \sqcup F_i(X^{\delta-1})$ si $\delta > 0$, $i \in J^{\delta}$, l'arc numéroté i de G est "simple"

(c) - $X_i^{\delta} = X_i^{\delta-1} \bar{\Delta} F_i(X^{\delta-1})$ si $\delta > 0$, $i \in J^{\delta}$, l'arc numéroté i de G est "tête de circuit"

(d) - $X_i^{\delta} = X_i^{\delta-1}$ si $\delta > 0$ et $i \notin J^{\delta}$

THEOREME 4.1.2.0.13 *Convergence d'une itération chaotique croissante avec rétrécissement supérieur*

Soient $D \in L^n(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ et $F \in mon(L^n \to L^n)$ alors une itération $\langle X^{\delta} : \delta \in \omega \rangle$ chaotique croissante avec rétrécissement supérieur pour F et D est une chaîne ascendante stationnaire après un nombre fini de pas dont chaque terme approche $lui s(\lambda X.X \sqcup F(X))(D)$ inférieurement.

Preuve : La preuve que $\langle X^{\delta} : \delta \in \omega \rangle$ est une chaîne ascendante stationnaire est tout à fait similaire à la preuve donnée pour le théorème 4.1.2.0.6.

Chaque terme de l'itération chaotique $\langle Y^\delta, \delta \in \omega\rangle$ partant de D et définie par $\lambda X.X \sqcup F(X)$ et $\langle J^\delta : \delta \in \omega\rangle$ est supérieur ou égal au terme correspondant de $\langle X^\delta : \delta \in \omega\rangle$. En effet $X^0 = Y^0 = D$. Supposons $X^{\delta-1} \sqsubseteq Y^{\delta-1}$. Si $i \notin J^\delta$ alors $X^\delta = X^{\delta-1} \sqsubseteq Y^{\delta-1} = Y^\delta$. Si $i \in J$ alors si l'arc numéroté i de G est "simple", on a $X_i^\delta = X_i^{\delta-1} \sqcup F_i(X^{\delta-1}) \sqsubseteq Y_i^{\delta-1} \sqcup F_i(Y^{\delta-1})$ car F_i est monotone sinon l'arc numéroté i de G est "tête de circuit" auquel cas $X_i^\delta = X_i^{\delta-1} \bar{\Delta} F_i(X^{\delta-1}) \sqsubseteq X_i^{\delta-1} \sqcup F_i(X^{\delta-1}) \sqsubseteq Y^{\delta-1} \sqcup F_i(Y^{\delta-1})$ par la condition 4.1.2.0.11.(a), hypothèse de récurrence et monotonie. Par récurrence sur δ on a $X^\delta \sqsubseteq Y^\delta$ pour tout $\delta \in \omega$. De plus on sait que $\langle Y^\delta : \delta \in \omega\rangle$ est une chaîne ascendante et que $Y^\omega \sqsubseteq \mathit{luis}(\lambda X.X \sqcup F(X))(D)$, par conséquent pour tout $\delta \in \omega$ on a bien $X^\delta \sqsubseteq \mathit{luis}(\lambda X.X \sqcup F(X))(D)$.
Fin de la preuve.

Remarque 4.1.2.0.14 : Les remarques 4.1.2.0.7 peuvent être reprises pour une itération chaotique avec rétrécissement supérieur, en particulier quand D est un pré-point fixe de F la condition 4.1.2.0.11.(a) peut être remplacée par $\{\forall x,y \in L,\ \{x \sqsubseteq y\} \Rightarrow \{x \sqsubseteq x \bar{\Delta} y \sqsubseteq y\}\}$ tandis que la règle 4.1.2.0.12.(b) se simplifie en $X_i^\delta = F_i(X^{\delta-1})$. On retrouve alors les conditions (duales) de Cousot & Cousot[1977a].
Fin de la remarque.

Itération chaotique décroissante avec rétrécissement inférieur

En appliquant le principe de dualité nous obtenons :

DEFINITION 4.1.2.0.15 *Rétrécissement inférieur*

Soit $L(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$ un treillis complet alors $\underline{\Delta} \in (L \times L \to L)$ est dite opération de rétrécissement inférieur si et seulement si :
(a) - $\{\forall x,y \in L,\ x \sqcap y \sqsubseteq x \underline{\Delta} y \sqsubseteq x\}$
(b) - Pour toute chaîne descendante $\langle x^\delta : \delta \in \omega\rangle$ d'éléments de L la suite $\langle y^\delta : \delta \in \omega\rangle$ définie par $y^0 = x^0$, $y^{\delta+1} = y^\delta \underline{\Delta} x^{\delta+1}$ est une chaîne non strictement descendante.

DEFINITION 4.1.2.0.16 *Itération chaotique décroissante avec rétrécissement inférieur*

Soient $L^n(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet et $F \in mon(L^n \to L^n)$ alors *une itération chaotique décroissante partant de* $D \in L^n$ *définie par* F, *le rétrécissement inférieur* $\underline{\Delta}$, *le graphe de dépendance* G *du système d'équations* X=F(X) *et la suite* $\langle J^\delta : \delta \in \omega \rangle$ (satisfaisant à la condition 2.9.2.0.1.(a)) est une suite $\langle X^\delta : \delta \in \omega \rangle$ d'éléments de L^n définie comme suit :

(a) - $X^0 = D$

(b) - $X_i^\delta = X_i^{\delta-1} \sqcap F_i(X^{\delta-1})$ si $\delta>0$, $i \in J^\delta$ et l'arc numéroté i de G est "simple"

(c) - $X_i^\delta = X_i^{\delta-1} \underline{\Delta} F_i(X^{\delta-1})$ si $\delta>0$, $i \in J^\delta$ et l'arc numéroté i de G est "tête de circuit"

(d) - $X_i^\delta = X_i^{\delta-1}$ si $\delta>0$ et $i \notin J^\delta$

THEOREME 4.1.2.0.17 *Convergence d'une itération chaotique décroissante avec rétrécissement inférieur*

Soient $D \in L^n(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ et $F \in mon(L^n \to L^n)$ alors une itération chaotique décroissante avec rétrécissement inférieur pour F et D est une chaîne descendante stationnaire après un nombre fini de pas dont chaque terme approche *lfis*$(\lambda X.X \sqcap F(X))(D)$ supérieurement.

Les divers résultats précédents utilisés avec la remarque 4.1.1.0.9 nous permettent maintenant d'approcher inférieurement et supérieurement les solutions extrêmes d'un système d'équations monotones sur un treillis complet.

Nous avons choisi de formuler les méthodes itératives d'approximation de points fixes basées sur une accélération de la convergence dans un contexte très général. En particulier la notion d'extrapolation a été formalisée en ne retenant que des hypothèses minimales qui ne permettent pas d'apporter une réponse conceptuelle aux problèmes de meilleur choix des opérations d'élargissement et rétrécissement tant au point de vue de l'efficacité des calculs qu'à celui de la précision de l'approximation. En fait un bon choix de l'extrapolation doit tenir compte des propriétés particulières des treillis et systèmes d'équations considérés comme le montre le chapitre 5 sur quelques exemples pratiques.

4.2 FERMETURES SUR UN TREILLIS COMPLET

Les algorithmes d'approximation de solutions d'un système d'équations monotones sur un treillis complet, basés sur une simplification des équations, reposent sur le fait que pour approcher supérieurement (respectivement inférieurement) le plus petit point fixe de $f \in mon(L \to L)$, il suffit de trouver g tel que $f \sqsubseteq g$ (respectivement $g \sqsubseteq f$) de sorte que le plus petit point fixe de g soit calculable ou approximable supérieurement (respectivement inférieurement) car $lfp(f) \sqsubseteq lfp(g)$ (respectivement $lfp(g) \sqsubseteq lfp(f)$). En particulier, on peut construire l'approximation g de f en utilisant une opération de fermeture sur L qui permet de restreindre l'espace des propriétés des programmes à un sous-espace modélisant les informations que l'on veut rassembler sur les programmes tout en éliminant les informations que l'on choisit a priori d'ignorer.

4.2.1 Définition, caractérisations et propriétés des fermetures

Rappelons les définitions des fermetures supérieures et inférieures énoncées au paragraphe 2.3 :

DEFINITION 4.2.1.0.1 *Fermeture supérieure* (Moore[1910])

Soit $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet. Un opérateur $\bar{\rho}$ de L est une fermeture supérieure si et seulement si :

(a) - $\bar{\rho}$ est monotone $\quad \{\forall x,y \in L,\ \{x \sqsubseteq y\} \Rightarrow \{\bar{\rho}(x) \sqsubseteq \bar{\rho}(y)\}\}$

(b) - $\bar{\rho}$ est extensive $\quad \{\forall x \in L,\ x \sqsubseteq \bar{\rho}(x)\}$

(c) - $\bar{\rho}$ est idempotente $\quad \{\forall x \in L,\ \bar{\rho}(x) = \bar{\rho}(\bar{\rho}(x))\}$

DEFINITION 4.2.1.0.2 *Fermeture inférieure*

Un opérateur $\underline{\rho}$ de $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ est une fermeture inférieure si et seulement si $\underline{\rho}$ est monotone, réductive $\{\forall x \in L,\ \underline{\rho}(x) \sqsubseteq x\}$ et idempotente.

Ces deux notions étant duales, nous étudierons plus particulièrement les fermetures supérieures. Nous commençons par rappeler quelques définitions des fermetures supérieures équivalentes à la définition classique

4.2.1.0.1.

CARACTERISATION 4.2.1.0.3 (Monteiro[1945])

$\rho \in (L \to L)$ est une fermeture supérieure si et seulement si $\{\forall x,y \in L, y \sqcup \rho(y) \sqcup \rho(\rho(x)) \sqsubseteq \rho(x \sqcup y)\}$.

CARACTERISATION 4.2.1.0.4 (Iseki[1951])

$\rho \in (L \to L)$ est une fermeture supérieure si et seulement si $\{\forall x,y \in L, x \sqcup \rho(\rho(x)) \sqsubseteq \rho(x \sqcup y)\}$.

CARACTERISATION 4.2.1.0.5 (Morgado[1962b])

$\rho \in (L \to L)$ est une fermeture supérieure si et seulement si $\{\forall x,y \in L, \{\{x \sqsubseteq \rho(y)\} \iff \{\rho(x) \sqsubseteq \rho(y)\}\}\}$.

CARACTERISATION 4.2.1.0.6 (Morgado[1965b])

$\rho \in (L \to L)$ est une fermeture supérieure si et seulement si $\{\forall f \in (L \to L), \{\{\lambda x.x \sqsubseteq f\} \Rightarrow \{\lambda x.x \sqsubseteq \rho \circ \rho \sqsubseteq \rho \circ f\}\}\}$.

CARACTERISATION 4.2.1.0.7 (Morgado[1965b])

$\rho \in (L \to L)$ est une fermeture supérieure si et seulement si $\{\forall f \in (L \to L), \{\{\lambda x.x \sqsubseteq \rho \circ f\} \iff \{\rho \sqsubseteq \rho \circ f\}\}\}$.

PROPOSITION 4.2.1.0.8 (Ward[1942])

Une fermeture supérieure ρ de $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ est un *quasi-morphisme complet supérieur* c'est-à-dire $\{\forall S \sqsubseteq L, \rho(\sqcup S)=\rho(\sqcup \rho(S))$ et $\sqcap \rho(S) = \rho(\sqcap \rho(S))\}$.

4.2.2 Caractérisation d'un sous-ensemble d'un treillis complet comme image de ce treillis par une fermeture supérieure

Les théorèmes 2.3.0.1 et 2.3.0.3 dus à Ward montrent que l'image d'un treillis complet par une fermeture supérieure est un treillis complet. Réciproquement Monteiro et Ribeiro ont étudié les propriétés des sous-ensembles d'un treillis complet qui peuvent être repésentés comme image de ce treillis

par une fermeture supérieure. Nous rappelons et complétons leurs résultats.

THEOREME 4.2.2.0.1 Monteiro & Ribeiro[1942,th.5.2]

Soit $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet. Un opérateur extensif ρ de L est uniquement déterminé par l'ensemble de ses points fixes si et seulement si ρ est une fermeture supérieure de L.

DEFINITION 4.2.2.0.2 *Partie de Moore inférieure*

Soit $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet, nous dirons que $M\subseteq L$ est une *partie de Moore inférieure* de L si et seulement si pour tout x de L l'ensemble $\{y\in M : x\sqsubseteq y\}$ n'est pas vide et admet un plus petit élément.

LEMME 4.2.2.0.3 *Caractérisation d'une partie de Moore inférieure*

Soit $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet alors $M\subseteq L$ est une partie de Moore inférieure de L si et seulement si :

(a) - $\{\top\in M\}$

(b) - $\{\forall S\subseteq M,\ (\sqcap S)\in M\}$.

Preuve : Soit $M\subseteq L$ tel que $\{\top\in M\}$ et $\{\forall S\subseteq M,\ (\sqcap S)\in M\}$ alors pour tout x de L l'ensemble $M'=\{y\in M : x\sqsubseteq y\}$ n'est pas vide (car $\top\in M$) et admet un plus petit élément $\sqcap M'$ (pour tout y de M', $(\sqcap M')\sqsubseteq y$ et $(\sqcap M')\in M'$).

Réciproquement soit M une partie de Moore inférieure de L. Pour $\top\in L$ l'ensemble $\{y\in M : \top\sqsubseteq y\}$ n'est pas vide donc il contient $\top\in M$. Soit $S\subseteq M$ tel que $(\sqcap S)\notin M$. Considérons $M'=\{y\in M : (\sqcap S)\sqsubseteq y\}$ et soit y_0 le plus petit élément de M'. On a $y_0\in M'$ et donc $(\sqcap S)\sqsubseteq y_0$. Mais $S\subseteq M'$ donc $y_0\sqsubseteq(\sqcap S)$ et par antisymétrie $y_0=(\sqcap S)\in M'\subseteq M$, en contradiction avec $(\sqcap S)\notin M$. Par l'absurde, on a $\{\forall S\subseteq M,\ (\sqcap S)\in M\}$.

Fin de la preuve.

THEOREME 4.2.2.0.4 Monteiro & Ribeiro[1942,th.5.3]

Soient $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet et $M\subseteq L$. Alors il existe une fermeture supérieure de L telle que $\rho(L)=M$ si et seulement si M est une partie de Moore inférieure de L (auquel cas $\rho=\lambda x.\sqcap\{y\in M : x\sqsubseteq y\}$).

THEOREME 4.2.2.0.5

Soient $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet et $S\subseteq L$. La fermeture supérieure

ρ de L telle que ρ(L) est la plus petite partie de Moore inférieure contenant S est $\rho=\lambda x.\sqcap\{y\in(S\cup\{T\}) : x\leq y\}$.

Preuve : Montrons tout d'abord que ρ est une fermeture supérieure :
- $\forall x\in L,\ x\leq\sqcap\{y\in(Su\{T\}) : x\leq y\}$ donc ρ est extensif.
- Si $x\leq z$ alors $\forall y\in(Su\{T\}),\ \{z\leq y\}$ implique $\{x\leq y\}$ donc $\sqcap\{y\in(Su\{T\}) : x\leq y\}\leq\sqcap\{y\in(Su\{T\}) : z\leq y\}$, ce qui montre la monotonie.
- Soit $y\in(Su\{T\})$ tel que $x\leq y$. Alors $(\sqcap\{z\in(Su\{T\}) : x\leq z\})\leq y$ et donc $\sqcap\{y\in(Su\{T\}) : (\sqcap\{z\in(Su\{T\}) : x\leq z\})\leq y\}\leq\sqcap\{y\in(Su\{T\}) : x\leq y\}$ soit $\rho(\rho(x))\leq\rho(x)$, de plus $\rho(x)\leq\rho(\rho(x))$ et donc par antisymétrie ρ est idempotente.

Maintenant :
- Pour tout $x\in S$, $\rho(x)=x$ donc $S\subseteq\rho(L)$.
- Soit θ une fermeture supérieure telle que $S\subseteq\theta(L)$. $\forall z\in\rho(L)$, $\exists y\in L$ tel que $z=\rho(y)=\sqcap\{x\in(Su\{T\}) : y\leq x\}$. Soit $R=\{t\in L : \theta(t)\in(Su\{T\})\}$. On a $z=\sqcap\{\theta(t) : y\leq\theta(t)$ et $t\in R\}$ qui est l'intersection d'éléments de θ(L) et donc d'après 2.3.0.1 $z\in\theta(L)$. Nous concluons $\rho(L)\subseteq\theta(L)$.

Fin de la preuve.

4.2.3 Treillis des fermetures supérieures d'un treillis complet et treillis des espaces induits

Les fermetures supérieures d'un treillis complet L sont partiellement ordonnées par l'ordre défini par les opérateurs de L c'est-à-dire $\{\rho\leq\eta\}\iff\{\forall x\in L,\ \rho(x)\leq\eta(x)\}$.

PROPOSITION 4.2.3.0.1 *Caractérisations de l'ordre sur les fermetures supérieures* (Ore[1943])

Soient ρ et η des fermetures supérieures de L, alors :

(a) - $\{\rho_1\leq\rho_2\}\iff\{\forall x\in L,\ \{\rho_2(x)=x\}\Rightarrow\{\rho_1(x)=x\}\}$

(b) - $\{\rho_1\leq\rho_2\}\iff\{\rho_2(L)\subseteq\rho_1(L)\}$

(c) - $\{\rho_1\leq\rho_2\}\iff\{\rho_1\circ\rho_2=\rho_2\}\iff\{\rho_2\circ\rho_1=\rho_2\}$

Il est bien connu que l'ensemble des fermetures supérieures d'un treillis complet est un treillis complet pour l'ordre ≤ défini ci-dessus :

PROPOSITION 4.2.3.0.2 *Treillis complet des fermetures supérieures d'un treillis complet* (Ward[1942])

Soit $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet. L'ensemble R des fermetures supérieures de L est un treillis complet $(\sqsubseteq, \lambda x.x, \lambda x.\top, \lambda S.\sqcap\{\eta \in R : \{\forall \rho \in S , \rho \sqsubseteq \eta\}\}, \sqcap)$.

Nous donnons maintenant une version constructive de ce théorème en utilisant notre version constructive du théorème de Tarski (2.5.5.0.1).

LEMME 4.2.3.0.3 *Treillis complet des opérateurs extensifs d'un treillis complet*

Soit $ext = \lambda f.(\lambda x.x \sqcup f(x))$. Alors *ext* est une fermeture supérieure de $L \to L$ et pour tout $f \in (L \to L)$, *ext*(f) est le plus petit opérateur extensif de L plus grand ou égal à f. L'ensemble $ext(L \to L)$ des opérateurs extensifs de L est un treillis complet $(\sqsubseteq, \lambda x.x, \lambda x.\top, \sqcup, \sqcap)$.

Preuve : Un opérateur de L est extensif si et seulement si $ext(f)=f$. On vérifie que *ext* est une fermeture supérieure de $L \to L$. De plus *ext* est un morphisme complet pour l'union et par conséquent $ext(L \to L)$ est un sous treillis complet de $L \to L$ (théorème 2.3.0.3). L'infimum de ce treillis est $ext(\lambda x.\bot) = \lambda x.x \sqcup \bot = \lambda x.x$.
Fin de la preuve.

LEMME 4.2.3.0.4 *Treillis complet des opérateurs monotones et extensifs d'un treillis complet*

- $mon \circ ext = ext \circ mon$ est une fermeture supérieure de $(L \to L)$
- $mon(L \to L) \cap ext(L \to L) = mon \circ ext(L \to L) = ext \circ mon(L \to L)$ est un sous-treillis complet $(\sqsubseteq, \lambda x.x, \top, \sqcup, \sqcap)$ de $(L \to L)$

Preuve : Comme $\sqsubseteq$ est réflexive, pour tout $f \in (L \to L)$ et $x \in L$ on a $x \sqsubseteq \sqcup\{y \sqcup f(y) : y \sqsubseteq x\}$ et par conséquent $mon(ext(f))(x) = \sqcup\{y \sqcup f(y) : y \sqsubseteq x\} = x \sqcup \{y \sqcup f(y) : y \sqsubseteq x\} = \sqcup\{x \sqcup y \sqcup f(y) : y \sqsubseteq x\} = \sqcup\{x \sqcup f(y) : y \sqsubseteq x\} = x \sqcup (\sqcup\{f(y) : y \sqsubseteq x\}) = ext(mon(f))(x)$. $mon \circ ext$ est composition d'opérateurs monotones et extensifs (2.4.0.3, 4.2.3.0.3) et par conséquent monotone et extensif. Comme *mon* et *ext* commutent et sont idempotents, $mon \circ ext$ est idempotent. $mon \circ ext$ est une fermeture supérieure de $(L \to L)$ qui est un morphisme

complet pour l'union et par conséquent $mon \circ ext(L \to L)$ est un sous-treillis complet $(\sqsubseteq, mon \circ ext(\bot) = \lambda x.x, \top, \sqcup, \sqcap)$ de $(L \to L)$ (théorème 2.3.0.3). Aussi $\forall f \in (L \to L)$, $(f \in mon(L \to L) \cap ext(L \to L)) \iff (mon(f)=f$ et $ext(f)=f) \iff (f=mon(ext(f)) \iff (f \in mon \circ ext(L \to L))$.
Fin de la preuve.

THEOREME 4.2.3.0.5 *Treillis complet des fermetures supérieures d'un treillis complet*

Soient $L(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$ un treillis complet, $idem = \lambda f.luis(\lambda g.g \circ g)(f)$ et $fers = idem \circ ext \circ mon = idem \circ mon \circ ext$. $fers$ est une fermeture supérieure de $(L \to L)$ et pour tout $f \in (L \to L)$, la plus petite fermeture supérieure de L plus grande ou égale à f est $fers(f)$. L'ensemble $fers(L \to L)$ des fermetures supérieures de L est un treillis complet $(\sqsubseteq, \lambda x.x, \top, \lambda S.fers(\sqcup S) = \lambda S.luis(\lambda g.g \circ g)(\sqcup S), \sqcap)$.

Preuve : D'après la définition 4.2.1.0.1 l'ensemble de toutes les fermetures supérieures du treillis complet L est l'ensemble des éléments de $(mon(L \to L) \cap ext(L \to L))$ qui sont idempotents c'est-à-dire points fixes de $\lambda g.g \circ g$. $\lambda g.g \circ g$ est un opérateur monotone de $(mon(L \to L) \cap ext(L \to L))$ car si $f \sqsubseteq g$ alors $f \in mon(L \to L)$ implique $f \circ f \sqsubseteq f \circ g$ et comme $f \circ g \sqsubseteq g \circ g$ on a $f \circ f \sqsubseteq g \circ g$. De plus $fp(\lambda g.g \circ g) = postfp(\lambda g.g \circ g)$ car si $f \in postfp(\lambda g.g\ g)$ alors $f \circ f \sqsubseteq f$ et $f \in ext(L \to L)$ implique $\lambda x.x \sqsubseteq f$ de sorte que $f \in mon(L \to L)$ implique $f \sqsubseteq f \circ f$ soit par antisymétrie $f = f \circ f$. D'après le théorème 2.5.3.0.2 l'ensemble des fermetures supérieures de L est donc le treillis complet $postfp(\lambda g.g \circ g)(\sqsubseteq, luis(\lambda g.g \circ g)(\lambda x.x) = \lambda x.x, \top, \lambda S.luis(\lambda g.g \circ g)(\sqcup S), \sqcap)$ qui est l'image du treillis complet $(mon(L \to L) \cap ext(L \to L))(\sqsubseteq, \lambda x.x, \top, \sqcup, \sqcap)$ par la fermeture supérieure $idem = \lambda f.luis(\lambda g.g \sqcup g \circ g)(f)$. Mais pour tout $f \in ext(L \to L)$ on a $f \in prefp(\lambda g.g \circ g)$ et par conséquent $idem = \lambda f.luis(\lambda g.g \circ g)(f)$.

$fers = idem \circ ext \circ mon$ est monotone et extensive en tant que composition d'opérateurs monotones et extensifs. Pour tout $f \in (L \to L)$, $fers(fers(f)) = idem \circ ext \circ mon(fers(f)) = idem \circ ext(fers(f)) = idem(fers(f)) = fers(f)$ car $fers(f)$ est monotone, extensive et idempotente. Donc $fers(f)$ est idempotente ce qui nous restait à montrer pour affirmer que c'est une fermeture supérieure.

D'après la proposition 2.3.0.4.(a) pour tout $f \in (L \to L)$ l'ensemble $\{\rho \in fers(L \to L) : f \sqsubseteq \rho\}$ des fermetures supérieures de L plus grandes ou égales à f n'est pas vide et admet un plus petit élément égal à $fers(f)$.
Fin de la preuve.

Remarquons qu'une fermeture supérieure ρ sur un treillis complet $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ est monotone (donc $\rho = mon(\rho)$, théorème 2.4.0.2), extensive (donc $\rho = ext(\rho)$, théorème 4.2.3.0.3) et idempotente (donc $\rho = \rho\circ\rho$) ce qui implique $\rho = mon(ext(\rho\circ\rho))$. Réciproquement si $\rho = mon(ext(\rho\circ\rho))$ alors ρ est monotone et donc $ext(\rho\circ\rho)$ l'est également ce qui implique que $\rho = ext(\rho\circ\rho)$. Comme ρ est extensive, $\rho\circ\rho$ qui l'est également est point fixe de *ext* ce qui implique que ρ est idempotente. Nous avons donc la caractérisation :

Si $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ est un treillis complet, $\rho \in (L\to L)$ alors
$\{\rho \in fers(L\to L)\} \Longleftrightarrow \{\rho=\lambda x.\sqcup\{y\sqcup\rho(\rho(y)) : (y\in L) \text{ et } (y\sqsubseteq x)\}$

Nous caractérisons *fers* de diverses manières utiles dans la suite :

PROPOSITION 4.2.3.0.6

$fers = \lambda f.(\lambda x.luis(\lambda y.ext(mon(f))(y))(x))$.

Preuve : D'après les théorèmes 2.5.2.0.5 et 2.5.3.0.1 nous savons que pour tout $f \in (L\to L)$, $\lambda x.luis(\lambda y.y\sqcup mon(f)(y))(x)$ est une fermeture supérieure de L plus grande que f. Soit ρ une fermeture supérieure de L telle que $f\sqsubseteq\rho$. Alors $ext(mon(f)) \sqsubseteq ext(mon(\rho)) = \rho$. Donc $\lambda x.luis(\lambda y.ext(mon(f))(y))(x) \sqsubseteq \lambda x.luis(\rho)(x) = \rho$ car ρ est idempotente. Nous en concluons que pour tout $f\in(L\to L)$, $\lambda x.luis(\lambda y.y\sqcup mon(f)(y))(x)$ est la plus petite fermeture supérieure de L plus grande ou égale à f, c'est donc $fers(f)$.
Fin de la preuve.

COROLLAIRE 4.2.3.0.7

Soit $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet, alors l'ensemble $fers(L\to L)$ des fermetures supérieures de L est un treillis complet ($\sqsubseteq$, $\lambda x.x$, $\lambda x.\top$, $\lambda S.luis(\sqcup S)$, $\sqcap$).

On sait (Devidé[1964]) que $\lambda x.lfp(\lambda y.x\sqcup f(y))$ est une fermeture supérieure de L quand f est monotone. Nous remarquons alors d'après le théorème 2.5.3.0.1 que $\lambda x.luis(\lambda y.y\sqcup f(y))(x) = \lambda x.luis(\lambda y.x\sqcup f(y))(x) = \lambda x.lfp(\lambda y.x\sqcup f(y))$ ce qui donne :

PROPOSITION 4.2.3.0.8

$fers = \lambda f.(\lambda x.lfp(\lambda y.x\sqcup mon(f)(y)))$.

COROLLAIRE 4.2.3.0.9

Soit $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet alors l'ensemble $fers(L\to L)$ des fermetures supérieures de L est un treillis complet $(\sqsubseteq, \lambda x.x, \lambda x.\top, \lambda S.(\lambda x.lfp(\lambda y.x\sqcup(\sqcup S)(y))), \sqcap)$.

On déduit immédiatement de 4.2.3.0.1.(b) le

THEOREME 4.2.3.0.10 (Ward[1942])

Soit $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet alors $fers(L\to L)$ est dual isomorphe au treillis complet des parties de Moore inférieures de L pour l'ordre $\subseteq$ (inclusion d'ensembles), infimum $\{\top\}$, supremum L, intersection $\cap$ (intersection d'ensembles) et union $\lambda S.\{\sqcap P : P\subseteq\cup S\}$.

4.2.4 Composition de fermetures supérieures d'un treillis complet

La composition $\rho_1\circ\rho_2$ de deux fermetures supérieures ρ_1 et ρ_2 de $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ est monotone et extensive mais pas nécessairement idempotente de sorte que $\rho_1\circ\rho_2$ n'est pas nécessairement une fermeture. Nous donnons donc des conditions nécessaires et suffisantes pour que la composition de deux fermetures supérieures soit une fermeture supérieure. Ayant démontré :

PROPOSITION 4.2.4.0.1

$\forall\rho_1,\rho_2\in fers(L\to L).\ fers(\rho_1\sqcup\rho_2) = fers(\rho_1\circ\rho_2) = fers(\rho_2\circ\rho_1)$.

Preuve : $\rho_1\sqsubseteq fers(\rho_1\sqcup\rho_2)$ et $\rho_2\sqsubseteq fers(\rho_1\sqcup\rho_2)$ impliquent par monotonie que $\rho_1\circ\rho_2\sqsubseteq fers(\rho_1\sqcup\rho_2)\circ fers(\rho_1\sqcup\rho_2) = fers(\rho_1\sqcup\rho_2)$. Comme $\lambda x.x\sqsubseteq\rho_2$ il vient $\rho_1\sqsubseteq\rho_1\circ\rho_2$ et aussi $\rho_2\sqsubseteq\rho_1\circ\rho_2$ car ρ_1 est extensive donc $\rho_1\sqcup\rho_2\sqsubseteq\rho_1\circ\rho_2\sqsubseteq fers(\rho_1\sqcup\rho_2)$. Donc $fers(\rho_1\sqcup\rho_2)\sqsubseteq fers(\rho_1\circ\rho_2)\sqsubseteq fers(fers(\rho_1\circ\rho_2)) = fers(\rho_1\sqcup\rho_2)$.
Fin de la preuve.

Les propositions suivantes dues à Ore[1943b,524-526] s'en déduisent immédiatement :

PROPOSITION 4.2.4.0.2 (Ore[1943b])

Si ρ_1 et ρ_2 sont des fermetures supérieures du treillis complet $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ alors $\mathit{fers}(\rho_1 \sqcup \rho_2)$ est la plus petite fermeture supérieure de L plus grande ou égale à $\rho_1 \circ \rho_2$.

PROPOSITION 4.2.4.0.3 (Ore[1943b])

Si ρ_1 et ρ_2 sont des fermetures supérieures du treillis complet $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ alors $\rho_1 \circ \rho_2$ est une fermeture supérieure de L si et seulement si $\rho_1 \circ \rho_2 = \mathit{fers}(\rho_1 \sqcup \rho_2)$.

PROPOSITION 4.2.4.0.4 (Ore[1943b])

Une condition nécessaire et suffisante pour que le produit $\rho_1 \circ \rho_2$ de deux fermetures supérieures ρ_1 et ρ_2 du treillis complet L soit une fermeture supérieure de L est que $\rho_2 \circ \rho_1 \circ \rho_2 = \rho_1 \circ \rho_2$.

THEOREME 4.2.4.0.5 (Ore[1943b])

Une condition nécessaire et suffisante pour que les produits $\rho_1 \circ \rho_2$ et $\rho_2 \circ \rho_1$ de deux fermetures supérieures ρ_1 et ρ_2 du treillis complet L soient tous les deux des fermetures supérieures de L est que ρ_1 et ρ_2 commutent (i.e. $\rho_1 \circ \rho_2 = \rho_2 \circ \rho_1$).

Pour définir des fermetures supérieures sur L nous procéderons souvent en deux temps : nous définirons d'abord $\rho \in \mathit{fers}(L \to L)$ puis $\eta \in \mathit{fers}(\rho(L) \to \rho(L))$ ce qui donne $\eta \circ \rho \in \mathit{fers}(L \to L)$. Le procédé peut être répété en cascade.

LEMME 4.2.4.0.6

Soient $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet et $\rho \in \mathit{fers}(L \to L)$. Alors si $\eta \in \mathit{fers}(\rho(L) \to \rho(L))$, $\eta \circ \rho \in \mathit{fers}(L \to L)$ et $\rho \sqsubseteq \eta \circ \rho$.

Preuve : Comme η est extensive, on a $\rho \sqsubseteq \eta \circ \rho$ donc d'après 4.2.3.0.1.(c) $\rho \circ \eta \circ \rho = \eta \circ \rho$ donc d'après 4.2.4.0.4 on obtient $\eta \circ \rho \in \mathit{fers}(L \to L)$.
Fin de la preuve.

THEOREME 4.2.4.0.7

Soient $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet et $\rho \in \mathit{fers}(L \to L)$. Alors le

treillis complet $fers(\rho(L)\to\rho(L))$ est isomorphe au treillis complet $\{\theta \in fers(L\to L) : \rho \sqsubseteq \theta\}$ par l'isomorphisme complet $\lambda\eta.\eta\circ\rho$, l'isomorphisme inverse est $\lambda\theta.\theta\circ\rho$.

Preuve : Soit $\eta \in fers(\rho(L)\to\rho(L))$, nous avons $\eta\circ\rho\circ\rho = \eta\circ\rho$ mais quand $\eta\circ\rho$ est appliqué à un élément x de $\rho(L)$ il vient $\eta(\rho(x)) = \eta(x)$. Réciproquement soit $\theta \in fers(L\to L)$ tel que $\rho \sqsubseteq \theta$ nous avons $(\theta\circ\rho)\circ\rho = \theta\circ\rho = \theta$ car $\rho \sqsubseteq \theta$ et théorème 4.2.3.0.1.(c). Comme $(\lambda\eta.\eta\circ\rho)\circ(\lambda\theta.\theta\circ\rho)$ et $(\lambda\theta.\theta\circ\rho)\circ(\lambda\eta.\eta\circ\rho)$ sont l'identité nous savons que $\lambda\eta.\eta\circ\rho$ est bijective et que $\lambda\theta.\theta\circ\rho$ est son inverse.

Soit $\{\theta_i : i\in I\}$ une famille d'éléments de $\{\theta \in fers(L\to L) : \rho \sqsubseteq \theta\}$, alors $(\bigsqcap_{i\in I}\theta_i)\circ\rho = \bigsqcap_{i\in I}(\theta_i\circ\rho)$. De même $fers(\bigsqcup_{i\in I}\theta_i)\circ\rho = fers(\bigsqcup_{i\in I}\theta_i\circ\rho)\circ\rho$ (car $\theta_i\circ\rho=\theta_i$ quand $\rho \sqsubseteq \theta_i$) et $fers(\bigsqcup_{i\in I}\theta_i\circ\rho) \in fers(\rho(L)\to\rho(L))$. En effet $fers(\bigsqcup_{i\in I}\theta_i\circ\rho) = fers((\bigsqcup_{i\in I}\theta_i)\circ\rho) = fers((\bigsqcup_{i\in I}\theta_i)\sqcup\rho) \sqsupseteq fers(\rho) = \rho$ et comme $\rho \sqsubseteq fers(\bigsqcup_{i\in I}\theta_i\circ\rho)$ nous avons $fers(\bigsqcup_{i\in I}\theta_i\circ\rho)(L) \leq \rho(L)$. Il vient donc $fers(\bigsqcup_{i\in I}\theta_i\circ\rho) = fers(\bigsqcup_{i\in I}\theta_i\circ\rho)\circ\rho$ et par transitivité $fers(\bigsqcup_{i\in I}\theta_i)\circ\rho = fers(\bigsqcup_{i\in I}\theta_i\circ\rho)$. Nous avons montré que $\lambda\theta.\theta\circ\rho$ est une bijection de $\{\theta \in fers(L\to L) : \rho \sqsubseteq \theta\}$ dans $fers(\rho(L)\to\rho(L))$ et également un morphisme complet. La composition de $\lambda\theta.\theta\circ\rho$ et $\lambda\eta.\eta\circ\rho$ étant l'identité nous en déduisons immédiatement que $\lambda\eta.\eta\circ\rho$ est un isomorphisme complet de $fers(\rho(L)\to\rho(L))$ dans $\{\theta \in fers(L\to L) : \rho \sqsubseteq \theta\}$.
Fin de la preuve.

4.2.5 Définition d'une fermeture supérieure par une famille d'idéaux principaux

PROPOSITION 4.2.5.0.1

Soient $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet et $\rho \in fers(L\to L)$. L'ensemble des éléments de L qui ont même fermeture par ρ est un sous-sup-demi-treillis de L complet et convexe. (Rappelons que $S\subseteq L$ est convexe si et seulement si $\{\forall x,y\in S,\ \forall t\in L,\ \{x \sqsubseteq t \sqsubseteq y\} \Rightarrow \{t\in S\}\}$).

Preuve : Soit $a \in \rho(L)$ et $S_a = \{x \in L : \rho(x)=a\}$. S_a n'est pas vide car $\rho(a)=a$. Soit S un sous-ensemble non vide de S_a. Alors $\forall x \in S$, $x \sqsubseteq \rho(x)=a$ et donc $(\bigsqcup S) \sqsubseteq a$ soit par monotonie $\rho(\bigsqcup S) \sqsubseteq \rho(a) = a$. Encore par monotonie $\rho(\bigsqcup S) \sqsupseteq \bigsqcup \rho(S) = a$ et donc $\rho(\bigsqcup S)=a$ ce qui montre que $(\bigsqcup S) \in S_a$ et implique que S_a est un sous-sup-demi-treillis complet de L. Maintenant si $x,y \in S_a$ et $x \sqsubseteq t \sqsubseteq y$ on a $a = \rho(x) \sqsubseteq \rho(t) \sqsubseteq \rho(y) = a$ soit $\rho(t) = a$ ce qui implique $t \in S_a$ et montre que S_a est convexe.
Fin de la preuve.

Un *idéal* J d'un treillis complet L est un sous-ensemble non vide de L tel que (i) : $\{\{a \in J,\ x \in L,\ x \sqsubseteq a\} \Rightarrow \{x \in J\}\}$, (ii) : $\{\{a \in J,\ b \in J\} \Rightarrow \{(a \sqcup b) \in J\}\}$. Une caractérisation équivalente est qu'un idéal J de L est tel que $\{J \sqsubseteq L,\ J \neq \emptyset,\ \{a \in J,\ b \in J\} \Leftrightarrow \{(a \sqcup b) \in J\}\}$. L'intersection d'une famille infinie d'idéaux de L est un idéal de L.

Les *idéaux principaux* de L sont les sous-ensembles $\{x \in L : x \sqsubseteq a\}$ pour tous les a de L. L'intersection d'une famille infinie d'idéaux principaux de L est encore un idéal principal de L. En particulier dans un treillis satisfaisant la condition de chaîne ascendante tout idéal est principal.

Un *semi-idéal* I d'un treillis complet L est un sous-ensemble de L tel que $\{\forall a \in I,\ \{x \in L \text{ et } x \sqsubseteq a\} \Rightarrow \{x \in I\}\}$. Un *semi-idéal dual* J de L est tel que $\{\forall a \in J,\ \{x \in L \text{ et } a \sqsubseteq x\} \Rightarrow \{x \in J\}\}$.

PROPOSITION 4.2.5.0.2

Soient I un idéal principal du treillis complet $L(\sqsubseteq, \bot, \top, \bigsqcup, \bigsqcap)$ et J un semi-idéal dual de L. Si $(I \cap J) \neq \emptyset$ alors $I \cap J$ est un sous-sup-demi-treillis complet et convexe de L. De plus tout sous-sup-demi-treillis complet et convexe C de L peut s'exprimer sous la forme $I \cap J$ où $I = \{x \in L : x \sqsubseteq (\bigsqcup C)\}$ et $\{x \in L : \{\exists y \in C : y \sqsubseteq x\}\} \sqsubseteq J$.

Preuve : Posons $D = I \cap J$ et soit $S \sqsubseteq D$ tel que $S \neq \emptyset$. Alors $S \sqsubseteq I$ et donc $(\bigsqcup S) \sqsubseteq (\bigsqcup I)$ de sorte que $(\bigsqcup S) \in I$. Comme S n'est pas vide $\{\exists x \in S : x \in J \text{ et } x \sqsubseteq (\bigsqcup S)\}$ et donc $(\bigsqcup S) \in J$ soit $(\bigsqcup S) \in D$ qui est donc un sous-sup-demi-treillis complet de L. Si $x, y \in D$ et $t \in L$ est tel que $x \sqsubseteq t \sqsubseteq y$ alors $t \sqsubseteq y$ et $y \in I$ impliquent $t \in I$. Egalement $x \sqsubseteq t$ et $x \in I$ impliquent $t \in J$ soit $t \in D$ ce qui montre que D est convexe.

Soit C un sous-sup-demi-treillis complet et convexe de L. Posons $I=\{x\in L : x\leq(\sqcup C)\}$. C'est un idéal principal de L. Posons $J=\{x\in L : \exists y\in C : y\leq x\}$, c'est un semi-idéal dual de L. Alors $C\subseteq(I\cap J)$ car $\forall x\in C$, $x\leq x\leq(\sqcup C)$. Si $t\in(I\cap J)$ alors $t\in I$ et donc $t\leq(\sqcup C)$ avec $(\sqcup C)\in C$. Aussi $t\in J$ de sorte que $\exists c\in C$ tel que $c\leq t$. Comme C est convexe $t\in C$ ce qui montre que $C=I\cap J$.

Supposons maintenant que C s'exprime sous la forme $C=I_1\cap J_1$. Comme $C\subseteq I_1$ il vient $\{x\in L : x\leq(\sqcup C)\}\leq I_1$. Soient $a\in I_1$ et x un élément arbitraire de C. Alors $(a\sqcup x)\in I_1$. D'autre part $(a\sqcup x)\geq x$ et $x\in J_1$ impliquent que $(a\sqcup x)\in J_1$ et $(a\sqcup x)\in(I_1\cap J_1)=C$. Donc $a\leq(a\sqcup x)\leq(\sqcup C)\in\{x\in L : x\leq(\sqcup C)\}$ soit $I_1\subseteq\{x\in L : x\leq(\sqcup C)\}$ et par antisymétrie $I_1=\{x\in L : x\leq(\sqcup C)\}$. De même comme $C\subseteq J_1$ nous avons $J=\{x\in L : \{\exists y\in C : y\leq x\}\}\subseteq J_1$. Toutefois le choix de J_1 n'est pas unique comme le montre le contre-exemple suivant :

Fin de la preuve.

THEOREME 4.2.5.0.3

Soit $\{I_i : i\in\Delta\}$ une famille d'idéaux principaux du treillis complet $L(\subseteq,\perp,\top,\sqcup,\sqcap)$. Alors $\lambda x.\ \sqcup(\cap\{J\in(\{L\}\cup\{I_i : i\in\Delta\}) : x\in J\})$ est la fermeture supérieure de L générée par $\{I_i : i\in\Delta\}$.

Preuve : Pour $x\in L$ posons $S_x=\cap\{J\in(\{L\}\cup\{I_i : i\in\Delta\}) : x\in J\}$ et $\rho(x)=(\sqcup S_x)$. Comme $x\in S_x$ on a $x\in S_x$ et $x\leq(\sqcup S_x)$ ce qui montre que ρ est extensive. Si $x\leq y$ alors $S_x\subseteq S_y$ car $y\in J$, $x\in L$ et $x\leq y$ impliquent $x\in J$ pour tout idéal J de L. Donc $\rho(x)=(\sqcup S_x)\leq(\sqcup S_y)=\rho(y)$ ce qui montre que ρ est monotone. Pour tout $J\in(\{L\}\cup\{I_i : i\in\Delta\})$ on a $x\in J$ implique que $(\sqcup S_x)\in J$ car $x\in J$ implique $S_x\subseteq J$. Donc $\cap\{J\in(\{L\}\cup\{I_i : i\in\Delta\}) : (\sqcup S_x)\in J\}$ est inclus dans $\cap\{J\in(\{L\}\cup\{I_i : i\in\Delta\}) : x\in J\}$ et alors $\rho(\rho(x))\leq\rho(x)$. De plus $\rho(x)\leq\rho(\rho(x))$ car ρ est extensive et monotone et par antisymétrie nous concluons que ρ est idempotente.
Fin de la preuve.

COROLLAIRE 4.2.5.0.4

Soient $L(\sqsubseteq,\perp,\top,\sqcup,\sqcap)$ un treillis complet et $\rho \in fers(L \to L)$ alors ρ est égal à la fermeture supérieure de L générée par $\{\{y \in L : y \sqsubseteq a\} : a \in \rho(L)\}$.

Preuve : Comme $\top \in \rho(L)$ nous avons $\lambda x.\sqcup\cap\{J \in \{\{y \in L : y \sqsubseteq a\} : a \in \rho(L)\} : x \in J\} = \lambda x.\sqcup\cap\{\{y \in L : y \sqsubseteq a\} : a \in \rho(L)$ et $x \sqsubseteq a\} = \lambda x.\sqcap\{a \in \rho(L) : x \sqsubseteq a\} = \rho(x)$.
Fin de la preuve.

4.2.6 Définition d'une fermeture supérieure par une relation de congruence complète pour l'union

DEFINITION 4.2.6.0.1 *Relation de congruence complète pour l'union*

Soit $L(\sqsubseteq,\perp,\top,\sqcup,\sqcap)$ un treillis complet alors une *relation de congruence complète pour l'union* sur L est une relation d'équivalence θ satisfaisant la *propriété de substitution pour l'union* $\{\forall x, y, u \in L, \{\{x \equiv y(\theta)\} \Rightarrow \{(x \sqcup u) \equiv (y \sqcup u)(\theta)\}\}$ et ayant la *propriété de complétude* $\{\forall x \in L, x \equiv \sqcup([x]\theta)(\theta)\}$ où $[x]\theta = \{y \in L : x \equiv y(\theta)\}$.

La propriété de substitution pour l'union peut s'exprimer sous la forme équivalente $\{\forall x_1, y_1, x_2, y_2 \in L, \{\{x_1 \equiv y_1(\theta)\}$ et $\{x_2 \equiv y_2(\theta)\}\} \Rightarrow \{(x_1 \sqcup x_2) \equiv (y_1 \sqcup y_2)(\theta)\}\}$.

PROPOSITION 4.2.6.0.2

Soient $L(\sqsubseteq,\perp,\top,\sqcup,\sqcap)$ un treillis complet et $\rho \in fers(L \to L)$ alors la relation (ρ) définie par $\{x \equiv y(\rho)\} \Leftrightarrow \{\rho(x) = \rho(y)\}$ est une relation de congruence complète pour l'union.

Preuve : Pour tout $\rho \in (L \to L)$ il est connu que (ρ) est une relation d'équivalence. Supposons que $\rho \in fers(L \to L)$ et $x, y, u \in L$ alors $\rho(x) = \rho(y)$ implique $\rho(x \sqcup u) = \rho(\rho(x) \sqcup \rho(u)) = \rho(\rho(y) \sqcup \rho(u)) = \rho(y \sqcup u)$ car ρ est un quasi-morphisme complet supérieur (4.2.1.0.8). Par conséquent $x \equiv y(\rho)$ implique $(x \sqcup u) \equiv (y \sqcup u)(\rho)$. De plus $\forall y \in [x](\rho)$, $\rho(y) = \rho(x)$ de-sorte que $\sqcup\{\rho(y) : y \in [x](\rho)\} = \rho(x)$. Maintenant $\rho(\sqcup([x](\rho))) = \rho(\sqcup\rho([x](\rho))) = \rho(\sqcup\{\rho(y) : y \in [x](\rho)\}) = \rho(\rho(x)) = \rho(x)$ soit $x \equiv \sqcup([x](\rho))(\rho)$.
Fin de la preuve.

COROLLAIRE 4.2.6.0.3

Soit $\rho \in \mathit{fers}(L \to L)$ alors $\rho = \lambda x.\bigsqcup([x](\rho))$.

THEOREME 4.2.6.0.4

Soit θ une relation de congruence complète pour l'union sur L, alors pour tout x de L, $[x]\theta$ est un sous-sup-demi-treillis complet et convexe de L. De plus $\lambda x.\bigsqcup([x]\theta) \in \mathit{fers}(L \to L)$.

Preuve : Soit $x \in L$ et $y, z \in ([x]\theta)$. Soit $t \in L$ tel que $y \sqsubseteq t \sqsubseteq z$. Comme $y \equiv z(\theta)$, il vient $(y \sqcup t) \equiv (z \sqcup t)(\theta)$ de plus $t = y \sqcup t$ et $z \sqcup t = z$ donc $t \equiv z(\theta)$ et $z \equiv x(\theta)$ qui impliquent $t \equiv x(\theta)$ et $t \in ([x]\theta)$ donc $[x]\theta$ est un sous-ensemble convexe de L.

Soit $S \subseteq ([x]\theta)$ tel que $S \neq \emptyset$. $\exists y \in S$ tel que $y \equiv x(\theta)$. D'autre part $\bigsqcup([x]\theta) \equiv x(\theta)$ et $y \sqsubseteq \bigsqcup S \sqsubseteq \bigsqcup([x]\theta)$ donc $(\bigsqcup S) \in ([x]\theta)$, ce qui montre que $[x]\theta$ est un sous-sup-demi treillis complet de L.

Posons $\rho = \lambda x.\bigsqcup([x]\theta)$. ρ est extensive car $x \in ([x]\theta)$ implique $x \sqsubseteq \bigsqcup([x]\theta) = \rho(x)$. De plus $\bigsqcup([x]\theta) \in ([x]\theta)$ de sorte que $\rho(\rho(x)) = \rho(\bigsqcup([x]\theta)) = \bigsqcup[\bigsqcup([x]\theta)]\theta = \bigsqcup([x]\theta) = \rho(x)$ et ρ est idempotente. Si $x \sqsubseteq y$ alors $y = x \sqcup y \equiv (\rho(x) \sqcup \rho(y))(\theta)$ par la propriété de substitution pour l'union. Donc $y \equiv \rho(y)(\theta)$ et par transitivité il vient que $\rho(x) \sqcup \rho(y)$ appartient à $[\rho(y)]\theta$ et par conséquent $(\rho(x) \sqcup \rho(y)) \sqsubseteq \bigsqcup([\rho(y)]\theta) = \rho(y)$. Comme $\rho(y) \sqsubseteq \rho(x) \sqcup \rho(y)$ nous concluons par antisymétrie que $\rho(x) \sqsubseteq \rho(y)$ et ρ est monotone. *Fin de la preuve.*

De longs calculs sont parfois nécessaires pour montrer qu'une relation binaire donnée est une relation de congruence complète pour l'union. Les calculs sont souvent facilités par le théorème suivant (que l'on peut comparer au théorème établi pour les congruences dans Grätzer & Schmidt[1958]).

PROPOSITION 4.2.6.0.5

Une relation binaire réflexive et symétrique θ sur un treillis complet $L(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$ est une relation de congruence pour l'union si et seulement si les trois propriétés suivantes sont satisfaites pour tous $x, y, z, t \in L$ et $S \subseteq L$:

(a) - $\{x \equiv y(\theta)\} \iff \{\exists u \in L : (x \sqcup y) \sqsubseteq u \text{ et } u \equiv x(\theta) \text{ et } u \equiv y(\theta)\}$
(b) - $\{x \sqsubseteq y \sqsubseteq z \text{ et } x \equiv y(\theta) \text{ et } y \equiv z(\theta)\} \Rightarrow \{x \equiv z(\theta)\}$
(c) - $\{x \sqsubseteq y \text{ et } x \equiv y(\theta)\} \Rightarrow \{(x \sqcup t) \equiv (y \sqcup t)(\theta)\}$

Preuve : Une relation de congruence complète pour l'union satisfait (a) car $\{x \equiv y(\theta)\} \Rightarrow \{(x \sqcup y) \equiv x(\theta) \text{ et } (x \sqcup y) \equiv y(\theta)\}$ et $\{\exists u \in L : u \equiv x(\theta) \text{ et } u \equiv (\sqcup S)(\theta)\} \Rightarrow \{x \equiv (\sqcup S)(\theta)\}$. (b) est vrai par transitivité et (c) est vrai à cause de la réflexivité et la propriété de substitution pour l'union.

Réciproquement soit une relation θ binaire, réflexive, symétrique et satisfaisant (a), (b) et (c).

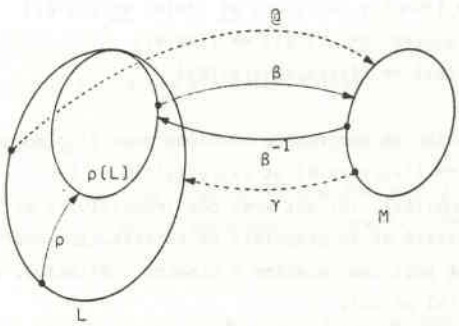
- θ est transitive car $x \equiv y(\theta)$ et $y \equiv z(\theta)$ et (a) impliquent que $\{\exists u, u' \in L : x \sqsubseteq u,\ y \sqsubseteq u,\ z \sqsubseteq u',\ y \sqsubseteq u',\ x \equiv u(\theta),\ y \equiv u(\theta),\ y \equiv u'(\theta),\ z \equiv u'(\theta)$. Alors d'après (c) il vient $u' = (y \sqcup u') \equiv (u \sqcup u')(\theta)$ et $u = (y \sqcup u) \equiv (u \sqcup u')(\theta)$. D'après (b), $x \sqsubseteq u \sqsubseteq (u \sqcup u')$ donc $x \equiv (u \sqcup u')(\theta)$, de même $z \equiv (u \sqcup u')(\theta)$. Il vient $(x \sqcup z) \equiv (u \sqcup u')(\theta)$ et donc d'après (a), nous concluons $x \equiv z(\theta)$.
- θ satisfait la propriété de substitution pour l'union. Il faut montrer que $x \equiv y(\theta)$ implique $(x \sqcup t) \equiv (y \sqcup t)(\theta)$. D'après (a) $\{\exists u \in L : x \sqsubseteq u,\ y \sqsubseteq u,\ x \equiv u(\theta),\ y \equiv u(\theta)\}$ donc d'après (c) $(x \sqcup t) \equiv (u \sqcup t)(\theta)$ et $(y \sqcup t) \equiv (u \sqcup t)(\theta)$. La propriété de transitivité que nous venons de démontrer implique $(x \sqcup t) \equiv (y \sqcup t)(\theta)$.

Fin de la preuve.

Pour vérifier que la relation de congruence est complète pour l'union il suffit de montrer la propriété supplémentaire $\{\forall x \in L,\ x \equiv \sqcup([x]\theta)(\theta)\}$.

4.2.7 Définition d'une fermeture supérieure par une paire de fonctions adjointes

Soient L un treillis complet et $\rho \in \mathit{fers}(L \to L)$. Pour représenter les éléments de $\rho(L)$, nous utiliserons un treillis complet $M(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$ isomorphe à L par un isomorphisme complet $\beta \in (\rho(L) \to M)$. Soit @ égale à $\beta \circ \rho$ et soit γ l'extension de β^{-1} à L :



Par référence à nos premiers travaux (Cousot & Cousot[1975], Cousot & Cousot[1976]), nous appellerons @ une opération d'*abstraction* et γ une opération de *concrétisation*. Il est facile de vérifier que @ est monotone et surjective, γ est monotone et injective, $@\circ\gamma=\lambda x.x$ et $\gamma\circ@ \sqsupseteq \lambda x.x$ ce qui implique immédiatement que $\{\forall x \in L, \forall y \in M, \{x \sqsubseteq \gamma(y)\} \Longleftrightarrow \{@(x) \sqsubseteq y\}\}$. De plus γ est un morphisme complet et @ un morphisme complet pour l'union ; (en effet, soit $\{x_i : i\in\Delta\}$ une famille d'éléments de L, alors par monotonie $@(\bigsqcup_{i\in\Delta} x_i) \sqsupseteq \bigsqcup_{i\in\Delta} @(x_i)$. De plus $@(\bigsqcup_{i\in\Delta} x_i) \sqsubseteq @(\bigsqcup_{i\in\Delta} \gamma(@(x_i))) = @(\gamma(\bigsqcup_{i\in\Delta} @(x_i))) = \bigsqcup_{i\in\Delta} @(x_i)$. Par antisymétrie, il vient $@(\bigsqcup_{i\in\Delta} x_i) = \bigsqcup_{i\in\Delta} @(x_i)$).

DEFINITION 4.2.7.0.1 *Paire de fonctions adjointes supérieure*

Soient $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ et $M(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ deux treillis complets. Une paire de fonctions monotones $@ \in (L \to M)$ et $\gamma \in (M \to L)$ est dite paire de fonctions adjointes supérieure si $\{\forall x \in L, \forall y \in M, \{\{x \sqsubseteq \gamma(y)\} \Longleftrightarrow \{@(x) \sqsubseteq y\}\}\}$.

La notion de paire de fonctions adjointes se trouvait déjà dans Cousot & Cousot[1975] avec les mêmes hypothèses, cependant nous avons emprunté le terme "paire de fonctions adjointes" à Scott[1977] qui l'utilise pour une notion duale avec les hypothèses supplémentaires que @ et γ sont continues supérieurement et L et M des "continuous lattices". Le théorème 4.2.7.0.3 généralise un résultat de Scott et montre que ces hypothèses supplémentaires sont inutiles.

PROPOSITION 4.2.7.0.2

Soient $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ et $M(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ deux treillis complets et $\rho \in \mathit{fers}(L \to L)$ tels que $\beta \in (\rho(L) \to M)$ est un isomorphisme complet. Alors $(@,\gamma)$ est une paire de fonctions adjointes supérieure, $@$ est surjective, γ est injective, $@$ est un morphisme complet pour l'union, γ est un morphisme complet.

Inversement, nous utiliserons des paires de fonctions adjointes pour définir des fermetures supérieures :

THEOREME 4.2.7.0.3

Soient $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ et $M(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ deux treillis complets et $(@,\gamma)$ où $@ \in (L \to M)$ et $\gamma \in (M \to L)$ une paire de fonctions adjointes supérieure.

(a) - Dans une paire $(@,\gamma)$ de fonctions adjointes supérieure chaque fonction détermine l'autre de manière unique et $\lambda x.x \sqsubseteq \gamma\circ@$ et $@\circ\gamma \sqsubseteq \lambda y.y$.

(b) - $@$ est surjective si et seulement si γ est injective.

(c) - Si $@$ est surjective ou γ est injective alors :

- $\gamma\circ@ \in \mathit{fers}(L \to L)$ et $@\circ\gamma = \lambda y.y$,
- $@$ est un morphisme complet pour l'union et γ est un morphisme complet pour l'intersection,
- $@ = \lambda x.\sqcap\{y \in M : x \sqsubseteq \gamma(y)\}$ et $\gamma = \lambda y.\sqcup\{x \in L : @(x) \sqsubseteq y\}$,
- $\gamma\circ@(L)$ et M sont isomorphes par l'isomorphisme complet $(@|\gamma\circ@(L))$ dont l'inverse est γ.

Preuve :

(a) - $\forall x \in L$, $@(x) \sqsubseteq @(x)$ et d'après 4.2.7.0.1 ceci implique que $x \sqsubseteq \gamma(@(x))$. De même $\forall y \in M$, $\gamma(y) \sqsubseteq \gamma(y)$ implique $@(\gamma(y)) \sqsubseteq y$.

Soit f telle que (f,γ) est une paire de fonctions adjointes supérieure. Alors $\forall x \in L$, $x \sqsubseteq \gamma(f(x))$ et comme $@$ est monotone nous avons $@(x) \sqsubseteq @(\gamma(f(x))) \sqsubseteq f(x)$. De plus $\forall x \in L$, $x \sqsubseteq \gamma(@(x))$ et comme f est monotone il vient $f(x) \sqsubseteq f(\gamma(@(x))) \sqsubseteq @(x)$. Par antisymétrie nous concluons que $f=@$.

Soit f telle que $(@,f)$ est une paire de fonctions adjointes supérieure. Alors $\forall y \in L$, $@(f(y)) \sqsubseteq y$ donc comme γ est monotone il vient $\gamma(y) \sqsupseteq \gamma(@(f(y))) \sqsupseteq f(y)$. De même $f(y) \sqsupseteq f(@(\gamma(y)) \sqsupseteq \gamma(y)$ et par antisymétrie nous concluons que $f=\gamma$.

(b) - Supposons γ injective et montrons que @ est surjective. Pour tout y de M on a $@(\gamma(y)) \sqsubseteq y$ et par monotonie $\gamma(@(\gamma(y))) \sqsubseteq \gamma(y)$. De plus $\gamma(y) \sqsubseteq \gamma(@(\gamma(y)))$ donc par antisymétrie $\gamma(y) = \gamma(@(\gamma(y)))$. Comme γ est injective ceci implique que $@(\gamma(y)) = y$. Donc pour tout y de M il existe $x=\gamma(y)\in L$ tel que $y=@(x)$ et donc @ est surjective.

Supposons @ surjective et montrons que γ est injective. En effet, $\forall y\in M,\ \exists x\in L : y=@(x)$. Donc $\gamma(y) = \gamma(@(x)) \sqsupseteq x$. La monotonie de @ implique que $@(\gamma(y)) \sqsupseteq @(x) = y$. De plus, il est toujours vrai que $@(\gamma(y)) \sqsubseteq y$ donc par antisymétrie $@(\gamma(y))=y$. Soient $y_1, y_2\in M$ alors $\gamma(y_1)=\gamma(y_2)$ implique $@(\gamma(y_1)) = @(\gamma(y_2))$ qui implique $y_1 = y_2$ c'est-à-dire que γ est injective.

(c) - On aura noté que si γ est injective ou @ surjective on a $@\circ\gamma=\lambda y.y$. Montrons que $\gamma\circ@$ est une fermeture supérieure de L. $\forall x \in L$, $x \sqsubseteq \gamma(@(x))$ de sorte que $\gamma\circ@$ est extensive. $\gamma\circ@$ est composition de fonctions monotones donc monotone. Comme $@\circ\gamma=\lambda y.y$ on a $(\gamma\circ@)\circ(\gamma\circ@) = \gamma\circ@$ ce qui montre l'idempotence.

- Soit $\{x_i : i\in\Delta\}$ une famille d'éléments de L. Alors par monotonie $@(\bigsqcup_{i\in\Delta} x_i) \sqsupseteq \bigsqcup_{i\in\Delta} @(x_i)$. De plus $@(\bigsqcup_{i\in\Delta} x_i) \sqsubseteq @(\bigsqcup_{i\in\Delta} \gamma(@(x_i))) \sqsubseteq @(\gamma(\bigsqcup_{i\in\Delta} @(x_i)))$ $= \bigsqcup_{i\in\Delta} @(x_i)$ par monotonie de γ, @ et $@\circ\gamma=\lambda y.y$. Par antisymétrie @ est un morphisme complet pour l'union. De même par monotonie $\gamma(\bigsqcap_{i\in\Delta} x_i) \sqsubseteq \bigsqcap_{i\in\Delta} \gamma(x_i)$ et $\bigsqcap_{i\in\Delta} \gamma(x_i) \sqsubseteq \gamma(@(\bigsqcap_{i\in\Delta} \gamma(x_i))) \sqsubseteq \gamma(\bigsqcap_{i\in\Delta} @(\gamma(x_i))) = \gamma(\bigsqcap_{i\in\Delta} x_i)$ et par antisymétrie nous concluons que γ est un morphisme complet pour l'intersection.

- Soit $f = \lambda y.\bigsqcup\{x\in L : @(x) \sqsubseteq y\}$. Comme @ est surjective pour tout y de M il existe $x \in L$ tel que $@(x)=y$ donc $\{x \in L : @(x) \sqsubseteq y\}$ n'est pas vide et comme L est un treillis complet $\bigsqcup\{x \in L : @(x) \sqsubseteq y\}$ existe ce qui montre que f est une application totale de M dans L.

Soient t, u appartenant à M tels que $t \sqsubseteq u$. Alors $\forall x \in L$, $\{@(x) \sqsubseteq t\} \Rightarrow \{@(x) \sqsubseteq u\}$ donc $\bigsqcup\{x \in L : @(x) \sqsubseteq t\} \sqsubseteq \bigsqcup\{x \in L : @(x) \sqsubseteq u\}$ soit $f(t) \sqsubseteq f(u)$, ce qui montre que f est monotone.

$\forall x \in L$, $\forall y \in M$, $\{x \sqsubseteq f(y)\} \Rightarrow \{@(x) \sqsubseteq @(f(y))\}$ car @ est monotone. $@(f(y)) = @(\bigsqcup\{z \in L : @(z) \sqsubseteq y\}) = \bigsqcup\{@(z) : @(z) \sqsubseteq y\}$ car @ est un mor-

phisme complet pour l'union. Donc $@(f(y)) \sqsubseteq y$. Par transitivité $\{x \sqsubseteq f(y)\} \Rightarrow \{@(x) \sqsubseteq y\}$. Supposons maintenant que $@(x) \sqsubseteq y$ alors $f(@(x)) \sqsubseteq f(y)$ car f est monotone. Comme $f(@(x)) = \bigsqcup\{z \in L : @(z) \sqsubseteq @(x)\} \sqsupseteq \bigsqcup\{z \in L : z \sqsubseteq x\} = x$ on a par transitivité que $\{@(x) \sqsubseteq y\} \Rightarrow \{x \sqsubseteq f(y)\}$. Donc $(@,f)$ est une paire de fonctions adjointes supérieure et comme l'une détermine l'autre nous concluons que $f=\gamma$.

- Comme @ est monotone et surjective, on a $@(T)=T$. Donc $T \sqsubseteq \gamma\circ@(T) = \gamma(T) \sqsubseteq T$ soit $\gamma(T)=T$. Soit donc γ un morphisme injectif complet pour l'intersection de M dans L tel que $\gamma(T)=T$. Posons $f = \lambda x.\sqcap\{y \in M : x \sqsubseteq \gamma(y)\}$. Pour tout x de L on a $x \sqsubseteq T = \gamma(T)$ donc $\{y \in M : x \sqsubseteq \gamma(y)\}$ n'est pas vide et comme M est un treillis complet f est une fonction totale de L dans M.

  Soit $t \sqsubseteq u$ alors $\forall y \in M$, $\{u \sqsubseteq \gamma(y)\} \Rightarrow \{t \sqsubseteq \gamma(y)\}$ donc $\sqcap\{y \in M : t \sqsubseteq \gamma(y)\} \sqsubseteq \sqcap\{y \in M : u \sqsubseteq \gamma(y)\}$ et f est monotone.

  $\forall x \in L$, $\forall y \in M$, $\{x \sqsubseteq \gamma(y)\} \Rightarrow \{f(x) \sqsubseteq f(\gamma(y))\}$ par monotonie. $f(\gamma(y)) = \sqcap\{z \in M : \gamma(y) \sqsubseteq \gamma(z)\} \sqsubseteq \sqcap\{z \in M : y \sqsubseteq z\} = y$. Par transitivité $\{x \sqsubseteq \gamma(y)\} \Rightarrow \{f(x) \sqsubseteq y\}$. Supposons maintenant que $f(x) \sqsubseteq y$ alors $\gamma(f(x)) \sqsubseteq \gamma(y)$. On a $\gamma(f(x)) = \gamma(\sqcap\{z \in M : x \sqsubseteq \gamma(z)\}) = \sqcap\{\gamma(z) : x \sqsubseteq \gamma(z)\} \sqsupseteq x$ car γ est un morphisme complet pour l'intersection. Donc $\{f(x) \sqsubseteq y\} \Rightarrow \{x \sqsubseteq \gamma(y)\}$ et (f,γ) est une paire de fonctions adjointes supérieure et comme l'une détermine l'autre nous concluons que $f=@$.

- Montrons que $\gamma\circ@(L)$ et M sont isomorphes par l'isomorphisme complet $(@|\gamma\circ@(L))$. Montrons que la restriction $(@|\gamma\circ@(L))$ de @ à $\gamma\circ@(L)$ est bijective. $\forall y \in M$, $y \in @(L)$ car @ est surjective donc $\gamma(y) \in (\gamma\circ@(L))$ et $@\circ\gamma(y)=y$ donc $\forall y \in M$, $\exists x \in (\gamma\circ@(L))$ tel que $@(x)=y$ et $(@|\gamma\circ@(L))$ est surjective. Maintenant $\gamma\circ@(L) = \gamma(M)$ et γ de M dans $\gamma(M)$ est surjective donc $\forall x \in (\gamma\circ@(L))$, $\exists y \in M$ tel que $x=\gamma(y)$. Alors $(@|\gamma\circ@(L))(x) = (@|\gamma\circ@(L))\circ\gamma(y) = y$ donc $\gamma\circ(@|\gamma\circ@(L))(x) = \gamma(y) = x$. Maintenant soient $x_1, x_2 \in \gamma\circ@(L)$ tels que $(@|\gamma\circ@(L))(x_1) = (@|\gamma\circ@(L))(x_2)$. Alors $x_1 = \gamma((@|\gamma\circ@(L))(x_1)) = \gamma((@|\gamma\circ@(L))(x_2)) = x_2$ ce qui montre que $(@|\gamma\circ@(L))$ est injective. De plus, son inverse est γ.

  Il nous reste à montrer que $(@|\gamma\circ@(L))$ est un isomorphisme complet. Comme $(\gamma\circ@) \in \mathit{fers}(L \to L)$, on sait que $\gamma\circ@(L)$ est un treillis complet $(\sqsubseteq, \gamma\circ@(\bot), \top, \lambda S.\gamma\circ@(\bigsqcup S), \sqcap)$. Soit $S \subseteq \gamma\circ@(L)$. La borne supérieure de

S dans $\gamma\circ@(L)$ est $\gamma\circ@(\bigsqcup S)$ et $@(\gamma\circ@(\bigsqcup S)) = @(\bigsqcup S) = \bigsqcup @(S)$ donc $(@|\gamma\circ@(L))(\gamma\circ@(\bigsqcup S)) = \bigsqcup(@|\gamma\circ@(L))(S)$. D'autre part $\gamma(\bigsqcap@(S)) = \bigsqcap\gamma(@(S)) = \gamma\circ@(\bigsqcap\gamma\circ@(S)) = \gamma\circ@(\bigsqcap S)$ car $\gamma\circ@(S)$ est un ensemble de points fixes de la fermeture supérieure $\gamma\circ@$ et donc $\bigsqcap\gamma\circ@(S) = (\bigsqcap S) \in (\gamma\circ@(L))$. Comme γ est injective, nous avons $\bigsqcap@(S) = @(\bigsqcap S)$ et $\bigsqcap(@|\gamma\circ@(L))(S) = (@|\gamma\circ@(L))(\bigsqcap S)$.

Fin de la preuve.

COROLLAIRE 4.2.7.0.4

Soient $L(\sqsubseteq,\bot,\top,\bigsqcup,\bigsqcap)$ et $M(\sqsubseteq,\bot,\top,\bigsqcup,\bigsqcap)$ des treillis complets et $@$ un morphisme complet surjectif pour l'union de L dans M. Alors $(@\ ,\ \lambda y.\bigsqcup\{x \in L\ :\ @(x) \sqsubseteq y\})$ est une paire de fonctions adjointes supérieure.

COROLLAIRE 4.2.7.0.5

Soient $L(\sqsubseteq,\bot,\top,\bigsqcup,\bigsqcap)$ et $M(\sqsubseteq,\bot,\top,\bigsqcup,\bigsqcap)$ des treillis complets et γ un morphisme injectif complet pour l'intersection de M dans L tel que $\gamma(\top)=\top$. Alors $(\lambda x.\bigsqcap\{y \in M\ :\ x \sqsubseteq \gamma(y)\}\ ,\ \gamma)$ est une paire de fonctions adjointes supérieure.

DEFINITION 4.2.7.0.6 *Image approchée supérieurement d'un treillis complet*

Soient $L(\sqsubseteq,\bot,\top,\bigsqcup,\bigsqcap)$ et $M(\sqsubseteq,\bot,\top,\bigsqcup,\bigsqcap)$ deux treillis complets. Nous dirons que M est l'image approchée supérieurement de L (noté $L \overline{\rhd} M$) si et seulement si il existe $\rho \in \mathit{fers}(L \to L)$ tel que $\rho(L)(\sqsubseteq,\rho(\bot),\top,\lambda S.\rho(\bigsqcup S),\bigsqcap)$ et $M(\sqsubseteq,\bot,\top,\bigsqcup,\bigsqcap)$ sont complètement isomorphes.

COROLLAIRE 4.2.7.0.7

Soient $L(\sqsubseteq,\bot,\top,\bigsqcup,\bigsqcap)$ et $M(\sqsubseteq,\bot,\top,\bigsqcup,\bigsqcap)$ deux treillis complets.

$\{L \overline{\rhd} M\} \iff \{\exists @ \in (L \to M)$ surjective, $\exists \gamma \in (M \to L)$ injective : $(@,\gamma)$ est une paire de fonctions adjointes supérieure$\}$

$\{L \overline{\rhd} M\} \iff \{\exists \gamma \in (M \to L)$: injective, morphisme complet pour l'intersection et $\gamma(\top)=\top\}$

$\{L \overline{\rhd} M\} \iff \{\exists @ \in (L \to M)$: surjective, morphisme complet pour l'union$\}$.

4.2.8 Fermeture induite sur l'espace des opérateurs monotones sur un treillis complet L par une fermeture sur L

THEOREME 4.2.8.0.1

Soient $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet et $\rho \in fers(L \to L)$, alors $\lambda f.\rho\circ f\circ\rho \in fers(mon(L \to L) \to mon(L \to L))$.

Preuve : Soient $f, g \in mon(L \to L)$ tels que $f \sqsubseteq g$, alors $\rho\circ f\circ\rho \sqsubseteq \rho\circ g\circ\rho$ car ρ est monotone et $\lambda f.\rho\circ f\circ\rho$ est monotone. Aussi $f \sqsubseteq \rho\circ f\circ\rho$ car $\forall x \in L$, $f(x) \sqsubseteq \rho(f(\rho(x)))$ car ρ est extensive et f et ρ sont monotones. Enfin $\rho\circ\rho\circ f\circ\rho\circ\rho = \rho\circ f\circ\rho$ ce qui montre que $\lambda f.\rho\circ f\circ\rho$ est idempotente.
Fin de la preuve.

DEFINITION 4.2.8.0.2

Soient $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet et ρ une fermeture supérieure de L. Nous appellerons $\bar{\rho}=\lambda f.\rho\circ f\circ\rho$ la *fermeture supérieure induite sur l'espace des opérateurs monotones de* L *par* ρ.

On remarquera que $\bar{\rho}(f)$ est la plus petite fonction monotone de $(\rho(L) \to \rho(L))$ plus grande que f restreinte à $\rho(L)$. (En effet soit $g \in mon(\rho(L) \to \rho(L))$ tel que $(f|\rho(L)) \sqsubseteq g$ alors $\rho\circ f\circ\rho \sqsubseteq \rho\circ g\circ\rho = g$).

THEOREME 4.2.8.0.3

$\{L \trianglerighteq(@,\gamma)\ M\} \Rightarrow \{mon(L \to L) \trianglerighteq(\lambda f.@\circ f\circ\gamma, \lambda f.\gamma\circ f\circ@)\ mon(M \to M)\}$.

COROLLAIRE 4.2.8.0.4

Soient L, M des treillis complets tels que $L \trianglerighteq(@,\gamma)\ M$ et $mon(L \to L) \trianglerighteq(\bar{@},\bar{\gamma})\ mon(M \to M)$ avec $\bar{@}=\lambda f.@\circ f\circ\gamma$ et $\bar{\gamma}=\lambda f.\gamma\circ f\circ@$. Soient F, $G \in mon(L \to L)$ alors :

(a) - $\{lfp(F) \sqsubseteq \gamma(lfp(\bar{@}(F)))\}$

(b) - $\{\bar{F} \in mon(M \to M)$ et $\bar{@}(F) \sqsubseteq \bar{F}\} \Rightarrow \{lfp(F) \sqsubseteq \gamma(lfp(\bar{F}))\}$

(c) - $\{lfp(F\circ G) \sqsubseteq \gamma(lfp(\bar{@}(F)\circ\bar{@}(G)))\}$

(d) - $\{\bar{F}, \bar{G} \in mon(M \to M)$ et $\bar{@}(F) \sqsubseteq \bar{F}$ et $\bar{@}(G) \sqsubseteq \bar{G}\} \Rightarrow \{lfp(F\circ G) \sqsubseteq \gamma(lfp(\bar{F}\circ\bar{G}))\}$

(et même chose avec *gfp*).

PROPOSITION 4.2.8.0.5

Soient $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet, ρ_1, $\rho_2 \in fers(L\to L)$ et $f \in mon(L\to L)$ alors :

$$lfp((\rho_1\sqcap\rho_2)\circ f\circ(\rho_1\sqcap\rho_2)) \sqsubseteq lfp(\rho_1\circ f\circ\rho_1)\sqcap lfp(\rho_2\circ f\circ\rho_2).$$

Preuve : $\rho_1\sqcap\rho_2 \sqsubseteq \rho_1$ et f monotone impliquent $(\rho_1\sqcap\rho_2)\circ f\circ(\rho_1\sqcap\rho_2) \sqsubseteq (\rho_1\circ f\circ\rho_1)$. Comme lfp est monotone il vient $lfp((\rho_1\sqcap\rho_2)\circ f\circ(\rho_1\sqcap\rho_2)) \sqsubseteq lfp(\rho_1\circ f\circ\rho_2)$. Même chose pour ρ_2.
Fin de la preuve.

4.3 APPROXIMATION DE POINTS FIXES D'UN OPERATEUR PAR APPROXIMATION DE L'OPERATEUR

Le calcul complexe des points fixes extrêmes d'un opérateur F peut être remplacé par le calcul plus simple des points fixes d'un opérateur $\bar{F}$ approchant F comme suit :

THEOREME 4.3.0.1

Soient $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet et $F, G \in mon(L\to L)$ alors $\{\{F\sqsubseteq G\} \Rightarrow \{\{lfp(F)\sqsubseteq lfp(G)\}$ et $\{gfp(F)\sqsubseteq gfp(G)\}\}\}$.

Preuve : Rappelons que $lfp(F) = \sqcap\{X\in L : F(X)\sqsubseteq X\}$ et $lfp(G) = \sqcap\{X\in L : G(X)\sqsubseteq X\}$. Comme $F\sqsubseteq G$ il vient $\{\forall X\in L, \{G(X)\sqsubseteq X\} \Rightarrow \{F(X)\sqsubseteq X\}\}$ ce qui implique $lfp(F)\sqsubseteq lfp(G)$ et par dualité $\{\{F\sqsupseteq G\} \Rightarrow \{gfp(F)\sqsupseteq gfp(G)\}$.
Fin de la preuve.

4.3.1 Approximation d'un opérateur sur un treillis induite par une image approchée du treillis

Pour approcher les points fixes d'un opérateur monotone F sur un treillis complet $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ nous chercherons à calculer les points fixes d'un opérateur plus simple $\bar{F}$ choisi de sorte que l'algorithme pour calculer

$\bar{F}$ soit moins complexe que l'algorithme pour calculer F. Plaçons-nous dans le cas d'une approximation supérieure, (celui d'une approximation inférieure étant dual), c'est-à-dire qu'il faut choisir $\bar{F}$ tel que $F \sqsubseteq \bar{F}$ (théorème 4.3.0.1). Le problème est de trouver un moyen de déterminer $\bar{F}$ à partir de F.

Nous proposons de définir un sous-ensemble M de L dans lequel seront choisies les approximations des points fixes de F. Pour cela, ayant fixé M nous choisirons $\bar{F}$ égale à la meilleure approximation supérieure de F qui appartient à $(L \to M)$.

Nous nous proposons également de choisir M indépendamment de F et par conséquent nous choisirons un sous-espace M de L de sorte que tout élément x de L ait une approximation supérieure dans M. Soit $\rho \in (L \to M)$ un opérateur qui à tout x de L associe une approximation supérieure $\rho(x)$ de x dans M. Comme ρ est extensif il est uniquement déterminé par le choix de M si et seulement si c'est une fermeture supérieure de L (4.2.2.0.1) et M une partie de Moore inférieure de L (4.2.2.0.4). En effet, tous les y de M plus grands que x sont une approximation supérieure de x. En particulier le supremum $\top$ de L ne peut être approché supérieurement que par $\top$ et donc $\top \in M$. Parmi les approximations supérieures de x dans M certaines sont meilleures que d'autres. Le critère de comparaison étant l'ordre $\sqsubseteq$ sur L, $a \in M$ est une meilleure approximation de $x \in L$ que $b \in M$ si $x \sqsubseteq a \sqsubseteq b$. Les meilleures approximations de x dans M sont les éléments minimaux de $\{y \in M : x \sqsubseteq y\}$. Il existe une unique meilleure approximation d'un élément x quelconque de L dans M si $\{y \in M : x \sqsubseteq y\}$ admet un plus petit élément c'est-à-dire si M est une partie de Moore inférieure (4.2.2.0.2).

Supposons qu'on ait choisi un sous-espace M de L qui ne soit pas une partie de Moore inférieure. Dans ce cas certains éléments de L ont plusieurs approximations supérieures minimales dans M et il n'est pas possible d'en choisir une meilleure indépendamment d'un F donné. Dans ce cas, le théorème 4.2.2.0.5 indique quels sont les éléments de L à rajouter en nombre minimum à M pour éviter l'équivoque.

Une partie de Moore inférieure M induit une fermeture supérieure unique ρ telle que $\rho(L)=M$ (4.2.2.0.5) et de même une fermeture supérieure ρ détermine de manière unique une partie de Moore inférieure (2.3.0.1) si bien que le choix de l'ensemble M peut être remplacé par celui d'une fermeture supérieure ρ en définissant $M=\rho(L)$. Nous avons donc rappelé plusieurs caractérisations des fermetures supérieures (4.2.1).

Si on a choisi un opérateur ρ qui n'est pas une fermeture supérieure

le théorème 4.2.3.0.5 indique que *fers*(ρ) est la plus petite fermeture supérieure de L telle que tout x de L ait une meilleure approximation supérieure *fers*(ρ)(x) plus grande que ρ(x). Nous avons donné plusieurs définitions équivalentes de *fers* (4.2.3.0.5, 4.2.3.0.6, 4.2.3.0.8).

Nous avons étudié diverses manières de construire des fermetures supérieures (4.2.1, 4.2.5, 4.2.6, 4.2.7) et de les combiner (4.2.3, 4.2.4). En particulier, la définition d'une fermeture supérieure par une famille d'idéaux principaux (4.2.5.0.3) ou mieux l'utilisation d'une relation de congruence complète pour l'union (4.2.6.0.4) met bien en évidence l'idée de ne pas distinguer entre les éléments d'une même classe d'équivalence et qui sont donc tous approchés par le supremum de la classe.

Nous utiliserons fréquemment des combinaisons de fermetures pour renforcer une approximation (4.2.3.0.5, 4.2.4.0.7) ou au contraire pour la raffiner (intersection de fermetures 4.2.4.0.5). (On notera que les propositions 4.2.4.0.2 et 4.2.4.0.3 indiquent qu'il est toujours préférable d'utiliser l'union $fers(\rho_1 \sqcup \rho_2)$ plutôt que la composition $\rho_1 \circ \rho_2$ de fermetures ρ_1 et ρ_2 (puisque $fers(\rho_1 \sqcup \rho_2)$ est toujours une fermeture tandis que ce n'est pas nécessairement le cas de $\rho_1 \circ \rho_2$ et que si $\rho_1 \circ \rho_2$ est une fermeture alors $\rho_1 \circ \rho_2 = fers(\rho_1 \sqcup \rho_2)$)).

Le choix de l'espace M des valeurs approchées ou de la fermeture supérieure induite dépend du problème à résoudre, de la précision désirée dans les approximations et du coût qu'on est prêt à payer pour le résoudre et ceci sera largement illustré au chapitre 5. Les théorèmes 4.2.4.0.5 et 4.2.3.0.10 permettent d'ordonner partiellement les approximations selon leur précision grâce à l'ordre partiel sur les fermetures ou parties de Moore correspondantes.

Ayant choisi une fermeture supérieure ρ, la meilleure approximation supérieure de $F \in mon(L \to L)$ dans $mon(L \to \rho(L))$ est $\rho \circ F$. Toutefois pour restreindre le domaine des calculs à ρ(L), nous choisirons $\bar{F} = \rho \circ F \circ \rho$ qui est la plus petite fonction monotone de $(\rho(L) \to \rho(L))$ plus grande que f et restreinte à ρ(L) (4.2.8). Alors $lfp(F) \sqsubseteq lfp(\bar{F}) = luis(\rho \circ F \circ \rho)(\perp) = luis(\rho \circ F)(\rho(\perp)) = lfp((\rho \circ F \mid \rho(L)))$. De même $gfp(F)$ est approché supérieurement par $gpf((\rho \circ F \mid \rho(L)))$.

Quand les éléments de ρ(L) ne sont pas représentables en machine, nous utiliserons un espace de valeurs approchées M tel que $L \overrightarrow{\triangleright}(@,\gamma)$ M avec $\rho = \gamma \circ @$ en choisissant M de sorte que ses éléments soient aisément représentables en machine. Alors la meilleure approximation de $F \in mon(L \to L)$ dans

$mon(M \to M)$ est $@ \circ F \circ \gamma$ (4.2.8.0.3). Quand il est difficile d'écrire l'algorithme réalisant $@ \circ F \circ \gamma$ nous choisirons de réaliser $\bar{F} \in mon(M \to M)$ tel que $@ \circ F \circ \gamma \sqsubseteq \bar{F}$. Alors $lfp(F) \sqsubseteq \gamma(lfp(\bar{F}))$ et $gfp(F) \sqsubseteq \gamma(gfp(\bar{F}))$ (4.2.8.0.4.(b)). Généralement F est composition de fonctions élémentaires ce qui permet de construire $\bar{F}$ à partir de F d'une manière systématique en approchant supérieurement toutes les fonctions élémentaires $f \in mon(L \to L)$ par une fonction approchée élémentaire $\bar{f} \in mon(M \to M)$ telle que $\bar{f} \sqsubseteq \gamma \circ f \circ @$ (4.2.8.0.4.(d)).

Noter qu'au lieu d'approcher un opérateur monotone F par un opérateur $\bar{F}$ monotone, nous pourrions également utiliser un opérateur extensif (ou dualement réductif) :

PROPOSITION 4.3.1.0.1

Soient $L(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$ un treillis complet et $f \in ext(L \to L)$, alors $fp(f) = luis(f)(\bot)$.

PROPOSITION 4.3.1.0.2

Soient $L(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$ un treillis complet, $f \in mon(L \to L)$, $\bar{f} \in (L \to L)$ tels que $f \sqsubseteq \bar{f}$ alors $lfp(f) \sqsubseteq luis(ext(\bar{f}))(\bot)$.

4.3.2 Amélioration d'une approximation d'un point fixe d'un opérateur monotone

Les résultats du paragraphe 4.1.1 permettent d'améliorer une approximation d'un point fixe extrême d'un opérateur monotone f sur un treillis complet L. Nous spécialisons maintenant ces résultats au cas où nous disposons d'une approximation g de f, ($g \sqsubseteq f$ ou $f \sqsubseteq g$).

THEOREME 4.3.2.0.1

Soient $L(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$ un treillis complet et $f, g \in mon(L \to L)$

$\{\{\exists y \in L : y \sqsubseteq lfp(f)\}$ et $\{g \sqsubseteq f\}\}$

$$\Rightarrow \{y \sqsubseteq lfp(\lambda x.y \sqcup g(x)) \sqsubseteq luis(\lambda x.x \sqcup g(x))(y) \sqsubseteq lfp(f)\}.$$

Preuve : Soit $y \sqsubseteq lfp(f)$ et $g \sqsubseteq f$. Considérons l'itération croissante $\langle x^{\delta} : \delta \in \mu(L) \rangle$ partant de y et définie par $\lambda x.x \sqcup g(x)$, (2.5.1.0.1). C'est une

itération croissante partant de y et approchée inférieurement pour f, (4.1.1.0.5). En effet, $x^0 = y$, quand δ est un ordinal successeur alors $x^{\delta-1} \sqsubseteq x^{\delta} = x^{\delta-1} \sqcup g(x^{\delta-1}) \sqsubseteq x^{\delta-1} \sqcup f(x^{\delta-1})$ car $g \sqsubseteq f$ et quand δ est un ordinal limite $x^{\delta} = \bigsqcup_{\alpha<\delta} x^{\alpha}$. Par conséquent le théorème 4.1.1.0.6 implique que la limite $\mathit{luis}(\lambda x.x \sqcup g(x))(y)$ de $\langle x^{\delta} : \delta \in \mu(L) \rangle$ approche $\mathit{luis}(\lambda x.x \sqcup f(x))(y)$ inférieurement. Mais d'après les théorèmes 2.5.2.0.5 et 2.5.3.0.1 $\mathit{luis}(\lambda x.x \sqcup g(x))(y) = \mathit{luis}(\lambda x.y \sqcup g(x))(y) \sqsupseteq \mathit{luis}(\lambda x.y \sqcup g(x))(\bot) = \mathit{lfp}(\lambda x.y \sqcup g(x)) \sqsupseteq y$. Egalement $\bot \sqsubseteq y \sqsubseteq \mathit{lfp}(f)$ implique $\mathit{lfp}(f) = \mathit{luis}(\lambda x.x \sqcup f(x))(\bot) \sqsubseteq \mathit{luis}(\lambda x.x \sqcup f(x))(y) \sqsubseteq \mathit{luis}(\lambda x.x \sqcup f(x))(\mathit{lfp}(f)) = \mathit{lfp}(f)$ ce qui permet de conclure que $y \sqsubseteq \mathit{lfp}(\lambda x.y \sqcup g(x)) \sqsubseteq \mathit{luis}(\lambda x.x \sqcup g(x))(y) \sqsubseteq \mathit{lfp}(f)$.
Fin de la preuve.

Remarque 4.3.2.0.2

Soit y une approximation inférieure de $\mathit{lfp}(f)$. Connaissant $g \sqsubseteq f$ nous pouvons améliorer y car $y \sqsubseteq \mathit{luis}(\lambda x.x \sqcup g(x))(y) \sqsubseteq \mathit{lfp}(f)$. Il n'est pas possible d'améliorer $\mathit{luis}(\lambda x.x \sqcup g(x))(y)$ en répétant ce processus car $\mathit{luis}(\lambda x.x \sqcup g(x))$ est idempotent. De plus le théorème 2.5.4.0.2 montre que $\mathit{luis}(\lambda x.x \sqcup g(x))(y)$ est la plus grande valeur de L qu'il est possible d'obtenir à partir de y en utilisant $\sqcup$, $\sqcap$ et g. $\mathit{luis}(\lambda x.x \sqcup g(x))(y)$ est donc la meilleure approximation inférieure de $\mathit{lfp}(f)$ qui peut être obtenue en n'utilisant que ces éléments.
Fin de la remarque.

Appliquant le principe de dualité, nous obtenons :

THEOREME 4.3.2.0.3

Soient $L(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$ un treillis complet et $f, g \in \mathit{mon}(L \to L)$.
$\{\{\exists y \in L : \mathit{gfp}(f) \sqsubseteq y\}$ et $\{f \sqsubseteq g\}\}$
$\Rightarrow \{\mathit{gfp}(f) \sqsubseteq \mathit{llis}(\lambda x.x \sqcap g(x))(y) \sqsubseteq \mathit{gfp}(\lambda x.y \sqcap g(x)) \sqsubseteq y\}$.

THEOREME 4.3.2.0.4

Soient $L(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$ un treillis complet et $f, g \in \mathit{mon}(L \to L)$.
$\{\{\exists y \in L : \mathit{lfp}(f) \sqsubseteq y\}$ et $\{f \sqsubseteq g\}\}$
$\Rightarrow \{\mathit{lfp}(f) \sqsubseteq \mathit{lfp}(\lambda x.y \sqcap g(x)) \sqsubseteq \mathit{llis}(\lambda x.x \sqcap g(x))(y) \sqsubseteq y\}$.

Preuve : Soient $lfp(f) \sqsubseteq y$ et $f \sqsubseteq g$. Considérons l'itération croissante $<x^\delta : \delta \in \mu(L)>$ partant de $\bot$ et définie par $\lambda x.y \sqcap g(x)$ (2.5.1.0.1). C'est une chaîne ascendante de limite $P = lfp(\lambda x.y \sqcap g(x))$ (2.5.3.0.1 et 2.5.3.0.2). Comme $P = y \sqcap g(P) \sqsubseteq y$ nous avons $x^\delta \sqsubseteq P \sqsubseteq y$ pour tout $\delta \in \mu(L)$. Pour tout ordinal successeur $\delta \in \mu(L)$ nous avons $x^{\delta-1} \sqsubseteq x^\delta = (y \sqcap g(x^{\delta-1})) \sqsubseteq g(x^{\delta-1})$. Montrons que $<x^\delta : \delta \in \mu(L)>$ est une itération croissante partant de $\bot$ et approchée supérieurement pour $\lambda x.y \sqcap f(x)$, (4.1.1.0.1). En effet, $x^0 = \bot$. Pour tout ordinal successeur nous avons $x^{\delta-1} \sqcup (y \sqcap f(x^{\delta-1})) \sqsubseteq x^\delta = y \sqcap g(x^{\delta-1})$ car $x^{\delta-1} \sqsubseteq y$, $y \sqcap f(x^{\delta-1}) \sqsubseteq y$, $x^{\delta-1} \sqsubseteq g(x^{\delta-1})$ et $y \sqcap f(x^{\delta-1}) \sqsubseteq g(x^{\delta-1})$ puisque $y \sqcap f(x^{\delta-1}) \sqsubseteq f(x^{\delta-1}) \sqsubseteq g(x^{\delta-1})$ qui vient de $f \sqsubseteq g$. Le théorème 4.1.1.0.2 implique alors que $lfp(\lambda x.y \sqcap g(x))$ approche $luis(\lambda x.x \sqcup (y \sqcap f(x)))(\bot)$ supérieurement. Soit $<t^\delta : \delta \in \mu(L)>$ l'itération croissante partant de $\bot$ et définie par f et soit $<z^\delta : \delta \in \mu(L)>$ l'itération croissante partant de $\bot$ et définie par $\lambda x.x \sqcup (y \sqcap f(x))$. Nous avons $t^0 = z^0 = \bot$. Supposons que $t^\alpha = z^\alpha$ pour tout $\alpha < \delta$. Si δ est un ordinal limite alors $t^\delta = \bigsqcup_{\alpha<\delta} t^\alpha = \bigsqcup_{\alpha<\delta} z^\alpha = z^\delta$. Si δ est un ordinal successeur alors en particulier $t^{\delta-1} = z^{\delta-1}$. Comme $t^{\delta-1} \sqsubseteq t^\delta = f(t^{\delta-1}) \sqsubseteq lfp(f) \sqsubseteq y$ nous avons $y \sqcap f(z^{\delta-1}) = y \sqcap f(t^{\delta-1}) = f(t^{\delta-1}) = f(z^{\delta-1})$. Comme $z^{\delta-1} \sqsubseteq f(z^{\delta-1})$ il vient $t^\delta = f(t^{\delta-1}) = f(z^{\delta-1}) = z^{\delta-1} \sqcup (y \sqcap f(z^{\delta-1}))$. Par induction transfinie nous concluons $\{\forall \delta \in \mu(L),\ t^\delta = z^\delta\}$ et en particulier $luis(\lambda x.x \sqcup (y \sqcap f(x)))(\bot) = luis(f)(\bot) = lfp(f)$. En plus il est facile de montrer que chaque terme de $<x^\delta : \delta \in \mu(L)>$ est inférieur au terme correspondant de l'itération décroissante partant de y et définie par $\lambda x.x \sqcap g(x)$ (2.5.1.0.2) ce qui permet de conclure que $lfp(f) \sqsubseteq lfp(\lambda x.y \sqcap g(x)) \sqsubseteq llis(\lambda x.x \sqcap g(x))(y) \sqsubseteq y$.

Fin de la preuve.

THEOREME 4.3.2.0.5

Soient $L(\sqsubseteq, \bot, \top, \sqcup, \sqcap)$ un treillis complet et $g \in mon(L \to L)$ alors $\lambda y.lfp(\lambda x.y \sqcap g(x))$ est une fermeture inférieure de L plus petite que g.

Preuve : Posons $h = \lambda y.lfp(\lambda x.y \sqcap g(x))$. Le théorème 4.3.2.0.5 montre que h est réductive. De plus $y \sqsubseteq z$ implique $\lambda x.y \sqcap g(x) \sqsubseteq \lambda x.z \sqcap g(x)$ et $lfp(\lambda x.y \sqcap g(x)) \sqsubseteq lfp(\lambda x.z \sqcap g(x))$ (4.3.0.1) ce qui montre que h est monotone. Soient $<x^\delta : \delta \in \mu(L)>$ et $<y^\delta : \delta \in \mu(L)>$ les itérations croissantes partant de $\bot$ définies par $\lambda x.y \sqcap g(x)$ et $\lambda x.h(y) \sqcap g(x)$ respectivement (2.5.1.0.1). Comme $<x^\delta : \delta \in \mu(L)>$ est une chaîne stationnaire de limite $h(y)$ nous avons pour tout $\delta \in \mu(L)$, $x^\delta \sqsubseteq h(y)$. Pour $\delta = 0$ on a $x^0 = y^0 = \bot$. Supposons

que pour tout $\alpha<\delta$ nous ayons $x^\alpha=y^\alpha$. Si δ est un ordinal limite il vient $x^\delta = \bigsqcup_{\alpha<\delta} x^\alpha = \bigsqcup_{\alpha<\delta} y^\alpha = y^\delta$. Si δ est un ordinal successeur alors $x^{\delta-1}=y^{\delta-1}$ et $y^\delta = (h(y)\sqcap g(y^{\delta-1})) = h(y)\sqcap g(x^{\delta-1}) = y\sqcap g(h(y))\sqcap g(x^{\delta-1})$. Comme $x^{\delta-1} \sqsubseteq h(y)$ et g est monotone nous avons $g(x^{\delta-1}) \sqsubseteq g(h(y))$ donc $g(h(y))\sqcap g(x^{\delta-1}) = g(x^{\delta-1})$ ce qui donne $y^\delta = y\sqcap g(x^{\delta-1}) = x^\delta$. Par induction transfinie nous avons en particulier $h(y) = \mathit{luis}(\lambda x.y\sqcap g(x))(\bot) = \mathit{luis}(\lambda x.h(y)\sqcap g(x))(\bot) = h(h(y))$. Finalement $\forall y\in L,\ h(y) = y\sqcap g(H(y)) \sqsubseteq g(h(y)) \sqsubseteq g(y)$.
Fin de la preuve.

Remarque 4.3.2.0.6

Connaissant $g\sqsubseteq f$ une approximation supérieure y de $\mathit{lfp}(f)$ peut, comme précédemment, être améliorée en $\mathit{lfp}(\lambda x.y\sqcap g(x))$. Le procédé ne peut pas être répété pour une nouvelle amélioration de l'approximation puisque $\lambda y.\mathit{lfp}(\lambda x.y\sqcap g(x))$ est idempotente. Notons que $\mathit{lfp}(\lambda x.y\sqcap g(x))$ est une meilleure approximation de $\mathit{lfp}(f)$ que $\mathit{llis}(\lambda x.x\sqcap g(x))(y)$ qui d'après le dual du théorème 2.5.4.0.2 est la plus petite valeur de L qui peut être obtenue en utilisant $\sqcup$, $\sqcap$, g et y. Comme nous l'avions observé à la remarque 4.1.1.0.9 nous avons obtenu une meilleure amélioration de l'approximation supérieure y de $\mathit{lfp}(f)$ en utilisant 4.1.1.0.1 avec $\lambda x.y\sqcap f(x)$ plutôt qu'en utilisant 4.1.1.0.7 en partant de y. Ceci nous amène à revenir sur la remarque 4.1.1.0.9 pour noter que si y=IDAS(f,ICAS(f,$\bot$)) nous avons ICAS($\lambda x.y\sqcap f(x),\bot$)=y quand la même technique d'extrapolation est utilisée pour calculer ICAS(f,$\bot$) et ICAS($\lambda x.y\sqcap f(x),\bot$), (par exemple, en choisissant le même $\bar\nabla$ dans 4.1.2.0.5). A nouveau il n'est donc pas possible d'améliorer y.
Fin de la remarque.

THEOREME 4.3.2.0.7

Soient $L(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$ un treillis complet et $f,\ g\in \mathit{mon}(L\to L)$.
$\{\{\exists y\in L : y\sqsubseteq \mathit{gfp}(f)\}$ et $\{g\sqsubseteq f\}\}$
$\Rightarrow \{y \sqsubseteq \mathit{luis}(\lambda x.x\sqcup g(x))(y) \sqsubseteq \mathit{gfp}(\lambda x.y\sqcup g(x)) \sqsubseteq \mathit{gfp}(f)\}$.

4.4 NOTES BIBLIOGRAPHIQUES

Les techniques d'approximation de points fixes connues en analyse numérique (par exemple Amman[1976], Kuhn & MacKinnon[1975], Rockafellar[1976], Scarf[1967], Todd[1976], etc...) ne nous sont pas très utiles car nous travaillons sur des espaces non numériques pour lesquels nous ne disposons pas de notion de distance. En particulier nous ne savons pas donner de mesure de précision des approximations.

L'idée d'approcher inférieurement et supérieurement un point fixe est classique en analyse fonctionnelle (Krasnosel'skii[1964]) et se retrouve d'un point de vue différent dans Cousot & Cousot[1977a]. Toutefois les exemples étaient donnés pour des approximations supérieures ce qui ne mettait pas en valeur les approximations inférieures. C'est pourquoi nous avons systématiquement énoncé le dual de toutes les techniques d'approximation dans le paragraphe 4.1. En revanche dans le paragraphe 4.2 les résultats duaux sont laissés implicites.

Le paragraphe 4.1 sur les méthodes itératives d'approximation de points fixes basées sur une accélération de la convergence par extrapolation est une formalisation des algorithmes utilisés dans Cousot & Cousot[1975a, 1976] bien que l'idée que ces algorithmes reposent sur l'approximation de solutions de systèmes d'équations associés à un programme y soit encore implicite. Elle est explicite dans Cousot & Cousot[1977a]. Parallélement M. Sintzoff et A. van Lamsweerde ont développé des méthodes de calcul ou d'approximation de points fixes pour résoudre leurs problèmes de construction ou d'amélioration de programmes non-déterministes (Sintzoff[1977a, 1977b], van Lamsweerde[1977]) ou parallèles (van Lamsweerde & Sintzoff[1976]).

La formalisation de l'idée de simplification de systèmes d'équations (paragraphe 4.3) au moyen d'une fermeture (paragraphe 4.2) a son origine dans Cousot & Cousot[1975, 1976, 1977a] où nous utilisions une paire de fonctions adjointes. Le lien avec la notion de fermeture est fait dans Cousot & Cousot[1977d], Cousot[1977b, 1977c]. Outre les résultats connus sur les fermetures que nous avons rappelés dans le paragraphe 4.2 sans en donner les preuves et en mentionnant l'auteur, nous pouvons citer Achache[1969], Dubreil-Jacotin et al.[1963], Dwinger[1954, 1955], Iseki[1951], Ladegaillerie[1973], Monteiro[1945], Monteiro & Ribeiro[1942].

Morgado[1960, 1961, 1962a, 1962b, 1963, 1964, 1965a, 1965b, 1966], Ore[1943a, 1943b], Ward[1942]. La seule utilisation (autre que la nôtre) de la notion de fermeture que nous ayons trouvée en informatique est Scott[1976].

CHAPITRE 5

ANALYSE SEMANTIQUE APPROCHEE DES PROGRAMMES ET APPLICATIONS

5. ANALYSE SEMANTIQUE APPROCHEE DES PROGRAMMES ET APPLICATIONS

5. ANALYSE SEMANTIQUE APPROCHEE DES PROGRAMMES ET APPLICATIONS

Dans le chapitre 3 nous avons ramené le problème de l'analyse sémantique d'un programme au problème de résolution des systèmes d'équations sémantiques en avant et en arrière associés à ce programme. Certains problèmes d'analyse sémantique des programmes comme celui de la terminaison (Manna [1974, §4.2.]) ou celui qui peut paraître plus simple de déterminer les expressions constantes d'un programme (Reif & Lewis[1977]) étant indécidables, la solution de ces équations sémantiques n'est pas calculable automatiquement. Même une résolution manuelle est souvent extrêmement difficile (comme, par exemple, dans le problème ouvert de montrer que le programme suivant se termine par toute valeur initiale entière positive de n :

```
tantque  n≠1 faire
    n := si pair(n) alors n/2 sinon 3n+1 finsi;
refaire).
```

En pratique il faut donc se contenter de réponses partielles aux questions sur la sémantique d'un programme. Par exemple il est facile de montrer que le programme ci-dessus se termine pour toute valeur initiale de n positive et égale à une puissance de deux et ceci constitue une réponse partielle au problème de terminaison du programme. Dans un autre domaine un compilateur peut se contenter d'une réponse par "oui" ou "je ne sais pas" à la question "est-ce que telle expression est constante en tel point d'un programme?". En effet si le compilateur peut montrer que l'expression est constante il en calculera la valeur une fois pour toutes avant l'exécution tandis que s'il ne sait pas il produira du code machine pour que ce calcul soit fait à l'exécution du programme.

Pour procéder à l'analyse sémantique approchée d'un programme nous proposons de calculer une solution approchée des équations sémantiques en avant et en arrière associées à ce programme. Il s'agit donc d'appliquer les méthodes de résolution approchée développées au chapitre 4 aux équations sémantiques considérées au chapitre 3. Nous montrons comment, ayant choisi une classe particulière de propriétés des programmes apportant des réponses à un

problème donné, les résultats du chapitre 4 permettent de construire un algorithme réalisant automatiquement l'analyse d'un programme quelconque pour cette classe de propriétés. Nous illustrons la méthode en construisant une liste d'exemples d'algorithmes d'analyse sémantique approchée de programmes. Cette liste ne saurait être exhaustive car nous avons préféré développer une méthodologie de l'analyse sémantique approchée des programmes plutôt que, comme c'est souvent le cas pour les recherches en informatique, de traiter des problèmes particuliers pour lesquels on fournit une implémentation hâtive dont il est impossible de tirer le moindre enseignement. Enfin les exemples que nous avons sélectionnés sont principalement orientés vers des problèmes de compilation car les solutions jusqu'ici non satisfaisantes de ces problèmes sont d'un intérêt économique considérable.

5.1 CONCEPTION D'UNE TECHNIQUE D'ANALYSE APPROCHEE DES PROGRAMMES POUR UNE CLASSE DONNEE DE PROPRIETES SEMANTIQUES

Pour concevoir une méthode d'analyse sémantique des programmes il faut d'abord envisager les questions que l'on veut se poser sur les programmes. Ensuite nous proposons de déterminer à l'aide des résultats du chapitre 3 comment une approximation supérieure ou inférieure des solutions des systèmes d'équations sémantiques associés au programme permet de répondre à ces questions. Enfin le chapitre 4 offre un choix de méthodes de résolution approchée des équations sémantiques.

Exemple: Au problème "trouver le domaine de terminaison d'un programme π" nous pouvons donner une réponse en calculant $lfp(B_\pi(\lambda x.\underline{vrai}))_\varepsilon$, (définition 3.5.0.1 et théorème 3.5.0.5). Une réponse partielle peut consister à déterminer un sous-ensemble du domaine de terminaison. Ce type de réponse approchée est alors obtenu en caractérisant un sous-domaine de terminaison par un prédicat R tel que $R \Rightarrow lfp(B_\pi(\lambda x.\underline{vrai}))_\varepsilon$. Ainsi pour calculer une approximation inférieure du plus petit point fixe de $\lambda P.\{\lambda n.[[(n\neq 1)\ \underline{et}\ \underline{pair}(n)\ \underline{et}\ P(n/2)\ \underline{ou}\ ((n\neq 1)\ \underline{et}\ \underline{impair}(n)\ \underline{et}\ P(3n+1))\ \underline{ou}\ (n=1)]]\}$ nous pouvons simplifier en $\lambda P.\{\lambda n.[(\underline{pair}(n)\ \underline{et}\ P(n/2))\ \underline{ou}\ (n=1)]]\}$ ce qui donne $R=\lambda n.\{\exists k\geq 0 : n=2^k\}$. *Fin de l'exemple.*

Si dans le cas d'une analyse manuelle nous pouvons nous contenter d'énoncer des principes de résolution approchée des équations sémantiques (ce qui laisse la liberté de choisir la meilleure méthode en fonction de chaque programme particulier) il faut dans le cas d'une analyse automatique fournir un algorithme de résolution approchée des équations sémantiques qui permette d'analyser n'importe quel programme. Dans ce dernier cas nous proposons pour un problème donné de choisir un type particulier de réponses au problème posé. Alors le chapitre 3 offre une méthode de conception d'un technique d'analyse approchée des programmes pour cette classe donnée de propriétés sémantiques.

Par exemple soit π un programme comportant n variables $x_1,\ldots,x_n$ à valeurs dans U et α points de programme $a_1,\ldots,a_\alpha$ où a_ε et a_σ sont respectivement les points d'entrée et de sortie du programme. Pour répondre à la question "quel est le domaine des valeurs possibles des variables du programme au cours d'une exécution quelconque du programme" nous pouvons calculer $lfp(F_\pi(\phi))$ (définition 3.3.0.1 et théorème 3.3.0.2). En fait tout $P\in(P_n)^\alpha=(U^n\to\{\underline{vrai},\underline{faux}\})^\alpha$ tel que $lfp(F_\pi(\phi))\Rightarrow P$ apporte une réponse partielle au problème en caractérisant un sur-domaine des valeurs possibles en chaque point du programme. Il y a généralement une infinité de P qui sont une approximation supérieure de $lfp(F_\pi(\phi))$ mais certains sont plus instructifs que d'autres pour résoudre un problème donné. Dans une analyse manuelle on jugera intuitivement de la forme la plus intéressante de P au cours du calcul de P lui-même. Au contraire dans une analyse automatique nous ne pouvons pas compter sur l'intuition pour orienter le calcul de la solution approchée. C'est pourquoi nous proposons de déterminer la forme la plus intéressante des solutions approchées avant de commencer l'analyse, ce qui revient à déterminer a priori un sous-ensemble R de $(P_n)^\alpha$ de sorte qu'une approximation supérieure P de $lfp(F_\pi(\phi))$ dans R apporte une réponse partielle au problème donné qui soit intuitivement satisfaisante. Pour que tout prédicat P de $(P_n)^\alpha$ ait une meilleure approximation supérieure $\overline{\rho}(P)$ dans R il faut et il suffit que R soit une partie de Moore inférieure, c'est-à-dire que $\overline{\rho}$ doit être une fermeture supérieure de $(P_n)^\alpha$ telle que $\overline{\rho}(P_n)^\alpha=R$, (théorèmes 4.2.2.0.1 et 4.2.2.0.4). Si R n'est pas une partie de Moore inférieure nous savons construire la plus petite partie de Moore inférieure $\overline{R}$ contenant R, (théorème 4.2.2.0.5). La plus petite fonction monotone de $\overline{R}$ dans $\overline{R}$ plus grande que $F_\pi(\phi)$ restreinte à $\overline{R}$ est alors $\overline{\rho}\circ F_\pi(\phi)\circ\overline{\rho}$ (paragraphe 4.2.8) et d'après le théorème 4.3.1.0.1 nous avons $lfp(F_\pi(\phi))\Rightarrow lfp(\overline{\rho}\circ(F_\pi(\phi))\circ\overline{\rho})$.

Le choix d'un espace $\overline{R}$ de propriétés approchées a en quelque sorte permis de simplifier le système d'équations à résoudre puisque le calcul de $lfp(\overline{\rho}\circ(F_\pi(\phi))_\circ\overline{\rho})$ revient à calculer la plus petite solution du système d'équations approchées $X=\overline{\rho}(F_\pi(\phi))(X)$ défini sur $\overline{R}$. Ce système d'équations approchées est plus simple que $X=F_\pi(\phi)$ car il porte sur un espace $\overline{R}$ de propriétés plus simple que $(P_n)^\alpha$. En particulier $\overline{R}$ sera choisi pour que ses éléments soient facilement représentables en machine. Toutefois l'évaluation de $\overline{\rho}\circ F_\pi(\phi)$ en machine pose les mêmes problèmes que celle de $F_\pi(\phi)$ puisque les éléments de $(P_n)^\alpha$ n'ont pas de représentation canonique ce qui rend le calcul difficilement automatisable principalement à cause des problèmes de simplification. C'est pourquoi nous proposons d'évaluer $\overline{\rho}\circ F_\pi(\phi)$ à la main, ou plus précisément, d'élaborer manuellement un algorithme pour calculer $\overline{\rho}\circ F_\pi(\phi)$ ou, si c'est trop difficile, de construire un algorithme pour calculer $\overline{F}_\pi \in mon(\overline{R}\to\overline{R})$ tel que $\overline{\rho}\circ F_\pi(\phi) \Rightarrow \overline{F}_\pi$ de sorte que $lfp(\overline{\rho}\circ F_\pi(\phi)) \Rightarrow lfp(\overline{F}_\pi)$ (voir théorème 4.3.0.1 également 4.3.1.0.1-2). Pour éviter d'avoir à faire ce travail pour chaque programme particulier π on élaborera directement un algorithme de construction du système d'équations approché $X=\overline{F}_\pi(X)$ associé à un programme π quelconque. Très souvent toutes les propriétés approchées à déterminer en chaque point du programme seront de la même forme ce qui revient à choisir $\overline{\rho}\in((P_n)^\alpha \to (P_n)^\alpha)$ de la forme $\overline{\rho}=(\overline{\rho}_1,\ldots\overline{\rho}_\alpha)$ où $\overline{\rho}_1=\overline{\rho}_2=\ldots=\overline{\rho}_\alpha=\rho$ avec $\rho\in(P_n\to P_n)$. Dans ce cas à chaque règle $X_i=F_i(X_1,\ldots,X_\alpha)$ de la définition 3.3.0.1 correspond une règle approchée $X_i=\overline{F}_i(X_1,\ldots,X_n)$ où $X_j\in\overline{R}$ et $\overline{\rho}\circ F_i \Rightarrow \overline{F}_i$. L'algorithme permettant d'associer à tout programme π le système d'équations approché $X=\overline{F}_\pi(X)$ peut être écrit une fois pour toutes les applications en le paramétrant par la représentation machine des éléments de $\overline{R}$ et les fonctions correspondant aux règles élémentaires. De même le travail d'écriture des algorithmes de résolution des systèmes d'équations approchés $X=F_\pi(X)$ sera fait une fois pour toutes. On distinguera le cas où $\overline{R}$ satisfait la condition de chaîne ascendante ce qui permet d'utiliser l'un des théorèmes 2.9.1.0.2, 2.9.2.0.2 ou 2.9.3.0.9 pour calculer itérativement à partir de l'infimum de $\overline{R}$ la plus petite solution de $X=\overline{F}_\pi(X)$ et ce, en un nombre fini de pas. Dans le cas où la convergence n'est pas naturellement garantie la remarque 4.1.1.0.9. nous indique d'utiliser une itération croissante approchée supérieurement (4.1.1.0.1) suivie d'une itération décroissante approchée supérieurement (4.1.1.0.7). Cet algorithme sera paramétré par un élargissement supérieur (4.1.2.0.5) et un retrécissement

inférieur (4.1.2.0,16) qui doivent être choisis pour chaque application.

En résumé, pour définir un *algorithme d'analyse approchée des programmes pour une classe donnée de propriétés sémantiques* nous proposons:

1 - De choisir une fermeture (paramétrée par le nombre n de variables) de P_n ce qui détermine un sous-espace $\overline{R}$ de propriétés approchées,

2 - D'utiliser cette fermeture et les définitions des règles de construction des systèmes d'équations sémantiques en avant ou en arrière pour élaborer à la main les règles de construction des systèmes d'équations approchés,

3 - D'écrire l'algorithme permettant d'associer à tout programme les systèmes d'équations approchés (en avant et/ou en arrière),

4 - D'écrire un algorithme de résolution de ces systèmes (en utilisant éventuellement les méthodes d'accélération de la convergence si celle-ci n'est pas naturellement garantie en un nombre fini de pas).

Nous donnons tout d'abord des exemples très simples dont l'intérêt est d'illustrer la méthode en détail. Nous abordons ensuite des exemples plus complexes qui traitent de problèmes concrets. En particulier l'utilisation d'analyses sémantiques en arrière et la méthode de combinaison des analyses sémantiques en avant et en arrière est développée et illustrée dans les paragraphes 5.6 et 5.7.

5.2 EXEMPLE DE LA DECOUVERTE AUTOMATIQUE DU SIGNE DES VARIABLES NUMERIQUES D'UN PROGRAMME PAR UNE ANALYSE SEMANTIQUE EN AVANT APPROCHEE SUPERIEUREMENT

L'intérêt de cet exemple très simple est de fournir un support intuitif illustrant notre technique d'analyse approchée de propriétés sémantiques des programmes.

5.2.1 Définition d'un espace de propriétés approchées par une fermeture supérieure

Pour étudier le signe des valeurs des variables d'un programme π il faut déterminer en chaque point du programme des invariants du type $\underline{\text{ET}}_{i=1}^{n}(x_i r_i 0)$ où r_i est une relation d'inégalité $\geq$ ou $\leq$. Ayant choisi cette classe particulière de propriétés sémantiques, nous procédons à la construction de la fermeture supérieure correspondante.

Nous retrouverons souvent le cas d'un espace de propriétés approchées ne faisant pas intervenir de relations entre les variables $x_1,\ldots,x_n$. Par exemple une approximation supérieure de $P=\lambda(x,y).[(x\geq y>0)$ <u>ou</u> $(x=y=0)]$ où l'on ne considère que des relations faisant intervenir une seule variable est $\lambda(x,y).(x\geq 0$ <u>et</u> $y\geq 0)$. Pour définir une telle approximation par une fermeture supérieure nous procédons comme suit:

Posons pour $j = 1,\ldots,n$:

$$\sigma_j^n \in P_n \to P_1 \text{ où } P_n=(U^n \to B) \text{ et } B=\{\underline{\text{vrai}},\underline{\text{faux}}\}$$

$$\sigma_j^n = \lambda P.[\lambda x.\{\exists (v_1,\ldots,v_{j-1},v_{j+1},\ldots,v_n)\in U^{n-1} : P(v_1,\ldots,v_{j-1},x,v_{j+1},\ldots,v_n)\}]$$

$$@ \in P_n \to (P_1)^n$$

$$@ = \lambda P.(\sigma_1^n(P),\sigma_2^n(P),\ldots,\sigma_n^n(P))$$

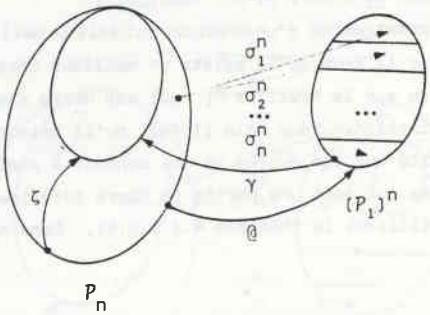
$$\gamma \in (P_1)^n \to P_n$$

$$\gamma = \lambda(P_1,\ldots,P_n).[\lambda(x_1,\ldots,x_n).(\underline{\text{ET}}_{j=1}^{n} P_j(x_j))]$$

$$\zeta = \gamma\circ@ \in (P_n \to P_n)$$

$$\zeta = \lambda P.[\lambda(x_1,\ldots,x_n).(\underline{\text{ET}}_{j=1}^{n} \sigma_j^n(P)(x_j))]$$

Schématiquement on a:



On vérifie que ζ est une fermeture supérieure de P_n (définition 4.2.1.0.1) et que $\zeta(P_n)$ est l'ensemble des prédicats portant sur $x_1,\ldots,x_n$ et n'exprimant pas de relations entre ces variables, c'est-à-dire que $\zeta(P_n)$ est complètement isomorphe à $(P_1)^n = (U \to \{\underline{vrai},\underline{faux}\})^n$ par l'isomorphisme complet γ dont l'inverse est @.

Les prédicats portant sur une seule variable pouvant encore être très complexes nous introduisons une fermeture supérieure η sur P_1 pour ne retenir que les propriétés des variables qui sont intéressantes pour un problème donné. Par exemple, pour étudier le signe des valeurs des variables nous pouvons choisir des prédicats de la forme $\lambda x.(x=\Omega)$, $\lambda x.(x\geq 0 \ \underline{ou}\ x=\Omega)$, $\lambda x.(x>0 \ \underline{ou}\ x=\Omega)$, $\lambda x.(x\leq 0 \ \underline{ou}\ x=\Omega)$, $\lambda x.(x<0 \ \underline{ou}\ x=\Omega)$. On suppose que $\Omega\epsilon U$ est la valeur des variables non initialisées. Nous avons choisi des prédicats en disjonction avec $\lambda x.(x=\Omega)$ puisqu'il semble a priori difficile d'étudier la bonne initialisation des variables sur l'unique base de leur signe. On notera également que cet ensemble n'est pas une partie de Moore inférieure puisqu'il ne contient pas $\lambda x.(x\geq 0 \ \underline{ou}\ x=\Omega)$ et $\lambda x.(x\leq 0 \ \underline{ou}\ x=\Omega) = \lambda x.(x=0 \ \underline{ou}\ x=\Omega)$. Dans ce cas il y a deux approximations possibles pour le prédicat $\lambda x.(x=0)$ en l'occurrence $\lambda x.(x\geq 0 \ \underline{ou}\ x=\Omega)$ ou $\lambda x.(x\leq 0 \ \underline{ou}\ x=\Omega)$. Indépendamment d'un programme particulier il n'y a pas de meilleure approximation du prédicat $\lambda x.(x=0)$ car par exemple, l'approximation $\lambda x.(x\geq 0 \ \underline{ou}\ x=\Omega)$ dans:

x:=0 {$\lambda x.(x\geq 0 \ \underline{ou}\ x=\Omega)$}; x:=x+1 {$\lambda x.(x\geq 0 \ \underline{ou}\ x=\Omega)$};

conduit à une analyse plus fine que l'approximation $\lambda x.(x\leq 0 \ \underline{ou}\ x=\Omega)$:

x:=0 {$\lambda x.(x\leq 0 \ \underline{ou}\ x=\Omega)$}; x:=x+1 {$\lambda x.\underline{vrai}$};

tandis que,

x:=0 {$\lambda x.(x\leq 0 \ \underline{ou}\ x=\Omega)$}; x:=x-1 {$\lambda x.(x\leq 0 \ \underline{ou}\ x=\Omega)$}

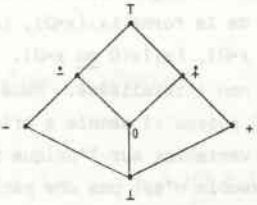
est meilleure que :

x:=0 {λx.(λx.(x≥0 ou x=Ω)}; x:=x-1 {λx.vrai};

Pour définir une approximation d'un prédicat qui soit la meilleure pour tous les programmes à analyser il faut qu'il existe un meilleur choix possible sur la seule base de l'ordre sur le treillis P_1 qui est notre seule "mesure" de la qualité de l'approximation. Pour cela il faut qu'il existe une approximation supérieure plus petite que les autres ce qui conduit à choisir un ensemble de propriétés approchées qui soit une partie de Moore inférieure (ce qui est toujours possible en utilisant le théorème 4.2.2.0.5). Dans notre exemple nous définirons donc:

```
η = λP.cas
        P => λx.(x=Ω)          → λx.(x=Ω);
        P => λx.(x=0 ou x=Ω)   → λx.(x=0 ou x=Ω);
        P => λx.(x>0 ou x=Ω)   → λx.(x>0 ou x=Ω);
        P => λx.(x<0 ou x=Ω)   → λx.(x<0 ou x=Ω);
        P => λx.(x≥0 ou x=Ω)   → λx.(x≥0 ou x=Ω);
        P => λx.(x≤0 ou x=Ω)   → λx.(x≤0 ou x=Ω);
        P => λx.vrai           → λx.vrai;
      fincas;
```

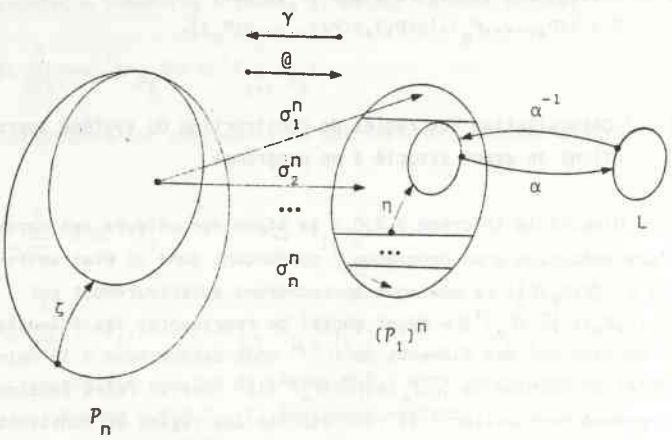
Pour représenter les éléments de $\eta(P_1)$ en machine nous utiliserons le treillis suivant:



qui est isomorphe au treillis $\eta(P_1)$ par l'isomorphisme α:

```
α = λP.cas P dans
        λx.(x=Ω)          → ⊥ ;
        λx.(x=0 ou x=Ω)   → 0 ;
        λx.(x>0 ou x=Ω)   → + ;
        λx.(x<0 ou x=Ω)   → - ;
        λx.(x≥0 ou x=Ω)   → ‡ ;
        λx.(x≤0 ou x=Ω)   → ± ;
        λx.vrai           → T ;
      fincas;
```

dont l'inverse sera noté α^{-1}. A ce point la situation est la suivante:



Le plus simple étant de faire la même approximation pour toutes les variables du programme on vérifie aisément que:

$\overline{\eta} = \lambda(P_1,\ldots,P_n).(\eta(P_1),\eta(P_2),\ldots,\eta(P_n))$

est une fermeture supérieure de $(P_1)^n$ et que $\overline{\eta}(P_1^n)$ est complètement isomorphe à L^n par l'isomorphisme complet:

$\overline{\alpha} = \lambda(P_1,\ldots,P_n).(\alpha(P_1),\alpha(P_2),\ldots,\alpha(P_n))$

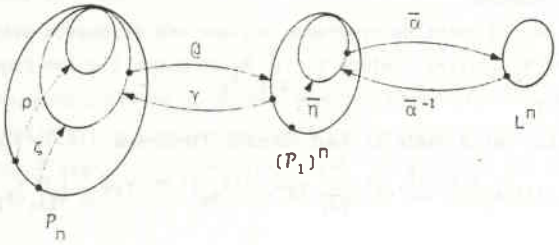
dont l'inverse est:

$\overline{\alpha}^{-1} = \lambda(v_1,\ldots,v_n).(\alpha^{-1}(v_1),\alpha^{-1}(v_2),\ldots,\alpha^{-1}(v_n))$

D'après le théorème 4.2.4.0.6 on a défini une fermeture ρ sur P_n par:

$\rho = \gamma\circ\overline{\eta}\circ@\circ\zeta$

et $\rho(P_n)$ est isomorphe à L^n par $\overline{\alpha}\circ@$ dont l'inverse est $\gamma\circ\overline{\alpha}^{-1}$. On a donc une image approchée supérieurement de P_n facilement représentable en machine:



Ayant défini une fermeture ρ sur P_n il est naturel de définir la même approximation en chaque point du programme (mais ce n'est pas une nécessité) et l'on obtient une fermeture $\overline{\rho}$ sur $(P_n)^\alpha$ par:

$$\overline{\rho} = \lambda(P_1,\ldots,P_\alpha).(\rho(P_1),\rho(P_2),\ldots,\rho(P_\alpha)).$$

5.2.2 Détermination des règles de construction du système approché d'équations en avant associé à un programme

D'après le théorème 3.3.0.2 le signe des valeurs des variables au cours d'une exécution d'un programme π commençant dans un état satisfaisant ϕ est $\overline{\rho}(lfp(F_\pi(\phi)))$ ce que nous approcherons supérieurement par $lfp(\overline{\rho}\circ F_\pi(\phi)|\overline{\rho}((P_n)^\alpha))$. Ayant choisi de représenter les éléments de $\overline{\rho}((P_n)^\alpha)$ en machine par des éléments de $(L^n)^\alpha$ nous déterminons à la main une représentation machine de $(\overline{\rho}\circ F_\pi(\phi)|\overline{\rho}((P_n)^\alpha))$. Pour le faire indépendamment d'un programme particulier π il faut établir les règles de construction d'un système d'équations approché $X=\overline{F}_\pi(\overline{\phi})(X)$ sur $(L^n)^\alpha$. Ces règles seront justifiées en démontrant que $\{(\rho\circ F_\pi(\phi)_j\circ\rho) \Rightarrow (\gamma\circ\overline{\alpha}^{-1}\circ\overline{F}_\pi(\overline{\phi})_j\circ\alpha\circ@)\}$ car d'après le théorème 4.3.0.1 ceci implique pour tout $j=1,\ldots,\alpha$ que $\{lfp(F_\pi(\phi))_j \Rightarrow lfp(\overline{\rho}\circ F_\pi(\phi)\circ\overline{\rho})_j \Rightarrow \gamma\circ\alpha^{-1}(lfp(\overline{F}_\pi(\overline{\phi}))_j)\}$. Nous établissons et justifions ces règles comme suit:

Entrée du programme:

Si a_j est le point d'entrée du programme alors $X_j=\overline{\alpha}\circ@\circ\rho(\phi)$ et ce calcul est fait à la main, l'utilisateur spécifiant directement les conditions d'entrée par un élément de L^n plutôt que par un élément de P_n. D'ailleurs dans la plupart des cas la spécification d'entrée est standard (les plus simples étant aucune des variables n'est initialisée $(\bot,\bot,\ldots,\bot)$ ou toutes les variables sont initialisées à des valeurs inconnues $(\top,\top,\ldots,\top)$)

Jonction de chemins:

Si a_j est le point de programme suivant une étiquette précédée des points de programme $a_{i_1},\ldots,a_{i_k}$, alors $X_j=\bigsqcup_{\ell=1}^{k} X_{i_\ell}$ où $\bigsqcup$ est l'union dans le treillis $L^n(\sqsubseteq,\bot,\top,\sqcup,\sqcap)$.

Pour justifier cette règle il faut montrer (théorème 4.3.0.1) que :

$$\forall(X_{i_1},\ldots,X_{i_k})\in(L^n)^k,\ \rho(\overline{\bigcup_{\ell=1}^{k}}\ \gamma\circ\overline{\alpha}^{-1}(X_{i_\ell})) \Rightarrow \gamma\circ\overline{\alpha}^{-1}(\bigsqcup_{\ell=1}^{k} X_{i_\ell})$$

(Intuitivement, les calculs sur L^n doivent être une approximation supérieure des calculs sur $\rho(P_n)$, l'idéal étant d'obtenir une égalité, ce qui est le cas pour cet exemple).

Comme $\rho = \gamma\circ\overline{\eta}\circ@\circ\zeta = \gamma\circ\overline{\eta}\circ@\circ\gamma\circ@ = \gamma\circ\overline{\eta}\circ@$ il suffit de montrer que:

$$\overline{\eta}\circ@(\underset{\ell=1}{\overset{k}{\underline{OU}}}\ \gamma\circ\alpha^{-1}(X_{i_\ell})) = \overline{\alpha}^{-1}(\underset{\ell=1}{\overset{k}{\sqcup}} X_{i_\ell})$$

On procède composante par composante ce qui revient à montrer que $\forall m=1,\ldots,n$ on a:

$$\eta\circ\sigma^n_m(\underset{\ell=1}{\overset{k}{\underline{OU}}}\ \gamma\circ\alpha^{-1}(X_{i_\ell})) = \alpha^{-1}(\underset{\ell=1}{\overset{k}{\sqcup}} (X_{i_\ell})_m)$$

soit

$$\eta\circ\sigma^n_m(\underset{\ell=1}{\overset{k}{\underline{OU}}}\ (\lambda(x_1,\ldots,x_n).(\underset{j=1}{\overset{n}{\underline{ET}}}\ \alpha^{-1}(X_{i_\ell})_j(x_j)))) = \alpha^{-1}(\underset{\ell=1}{\overset{k}{\sqcup}} (X_{i_\ell})_m)$$

α^{-1} est un isomorphisme complet de L dans $\eta(P_1)$ et l'union dans $\eta(P_1)$ est donnée par le théorème 2.3.0.1. Il faut donc montrer:

$$\eta\circ\sigma^n_m(\lambda(x_1,\ldots,x_n).(\underset{\ell=1}{\overset{k}{\underline{OU}}}(\underset{j=1}{\overset{n}{\underline{ET}}}\ \alpha^{-1}(X_{i_\ell})_j(x_j)))) = \eta(\underset{\ell=1}{\overset{k}{\underline{OU}}}\ \alpha^{-1}((X_{i_\ell})_m))$$

Eliminant η et remplaçant σ^n_m par sa définition, il vient:

$$\lambda x.\{\}(v_1,\ldots,v_{m-1},v_{m+1},\ldots,v_n)\in U^{n-1} : \underset{\ell=1}{\overset{k}{\underline{OU}}}((\underset{j=1,j\neq m}{\overset{n}{\underline{ET}}}\ \alpha^{-1}(X_{i_\ell})_j(v_j)) \underline{et} (\alpha^{-1}(X_{i_\ell})_m(x)))\}$$

Pour $j\in\{1,\ldots,n\}-\{m\}$, en choisissant $v_j=\Omega$ on a toujours $\alpha^{-1}(X_{i_\ell})_j(v_j)$ qui est vrai car $\alpha^{-1}(X_{i_\ell})$ est de la forme $\lambda x.((x=\Omega)\underline{ou}\ldots)$ et l'on obtient:

$$\underset{\ell=1}{\overset{k}{\underline{OU}}}\ (\alpha^{-1}(X_{i_\ell})_m)$$

ce qui termine la preuve.

Test :

Nous examinons le cas où a_j est le point de sortie vrai d'un test p. Pour simplifier nous supposons que p est de la forme $\lambda(x,y).(x=y)$, $\lambda(x,y).(x\neq y)$, $\lambda(x,y).(x>y)$ ou $\lambda(x,y).(x\geq y)$. (Ceci n'est pas une limitation théorique car les autres cas peuvent se traiter par composition en utilisant

le théorème 4.2.8.0.4.(c)-(d). Ainsi (si x<y alors I_1 sinon I_2 finsi) est équivalent à (si y>x alors I_2 sinon I_1 finsi), de même (si x=0 alors ...) est équivalent à (z:=0; si x=z alors ...),(si (x=z) et (y>z) alors ...) est équivalent à (si x=y alors (si y>z alors ...)finsi;), etc. Bien sûr en pratique il est plus efficace de considérer les tests sous la forme générale donnée dans le langage, mais les calculs sont trop longs pour les donner ici et ils n'apportent rien à la compréhension du sujet).

Soient X_j et $X_i \in L^n$ respectivement associés aux points de sortie vrai a_j et d'entrée a_i d'un test de la condition p. Connaissant X_i il faut trouver la valeur de X_j qui doit être égale ou supérieure à:

$$\bar{\alpha}\circ@\circ\rho(\underline{test}(p)(\gamma\circ\alpha^{-1}(X_i)))$$

Comme $\bar{\alpha}\circ@\circ\rho = \bar{\alpha}\circ@\circ\gamma\circ\bar{\eta}\circ@\circ\zeta = \bar{\alpha}\circ\bar{\eta}\circ@\circ\gamma\circ@ = \bar{\alpha}\circ\bar{\eta}\circ@$ il faut trouver d'après la définition 3.3.0.1 une approximation supérieure de:

$$\bar{\alpha}\circ\bar{\eta}\circ@(\lambda(x_1,\ldots,x_n).\{\gamma\circ\bar{\alpha^{-1}}(X_i)(x_1,\ldots,x_n) \text{ et } (x_1,\ldots,x_n)\in dom(p) \text{ et } p(x_1,\ldots,x_n)\}$$

$$\Rightarrow \bar{\alpha}\circ\bar{\eta}\circ@(\lambda(x_1,\ldots,x_n).\{\gamma\circ\bar{\alpha^{-1}}(X_i)(x_1,\ldots,x_n) \text{ et } p(x_1,\ldots,x_n)\}$$

car $\bar{\alpha}$, $\bar{\eta}$ et @ sont monotones. On ignore donc la possibilité que le test soit incorrect. Nous poursuivons le calcul composante par composante, et pour q=1,...,n il faut trouver une approximation supérieure de:

$$\alpha\circ\eta\circ\sigma^n_q(\lambda(x_1,\ldots,x_n).\{\gamma\circ\bar{\alpha^{-1}}(X_i)(x_1,\ldots,x_n) \text{ et } p(x_1,\ldots,x_n)\})$$

$$= \alpha\circ\eta(\lambda x.\{\exists v_1,\ldots,v_{q-1},v_{q+1},\ldots,v_n)\in U^{n-1} : \gamma\circ\bar{\alpha^{-1}}(X_i)(v_1,\ldots,x,\ldots,v_n) \text{ et } p(v_1,\ldots,x,\ldots,v_n)\})$$

$$= \alpha\circ\eta(\lambda x.\{\exists(v_1,\ldots,v_n)\in U^{n-1} : \underset{\ell=1,\ell\neq q}{\overset{n}{ET}} (\alpha^{-1}(X_i)_\ell)(v_\ell) \text{ et } (\alpha^{-1}(X_i)_q)(x) \text{ et } p(v_1,\ldots,x,\ldots,v_n)\})$$

- Si la variable x_q n'intervient pas dans le test on a pour tout $x\in U^n$, $p(v_1,\ldots,x,\ldots,v_n) = p(v_1,\ldots,\Omega,\ldots,v_n)$. Donc choisissant, $v_1 = \ldots = v_{q-1} = v_{q+1} = \ldots = v_n = \Omega$, on peut simplifier:

$$= \alpha\circ\eta(\alpha^{-1}(X_i)_q) = (X_i)_q$$

Pour toutes les variables n'intervenant pas dans le test on choisira donc $(X_j)_q = (X_i)_q$. Pour les autres variables le choix dépend bien évidemment du test.

- Si $p=\lambda(x_1,\ldots,x_m).(x_q=x_r)$ alors il faut trouver une approximation supérieure de:

$$\alpha\circ\eta(\lambda x.\{\exists(v_1,\ldots,v_n)\in U^{n-1} : \underset{\ell=1,\ell\neq q}{\overset{n}{\underline{ET}}} (\alpha^{-1}(X_i)_\ell)(v_\ell) \underline{\text{ et }} (\alpha^{-1}(X_i)_q)(x) \underline{\text{ et }} x=v_r\})$$

$$= \alpha\circ\eta(\lambda x.\{\exists v_r\in U : (\alpha^{-1}(X_i)_q)(x) \underline{\text{ et }} (\alpha^{-1}(X_i)_r)(v_r) \underline{\text{ et }} x=v_r\})$$

$$= \alpha\circ\eta\{\alpha^{-1}(X_i)_q \underline{\text{ et }} \alpha^{-1}(X_i)_r\}$$

$$= \{\alpha\circ\eta\circ\alpha^{-1}(X_i)_q \sqcap \alpha\circ\eta\circ\alpha^{-1}(X_i)_r\}$$ par η est une fermeture, le théorème 2.3.0.1 et α est un isomorphisme complet de $\eta(P_1)$ dans L.

$$= (X_i)_q \sqcap (X_i)_r$$

On conclut donc que si X_i est associé au point d'entrée a_i d'un test $x_q=x_r$, on a $X_j = X_i(x_q \leftarrow x_q \sqcap x_r,\ x_r \leftarrow x_q \sqcap x_r)$ sur l'arc de sortie vrai, cette notation signifiant $(X_j)_m=(X_i)_m$ pour $m\in\{1,\ldots,n\}-\{q,r\}$, $(X_j)_q=(X_i)_q\sqcap(X_i)_r$ et $(X_j)_r=(X_i)_q\sqcap(X_i)_r$

- Si $p=\lambda(x_1,\ldots,x_n).(x_q\geq x_r)$ alors il faut trouver une approximation supérieure de:

$$\alpha\circ\eta(\lambda x.\{\exists v_1,\ldots,v_n\in U^{n-1} : \underset{\ell=1,\ell\neq q,\ell\neq r}{\overset{n}{\underline{ET}}} (\alpha^{-1}(X_i)_\ell)(v_\ell) \underline{\text{ et }} (\alpha^{-1}(X_i)_q)(x) \underline{\text{ et }} (\alpha^{-1}(X_i)_r)(v_r) \underline{\text{ et }} x\geq v_r\})$$

$$= \alpha\circ\eta(\lambda x.\{\exists v_r : (\alpha^{-1}(X_i)_q)(x) \underline{\text{ et }} (\alpha^{-1}(X_i)_r)(v_r) \underline{\text{ et }} x\geq v_r\})$$

On peut poursuivre l'étude par cas:

- Si $(\alpha^{-1}(X_i)_r)(v_r) = (v_r=\Omega)$ alors $x\geq\Omega$ n'est pas défini et l'on peut considérer que la sémantique du langage spécifie qu'aucune des deux branches de sortie du test n'est empruntée. Ceci revient à dire que le test est <u>faux</u> pour la branche de sortie <u>vrai</u> (et <u>vrai</u> pour la branche de sortie <u>faux</u>):
  $\alpha\circ\eta(\lambda x.\underline{\text{faux}}) = \alpha(\lambda x.(x=\Omega)) = \bot$
- Si $(\alpha^{-1}(X_i)_r)(v_r) = (v_r=0 \underline{\text{ ou }} v_r=\Omega)$ alors:
  $= \alpha\circ\eta(\lambda x.\{\exists v_r : (\alpha^{-1}(X_i)_q)(x) \underline{\text{ et }} ((v_r=0 \underline{\text{ ou }} v_r=\Omega) \underline{\text{ et }} x\geq v_r)\})$

$= \alpha\circ\eta(\lambda x.\{\exists v_r:(\alpha^{-1}(X_i)_q)(x)$ et $((v_r=0$ et $x\geq v_r)$ ou $(v_r=\Omega$ et $x\geq v_r))\})$

Comme $x\geq\Omega$ est faux, on obtient:

$= \alpha\circ\eta(\lambda x.\{(\alpha^{-1}(X_i)_q)(x)$ et $(x\geq 0)\}$

$\Rightarrow \alpha\circ\eta(\lambda x.\{(\alpha^{-1}(X_i)_q)(x)$ et $(x\geq 0$ ou $x=\Omega)\}$

$= \alpha\circ\eta\circ\alpha^{-1}(X_i)_q \sqcap \alpha\circ\eta\circ\alpha^{-1}(\dot{+})$

$= (X_i)_q \sqcap \dot{+}$

On voit que l'on obtient la même chose quand $(\alpha^{-1}(X_i)_r)(v_r) = (v_r\geq 0$ ou $v_r=\Omega)$ tandis que pour $(\alpha^{-1}(X_i)_r)(v_r) = (v_r>0$ ou $v_r=\Omega)$ on obtient $(X_i)_q \sqcap +$.

- Si $(\alpha^{-1}(X_i)_r)(v_r) = (v_r\leq 0$ ou $v_r=\Omega)$ on doit approcher supérieurement $\alpha\circ\eta(\lambda x.\{\exists v_r:(\alpha^{-1}(X_i)_q)(x)$ et $(v_r\leq 0$ et $x\geq v_r)\}) = (X_i)_q$

  Même résultat quand $(X_i)_r = \tau$ ou $-$.

On peut poursuivre cette étude pour toutes les alternatives pour p et construire les règles de compositions pour les tests comportant des et ou et non. Il est inutile de la mener plus avant parce que dans cet exemple on peut faire confiance à l'intuition. Nous retiendrons toutefois, que la théorie fournit un excellent guide pour déterminer les règles de construction du système d'équations approché. De plus on a une preuve qu'elles sont correctes ce qui démontre du même coup que l'analyse approchée de tout programme sera correcte.

Affectation :

Enonçons tout de même brièvement la règle pour l'instruction d'affectation. Il suffit de remplacer dans le membre droit de l'affectation les variables par leur signe ainsi que les constantes dont la valeur absolue est différente de 1 puis d'appliquer la règle des signes comme $\bot\dot{+}\dot{+}=\bot$, $+\dot{+}\dot{+}=+$, $+\dot{-}1=\dot{+}$, $\dot{+}\dot{+}\dot{-}=\tau$, etc... On obtient alors le signe de l'expression membre droit de l'affectation que l'on affecte à la variable du membre gauche. En résumé, l'équation associée à l'affectation,

"$a_j : x_q := f(x_1,\ldots,x_n); a_i : \ldots$"

est:

$$X_j = X_i(x_q \leftarrow \overline{f}((X_i)_1,\ldots,(X_i)_n))$$

où $\overline{f}$ est obtenu à partir de f en appliquant la règle des signes.

5.2.3 Résolution itérative du système d'équations approché dans le cas d'une convergence naturelle et exemple

Donnons maintenant un exemple (Manna[1974], p.185). Il s'agit de calculer $\lceil\sqrt{x}\rceil$ quand $x \geq 0$:

```
{1}
        (y1,y2,y3):=(0,0,1);
{2}
    L:
{3}
        y2:=y2+y3;
{4}
        si y2≤x alors
{5}
            (y1,y3):=(y1+1,y3+2);
{6}
            allera L;
        finsi;
{7}
```

Les règles que nous venons d'établir permettent d'associer automatiquement le système d'équations approché suivant à ce programme:

$$
\begin{cases}
P_1 = \langle (x,\dot{+}),(y_1,\perp),(y_2,\perp),(y_3,\perp)\rangle \\
P_2 = P_1(y_1 \leftarrow 0, y_2 \leftarrow 0, y_3 \leftarrow +) \\
P_3 = P_2 \sqcup P_6 \\
P_4 = P_3(y_2 \leftarrow y_2 + y_3) \\
P_5 = P_4(y_2 \leftarrow \underline{si}\ x=\perp\ \underline{alors}\ \perp\ \underline{sinon}\ y_2 \sqcap (x \sqcup -)\ \underline{finsi}, \\
\qquad x \leftarrow \underline{si}\ y_2=\perp\ \underline{alors}\ \perp\ \underline{sinon}\ x \sqcap (y_2 \sqcup +)\ \underline{finsi}) \\
P_6 = P_5(y_1 \leftarrow y_1+1, y_3 \leftarrow y_3++) \\
P_7 = P_4(y_2 \leftarrow \underline{si}\ x=\perp\ \underline{alors}\ \perp\ \underline{sinonsi}\ (x=0)\ \underline{ou}\ (x=+)\ \underline{ou}\ (x=\dot{+})\ \underline{alors}\ y_2 \sqcap + \\
\qquad\qquad \underline{sinon}\ y_2\ \underline{finsi}, \\
\qquad x \leftarrow \underline{si}\ y_2=\perp\ \underline{alors}\ \perp\ \underline{sinonsi}\ (y_2=0)\ \underline{ou}\ (y_2=-)\ \underline{ou}\ (y_2=\dot{+})\ \underline{alors}\ x \sqcap - \\
\qquad\qquad \underline{sinon}\ x\ \underline{finsi})
\end{cases}
$$

Comme L est fini, $(L^{\sqcap})^{\alpha}$ est fini et le calcul de $lfp(\overline{F}_{\pi}(\overline{\alpha}))$ par approximations successives converge en un nombre fini de pas.
Nous avons trouvé les résultats suivants:

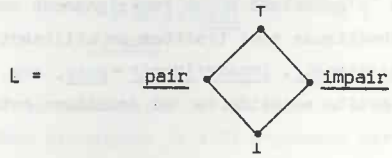
	x	y_1	y_2	y_3
{1}	⋮	⊥	⊥	⊥
{2}	⋮	0	0	+
{3}	⋮	⋮	⋮	+
{4}	⋮	⋮	+	+
{5}	+	⋮	+	+
{6}	+	+	+	+
{7}	⋮	⋮	+	+

A la suite de cet exemple nous retiendrons que dans la conception d'un algorithme d'analyse sémantique aprochée des programmes nous avons laissé à l'initiative humaine le choix de l'espace de propriétés approchées jugé satisfaisant pour résoudre un problème donné. Ce choix étant fait nous offrons une théorie pour déterminer les règles de construction du système d'équations approché. Comme l'exemple l'a montré il s'agit uniquement de simplifier les fonctions de transformation des prédicats approchés associés aux divers points du programme. Nous avons laissé à l'être humain le soin d'effectuer cette simplification mais une assistance de la machine dans les calculs symboliques peut être envisagée. Ce travail ayant été fait une fois pour toutes, c'est la machine qui associera un système d'équations approché à un programme particulier et le résoudra en utilisant une méthode itérative. Dans le cas d'une application où la convergence n'est pas naturellement garantie il est fait une fois pour toutes appel à l'intuition humaine pour déterminer la méthode d'approximation dynamique à appliquer, c'est-à-dire principalement pour choisir un élargissement et un retrécissement (voir paragraphe 4.1.2).

5.3 EXEMPLE DE LA DECOUVERTE AUTOMATIQUE APPROCHEE DE LA PARITE DES VARIABLES ENTIERES D'UN PROGRAMME

Un autre exemple aussi simple est celui de la détermination de la parité des variables d'un programme. Soit n le nombre de variables entières $x_1, \ldots, x_n$ du programme. Nous considérons une image approchée supérieurement L^n du treillis complet P_n des prédicats sur ces variables définie

comme suit:



$$\gamma = \lambda(v_1,\ldots,v_n).[\lambda(x_1,\ldots,x_n).\ \underset{j=1}{\overset{n}{\text{ET}}}\ \gamma_1(v_j)(x_j)]$$

$\gamma_1 = \lambda w.$ **cas** v **dans**
⊥ → λx.(x=Ω);
pair → λx.((x **modulo** 2 = 0) **ou** (x=Ω));
impair → λx.((x **modulo** 2 = 1) **ou** (x=Ω));
T → λx.**vrai**;
fincas;

On notera que γ est un morphisme injectif complet pour l'intersection de sorte que le théorème 4.2.7.0.5 fournit l'abstraction correspondante. Nous ne détaillerons pas la méthode pour associer un système d'équations approché à un programme (5.2.2) et nous donnerons simplement un exemple (produit de deux entiers a et b):

```
{1}   r:=0; i:=1;
      jusqu'à i=abs(b) faire
{2}
{3}         r:=r+a;  i:=i+1;
      refaire;
{4}
      si b<0 alors r:=-r; finsi;
{5}
```

Soient α,β les parités des valeurs initiales des variables a et b. Le système d'équations approché associé à ce programme est:

$$\begin{cases} X_1 = \langle (a,\alpha),(b,\beta),(r,\underline{\text{pair}}),(i,\underline{\text{impair}})\rangle \\ X_2 = X_1 \sqcup X_3 \\ X_3 = X_2(r \leftarrow r+a, i \leftarrow i+\underline{\text{impair}}) \\ X_4 = (X_1 \sqcup X_3)(i \leftarrow i \sqcap b, b \leftarrow i \sqcap b) \\ X_5 = X_4 \end{cases}$$

On notera que le test i=abs(b) apporte l'information que la parité de i et abs(b) c'est-à-dire de i et b doit être la même quand ce test est vrai. Par contre les tests i≠abs(b) et b<0 n'apportent aucun renseignement concernant la parité. Les expressions arithmétiques sont traitées en utilisant les règles pair+pair→pair, pair+impair→impair, impair+impair→pair, etc... Le treillis L^n étant fini, la plus petite solution de ces équations est obtenue par une itération finie.

Si a est initialement pair et b impair nous avons obtenu:

$$\begin{cases} X_1 = \langle(a,\underline{pair}),(b,\underline{impair}),(r,\underline{pair}),(i,\underline{impair})\rangle \\ X_2, X_3 = \langle(a,\underline{pair}),(b,\underline{impair}),(r,\underline{pair}),(i,\tau)\rangle \\ X_4, X_5 = \langle(a,\underline{pair}),(b,\underline{impair}),(r,\underline{pair}),(i,\underline{impair})\rangle \end{cases}$$

La perte de précision dans l'approximation est mise en évidence quand on considère les valeurs initiales a impair et b pair puisque nous trouvons:

$$\begin{cases} X_1 = \langle(a,\underline{impair}),(b,\underline{pair}),(r,\underline{pair}),(i,\underline{impair})\rangle \\ X_2, X_3 = \langle(a,\underline{impair}),(b,\underline{pair}),(r,\tau),(i,\tau)\rangle \\ X_4, X_5 = \langle(a,\underline{impair}),(b,\underline{pair}),(r,\tau),(i,\underline{pair})\rangle \end{cases}$$

En fait quand la valeur initiale de a est impaire et celle de b est paire la valeur finale de r à la sortie du programme sera paire. Pour le démontrer on utilise le fait qu'en additionnant un nombre pair de fois un nombre impair à lui-même on obtient un nombre pair. Le raisonnement tenant compte de la valeur de b, il est impossible de trouver ce résultat par une approximation ignorant cette valeur.

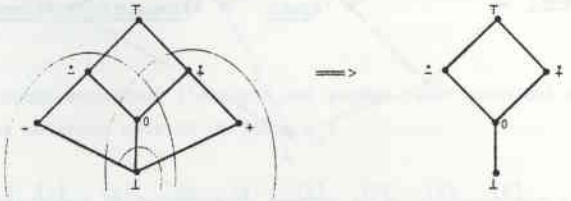
5.4 COMBINAISON D'ANALYSES APPROCHEES : SIGNE ET PARITE DES VARIABLES ENTIERES D'UN PROGRAMME

L'intérêt de cet exemple est d'illustrer les résultats du paragraphe 4.2.3 sur la combinaison d'analyses approchées d'un programme par combinaison des fermetures correspondantes.

Pour la règle des signes (sans considérer la possibilité d'être strictement positif ou négatif) la fermeture supérieure est donnée par:

```
ρ1 = λP.cas
       P ⇒ λx.(x=Ω)                  → ⊥;
       P ⇒ λx.(x=0 ou x=Ω)           → 0;
       P ⇒ λx.(x≥0 ou x=Ω)           → ↑;
       P ⇒ λx.(x≤0 ou x=Ω)           → ↓;
       P ⇒ λx.vrai                   → T;
     fincas;
```

(Noter au passage que ρ_1 est obtenue à partir de η (5.2.1) par la famille d'idéaux principaux (4.2.5) engendrés par les suprema $\dot{+}$, $\dot{-}$ et $\perp$:



)

Pour l'étude de parité on avait:

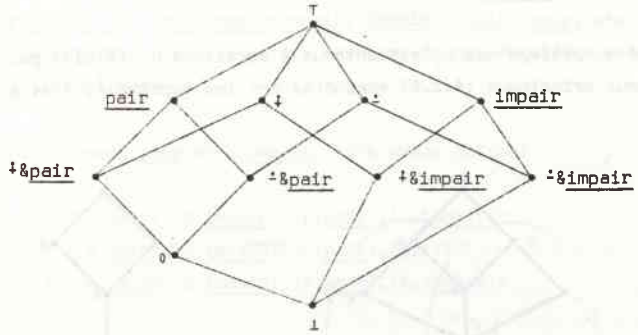
```
ρ2 = λP.cas
       P ⇒ λx.(x=Ω)                              → ⊥;
       P ⇒ λx.(x modulo 2 = 0 ou x=Ω)            → pair;
       P ⇒ λx.(x modulo 2 = 1 ou x=Ω)            → impair;
       P ⇒ λx.vrai                               → T;
     fincas;
```

D'après le théorème 4.2.3.0.10 pour calculer $\rho_1 \sqcup \rho_2$ il suffit de calculer l'intersection des ensembles $\{\perp,0,\dot{+},\dot{-},T\}$ et $\{\perp,\underline{pair},\underline{impair},T\}$ soit $\{\perp,T\}$ qui est une famille de Moore à laquelle correspond la fermeture:

```
ρ1 ⊔ ρ2 = λP.cas
            P ⇒ λx.(x=Ω) → ⊥;
            P ⇒ λx.vrai  → T;
          fincas;
```

L'approximation obtenue dans cet exemple est tellement grossière qu'elle peut paraître inutile mais ce n'est pas le cas puisqu'on peut l'utiliser pour montrer que le graphe du programme est connexe! .

Le théorème 4.2.3.0.10 indique aussi que pour obtenir $\rho_1 \sqcap \rho_2$ il suffit de construire la plus petite famille de Moore contenant $\{\bot, 0, \dot{+}, \dot{-}, \top\}$ et $\{\bot, \underline{pair}, \underline{impair}, \top\}$. On l'obtient en rajoutant toutes les intersections qui ne sont pas dans l'ensemble:



En faisant un tri topologique on obtient directement la fermeture correspondante:

```
ρ1 ⊓ ρ2 = λP.cas
            P => λx.(x=Ω)                                          → ⊥;
            P => λx.(x=0 ou x=Ω)                                   → 0;
            P => λx.(((x≥0) et (x modulo 2 = 0)) ou (x=Ω))         → +&pair;
            P => λx.(((x≤0) et (x modulo 2 = 0)) ou (x=Ω))         → -&pair;
            P => λx.(((x≥0) et (x modulo 2 = 1)) ou (x=Ω))         → +&impair;
            P => λx.(((x≤0) et (x modulo 2 = 1)) ou (x=Ω))         → -&impair;
            P => λx.((x modulo 2 = 0) ou (x=Ω))                    → pair;
            P => λx.(x≥0 ou x=Ω)                                   → +;
            P => λx.(x≤0 ou x=Ω)                                   → -;
            P => λx.((x modulo 2 = 1) ou (x=Ω))                    → impair;
            P => λx.vrai                                           → T;
          fincas;
```

L'intérêt de cette dernière combinaison des deux analyses est qu'elle donne de meilleurs résultats que chacune des deux analyses séparées. Considérons par exemple le programme suivant (Manna[1974],p.179) qui calcule $y^3 = x_1^{x_2}$ (avec la convention $0^0=1$) pour tout entier x_1 et tout nombre naturel x_2:

```
{1}   <y1,y2,y3> := <x1,x2,1>;
{2}   jusqu'à y2=0 faire
{3}      si impair(y2) alors
{4}         <y2,y3> := <y2-1,y1*y3>
{5}      sinon
{6}         <y1,y2> := <y1*y1,y2/2>
{7}      finsi;
{8}   refaire;
```

En appliquant seulement l'analyse des signes (correspondant à la fermeture ρ_1), nous obtenons pour la variable y_2 :

	{1}	{2}	{3}	{4}	{5}	{6}	{7}	{8}
y_2	⊥	T	T	T	T	T	T	0

En appliquant seulement l'analyse de parité (correspondant à la fermeture ρ_2), nous obtenons:

	{1}	{2}	{3}	{4}	{5}	{6}	{7}	{8}
y_2	T	T	i	p	p	T	T	p

L'intersection des résultats de ces deux analyses indépendantes n'apporte aucun renseignement nouveau sur le signe de y_2. Par contre la combinaison de ces analyses sous forme d'une analyse correspondant à la fermeture $\rho_1 \sqcap \rho_2$ permet de tenir compte de la règle $(\dot{+}\&i)-1=(\dot{+}\&p)$ ce qui donne une meilleure information sur le signe de y_2 :

	{1}	{2}	{3}	{4}	{5}	{6}	{7}	{8}
y_2	$\dot{+}$	$\dot{+}$	$\dot{+}\&i$	$\dot{+}\&p$	$\dot{+}\&p$	$\dot{+}$	$\dot{+}$	0

En conclusion de cet exemple nous retiendrons que lorsque l'on veut analyser des propriétés différentes des programmes (c'est-à-dire correspondant à des fermetures ρ_1 et ρ_2 non comparables) on a toujours intérêt (aussi bien sur le plan de la rapidité des calculs que sur le plan de la précision

des résultats) à les combiner en suivant les directives du paragraphe 4.2.3. En toute généralité, le théorème 4.2.8.0.5 montre que l'analyse correspondant à $\rho_1 \sqcap \rho_2$ donne un résultat meilleur que l'intersection des analyses indépendantes correspondant à ρ_1 et ρ_2.

5.5 TECHNIQUES CLASSIQUES D'OPTIMISATION DES PROGRAMMES

De très nombreuses techniques d'optimisation de programmes comme la détermination des variables vivantes en chaque point d'un programme sont purement syntaxiques. On détermine des propriétés des programmes qui sont indépendantes des valeurs des variables. D'autres comme la propagation des constantes sont sémantiques. Dans tous les cas elles se ramènent à la résolution d'un système d'équations associé au programme.

5.5.1 Techniques booléennes d'optimisation des programmes

5.5.1.1 Variables vivantes d'un programme

Une variable est "vivante" en un point d'un programme (Aho & Ullman[1977] Hecht & Ullman[1973], Kennedy[1971], Kennedy[1976], Ullman[1975]) si sa valeur est utilisée sur un chemin d'exécution du programme partant de ce point. (Par exemple dans début x:=1; y:=2; {1} y:=x+1 fin, x est vivante au point {1} tandis que y ne l'est pas. Cette information est évidemment utile pour l'allocation de registres). Soit pour chaque noeud b du programme l'ensemble utilisé(b) des variables dont la valeur est utilisée dans ce noeud et transparent(b) l'ensemble des variables dont la valeur n'est pas modifiée dans ce noeud. Alors on obtient l'ensemble vivante(b) des variables vivantes à l'entrée du noeud b du programme en calculant la plus petite solution (pour l'inclusion $\subset$) du système d'équations à point fixe associé au programme par la règle en arrière:

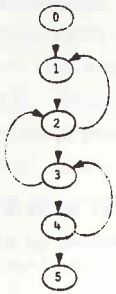
$$\text{vivante}(b) = \text{utilisé}(b) \cup \bigcup_{x \in \text{succ}(b)} (\text{transparent}(b) \cap \text{vivante}(x))$$

alors que pour les noeuds de sortie on a:

vivante(b) = ∅

(Une variable est vivante à l'entrée d'un bloc b si elle est utilisée dans ce bloc ou si sa valeur n'est pas modifiée dans le bloc b et qu'elle est utilisée sur un chemin d'exécution du programme partant de la sortie de ce bloc c'est-à-dire vivante à la sortie de ce bloc).

Considérons comme exemple un programme comportant deux variables α et β et pour lequel les vecteurs utilisé et transparent déterminés syntaxiquement sont supposés être:



noeud	0	1	2	3	4
utilisé	∅	{β}	∅	∅	{α}
transparent	∅	{α,β}	{α,β}	{α,β}	{α}

La plus petite solution du système d'équations établi ci-dessus est:

noeud	0	1	2	3	4	5
vivante	∅	{α,β}	{α,β}	{α,β}	{α}	∅

Dans le calcul itératif on initialise vivante avec utilisé qui est inférieur au plus petit point fixe, en machine on utlise une représentation de ces ensembles par des vecteurs booléens dont la longueur est le nombre de variables du programme.

5.5.1.2 Expressions disponibles

Une expression est disponible en un point d'un programme (Cocke[1970], Hecht & Ullman[1973], Morel & Renvoise[1974], Schaefer[1973], Ullman[1974], Urschler[1974]) si la valeur de l'expression a été calculée précédemment et depuis le dernier calcul de cette expression aucun argument de cette expression n'a eu sa valeur modifiée. Si la valeur d'une expression disponible en un point du programme a été sauvegardée elle n'a pas besoin d'être recalculée.

A chaque noeud b du programme on associe l'ensemble transparent(b) des

expressions élémentaires du programme dont la valeur d'aucun des arguments n'est modifiée dans ce noeud. Soit loc-disponible(b) l'ensemble des expressions élémentaires du programme qui sont localement disponibles au noeud b, c'est-à-dire évaluées dans ce noeud sans que ses arguments soient modifiés par la suite. Cette information est calculée en chaque point du programme par simple examen syntaxique. L'ensemble disponible(b) des expressions disponibles en sortie de chaque noeud du programme est alors donné par la plus grande solution (pour l'inclusion d'ensembles $\subseteq$) du système d'équations associé au programme par la règle en avant:

$$\text{disponible}(b) = \text{loc-disponible}(b) \cup \bigcap_{x \in \text{pred}(b)} (\text{transparent}(b) \cap \text{disponible}(x))$$

alors que pour les noeuds d'entrée on a:

$$\text{disponible}(b) = \emptyset$$

(Une expression est disponible à la sortie d'un bloc si sa valeur a été calculée dans le bloc ou si elle est disponible à l'entrée du bloc et aucun de ses arguments n'est modifié dans le bloc).

Par exemple dans le schéma de programme suivant:

```
1 :  début
2 :    I:=...; J:=...;
3 :    K:= I+J;
4 :    ...
5 :    si ... alors
6 :       I:=...;
7 :       K:=I+J;
          allera 4;
       sinon
8 :       ...
9 :       si ... K ... alors
             allera 10;
          finsi;
          allera 4;
       finsi;
10:  fin.
```

Nous aurons:

b	1	2	3	4	5	6	7	8	9
transparent(b)		ø	{I+J}	{I+J}	{I+J}	ø	{I+J}	{I+J}	{I+J}
loc-disponible(b)		ø	{I+J}	ø	ø	ø	{I+J}	ø	ø
disponible(b)	ø	ø	{I+J}	{I+J}	{I+J}	ø	{I+J}	{I+J}	{I+J}

5.5.2 Techniques non booléennes d'optimisation des programmes: exemple de la propagation des constantes

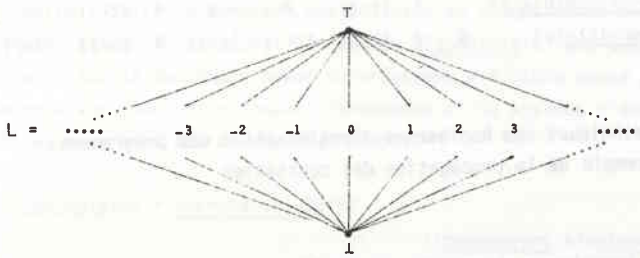
Les techniques classiques non booléennes sont assez rares, (Aho & Ullman[1977]) la plus connue étant certainement la "propagation des constantes" (Kam & Ullman[1976], Kam & Ullman[1977], Kildall[1973], Reif & Lewis[1977]). Elle est présentée comme une exécution symbolique du programme où les variables peuvent prendre éventuellement la valeur symbolique T (pour non-constant) quand, à une jonction de chemins d'exécution, une variable n'a pas la même valeur constante sur tous ces chemins. Ainsi donc l'exemple:

```
      a:=1; b:=2; c:=3; d:=3; e:=0;
{1}
      tantque ... faire
{2}
            b:=2*a; d:=d+1; e:=e-a;
{3}
            a:=b-a; c:=e+d;
{4}
      refaire;
```

l'exécution symbolique donne:

	a	b	c	d	e	
{1}	1	2	3	3	0	
{2}	1	2	3	3	0	
{3}	1	2	3	4	-1	
{4}	1	2	3	4	-1	
{2}	1	2	3	T	T	← Car les valeurs de d et e ont changé en ce point.
{3}	1	2	3	T	T	
{4}	1	2	T	T	T	← Car c est la somme de deux valeurs non constantes.
{2}	1	2	T	T	T	
{3}	1	2	T	T	T	
{4}	1	2	T	T	T	← L'exécution symbolique s'arrête quand les valeurs des variables en chaque point du programme ne changent plus.

On peut comprendre cette exécution symbolique comme une résolution chaotique d'un système d'équations approché construit en utilisant la fermeture supérieure ρ définie par (voir 5.2.1):



Sans détailler la détermination des règles de construction des équations approchées, le système associé au programme que nous avons donné en exemple est le suivant:

$$\begin{cases} P_1 = (a=1,\ b=2,\ c=3,\ d=3,\ e=0) \\ P_2 = P_1 \sqcup P_4 \\ P_3 = P_2(b \leftarrow 2*a,\ d \leftarrow d+1,\ e := e-a) \\ P_4 = P_3(a \leftarrow b-a,\ c \leftarrow e+d) \end{cases}$$

Le treillis $(L^5)^4$ est infini mais satisfait la condition de chaîne, nous pouvons résoudre le système d'équations approché par itérations en un nombre fini de pas. En choisissant une stratégie chaotique correspondant au graphe du programme nous obtenons l'explication classique en termes d'exécution symbolique qui donne:

$$\begin{cases} P_1 = (a=1,\ b=2,\ c=3,\ d=3,\ e=0) \\ P_2,\ P_3,\ P_4 = (a=1,\ b=2,\ c=\top,\ d=\top,\ e=\top) \end{cases}$$

Un compilateur peut optimiser le programme en remplaçant a et b par leurs valeurs 1 et 2 et en éliminant les variables a et b du programme objet. Noter que d'après l'approximation statique qui a été choisie on ne peut pas découvrir qu'à la ligne {3} on a e+d=3 et donc que c est une constante égale à 3 dans ce programme. On notera qu'une approximation moins grossière

mais plus coûteuse comme celle décrite au paragraphe 5.8 permet de découvrir automatiquement l'invariant de boucle {a=1, b=2, c=3, e≤0, d+e=3}.

On remarquera que l'on découvre en chaque point du programme des invariants de la forme {(a=i) ou (a=Ω)}. Si au point correspondant dans le programme objet la variable a est remplacée par la valeur i les programmes source et objet ne seront pas équivalents si a peut ne pas être initialisé en ce point du programme source. Pour exclure cé phénomène on pourrait choisir une fermeture du type:

```
λP.cas
      P =  λx.faux  → ⊥;
      P =  λx.(x=Ω) → Ω;
      P => λx.(x=i) → i;
      P => λx.vrai  → ⊤;
   fincas;
```

(ce qui suffit pour définir complètement la méthode d'analyse correspondante).

5.6 DECOUVERTE AUTOMATIQUE DU TYPE DES VARIABLES D'UN PROGRAMME

Dans les langages classiques le type d'une variable en un point d'un programme est généralement un sous-type du type déclaré (même si ce sous-type n'existe pas de façon explicite dans la définition du langage). Parexemple une variable de type global "pointeur sur un enregistrement de type R" peut être localement du sous-type "pointeur non-nul sur un enregistrement de type R". Cette information est très importante dans la phase de génération de code pour éviter par exemple d'avoir à tester dynamiquement que l'accès indirect à l'enregistrement est correct. De même dans d'autres langages comme APL (Iverson[1962]) ou SETL (Schwartz[1973]) il n'y a pas de déclarations et le type des objets manipulés par le programme doit être déterminé à l'exécution. L'implémentation du langage est alors interprétative et l'écriture d'un compilateur passe avant tout par la résolution du problème de la détermination statique du type des objets manipulés par les programmes.

5.6.1 Traitement des pointeurs

Dans des langages de programmation comme PASCAL (Jensen & Wirth[1976]) ou LIS (Ichbiah et al.[1974]) l'utilisation d'une variable de type pointeur pour accéder indirectement à un bloc de mémoire pose le problème de vérifier que le pointeur utilisé n'est pas nul. La plupart des compilateurs reportent cette vérification au moment de l'exécution des programmes. Le coût de ces tests à l'exécution est pratiquement nul quand ils sont mis en oeuvre à l'aide du mécanisme de protection mémoire. Outre que cette solution a l'inconvénient de découvrir très tardivement les fautes de programmation, elle ne peut pas être mise en oeuvre par un langage d'implémentation de systèmes comme LIS puisque certains programmes peuvent s'exécuter en mode maître. Dans ce cas la solution des tests à l'exécution est coûteuse et seule une analyse statique des programmes permet d'en réduire le coût.

Les pointeurs permettant un accès mémoire défini dynamiquement, ils posent à un compilateur le problème de savoir quels objets peuvent être référencés ou modifiés par leur intermédiaire. Illustrons le problème sur l'exemple suivant:

```
    type E = enregistrement A,B : entier fin;
    var P,Q : ref E;
    var x,y : entier;
    ...
{1} X := P.A+P.B;
{2} Q.A := 0;
{3} Y := P.A+P.B;
```

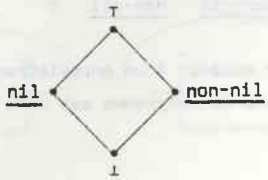
Si le compilateur peut démontrer statiquement que P et Q ne référencent pas le même enregistrement à la ligne {2}, alors l'expression P.A+P.B est disponible à la ligne {3} et n'a donc pas à être recalculée. Par contre si le compilateur est incapable de démontrer que P et Q pointent sur des enregistrements différents il doit supposer que l'affectation de la valeur 0 à Q.A peut également modifier P.A et le calcul de l'expression P.A+P.B doit être refait à la ligne {3}. Pour résoudre ce problème nous proposons au paragraphe 5.6.1.2 une analyse permettant de découvrir une partition des variables pointeurs en groupes tels que deux variables dans des groupes distincts ne puissent pas repérer indirectement le même objet.

(On notera que le problème n'est pas spécifique à l'emploi de pointeurs puisqu'il se retrouve aussi bien avec les indices de tableaux qui permettent

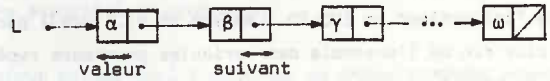
également de faire des accès mémoire dynamiques. Dans ce cas le problème est plus complexe (voir 5.8) puisque l'emploi de tableaux permet des calculs explicites d'adresses qui, du moins dans les langages de haut niveau, sont interdits avec les pointeurs).

5.6.1.1 Pointeurs nuls et non nuls

L'exemple est comparable aux applications 5.2 et 5.3 et n'est donc pas traité en détail. Nous choisissons comme image approchée des propriétés d'un pointeur le treillis:



Le programme suivant (Jensen & Wirth[1974]) cherche l'entier n dans une liste linéaire chaînée d'entiers:




```
    pt:=L; b:=vrai;
{1}
    tantque (pt≠nil et b) faire
{2}
        si pt↑.valeur=n alors
            b:=faux;
{3}
        sinon
{4}
            pt:=pt↑.suivant;
{5}
        finsi;
    refaire;
```

Le système d'équations associé est:

$$\begin{cases} P_1 = \tau & \text{(L peut être une liste vide ou non)} \\ P_2 = (P_1 \sqcup P_5) \sqcap \underline{\text{non-nil}} \\ P_3 = P_2 \\ P_4 = P_2 \\ P_5 = P_4 \sqcup \tau \end{cases}$$

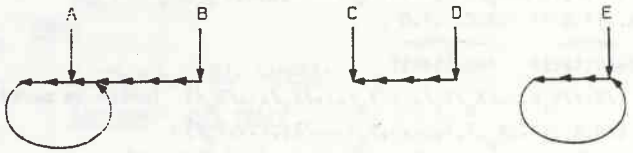
la solution est:

	{1}	{2}	{3}	{4}	{5}
pt	τ	non-nil	non-nil	non-nil	τ

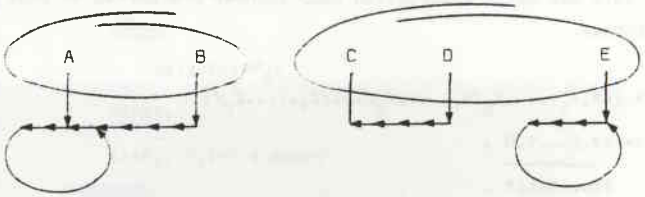
Quand pt est utilisé pour accéder à un enregistrement il n'est pas nil et l'emploi des pointeurs dans ce programme est donc correct.

5.6.1.2 Pointeurs repérant des enregistrements distincts

La seule réponse offerte dans les langages de programmation classiques à la question: "quels sont les objets qui peuvent être désignés par une variable pointeur?" est qu'un pointeur donné ne peut repérer que des enregistrements du même type. La notion de "domaine" en LIS (Ichbiah et al.[1974] reprise par celle de "collection" en EUCLID (Lampson et al.[1976]) permet un partitionnement plus fin de l'ensemble des variables pointeurs repérant des enregistrements de même type. Le progrès n'est pas suffisant dans la mesure où les déclarations globales ne peuvent pas être détaillées pour décrire la situation particulière en chaque point du programme. Nous proposons donc de construire en chaque point du programme une partition de l'ensemble des variables pointeurs repérant des enregistrements d'un type donné. Deux variables pointeurs seront équivalentes si elles peuvent désigner le même enregistrement. Deux variables pointeurs appartenant à des classes d'équivalence distinctes ne pourront pas repérer, même indirectement, le même objet. Par exemple une partition permettant de décrire la situation:



est:



que l'on note /A,B/C,D,E/.

Nous esquissons brièvement les règles de construction du système d'équations associé à un programme:

Pointeur nul ou seul à désigner un enregistrement nouvellement alloué:

$P = \{/X,X_1,\ldots,X_p/Y_1,\ldots,Y_q/\ldots/Z_1,\ldots,Z_r/\}$ (partition avant l'instruction)
X:=nil;
X:=Y; (où Y est nil par 5.6.1.1)
si X=nil alors ...
allouer(X);
$\varepsilon(X,P) = \{/X/X_1,\ldots,X_p/Y_1,\ldots,Y_q/\ldots/Z_1,\ldots,Z_r\}$(partition après l'instruction)

Affectation de pointeurs:

$P = \{/X,X_1,\ldots,X_p/Y,Y_1,\ldots,Y_q/\ldots/Z_1,\ldots,Z_r/\}$

$X\uparrow.A\ldots\uparrow.B := Y\uparrow.C\ldots\uparrow.D$;

facultatif facultatif

$P \sqcup \{/X,Y/X_1/\ldots/X_p/Y_1/\ldots/Y_q/\ldots/Z_1/\ldots/Z_r/\}$ (union de partitions)

$= \{/X,X_1,\ldots,X_p,Y,Y_1,\ldots,Y_q/\ldots/Z_1,\ldots,Z_r/\}$

(X et Y repérant indirectement le même objet, ils sont mis dans la même partie. Ne connaissant pas l'organisation exacte des données, nous ignorons le fait que cette instruction peut scinder une partie en deux parties disjointes).

$P = \{/X,X_1,\ldots,X_p/Y_1,\ldots,Y_q/\ldots/Z_1,\ldots,Z_r/\}$

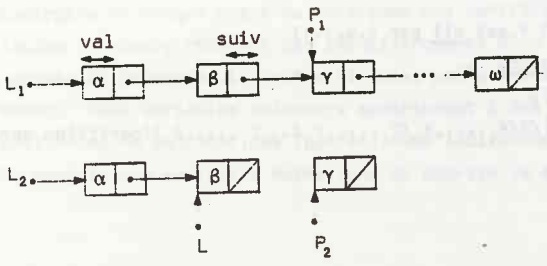
$X := Y\uparrow.C\ldots\uparrow.D$;

facultatif

$\varepsilon(X,P) \sqcup \{/X,Y/X_1/\ldots/Z_r/\} = \{/X_1,\ldots,X_p/X,Y,Y_1,\ldots,Y_q/\ldots/Z_1,\ldots,Z_r/\}$

(Après l'affectation, X ne peut pas repérer, même indirectement, un objet repéré par $X_1,\ldots,X_p$).

Comme exemple, considérons le programme suivant pour recopier une liste linéaire chaînée:



```
        procédure copie(L1:liste ; var L2:liste);
            var P1,P2,L : liste;
        début
{1}         P1:=L1; L2:=nil; L:=nil;
{2}         tantque P1≠nil faire
{3}             allouer(P2); P2↑.val:=P1↑.val; P2↑.suiv:=nil;
{4}             si L=nil alors
{5}                 L2:=P2;
{6}             sinon
{7}                 L↑.suiv:=P2;
{8}             finsi;
{9}             L:=P2; P1:=P1↑.suiv;
{10}        refaire;
{11}    fin;
```

Le système d'équations approchées correspondant est le suivant:

$$
\begin{cases}
S_1 = /L_1,L_2/P_1,P_2,L/ \\
S_2 = \varepsilon(L,\varepsilon(L_2,\varepsilon(P_1,S_1) \sqcup \{/P_1,L_1/L/L_2/P_2/\})) \\
S_3 = S_2 \sqcup S_{10} \\
S_4 = \varepsilon(P_2,S_3) \\
S_5 = \varepsilon(L,S_4) \\
S_6 = \varepsilon(L_2,S_5) \sqcup \{/L_2,P_2/L/L_1/P_1/\} \\
S_7 = S_4 \\
S_8 = S_7 \sqcup \{/L,P_2/L_1/L_2/P_1/\} \\
S_9 = S_6 \sqcup S_8 \\
S_{10} = \varepsilon(L,S_9) \sqcup \{/L,P_2/L_1/L_2/P_1/\} \\
S_{11} = \varepsilon(P_1,S_2 \sqcup S_{10})
\end{cases}
$$

A la ligne {1} les paramètres et les variables locales sont dans des parties disjointes. Aux lignes {3} et {7} les tests $P_1 \neq$ nil et $L \neq$ nil n'apportent aucune information sur la partition des variables pointeurs. A la ligne {4} l'affectation $P_2\uparrow.val:=P_1\uparrow.val$ d'une valeur qui n'est pas un pointeur et la modification profonde $P_2\uparrow.suiv:=$nil dans la partie de P_2 sont ignorées.

A la ligne {10} $P_1 := P_1\uparrow.suiv$ est ignoré car cette instruction ne peut pas faire repérer indirectement par P_1 un enregistrement auquel P_1 ne permettait pas déjà un accès indirect.

Le nombre de variables pointeurs étant fini, le treillis des partitions de l'ensemble de leurs noms est fini et le système peut être résolu par itération en un nombre fini de pas en partant de l'infimum $\{/L_1/L_2/L/P_1/P_2/\}$. Nous obtenons:

$$\begin{cases} S_1 = \{/L_1, L_2/P_1, P_2, L/\} \\ S_2 = \{/L_1, P_1/L_2/P_2/L/\} \\ S_3 = \{/L_1, P_1/L_2, P_2, L/\} \\ S_4 = \{/L_1, P_1/L_2, L/P_2/\} \\ S_5 = \{/L_1, P_1/L_2/P_2/L/\} \\ S_6 = \{/L_1, P_1/L_2, P_2/L/\} \\ S_7 = \{/L_1, P_1/L_2, L/P_2/\} \\ S_8 = \{/L_1, P_1/L_2, L, P_2/\} \\ S_9 = \{/L_1, P_1/L_2, P_2, L/\} \\ S_{10} = \{/L_1, P_1/L_2, P_2, L/\} \\ S_{11} = \{/L_1/P_1/L_2, P_2, L/\} \end{cases}$$

On a donc démontré automatiquement et par un moyen simple que si L_1 et L_2 peuvent repérer le même enregistrement avant l'appel de procédure *copie*(L_1,L_2) alors, après cet appel, ils ne permettent pas de repérer, même indirectement, le même objet.

On trouvera de plus amples détails sur des applications au traitement des pointeurs (notamment dans le cas d'enregistrements avec variantes) dans Cousot & Cousot[1977b]. On notera qu'une telle information peut aussi être utile au ramassage des miettes (surtout quand il est explicite, auquel cas il faut vérifier que la mémoire libérée n'est pas référencée).

5.6.2 Découverte du type des objets d'un programme dans un langage de très haut niveau sans déclarations

Nous choisissons un exemple trop simple pour illustrer la puissance d'un langage comme SETL (Schwartz[1973]) mais qui convient bien pour montrer

l'application des résultats du paragraphe 3.6.

Soit un langage avec les types de base entiers (ent), réels (reel), chaînes de caractères (car) et sans déclarations. Considérons le programme suivant:

```
{1}
        s:=0;
{2}
    L:
{3}
        lire(x,y);
{4}
        si x≥0 alors
{5}
           x:=(x modulo 2)+y;
{6}
           s:=s+x;
{7}
        finsi;
{8}
        si (y modulo x) ≠ 0 alors
{9}
            allera L;
        finsi;
{10}
```

Dans ce programme le type d'une variable peut être n'importe quel élément de l'ensemble L des sous-ensembles de $T=\{ent,reel,car\}$ avec l'interprétation qu'à l'exécution un objet de type $t\subseteq T$ doit être d'un des types élémentaires appartenant à t. Les opérateurs du langage sont polymorphes mais ne sont pas toujours définis pour tous les types de leurs arguments. La procédure lire est définie pour tous les types de ses arguments. La comparaison x≥0 n'est définie que si x a une valeur numérique. L'opération x modulo y n'est définie que si x et y sont des entiers. L'opération + est définie pour tous les types de ses arguments, elle représente la concaténation de chaînes de caractères ou la concaténation d'une chaîne et de la conversion en chaîne d'une valeur numérique ou la somme de deux valeurs numériques. Si x est de type α et y de type β alors x+y est du type $\alpha\tilde{+}\beta$ défini par:

$$\bar{+} = \lambda(\alpha,\beta).\underline{cas}\ \alpha,\beta\ \underline{dans}$$
$$\underline{car}\ ,\ ? \rightarrow \underline{car};$$
$$?\ ,\ \underline{car} \rightarrow \underline{car};$$
$$\underline{reel}\ ,\ ? \rightarrow \underline{reel};$$
$$?\ ,\ \underline{reel} \rightarrow \underline{reel};$$
$$?\ ,\ ? \rightarrow \underline{ent};$$
$$\underline{fincas};$$

5.6.2.1 Système approché d'équations en avant

Le système d'équations approché en avant $P = \overline{F}(\overline{\phi})(P)$ où $\overline{F}(\overline{\phi}) \in mon((L^3)^{10} \rightarrow (L^3)^{10})$ associé au programme est le suivant :

$$\begin{cases} P_1 = \overline{\phi} \\ P_2 = P_1(s \leftarrow \{\underline{ent}\}) \\ P_3 = P_2 \cup P_9 \\ P_4 = P_3(x \leftarrow \tau, y \leftarrow \tau) \\ P_5 = P_4(x \leftarrow x \cap \{\underline{ent},\underline{reel}\}) \\ P_6 = P_5(x \leftarrow \{\underline{ent}\} \bar{+} y) \\ P_7 = P_6(s \leftarrow s \bar{+} x) \\ P_8 = P_7 \cup P_4(x \leftarrow x \cap \{\underline{ent},\underline{reel}\}) \\ P_9 = P_8(x \leftarrow x \cap \{\underline{ent}\}, y \leftarrow y \cap \{\underline{ent}\}) \\ P_{10} = P_8(x \leftarrow x \cap \{\underline{ent}\}, y \leftarrow y \cap \{\underline{ent}\}) \end{cases}$$

Pour établir P_4 nous tenons compte du fait que lire permet d'affecter à x et y des valeurs d'un type élémentaire quelconque. Pour P_5 et P_8 nous prenons en considération le fait que le test $x \geq 0$ n'est défini que pour des valeurs numériques de x et y. De même pour P_{10} et P_9 le test (y modulo x)$\neq 0$ impose x et y entiers sinon le programme s'est terminé par une erreur. Enfin $\bar{\bar{+}} = \lambda(t_1,t_2).\{\alpha \bar{+} \beta : \alpha \in t_1 \text{ et } \alpha \in t_2\}$ donne l'ensemble des types possibles du résultat de l'opérateur + quand on connait l'ensemble des types possibles de ses paramètres.

5.6.2.2 Système approché d'équations en arrière

Le système d'équations approché en arrière $P = \overline{B}(\overline{\Psi})(P)$ où $\overline{B}(\overline{\Psi}) \in mon((L^3)^{10} \rightarrow (L^3)^{10})$ associé au programme est le suivant :

$$
\begin{cases}
P_1 = P_2(s\leftarrow\tau) \\
P_2 = P_3 \\
P_3 = P_4(x\leftarrow\tau, y\leftarrow\tau) \\
P_4 = (P_5 \cup P_8)(x\leftarrow x\underline{n}\{\underline{ent}, \underline{reel}\}) \\
P_5 = P_6(x\leftarrow x\underline{n}\underline{ent}, y\leftarrow y\underline{n}\underline{plus2}(x, \underline{ent}, y) \\
P_6 = P_7(x\leftarrow x\underline{n}\underline{plus2}(s,\tau,x), s\leftarrow s\underline{n}\underline{plus1}(s,\tau,x)) \\
P_7 = P_8 \\
P_8 = (P_9 \cup P_{10})(x\leftarrow x\underline{n}\{\underline{ent}\}, y\leftarrow y\underline{n}\{\underline{ent}\}) \\
P_9 = P_3 \\
P_{10} = \overline{\Psi}
\end{cases}
$$

Pour établir P_4 et P_8 nous tenons compte du fait que les tests doivent être définis. Pour P_6 nous prenons en considération le fait que (x modulo 2) n'est défini que si x est entier, auquel cas le résultat est un entier. De plus nous utilisons:

$$\underline{plus1}(t_0,t_1,t_2) = \{\beta\epsilon t_1 : \{\exists\alpha\epsilon t_0, \exists\gamma\epsilon t_2 : \alpha=\beta+\gamma\}\}$$
$$\underline{plus2}(t_0,t_1,t_2) = \{\gamma\epsilon t_2 : \{\exists\alpha\epsilon t_0, \exists\beta\epsilon t_1 : \alpha=\beta+\gamma\}\}$$

En effet, connaissant les types possibles t_0 du résultat et les types possibles t_1, ..., t_n des n arguments avant que l'opération $f(x_1,\ldots,x_n)$ ait été évaluée (qui est le même que le type après que l'opération ait été évaluée par les arguments non modifiés dans l'instruction) alors on sait qu'un certain nombre seulement de combinaisons n'étaient possibles avant d'évaluer cette opération pour que le type du résultat soit du type donné. Par exemple pour que 1+y soit une chaîne de caractères il faut que y soit du type $\underline{plus2}(\underline{car}, \underline{ent}, \tau) = \underline{car}$.

5.6.2.3 Principe de la méthode de résolution

Soient F_π et B_π les systèmes d'équations sémantiques associées au programme π ci-dessus et ϕ et Ψ les spécifications d'entrée et de sortie. (Dans l'exemple, nous prendrons $\phi=\lambda(x,y,s).(x=y=s=\Omega)$ et $\Psi=\lambda(x,y,s).\underline{vrai}$). Nous voulons approcher supérieurement $\mathit{lfp}(F_\pi(\phi))$ et $\mathit{lfp}(B_\pi(\Psi))$ c'est-à-dire déterminer en chaque point du programme π un sur-ensemble des valeurs des variables que l'on peut obtenir sur un chemin d'exécution quelconque du programme partant du point d'entrée dans un état satisfaisant les conditions d'entrée ϕ et arrivant à ce point (proposition 3.3.0.2) et qui ensuite se

termine dans un état satisfaisant la spécification de sortie Ψ (proposition 3.5.0.2). Nous disposons de $\overline{\phi} \supseteq \phi$, $\overline{\Psi} \supseteq \phi$, $\overline{F}(\overline{\phi}) \supseteq F_\pi(\phi)$ et $\overline{B}(\overline{\Psi}) \supseteq B_\pi(\phi)$.

Nous proposons de calculer la limite de la suite décroissante $P^1 = \mathcal{lfp}(\overline{F}(\overline{\phi}))$, $P^2 = \mathcal{lfp}(\lambda X.P^1 \underline{\text{et}}\ \overline{B}(\overline{\Psi})(X))$, ..., $P^{2k+1} = \mathcal{lfp}(\lambda X.P^{2k} \underline{\text{et}}\ \overline{F}(\overline{\phi})(X))$, $P^{2k+2} = \mathcal{lfp}(\lambda X.P^{2k+1} \underline{\text{et}}\ \overline{B}(\overline{\Psi})(X))$,... . D'après le théorème 4.3.2.0.5 cette suite est décroissante et comme dans notre cas l'espace des propriétés approchées $(L^3)^{10}$ satisfait la condition de chaîne descendante, cette suite est stationnaire à partir d'un rang fini. Pour montrer que ce choix est correct il suffit donc de montrer que chaque terme est plus grand que $(\mathcal{lfp}(F_\pi(\phi)) \underline{\text{et}}\ \mathcal{lfp}(B_\pi(\Psi)))$:

On a $P_1 = \mathcal{lfp}(\overline{F}(\overline{\phi})) \supseteq \mathcal{lfp}(F_\pi(\phi)) \supseteq (\mathcal{lfp}(F_\pi(\phi)) \underline{\text{et}}\ \mathcal{lfp}(B_\pi(\phi)))$. Comme $\lambda X.[P_1 \underline{\text{et}}\ \overline{B}(\overline{\Psi})(X)] \supseteq \lambda X.[\mathcal{lfp}(F_\pi(\phi)) \underline{\text{et}}\ \exists_\pi(\Psi)(X)]$ on en déduit par monotonie que $P_2 = \mathcal{lfp}(\lambda X.[P_1 \underline{\text{et}}\ \overline{B}(\overline{\Psi})(X)]) \supseteq \mathcal{lfp}(\lambda X.[\mathcal{lfp}(F_\pi(\phi)) \underline{\text{et}}\ B_\pi(\Psi)(X)])$ qui est égal d'après 3.7.0.1.(c) à $(\mathcal{lfp}(F_\pi(\phi)) \underline{\text{et}}\ \mathcal{lfp}(B_\pi(\Psi)))$. L'étape d'induction est la même en utilisant la proposition 3.7.0.1.(e)-(f).

5.6.2.4 Exemple

Pour notre exemple nous obtenons:

$\overline{\phi}$ = ((x=∅),(y=∅),(s=∅))

$\overline{\Psi}$ = ((x=T),(y=T),(s=T))

$P^1 = \mathcal{lfp}(\overline{F}(\overline{\phi}))$

	1	2	3	4	5	6	7	8	9	10
x	∅	∅	{ent}	T	{ent,reel}	T	T	{ent,reel}	{ent}	{ent}
y	∅	∅	{ent}	T	T	T	T	T	{ent}	{ent}
s	∅	{ent}	T	T	T	T	T	T	T	T

$P^2 = \mathcal{lfp}(\lambda X.P^1 \cap \overline{B}(\overline{\Psi}))$

	1	2	3	4	5	6	7	8	9	10
x	∅	∅	{ent}	{ent}	{ent}	{ent}	{ent}	{ent}	{ent}	{ent}
y	∅	∅	{ent}	{ent}	{ent}	{ent}	{ent}	{ent}	{ent}	{ent}
s	∅	{ent}	T	{ent}	T	T	T	T	T	T

$P^3 = lfp(\lambda X.P^2 \cap \overline{F}(\overline{\phi}))$

	1	2	3	4	5	6	7	8	9	10
x	ø	ø	{ent}	{ent}	{ent}	{ent}	{ent}	{ent}	{ent}	{ent}
y	ø	ø	{ent}	{ent}	{ent}	{ent}	{ent}	{ent}	{ent}	{ent}
s	ø	{ent}	{ent}	{ent}	{ent}	{ent}	{ent}	{ent}	{ent}	{ent}

Enfin $P^4 = lfp(\lambda X.P^3 \cap \overline{B}(\overline{\Psi}))$ donne le même résultat ce qui montre que x,y et s doivent être déclarés entiers si l'on ne veut pas que le programme ne se termine pas ou conduise à une erreur. Pour expliquer intuitivement ce résultat on remarque que si la valeur de x et y n'est pas entière, on doit toujours évaluer le test (y modulo x)≠0 ce qui conduit à une erreur pour des arguments réels ou chaînes de caractères. Comme x et y doivent être entiers, on voit que s l'est également puisqu'il est initialisé et incrémenté par une valeur entière. Ces deux étapes dans le raisonnement sont bien représentées par P^2 et P^3.

5.7 TECHNIQUES D'APPROXIMATION APPLICABLES AU CAS D'UN ESPACE DE PROPRIETES APPROCHEES INFINI: EXEMPLE DE LA DECOUVERTE AUTOMATIQUE D'UN INTERVALLE DE VALEURS DES VARIABLES NUMERIQUES D'UN PROGRAMME

Dans un programme PASCAL (Jensen & Wirth[1976]) on peut déclarer qu'un entier est compris entre deux bornes numériques. Outre l'avantage de déclarations plus précises, le compilateur peut faire une allocation de mémoire plus efficace (notamment pour les tableaux de mots, demi-mots et octets qu'il est très facile de gérer sur la plupart des machines). L'inconvénient est que le coût de la vérification de cohérence des programmes à l'exécution est prohibitif puisque le matériel n'offre généralement pas de dispositifs câblés pour tester les débordements de capacité dans les arithmétiques ne correspondant pas aux mots ou doubles-mots.

Pour mettre en évidence le côut des vérifications dynamiques, nous rappelons les chiffres (sur l'ordinateur CDC 6600, système d'exploitation SCOPE 3.4) donnés par Wirth[1976] à propos des programmes suivants: (1) Calcul de 2^k et $1/2^k$ pour k=1..90, (2) Trouver tous les nombres entre 1 et 1000 dont le carré est un palindrome, (3) Quicksort (Hoare[1962]), (4) Problème des 8 reines (Wirth[1971]) et (5) Calcul des 1000 premiers nombres premiers. Pour chaque programme nous donnons "n", le nombre de lignes PASCAL

du programme, "a" le nombre d'accès à des tableaux dans le texte du programme et "t_a", "t_s" les temps d'exécution en msec du programme respectivement avec et sans vérification dynamique des index de tableaux :

Programme	n	a	t_a	t_s
(1)	34	9	916	813
(2)	16	2	3466	2695
(3)	26	7	4098	2861
(4)	34	15	1017	679
(5)	30	8	1347	1061

A cause de ces gains de temps (en moyenne de l'ordre de 20%, mais parfois beaucoup plus considérables), les programmeurs utilisent presque toujours l'option de compilation permettant d'éliminer les tests à l'exécution. Les conséquences sont parfois surprenantes comme l'illustre l'exemple suivant:

```
début
   x : 0..255;
   x:=1;
   tantque x≠0 faire
      x:=x+1;
   refaire;
fin.
```

Avec un compilateur qui n'engendre pas de test pour vérifier que dans le corps de la boucle la valeur de x doit être comprise entre 0 et 255, voici ce que nous avons observé: x étant déclaré de type 0..255, le compilateur lui alloue un espace mémoire d'un octet. A l'exécution x est normalement incrémenté de 1 jusqu'à atteindre la valeur 255. A ce moment dans l'évaluation de x:=x+1 la valeur 255 de x est chargée dans un registre. La valeur du registre est incrémentée de 1 ce qui donne 256 et ne conduit pas à un débordement arithmétique dans un registre de 4 octets. Ensuite le dernier octet de ce registre qui est donc nul est recopié dans x. Comme x a atteint la valeur zéro, l'exécution de la boucle se termine normalement alors que le programme est manifestement incorrect.

Un tel exemple montre bien que la vérification des déclarations d'intervalles est nécessaire. La solution des tests à l'exécution étant très coûteuse (principalement pour les programmes éxécutés de nombreuses fois) il faudrait pouvoir éliminer la plupart des tests inutiles par une analyse assez fine à la compilation. Nous développons maintenant un modèle utile pour trai-

ter ce problème dans le cas de langages de programmation comme PASCAL où les déclarations de variables entières et tableaux comportent des bornes numériques.

5.7.1 Espace des propriétés approchées

Soit Z l'ensemble des entiers positifs ou négatifs ordonné par l'ordre naturel $\leq$ et $Z^* = Z \cup \{-\infty, +\infty\}$ où $-\infty \leq -\infty \leq i \leq +\infty \leq +\infty$ pour tout i de Z. Soit L le treillis complet dont l'infimum est $\perp$ et dont les autres éléments sont les couples $[a,b]$ où $a,b \in Z^*$ et $a \leq b$. L'ordre partiel $\sqsubseteq$ sur L est défini par $\perp \sqsubseteq x$ pour tout x de L et $\{[a,b] \sqsubseteq [c,d]\} \Leftrightarrow \{c \leq a \leq b \leq d\}$. L'union est définie par $[a,b] \sqcup [c,d] = [\min(a,c), \max(b,d)]$, son élément neutre est $\perp$ et l'élément absorbant est $\top = [-\infty, +\infty]$. L'intersection est définie par $[a,b] \sqcap [c,d] =$ **si** $\max(a,c) \leq \min(b,d)$ **alors** $[\max(a,c), \min(b,d)]$ **sinon** $\perp$ **finsi**, son élément neutre est $\top$ et l'élément absorbant est $\perp$. Nous définissons une fonction de concrétisation (4.2.7) comme suit:

$$\gamma \in L \to (2^{Z \cup \{\Omega\}} \to \{\underline{vrai}, \underline{faux}\}) = P_1$$

$$= \lambda x.\ \underline{cas}\ x\ \underline{dans}$$
$$\perp \to \lambda x.(x = \Omega);$$
$$[a,b] \to \lambda x.((a \leq x \leq b)\ \underline{ou}\ (x = \Omega));$$
$$\underline{fincas};$$

Soit n le nombre de variables d'un programme, alors L^n est une image approchée supérieurement de $P_n = (2^{Z \cup \{\Omega\}})^n \to \{\underline{vrai}, \underline{faux}\}$ par la fonction injective de concrétisation :

$$\overline{\gamma} = \lambda(v_1, \ldots, v_n).(\lambda(x_1, \ldots, x_n).(\underset{j=1}{\overset{n}{\underline{ET}}}\ \gamma(v_j)(x_j)))$$

La fonction d'abstraction correspondante est définie par:

$$@ \in \mathcal{P}_1 \to L$$
$$= \lambda P.\ \underline{si}\ P \Rightarrow \lambda x.(x=\Omega)\ \underline{alors}\ \perp\ \underline{sinon}\ [\underline{min}\{x\in Z : P(x)\}, \underline{max}\{x\in Z : P(x)\}]\ \underline{finsi};$$

Noter que cette définition de @ est correcte puisque $\underline{non}(P \Rightarrow \lambda x.(x=\Omega))$ implique $\{\exists x\in Z: P(x)\}$ et que $\underline{min}(Z)=-\infty$ et $\underline{max}(Z)=+\infty$. De plus, @ est surjectif.

Pour tout j = 1,...,n, nous définissons:

$$\sigma_j^n \in \mathcal{P}_n \to \mathcal{P}_1$$
$$= \lambda P.[\lambda x.\{\exists (v_1,\ldots,v_{j-1},v_{j+1},\ldots,v_n)\in(Z\cup\{\Omega\})^n : P(v_1,\ldots,v_{j-1},x,v_{j+1},\ldots,v_n)\}]$$

$$\overline{@} \in \mathcal{P}_n \to L^n$$
$$= \lambda P.(@(\sigma_1^n(P)), @(\sigma_2^n(P)),\ldots,@(\sigma_n^n(P)))$$

Montrons que $(\overline{\gamma},\overline{@})$ est une paire de fonctions adjointes supérieures (définition 4.2.7.0.1).

. Pour montrer que la concrétisation $\overline{\gamma}$ et l'abstraction $\overline{@}$ sont monotones il suffit de montrer que γ et @ le sont, ce qui est facile car $([a,b]\sqsubseteq[c,d]) \Rightarrow (c\le a\le b\le d) \Rightarrow (\lambda x.((a\le x\le b)\ \underline{ou}\ (x=\Omega))\Rightarrow\lambda x.((c\le x\le d)\ \underline{ou}\ (x=\Omega)))$ et d'autre part $\underline{non}(P\Rightarrow\lambda x.(x=\Omega))$ et $\underline{non}(Q\Rightarrow\lambda x.(x=\Omega))$ et $(P\Rightarrow Q)$ impliquent $\underline{min}\{x\in Z:Q(x)\} \le \underline{min}\{x\in Z:P(x)\} \le \underline{max}\{x\in Z:P(x)\} \le \underline{max}\{x\in Z:P(x)\}$.

. Montrons maintenant que $\forall Q\in\mathcal{P}_1$, $Q\Rightarrow\gamma\circ@(Q)$.

- Si $Q=\lambda x.\underline{faux}$ ou $Q=\lambda x.(x=\Omega)$ alors $\gamma\circ@(Q) = \gamma(\perp) = \lambda x.(x=\Omega)$ auquel cas $Q\Rightarrow\lambda x.(x=\Omega)$.
- Sinon on a $\underline{non}(Q\Rightarrow\lambda x.(x=\Omega))$ et
$\gamma\circ@(Q) = \gamma[\underline{min}\{x\in Z:Q(x)\},\underline{max}\{x\in Z:Q(x)\}]$
$= \lambda z.((\underline{min}\{x\in Z:Q(x)\}\le z\le\underline{max}\{x\in Z:Q(x)\})\ \underline{ou}\ (z=\Omega))$
et donc $\forall z\in(Z\cup\{\Omega\})$ on a $Q(z)\Rightarrow\gamma\circ@(Q)(z)$.

. Montrons maintenant que $\forall v \in L$, $v = @\circ\gamma(v)$

- Si $v=\perp$ alors $@\circ\gamma(\perp) = @(\lambda x.(x=\Omega)) = \perp$
- Si $v=[a,b]$ où $a,b \in Z^*$ et $a \leq b$ alors

$$@\circ\gamma([a,b]) = @(\lambda x.(a \leq x \leq b) \underline{ou} (x=\Omega))$$
$$= [\underline{min}\{x \in Z : a \leq x \leq b\}, \underline{max}\{x \in Z : a \leq x \leq b\}]$$
$$= [a,b]$$

. Soit $P \in P_1$ et $v \in L$ alors d'une part $(P \Rightarrow \gamma(v))$ implique par monotonie $@(P) \sqsubseteq @\circ\gamma(v) = v$ et d'autre part $(@(P) \sqsubseteq v)$ implique $P \Rightarrow \gamma\circ@(P) \Rightarrow \gamma(v)$ par conséquent $(@,\gamma)$ est une paire de fonctions adjointes supérieure.

. Montrons maintenant que $\forall Q \in P_n$, $Q \Rightarrow \overline{\gamma}\circ\overline{@}(Q)$:

$$\overline{\gamma}\circ\overline{@}(Q) = \overline{\gamma}(@(\sigma_1^n(Q)),\ldots,@(\sigma_n^n(Q)))$$
$$= \lambda(x_1,\ldots,x_n).\ \underset{j=1}{\overset{n}{ET}}\ \gamma(@(\sigma_n^j(Q)))(x_j)$$

Pour tout j, $\sigma_j^n(Q) \Rightarrow \gamma\circ@(\sigma_j^n(Q))$ donc $\overline{\gamma}\circ\overline{@}(Q) <= \lambda(x_1,\ldots,x_n).(\underset{j=1}{\overset{n}{ET}}\ \sigma_j^n(Q)(x_j)) \Leftarrow Q$

car $Q(x_1,\ldots,x_n) \Rightarrow \underset{j=1}{\overset{n}{ET}}\ (\nexists x_1,\ldots,x_{j-1},x_{j+1},\ldots,x_n \in (Z \cup \{\Omega\}) :$

$$Q(x_1,\ldots,x_j,\ldots,x_n)\}$$

. Si $v \in L^n$ alors $v = \overline{@}\circ\overline{\gamma}(v)$, en effet:

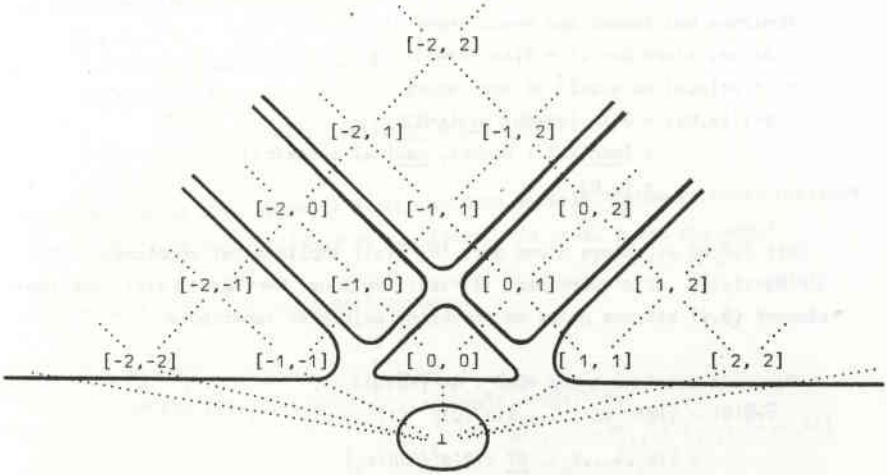
$$\overline{@}\circ\overline{\gamma}(v) = \overline{@}(\gamma(x_1,\ldots,x_n).(\underset{j=1}{\overset{n}{ET}}\ \gamma(v_j)(x_j)))$$

pour la jième composante:

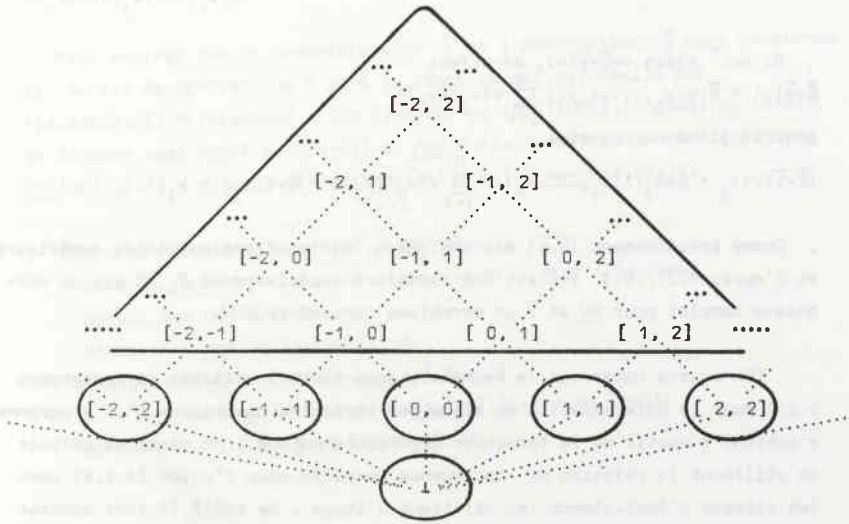
$$(\overline{@}\circ\overline{\gamma}(v))_j = @\circ\sigma_j^n(\lambda(x_1,\ldots,x_n).(\underset{j=1}{\overset{n}{ET}}\ \gamma(v_j)(x_j))) = @\circ\gamma(v_j) = v_j$$

. Comme précédemment $(\overline{@},\overline{\gamma})$ est une paire de fonctions adjointes supérieure et d'après 4.2.7.0.3 $\overline{\gamma}\circ\overline{@}$ est une fermeture supérieure de P_n, $@$ est un morphisme complet pour $\underline{OU}$ et $\overline{\gamma}$ un morphisme complet pour Γ.

(On pourra noter que la fermeture supérieure η utilisée au paragraphe 5.2.1 pour la détermination du signe des variables numériques d'un programme s'obtient à partir de la fermeture supérieure $\gamma\circ@$ que l'on vient de définir en utilisant la relation de congruence complète pour l'union (4.2.6) dont les classes d'équivalence (en utilisant l'image L de $\gamma\circ@(P_1)$) sont données comme suit:



tandis que la fermeture supérieure utilisée pour la propagation des constantes (5.4.2) est obtenue par la relation de congruence complète pour l'union dont les classes d'équivalences sont:



5.7.2 Règles de construction du système approché d'équations en avant associé à un programme

Nous illustrons quelques règles sans donner de détails (que l'on trouvera dans Cousot & Cousot [1975]).

Jonction de chemins d'exécution:

$$X_j = \bigsqcup_{i \in \underline{pred}(j)} X_i$$

Affectation:

Evaluer le membre droit en utilisant l'arithmétique sur les intervalles et affecter la valeur obtenue au membre gauche:

```
{((x=[1,10]),(y=[-2,3]))}
   x:=x+y+1;
{((x=[0,14]),(y=[-2,3]))}
   car [1,10]+[-2,3]+[1,1] = [1-2+1,10+3+1] = [0,14]

{((x=[1,+∞]),(y=[-∞,10]))}
   x:=x+y+1;
{((x=[-∞,+∞]),(y=[-∞,10]))}
   car [1,+∞]+[-∞,10]+[1,1] = [-∞+2,+∞+11] = [-∞,+∞]
```

Test:

```
{((x=[a,b]),(y=[c,d]))}
si x≤y alors
   {((x=[a,b]⊓[-∞,d]),(y=[c,d]⊓[a,+∞]))}
   ...
sinon
   {((x=[a,b]⊓[c+1,+∞]),(y=[c,d]⊓[-∞,b-1]))}
   ...
finsi;
```

```
{((x=[a,b]),(y=[c,d]))}
si x=y alors
   {((x=[a,b]⊓[c,d]),(y=[a,b]⊓[c,d]))}
   ...
sinon
   {si      a=b=c alors ((x=[a,b]),(y=[c+1,d]))
    sinonsi a=b=d alors ((x=[a,b]),(y=[c,d-1]))
    sinonsi a=c=d alors ((x=[a+1,b]),(y=[c,d]))
    sinonsi b=c=d alors ((x=[a,b-1]),(y=[c,d]))
    sinon   ((x=[a,b]),(y=[c,d]))
    finsi}
    ...
finsi;
```

(Avec la convention que [a,b]=⊥ quand b<a).

Exemple 5.7.2.0.1: Le système d'équations associé au programme:

```
        début  n : 0..1000; x: entier;
            lire(n); {0≤n≤1000 doit être vérifié à l'exécution}
{1}
            x:=0;
{2}
        L:
{3}
            si x≤n alors
{4}
                x:=x+2;
{5}
                allera L;
            finsi;
{6}
        fin.
```

est (en notant bs(⊥)=+∞, bs([a,b])=b, bi(⊥)=-∞, bi([a,b])=a):

$$\begin{cases} P_1 = ((x=\bot),(n=[0,1000])) \\ P_2 = ((x=[0,0]),(n=P_1(n))) \\ P_3 = P_2 \sqcup P_5 \\ P_4 = ((x=P_3(x)\sqcap[-\infty,bs(P_3(n))]),(n=P_3(n)\sqcap[bi(P_3(x)),+\infty])) \\ P_5 = ((x=P_4(x)+[2,2]),(n=P_4(n))) \\ P_6 = ((x=P_3(x)\sqcap[bi(P_3(n))+1,+\infty]),(n=P_3(n)\sqcap[-\infty,bs(P_3(x))-1])) \end{cases}$$

Fin de l'exemple.

5.7.3 Résolution du système d'équations approché par approximation dynamique

Le treillis L étant infini, il est facile de montrer que la résolution itérative des équations est en général non convergente en un nombre fini de pas. Nous utilisons donc les techniques d'approximation dynamique introduites au paragraphe 4.1.2.

5.7.3.1 Approximation de la plus petite solution par une itération chaotique croissante avec élargissement supérieur

Pour définir une fois pour toutes la méthode d'approximation utilisée pour contraindre une itération croissante à converger en un nombre fini de pas, nous introduisons un élargissement supérieur $\overline{\nabla}$ sur L (4.1.2.0.4) défini par:

- $\forall x \in L,\ \perp \overline{\nabla} x = x \overline{\nabla} \perp = x$
- $[a,b] \overline{\nabla} [c,d] = [$ si c<a alors $-\infty$ sinon a finsi,
  si d>b alors $+\infty$ sinon b finsi$]$

On vérifie que cet élargissement supérieur satisfait aux hypothèses de la définition 4.1.2.0.4. On notera que $\overline{\nabla}$ n'est pas monotone (voir 4.1.2.0.7(b)). L'élargissement $\overline{\overline{\nabla}}$ sur L^n est obtenu en appliquant $\overline{\nabla}$ composante par composante. On peut alors appliquer la définition 4.1.2.0.5 de sorte que le théorème 4.1.2.0.6 assure la convergence des itérations et la correction de l'approximation.

Nous illustrons la méthode sur l'exemple 5.7.2.0.1. Le graphe de programme correspondant étant réductible au sens d'Allen et Cocke, nous choisissons {3} comme tête de circuit (4.1.2.0.2). Par convention, nous abrégeons ((x=α),(n=β)) en (α ,β). En plus, nous utilisons la remarque 4.1.2.0.7.(a):

$P_j^0 = (\perp,\perp)$ pour j=1,...,6

$P_1^1 = (\perp,[0,1000])$

$P_2^1 = ([0,0],P_1^1(n)) = ([0,0],[0,1000])$

$P_3^1 = P_3^0 \overline{\overline{\nabla}} (P_2^1 \sqcup P_5^0) = (\perp,\perp) \overline{\overline{\nabla}} (P_2^1 \sqcup (\perp,\perp)) = ([0,0],[0,1000])$

$P_4^1 = (P_3^1(x) \sqcap [-\infty, \underline{bs}(P_3^1(n))], P_3^1(n) \sqcap [\underline{bi}(P_3^1(x)), +\infty])$

$= ([0,0] \sqcap [-\infty,1000],[0,1000] \sqcap [0,+\infty]) = ([0,0],[0,1000])$

$P_5^1 = (P_4^1(x)+[2,2],P_4^1(n)) = ([0,0]+[2,2],[0,1000]) = ([2,2],[0,1000])$

$P_3^2 = P_3^1 \overline{\nabla}(P_2^1 \sqcup P_5^1) = ([0,0]\overline{\nabla}([0,0] \sqcup [2,2]),[0,1000]\overline{\nabla}([0,1000] \sqcup [0,1000]))$
$= ([0,0]\overline{\nabla}[0,2],[0,1000]\overline{\nabla}[0,1000]) = ([0,+\infty],[0,1000])$

$P_4^2 = (P_3^2(x) \sqcap [-\infty, \underline{bs}(P_3^2(n))], P_3^2(n) \sqcap [\underline{bi}(P_3^2(x)),+\infty])$
$= ([0,+\infty] \sqcap [-\infty,1000],[0,1000] \sqcap [0,+\infty]) = ([0,1000],[0,1000])$

$P_5^2 = (P_4^2(x)+[2,2],P_4^2(n)) = ([2,1002],[0,1000])$

Comme $P_5^2 \sqcup P_2^1 \leq P_3^2$ la tête de circuit est stabilisée, il reste à calculer:

$P_6^2 = (P_3^2(x) \sqcap [\underline{bi}(P_3^2(n)+1,+\infty], P_3^2(n) \sqcap [-\infty, \underline{bs}(P_3^2(x))-1])$
$= ([0,+\infty] \sqcap [1,+\infty], [0,1000] \sqcap [-\infty,+\infty]) = ([1,+\infty],[0,1000])$

Nous avons donc calculé la solution approchée supérieurement suivante:

$$\begin{cases} P_1 = (\bot,[0,1000]) \\ P_2 = ([0,0],[0,1000]) \\ P_3 = ([0,+\infty],[0,1000]) \\ P_4 = ([0,1000],[0,1000]) \\ P_5 = ([2,1002],[0,1000]) \\ P_6 = ([1,+\infty],[0,1000]) \end{cases}$$

Une itération chaotique croissante sans élargissement se stabiliserait au bout de 500 pas de calculs tandis que l'utilisation d'un élargissement force à la convergence en 2 pas. La précision du résultat obtenu est évidemment moins bonne, mais d'après la remarque 4.1.1.0.9, la solution approchée dont nous disposons peut être améliorée.

5.7.3.2 Amélioration de la solution approchée par une itération chaotique décroissante avec retrécissement inférieur

Ayant obtenu un post-point-fixe P de X=F(X) on a $\mathit{lfp}(F) \leq \mathit{llis}(\lambda X.X \sqcap F(X))(P)$. Comme le calcul de $\mathit{llis}(\lambda X.X \sqcap F(X))(P)$ ne converge pas nécessairement en un nombre fini de pas, nous proposons d'en faire une approximation supérieure en utilisant la définition 4.1.2.0.16 et le théorème 4.1.2.0.17.
Choisissons un retrécissement inférieur $\underline{\Delta}$ sur L défini par:

- $\forall x \in L,\ \bot \underline{\Delta} x = x \underline{\Delta} \bot = \bot$
- $[a,b] \underline{\Delta} [c,d] =$ [si $a=-\infty$ alors c sinon min(a,c) finsi;
  si $b=-\infty$ alors d sinon max(b,d) finsi]

Les hypothèses de la définition 4.1.2.0.15 sont vérifiées (voir dual de la remarque 4.1.2.0.14). Comme précédemment $\underline{\Delta}$ sur L^n est défini en appliquant $\underline{\Delta}$ composante par composante. On reprend l'exemple 5.7.2.0.1 :

$P_1^0 = (\bot,[0,1000])$
$P_2^0 = ([0,0],[0,1000])$
$P_3^0 = ([0,+\infty],[0,1000])$
$P_4^0 = ([0,1000],[0,1000])$
$P_5^0 = ([2,1002],[0,1000])$
$P_6^0 = ([1,+\infty],[0,1000])$
$P_3^1 = (P_3^0 \underline{\Delta} (P_2^0 \sqcup P_5^0) = ([0,+\infty] \underline{\Delta} ([0,0] \sqcup [2,1002]),[0,1000] \underline{\Delta} ([0,1000] \sqcup [0,1000]))$
$= ([0,+\infty] \underline{\Delta} [0,1002],[0,1000] \underline{\Delta} [0,1000]) = ([0,1002],[0,1000])$
$P_4^1 = ([0,1002] \sqcap [-\infty,1000],[0,1000] \sqcap [0,+\infty]) = ([0,1000],[0,1000])$
$P_5^1 = ([0,1000] + [2,2],[0,1000]) = ([2,1002],[0,1000])$
$P_6^1 = ([0,1002] \sqcap [1,+\infty],[0,1000] \sqcap [-\infty,1001]) = ([1,1002],[0,1000])$

Ayant convergé vers un point fixe, le résultat final est:

$P_1 = (\bot,[0,1000])$
$P_2 = ([0,0],[0,1000])$
$P_3 = ([0,1002],[0,1000])$
$P_4 = ([0,1000],[0,1000])$
$P_5 = ([2,1002],[0,1000])$
$P_6 = ([1,1002],[0,1000])$

Dans une implémentation, il peut se produire des débordements de capacité dans les calculs arithmétiques sur les intervalles. Il faut alors récupérer l'interruption et retourner le résultat infini ($-\infty$ ou $+\infty$). De plus, on peut tenir compte des déclarations en introduisant des tests sur les bornes au niveau du langage intermédiaire, en appliquant notre méthode et en éliminant les branches mortes avant de générer le code.

5.7.4 Exemple de suppression des tests de bornes à l'exécution

Le premier exemple est celui de la recherche dichotomique d'une clé k dans une table R de 100 éléments dont les clés sont rangées par ordre croissant. Nous donnons directement les résultats de l'analyse en commentaires du programme:

```
type table = tableau [1,100] d'entiers;
procédure recherche-dichotomique(var R: table; k: valeur entier;
                                                  m: résultat entier);
    var bi, bs : entier;
début
    bi := bi(table); bs:=bs(table);
    {((bi=[1,1]),(bs=[100,100]),(m=⊥))}
    tantque bi≤bs faire
        {((bi=[1,100]),(bs=[1,100]),(m=[1,100]))}
        m:=(bi+bs) div 2;
        {((bi=[1,100]),(bs=[1,100]),(m=[1,100]))}
        si k=R(m) alors
            bi:=bs+1;
            {((bi=[2,101]),(bs=[1,100]),(m=[1,100]))}
        sinonsi k<R(m) alors
            bs:=m-1;
            {((bi=[1,100]),(bs=[0,99]),(m=[1,100]))}
        sinon
            bi:=m+1;
            {((bi=[2,101]),(bs=[1,100]),(m=[1,100]))}
        finsi;
        {((bi=[1,101]),(bs=[0,100]),(m=[1,100]))}
    refaire;
    {((bi=[1,101]),(bs=[0,100]),(m=[1,100]))}
    si R(m)≠k alors m:=bi(table)-1 finsi;
    {((bi=[1,101],(bs=[0,100]),(m=[0,100]))};
fin.
```

Le compilateur est capable de montrer que les accès au tableau R sont corrects (puisqu'à chaque fois 1≤m≤100) et de même qu'il n'y aura pas de débordements de capacité dans les additions et soustractions (il est donc inutile d'introduire du code pour signaler au programmeur les fautes en cours d'exécution). De plus, en faisant pour chaque variable l'union des intervalles obtenus en chaque point du programme, le compilateur est capable de déterminer que l'on aurait pu déclarer m:résultat 0..100, var bi:1..101, var bs:0..100 et en particulier l'information que le résultat m est un entier compris entre 0 et 100 peut être propagée dans tous les appels de la procédure. Enfin, dans l'instruction m:=(bi+bs)div 2 l'analyse a montré que (bi+bs)∈[2,200] et donc que bi+bs≥0. Par conséquent la division peut être implémentée par un décalage à droite (ce qui n'est pas possible si (bi+bs)<0). D'après Welsh[1977] l'introduction des tests à l'exécution et la non-optimisation de la division introduit une augmentation d'environ 50% de la taille du code généré et de 60% du temps moyen d'exécution.

Il est tout à fait intéressant de comparer notre méthode de *découverte* des intervalles de valeurs des variables entiers à la méthode de *vérification* de Welsh[1977]. La méthode de Welsh est une technique classique de compilation qui consiste pour chaque accès à une variable à supposer que sa valeur est dans l'intervalle déclaré par l'utilisateur et à vérifier pour chaque affectation à cette variable (en utilisant l'arithmétique sur les intervalles que nous avons définie pour le membre droit des affectations) que la valeur affectée est dans l'intervalle déclaré (en générant un test si ce n'est pas le cas). Cette méthode en un seul passage est économique mais pas très efficace. On constate en pratique que les programmeurs utilisent assez peu (et à tort) la possibilité introduite par Wirth en PASCAL de déclarer des variables entières dans un intervalle numérique. (Il n'est qu'à voir les exemples donnés par Wirth lui-même dans Wirth[1976], où cette possibilité n'est *jamais* utilisée). La raison en est que les bornes étant nécessairement des constantes numériques (et non comme en LIS (Ichbiah et al.[1974]) des constantes symboliques évaluables à la compilation) il faut généralement changer toutes les bornes quand on charge une déclaration. (Dans notre procédure *recherche-dichotomique* on peut déclarer m:0..100, bi:1..101, bs:0..100 mais si l'on change le nombre d'éléments dans la *table* il faut également changer manuellement les bornes de m, bi, bs ainsi que l'initialisation de bi et bs. En LIS au contraire on déclarerait m:bi(table)-1..bs(table),

bi:bi(table)..bs(table)+1, bs=bs(table)-1..bs(table) de sorte que si l'on change une borne des index de la *table* le compilateur peut automatiquement tenir compte de ce changement. Revenons maintenant à la méthode de Welsh. Si le programmeur déclare m, bi, bs:entier alors il faut 3 tests sur les bornes de tableau et 4 tests de débordement. Si le programmeur déclare m:0..100, bi:1..101, bs:0..100 (ce que nous avons trouvé automatiquement) les tests de débordement sont éliminés, l'optimisation de la division peut être faite mais il reste 3 tests sur les bornes inférieures du tableau (car dans l'accès R(m) il faut 1≤m≤100 alors qu'on a vérifié que 0≤m≤100). Ceci résulte en une longueur de code augmentée de 19% par rapport à notre solution optimale et une augmentation du temps moyen d'exécution de 17%, (chiffres donnés par Welsh). Une méthode de vérification basée uniquement sur les déclarations globales est forcément incomplète car le type d'une variable en un point d'un programme est presque toujours un sous-type du type déclaré globalement (Meertens[1975]). Ainsi dans notre exemple, pour signifier que le type de m n'est pas le même dans la boucle et à la sortie de la boucle, un programmeur bien informé des méthodes de compilation pourrait introduire une variable p supplémentaire et écrire:

```
procédure recherche-dichomotique(var R:table; k:valeur entier;
                                                 p:résultat 0..100);
    var bi:1..101; bs:0..100; m:1..100;
début
    bi:=1; bs:=100;
    tantque bi≤bs faire
        m:=(bi+bs) div 2;
        si R(m)=k alors
            bi:=bs+1;
        sinonsi R(m)>k alors
            bs:=m-1;
        sinon
            bi:=m+1;
        finsi;
    refaire;
    p:=si R(m)=k alors m sinon 0 finsi;
fin.
```

Toutefois cette solution n'est pas encore acceptable car dans l'instruction m:=(bi+bs) $\underline{div}$ 2 le type du membre droit est ([1,101]+[0,100]) $\underline{div}$ [2,2] = [0,100] et il faut introduire un test pour vérifier que (bi+bs) $\underline{div}$ 2≥1 ! Le problème est alors de comparer le coût de ce test à l'exécution au côut de l'analyse permettant de l'éviter. Un élément de comparaison plus important est certainement que notre approche permet la découverte d'un plus grand nombre de fautes de programmation *avant* l'exécution du programme. Enfin, notre approche reste valable dans le cas de bornes symboliques (voir §5.8).

5.7.5 Combinaison des analyses approchées avant et arrière

Nous illustrons la technique exposée au paragraphe 5.6.2.3. dans le cas où la convergence n'est pas naturellement garantie. Disposant des approximations supérieures $\overline{F}(\overline{\phi})$ et $\overline{B}(\overline{\Psi})$ des systèmes d'équations sémantiques en avant $F_\pi(\phi)$ et $B_\pi(\Psi)$ on calcule:

$$
\begin{array}{lll}
P^1 & \supseteq lfp(\overline{F}(\overline{\phi})) & \\
P^2 & = P^1 \underline{\Delta} X^2 & \text{où } X^2 \supseteq lfp(\lambda X.P^1 \sqcap \overline{B}(\Psi)(X)) \\
\ldots & & \\
P^{2k+1} & = P^{2k} \underline{\Delta} X^{2k+1} & \text{où } X^{2k+1} \supseteq lfp(\lambda X.P^{2k} \sqcap \overline{F}(\overline{\phi})(X)) \\
P^{2k+2} & = P^{2k+1} \underline{\Delta} X^{2k+2} & \text{où } X^{2k+2} \supseteq lfp(\lambda X.P^{2k+1} \sqcap \overline{B}(\overline{\phi})(X)) \\
\ldots & &
\end{array}
$$

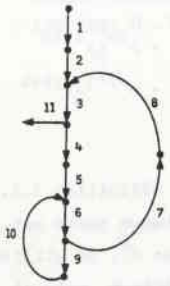
D'après la définition 4.2.1.0.15 du retrécissement inférieur $\underline{\Delta}$ la suite est finie et chaque terme est plus grand que $lfp(F_\pi(\phi))$ et $lfp(B_\pi(\phi))$. Pour calculer les X^k, on utilise une itération chaotique croissante avec élargissement supérieur, puis si la solution obtenue n'est pas un point fixe, on l'améliore par une itération chaotique décroissante avec retrécissement inférieur.

Nous illustrons la technique sur le programme de tri d'un tableau, par la méthode de "remontée des bulles", que nous avons pris dans Manna[1974, p.191]. En enlevant toutes les instructions manipulant le tableau à trier (dont on ne tient pas compte dans l'analyse approchée) ce programme s'écrit:

```
{1}
        i:=n;
{2}
    L:
{3}
        si i≠0 alors
{4}
            j:=0;
{5}
    M:
{6}
            si j=i alors
{7}
                i:=i-1;
{8}
                allera L;
            finsi;
{9}
            j:=j+1;
{10}
            allera M;
        finsi;
{11}
```

Le système approché d'équations en avant associé au programme est le suivant:

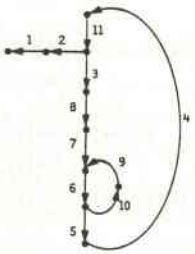
$$
\begin{cases}
P_1 = \overline{\phi} \\
P_2 = P_1(i \leftarrow n) \\
P_3 = P_2 \sqcup P_8 \\
P_4 = P_3(i \leftarrow i \neq 0) \\
P_5 = P_4(j \leftarrow [0,0]) \\
P_6 = P_5 \sqcup P_{10} \\
P_7 = P_6(i \leftarrow i \sqcap j, j \leftarrow i \sqcap j) \\
P_8 = P_7(i \leftarrow i-1) \\
P_9 = P_6(i \leftarrow i \neq j, j \leftarrow i \neq j) \\
P_{10} = P_9(j \leftarrow j+1) \\
P_{11} = P_3(i \leftarrow i \sqcap [0,0])
\end{cases}
$$



Graphe de dépendance
(tête de cycle: 6)

Le système approché d'équations en arrière associé au programme est le suivant:

$$\begin{cases} P_1 = P_2(i \leftarrow [-\infty,+\infty], n \leftarrow i) \\ P_2 = P_3 \\ P_3 = P_{11} \sqcup P_4 \\ P_4 = P_5(j \leftarrow [-\infty,+\infty]) \\ P_5 = P_6 \\ P_6 = P_7 \sqcup P_9 \\ P_7 = P_8(i \leftarrow i+1) \\ P_8 = P_3 \\ P_9 = P_{10}(j \leftarrow j-1) \\ P_{10} = P_6 \\ P_{11} = \overline{\Psi} \end{cases}$$



Graphe de dépendance
(tête de cycle: 6)

Pour analyser le programme, on utilise les spécifications:

$\overline{\phi} = ((i=\perp),(j=\perp),(n=[-\infty,+\infty]))$

$\overline{\Psi} = ((i=[-\infty,+\infty]),(j=[-\infty,+\infty]),(n=[-\infty,+\infty]))$

sans donner le détail des calculs:

$P^1 \supseteq lfp(\overline{F}(\overline{\phi}))$

	1	2	3	4	5	6	7	8	9	10	11
i	$\perp$	$[-\infty,+\infty]$	$[-\infty,+\infty]$	$[-\infty,+\infty]$	$[-\infty,+\infty]$	$[-\infty,+\infty]$	$[0,+\infty]$	$[-1,+\infty]$	$[-\infty,+\infty]$	$[-\infty,+\infty]$	$[0,0]$
j	$\perp$	$\perp$	$[0,+\infty]$	$[0,+\infty]$	$[0,0]$	$[0,+\infty]$	$[0,+\infty]$	$[0,+\infty]$	$[0,+\infty]$	$[1,+\infty]$	$[0,+\infty]$
n	$[-\infty,+\infty]$	$[-\infty,+\infty]$	$[-\infty,+\infty]$	$[-\infty,+\infty]$	$[-\infty,+\infty]$	$[-\infty,+\infty]$	$[-\infty,+\infty]$	$[-\infty,+\infty]$	$[-\infty,+\infty]$	$[-\infty,+\infty]$	$[-\infty,+\infty]$

Ce résultat est très décevant mis à part pour j où l'on a découvert qu'étant initialisé par zéro et incrémenté de un, il est positif. L'analyse en arrière apporte une amélioration:

$P^2 = P^1 \underline{\Delta} X^2$ où $X^2 \supseteq lfp(\lambda X.P^1 \sqcap \overline{B}(\overline{\Psi})(X))$

	1	2	3	4	5	6	7	8	9	10	11
i	$\perp$	$[0,+\infty]$	$[0,+\infty]$	$[1,+\infty]$	$[1,+\infty]$	$[1,+\infty]$	$[1,+\infty]$	$[0,+\infty]$	$[1,+\infty]$	$[1,+\infty]$	$[0,0]$
j	$\perp$	$\perp$	$[0,+\infty]$	$[0,+\infty]$	$[0,+\infty]$	$[0,+\infty]$	$[0,+\infty]$	$[0,+\infty]$	$[0,+\infty]$	$[1,+\infty]$	$[0,+\infty]$
n	$[0,+\infty]$	$[-\infty,+\infty]$	$[-\infty,+\infty]$	$[-\infty,+\infty]$	$[-\infty,+\infty]$	$[-\infty,+\infty]$	$[-\infty,+\infty]$	$[-\infty,+\infty]$	$[-\infty,+\infty]$	$[-\infty,+\infty]$	$[-\infty,+\infty]$

L'analyse en arrière a permis de découvrir que i étant décrémenté de un dans une boucle qui s'achève avec i=0, il faut nécessairement que i soit positif

avant l'entrée dans la boucle pour que cette boucle se termine. Les informations qui viennent d'être trouvées sont propagées en avant:

$P^3 = P^2 \underline{\Delta} X^3$ où $X^3 \supseteq lfp(\lambda X.P^2 \sqcap \overline{B}(\overline{\Psi})(X))$

	1	2	3	4	5	6	7	8	9	10	11
i	⊥	[0,+∞]	[0,+∞]	[1,+∞]	[1,+∞]	[1,+∞]	[1,+∞]	[0,+∞]	[1,+∞]	[1,+∞]	[0,0]
j	⊥	⊥	[1,+∞]	[1,+∞]	[0,0]	[0,+∞]	[1,+∞]	[1,+∞]	[0,+∞]	[1,+∞]	[1,+∞]
n	[0,+∞]	[0,+∞]	[0,+∞]	[0,+∞]	[0,+∞]	[0,+∞]	[0,+∞]	[0,+∞]	[0,+∞]	[0,+∞]	[0,+∞]

Une dernière étape montre que les résultats se stabilisent. Nous avons d'ailleurs trouvé le meilleur résultat que l'on peut obtenir avec les systèmes d'équations approchés dont nous disposons. En particulier, nous avons découvert que si $n \in [-\infty,-1]$ alors le programme ne se termine pas ou conduit à une erreur. D'ailleurs la spécification d'entrée $n \geq 0$ est donnée par Manna[1974, p.191], mais il est intéressant de noter que si elle n'est pas fournie par le programmeur, il est facile de montrer automatiquement et à peu de frais qu'il y a erreur.

Ayant découvert la condition d'entrée:

((i=⊥),(j=⊥),(n,[0,+∞]))

et les déclarations qui doivent figurer dans le programme

D = ((i=[0,+∞]),(j=[0,+∞]),(n=[0,+∞]))

il reste à déterminer où doivent être placés les tests à l'exécution. Connaissant les propriétés P^3_k des variables au point d'entrée a_k d'une instruction I, il suffit de vérifier en utilisant le système d'équations en avant, que le transformé P de P^3_k par I au point de sortie a_ℓ de I est inclus dans D. Si ce n'est pas le cas, un test à l'exécution est nécessaire, ainsi:

- Au point d'entrée il faut tester que $n \geq 0$.
- Quand on décrémente i on a $[1,+\infty]-[1,1] \subseteq [0,+\infty]$, aucun test n'est nécessaire.
- Quand on incrémente j on a $[0,+\infty]+[1,1] = [1,+\infty+1]$ le compilateur doit prévoir la détection possible par le matériel d'une faute par débordement supérieur lors de l'incrémentation de j. En fait ceci est inutile car on voit que l'on a toujours $j \leq i$. D'après l'approximation qui a été choisie en 5.6.1, il est impossible d'exprimer des relations entre ces variables et donc de découvrir cette inégalité. Nous présentons maintenant une technique d'approximation moins grossière qui permet de découvrir des relations entre variables numériques d'un programme.

5.8 DECOUVERTE AUTOMATIQUE DE RELATIONS LINEAIRES D'EGALITE OU D'INEGALITE ENTRE VARIABLES NUMERIQUES D'UN PROGRAMME

Il est assez rare qu'un raisonnement sur un programme ne nécessite pas la découverte de relations invariantes entre les variables x_1, ..., x_n du programme, (voir par exemple 5.6.1.2). Dans cette application nous proposons de découvrir des relations linéaires d'égalité ($ax_1+bx_2+\ldots+wx_n=\alpha$ où a, b, ..., w et α sont des constantes) ou d'inégalité ($ax_1+bx_2+\ldots+wx_n\leq\alpha$) entre les variables numériques x_1, ..., x_n d'un programme. Comme exemple introductif nous donnons l'analyse du programme de tri, par "remontée des bulles", de Knuth[1973, p.107]:

```
        procédure tri(N : valeur entier; K : tableau[1,N] de réel);
            var B, J, T : entier;
        début
            B:=N;
{1}
        tantque B≥1 faire
{2}
            J:=1; T:=0;
{3}
            tantque J≤(B-1) faire
{4}
                si K[J]>K[J+1] alors
{5}
                    échanger(J,J+1); {pas d'effets de bord sur N,B,J,T}
{6}
                    T:=J;
{7}
                finsi;
{8}
                J:=J+1;
{9}
            refaire;
{10}
            si T=0 alors retourner finsi;
{11}
            B:=T;
{12}
        refaire;
{13}
    fin;
```

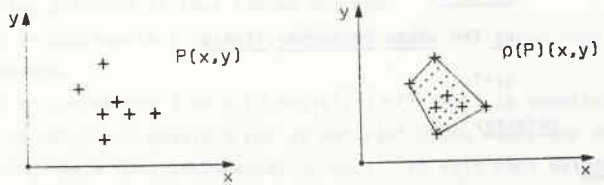
L'analyse automatique de cette procédure (qui a été effectuée grâce à l'implémentation expérimentale de cette application réalisée par N.Halbwachs), a permis de découvrir les invariants suivants:

$$
\begin{cases}
P_1 = \{B=N\} \\
P_2 = \{1 \le B \le N\} \\
P_3 = \{1 \le B \le N,\ J=1,\ T=0\} \\
P_4, P_5, P_6 = \{B \le N,\ T \ge 0,\ T+1 \le J,\ J+1 \le B\} \\
P_7 = \{B \le N,\ J \ge 1,\ J+1 \le B,\ J=T\} \\
P_8 = \{B \le N,\ J+1 \le B,\ J \ge 1,\ T \ge 0,\ T \le J\} \\
P_9 = \{B \le N,\ J \le B,\ J \ge 2,\ T \ge 0,\ T+1 \le J\} \\
P_{10}, P_{11} = \{B \le N,\ J \le B,\ T \ge 0,\ T+1 \le J,\ B \le J+1\} \\
P_{12} = \{J \le N,\ T \ge 0,\ T+1 \le J,\ T=B\} \\
P_{13} = \{B \le N,\ B \le 1\}
\end{cases}
$$

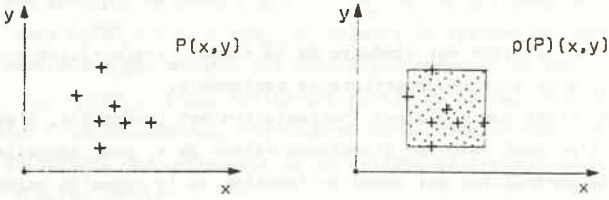
Cette information est utile pour vérifier que les index de tableaux sont compris entre les bornes surtout quand ces bornes sont données sous forme de constantes symboliques (alors que l'application 5.7 est mieux adaptée au cas de bornes numériques). De plus ces invariants sont indispensables à la justification du programme et sont complémentaires des spécifications données par le programmeur, qui bien souvent n'entre pas à ce niveau de détails.

5.8.1 Espace de propriétés approchées.

Soit $P \in P_n = \mathcal{U}^n \to \{\underline{vrai}, \underline{faux}\}$ une propriété sur n variables à valeurs dans l'ensemble $\mathcal{U}$ des rationnels. L'espace de propriétés approchées est défini par la fermeture supérieure ρ où $\rho(P)$ est la propriété caractéristique de l'ensemble des points de l'enveloppe convexe de l'ensemble caractérisé par P. L'interprétation géométrique est donc:



Pour comparer cette application à la précédente qui concernait la découverte d'un intervalle de valeurs des variables numériques, il faut observer que la fermeture utilisée au paragraphe 5.7 était plus simplement:



Pour l'application qui nous intéresse maintenant, il faut étudier l'ensemble $\rho(P_n)$. Soit $P \epsilon P_n$. Comme U est l'ensemble des entiers, rationnels ou réels que l'on peut représenter en machine, il est fini. Par conséquent le nombre de valeurs de U^n qui satisfont P est fini, et donc $\rho(P)$ est un polyhèdre convexe puisque c'est l'enveloppe convexe d'un nombre fini de points de l'espace U^n. En conclusion, on peut représenter $\rho(P)$ par un nombre fini de relations linéaires d'égalité ou d'inégalité entre les variables du programme. (En prenant un point de vue plus abstrait on peut oublier les limitations habituelles des langages de programmation et considérer que les variables du programme sont à valeurs dans l'ensemble R des réels. Dans ce cas $\rho(P_n)$ correspond aux sous-ensembles convexes de R^n et certains ne peuvent pas être décrits par un système fini de relations linéaires d'égalité ou d'inégalité. Alors l'espace des propriétés approchées qui correspond aux polyhèdres convexes n'est pas un treillis complet. Toutefois l'utilisation des algorithmes d'approximation dynamique développés au paragraphe 4.1 garantit que la résolution itérative des systèmes d'équations associés aux programmes se fait en un nombre fini de pas de sorte que la limite de la suite des itérés est un polyhèdre convexe, (voir remarque 4.1.2.0.7.(e)).

5.8.2 Règles de construction du système approché d'équations en avant associé à un programme

Affectation:

Soit X le vecteur colonne des variables du programme et $AX \leq B$ le

système de contraintes avant une instruction d'affectation (pour simplifier, les relations d'égalité son représentées par deux relations d'inégalité opposées):

- L'affectation n'est pas linéaire (comme $x:=x^2+yz-t$) alors le système de contraintes après l'instruction est obtenu en éliminant x dans le système $AX \leq B$.
- Sinon l'affectation est linéaire de la forme $x_k := a_1x_1 + \ldots + a_nx_n + b$ où $a_1, \ldots, a_n, b$ sont des coéfficients rationnels.
  - Si a_k n'est pas nul alors l'affectation est inversible, c'est-à-dire que l'on peut calculer l'ancienne valeur de x_k pour laquelle un système de contraintes est connu en fonction de la nouvelle valeur. Le système de contraintes après l'affectation est donc obtenu en remplaçant x_k par $(x_k - a_1x_1 - \ldots - a_{k-1}x_{k-1} - a_{k+1}x_{k+1} \ldots - a_nx_n - b)/a_k$ dans $AX \leq B$.
  - Si a_k est nul alors l'ancienne valeur de x_k est perdue et on élimine x_k du système de contraintes $AX \leq B$. On rajoute ensuite la contrainte $x_k = a_1x_1 + \ldots + a_{k-1}x_{k-1} + a_{k+1}x_{k+1} + \ldots + a_nx_n + b$.

Exemple:

Système de contraintes à l'entrée d'une affectation :

$$\begin{cases} x_2 \geq 1 \\ x_1 + x_2 \geq 5 \\ x_1 - x_2 \geq -1 \end{cases}$$

Affectation non linéaire :	Affectation linéaire non inversible :	Affectation linéaire inversible :
$x_2 := (x_1)^2 - (x_2)^2$	$x_2 := x_1 + 1$	$x_2 := x_1 + x_2/2 = 1$

Système de contraintes en sortie :

$$\begin{cases} x_1 \geq 2 \end{cases} \qquad \begin{cases} x_1 \geq 2 \\ -x_1 + x_2 = 1 \end{cases} \qquad \begin{cases} -2x_1 + 2x_2 \geq 3 \\ -x_1 + 2x_2 \geq 7 \\ +3x_1 - 2x_2 \geq -3 \end{cases}$$

Fin de l'exemple.

Test:

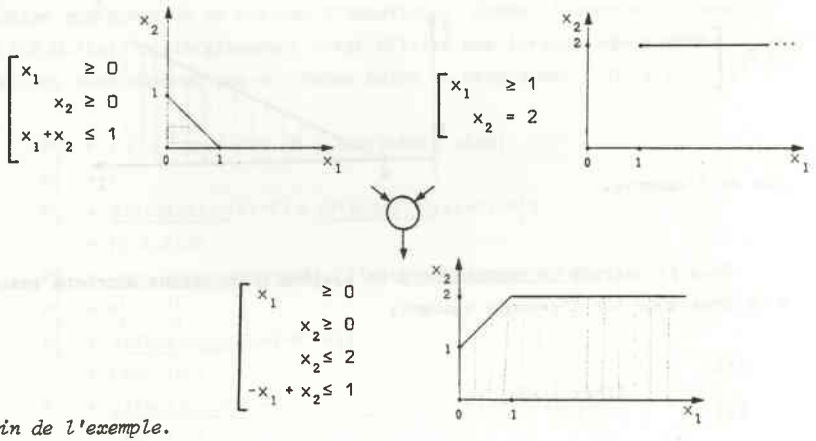
Si le test est linéaire, les contraintes en sortie sont obtenues en rajoutant la contrainte du test au système de contraintes en entrée. Si

le test n'est pas linéaire, il est simplement ignoré et les contraintes en entrée et sortie sont les mêmes.

Jonction de chemins d'exécution:

A une jonction de chemins $a_1, \ldots, a_k$ où les systèmes de contraintes respectifs sont $A_1X \leq B_1, \ldots, A_kX \leq B_k$, on calcule le système de contraintes correspondant à l'enveloppe convexe des polyhèdres convexes définis par $A_1X \leq B_1, \ldots, A_kX \leq B_k$. (Pour éviter une partie de ce calcul coûteux, on maintient dans l'implémentation, une double représentation des propriétés approchées: un système de contraintes et le système générateur du polyhèdre correspondant (Lanery[1966])).

Exemple:



Fin de l'exemple.

5.8.3 Résolution approchée du système d'équations par une itération chaotique croissante avec élargissement supérieur

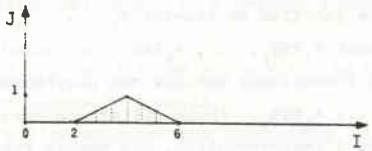
La résolution itérative du système d'équations associé à un programme étant en général non convergente en un nombre fini de pas, nous utilisons une itération chaotique croissante avec élargissement supérieur (définition 4.1.2.0.5).

L'élargissement supérieur $P_1 \overline{\nabla} P_2$ est constitué par les contraintes de P_1

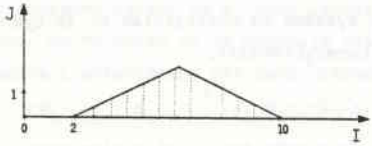
satisfaites par les éléments du système générateur de P_2. Comme le nombre de contraintes ne peut ainsi que diminuer, les conditions de la définition 4.1.2.0.4 sont satisfaites.

Exemple:

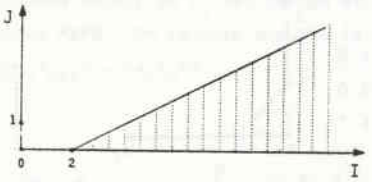
$$Q_1 \begin{cases} -I+2J \leq -2 \\ I+2J \leq 6 \\ J \geq 0 \end{cases}$$



$$Q_2 \begin{cases} -I+2J \leq -2 \\ I+2J \leq 10 \\ J \geq 0 \end{cases}$$



$$Q_1 \overline{\nabla} Q_2 \begin{cases} -I+2J \leq -2 \\ J \geq 0 \end{cases}$$



Fin de l'exemple.

Nous illustrons la résolution d'un système d'équations approché associé à un programme sur l'exemple suivant:

```
{1}
          I:=2; J:=0;
{2}
     L:
{3}
          si ... alors
{4}
               I:=I+4;
{5}
          sinon
{6}
               I:=I+2; J:=J+1;
{7}
          finsi;
{8}
          allera L;
```

(Le test d'une condition non linéaire est ignoré).

Les règles énoncées au paragraphe 5.8.2 permettent de construire le système approché d'équations en avant associé à ce programme:

$$\begin{cases} P_1 = T \\ P_2 = \underline{\text{affectation}}(I:=2) \circ \underline{\text{affectation}}(J:=0)(P_1) \\ P_3 = \underline{\text{enveloppe-convexe}}(P_2, P_8) \\ P_4 = P_3 \\ P_5 = \underline{\text{affectation}}(I:=I+4)(P_4) \\ P_6 = P_3 \\ P_7 = \underline{\text{affectation}}(J:=J+1) \circ \underline{\text{affectation}}(I:=I+2)(P_6) \\ P_8 = \underline{\text{enveloppe-convexe}}(P_5, P_7) \end{cases}$$

L'exemple étant très simple, nous pouvons maintenant illustrer la résolution approchée de ce système d'équations. Comme l'autorise la remarque 4.1.2.0.7(c), l'élargissement n'est utilisé que lorsque suffisamment d'informations sont rassemblées en chaque point du programme.

$P_1^0 = \bot$ pour i=1..8 (polyhèdre vide)

$P_1^1 = T$

$P_2^1 = \underline{\text{affectation}}(I:=2) \circ \underline{\text{affectation}}(J:=0)(P_1^1)$

$= \{I=2, J=0\}$

$P_3^1 = \underline{\text{enveloppe-convexe}}(P_2^1, P_8^0) = \underline{\text{enveloppe-convexe}}(P_2^1, \bot) = P_2^1$

$P_4^1 = P_6^1 = P_3^1$

$P_5^1 = \underline{\text{affectation}}(I:=I+4)(P_3^1)$

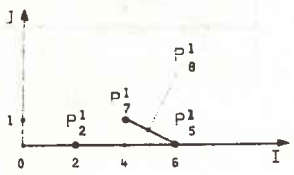
$= \{I=6, J=0\}$

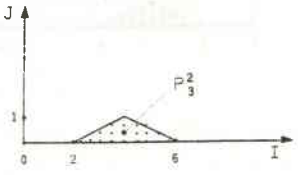
$P_7^1 = \underline{\text{affectation}}(J:=J+1) \circ \underline{\text{affectation}}(I:=I+2)(P_6^1)$

$= \{I=4, J=1\}$

$P_8^1 = \underline{\text{enveloppe-convexe}}(P_5^1, P_7^1)$

$= \{I+2J=6, 4 \leq I \leq 6\}$





P_3^2 = enveloppe-convexe(P_2^1,P_8^1)

= {2J+2≤I,I+2J≤6,0≤J}

P_4^2 = P_6^2 = P_3^2

P_5^2 = affectation(I:=I+4)(P_4^2)

= {2J+6≤I,I+2J≤10,0≤J}

(I est remplacé par I-4 dans P_4^2 car l'affectation est inversible)

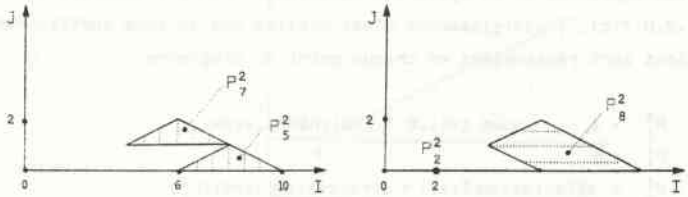
P_7^2 = affectation(J:=J+1) ∘ affectation(I:=I+2)(P_6^2)

= (2J+2≤I,I+2J≤10,1≤J}

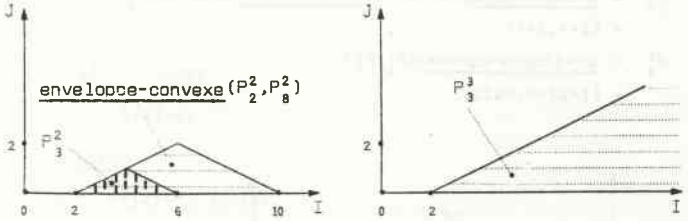
(I et J sont remplacés respectivement par I-2 et J-1 dans P_6^2).

P_8^2 = enveloppe-convexe(P_5^2,P_7^2)

= {2J+2≤I,6≤I+2J≤10,0≤J}



P_3^3 = P_3^2 $\overline{\nabla}$ enveloppe-convexe(P_2^2,P_8^2)



$P_3^3 = \{2J+2 \leq I, 0 \leq J\}$

$P_4^3 = P_6^3 = P_3^3$

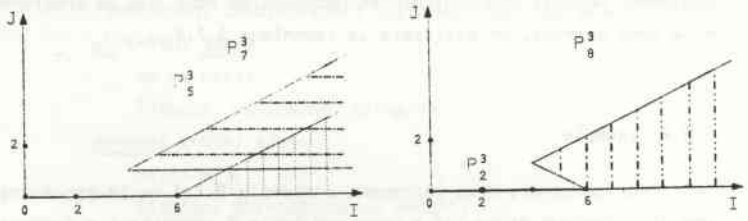
$P_5^3 = \underline{affectation}(I:=I+4)(P_4^3)$

$= \{2J+6 \leq I, 0 \leq J\}$

$P_7^3 = \underline{affectation}(J:=J+1) \circ \underline{affectation}(I:=I+2)(P_6^3)$

$= \{2J+2 \leq I, 1 \leq J\}$

$P_8^3 = \underline{enveloppe\text{-}convexe}(P_5^3, P_7^3)$



$P_8^3 = \{2J+2 \leq I, 6 \leq I+2J, 0 \leq J\}$

Alors $\underline{enveloppe\text{-}convexe}(P_2^3, P_8^3) = P_6^3$, donc a trouvé un point fixe du système d'équations et le résultat final est:

```
{1}
      I:=2;J:=0;
{2}   {I=2,J=0}
   L:
{3}   {2J+2≤I,0≤J}
      si ... alors
{4}       {2J+2≤I,0≤J}
          I:=I+4;
{5}       {2J+6≤I,0≤J}
      sinon
{6}       {2J+2≤I,0≤J}
          I:=I+2; J:=J+1;
{7}       {2J+2≤I,1≤J}
      finsi;
{8}   {2J+2≤I, 6≤I+2J,0≤J}
      allera L;
```

En particulier, nous avons trouvé un invariant de boucle {2J+2≤I,0≤J} qui exprime une relation invariante entre les deux variables I et J du programme bien que cette relation ne soit mise en évidence dans aucune instruction du programme.

On trouvera des informations complémentaires concernant cette application dans Cousot & Halbwachs[1978]. En complément de cette référence, ajoutons que les résultats obtenus par une itération croissante avec élargissement supérieur peuvent être améliorés par une itération décroissante avec retrécissement supérieur (comme au paragraphe 5.7.3.2). Si l'on désire également obtenir des contraintes nécessaires pour que le programme se termine sans erreurs, on utilisera la technique 5.7.5.

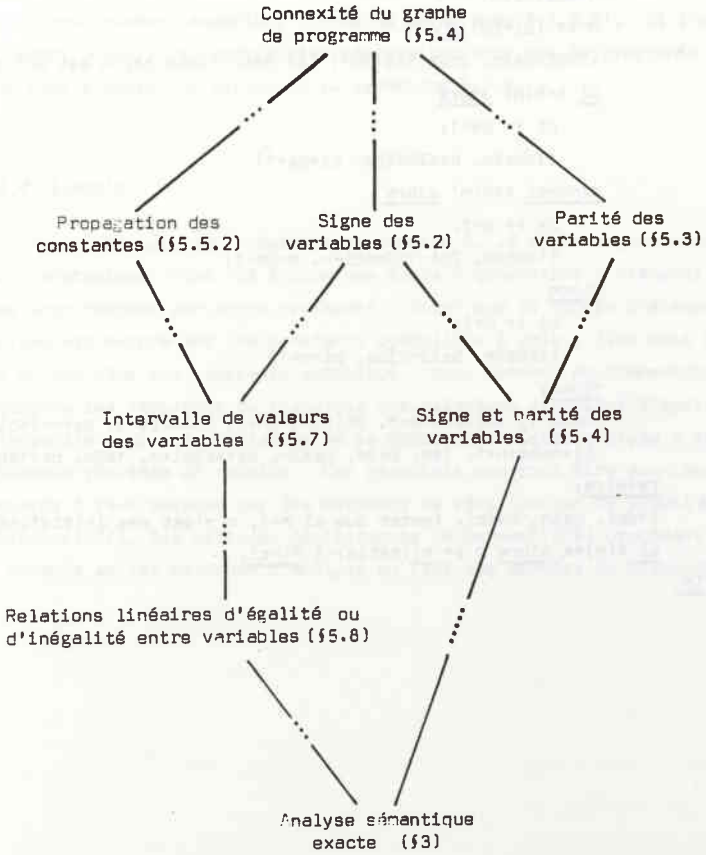
5.8.4 Exemple

Pour terminer, nous reprenons l'exemple 5.7.4 de la procédure de recherche dichotomique d'une clé k dans une table R comportant n éléments dont les clés sont rangées par ordre croissant. Noter que le nombre d'éléments du tableau est maintenant une constante symbolique à valeur fixe mais indéterminée et non plus une constante numérique. Nous donnons en commentaires de la procédure les résultats de l'analyse des relations linéaires d'égalité ou d'inégalité entre les variables de la procédure. Cette analyse a été automatiquement réalisée en machine. Ces résultats pourront être avantageusement comparés à ceux obtenus par les méthodes de vérification de Suzuki & Ishihata[1977], les méthodes heuristiques de German[1978] procédant par essais et erreurs et les méthodes d'analyse du flot des données de Gilbert[1977].

```
type table = tableau [1,n] d'entiers;
procédure recherche-dichotomique(var R: table; k: valeur entier;
                                 m: résultat entier);
    var bi, bs : entier;
début
    bi := bi(table); bs := bs(table);
    {bi=1, bs=n}
    tantque bi≤bs faire
        {1≤bi≤bs≤n}
        m := (bi+bs) div 2;
        {1≤bi≤bs≤n, 2m≤bi+bs≤2m+1 (et donc 1≤m≤n car m est entier)}
        si k=R(m) alors
            bi := bs+1;
            {1≤bs≤n, bs≤2m≤2bs, bi=bs+1}
        sinonsi k<R(m) alors
            bs := m-1;
            {1≤bi≤n, 2bi-1≤2m≤bi+n, m=bs+1}
        sinon
            bi := m+1;
            {1≤bs≤n, bs≤2m≤2bs, bi=m+1}
        finsi;
        {m≤bs+1, 3bi≤2bs+n+3, 3bi≤2bs+2n+3, 2bi≤2bs+3, bs+m+1≤bi+n,
         bi+m≤bs+n+1, 1≤n, bs≤n, bs≤2m, bs+4≤3bi+m, 1≤2m, bs+1≤bi+m}
    refaire;
    {1≤bi, bs≤n, bs<bi, (noter que si n<1, m n'est pas initialisé)}
    si R(m)≠k alors m := bi(table)-1 finsi;
fin.
```

5.9 HIERARCHIE DES APPLICATIONS

Pour terminer ce chapitre, nous pouvons ordonner partiellement quelques analyses approchées qui ont été données en exemples (en utilisant la notion intuitive de précision de l'approximation qui correspond à l'ordre $\sqsubseteq$ dans le treillis des fermetures):



Un travail qui reste à faire est de compléter ce treillis avec d'autres applications intéressantes. S'il peut être difficile d'imaginer des modèles approchés et utiles, chacun des exemples donnés dans ce chapitre a montré que le schéma théorique que nous proposons offre un guide très précis mettant en évidence les problèmes à résoudre pour chaque application particulière et proposant des méthodes générales pour les solutionner.

Enfin, la méthode d'analyse des programmes que nous proposons est la même qu'il s'agisse d'analyse sémantique exacte (§3) ou qu'il s'agisse d'analyse sémantique automatique mais approchée (§5). Entre ces deux extrêmes, on pourrait utiliser le même modèle pour effectuer à la main des analyses approchées mais trop complexes pour être complétement automatisables. Bien que nous n'ayons pas donné de tels exemples, il y a là un champ ouvert d'applications intéressantes.

5.10 NOTES BIBLIOGRAPHIQUES

Les techniques booléennes d'optimisation de programmes remontent à Vyssotsky et Wegner qui les ont mises en oeuvre dans un compilateur FORTRAN. Ils utilisèrent une méthode de résolution itérative. Plus tard Allen[1970], Allen[1971], Cocke[1970] introduisirent une méthode de résolution comparable à la méthode d'élimination de Gauss et basée sur les intervalles du graphe de programme. Cette méthode qui exige que les équations soient booléennes et le graphe de programme "réductible" n'est pas générale, (Earnest[1974], Graham & Wegman[1976], Hecht & Ullman[1972], Hecht & Ullman[1974], Kasvanov[1973], Kennedy[1972], Schaefer[1973], Tarjan[1974]). La résolution itérative des équations booléennes a été utilisée indépendamment par Ichbiah, Morel & Renvoise[1972],Hecht & Ullman[1973], elle n'impose évidemment aucune restriction. Les tentatives de comparaison entre les méthodes directes et itératives (Hecht & Ullman[1975], Kennedy[1976], Tarjan[1975]) n'apportent pas de conclusions très tranchées car les hypothèses qui permettent d'utiliser les méthodes directes assurent généralement une convergence rapide dans les méthodes itératives. Par exemple, dans le cas du problème des variables vivantes (4.2.4.1.1), on sait démontrer l'existence d'un ordre optimal de parcours du graphe de programme (Kennedy[1975], Tarjan[1976]) et quand le programme est réductible, on sait donner un algorithme pour construire cet ordre (Aho & Ullman[1976]). Cela montre d'ailleurs que le problème de l'étude théorique des stratégies chaotiques optimales peut recevoir une réponse dans

certains cas particuliers intéressants. Spillman[1971],Boom[1974] et Aho & Ullman[1977, ch.14.7] discutent l'emploi des techniques classiques d'optimisation des programmes en présence de pointeurs.

Notre méthode d'analyse des programmes généralise l'idée déjà ancienne d'effectuer une exécution symbolique sur des valeurs abstraites caractérisant les propriétés à découvrir, Jensen[1965], Naur[1965], Sintzoff[1972], Kildall[1973], Karr[1975], Schwartz[1975], Wegbreit[1975].

On trouvera des exemples de techniques pour déterminer à la compilation le type des objets manipulés par un langage de haut-niveau dans Bauer & Saal[1974], Tenenbaum[1974], Jones & Muchnick[1976] et Kaplan & Ullman[1978].

On pourra comparer notre application à la découverte d'un intervalle de valeur des variables numériques d'un programme (§5.7) à l'approche empirique de Harrison[1977]. Pour mémoire, rappelons les méthodes de vérification utilisées par Welsh[1977] et Suzuki & Ishihata[1977].

L'application à la découverte de relations linéaires d'égalité ou d'inégalité entre les variables d'un programme (§5.8) améliore les résultats de Karr[1976] sur le problème plus simple de déterminer des contraintes linéaires d'égalité. Il existe des approches différentes de la nôtre qui ont apporté des réponses partielles au problème de découvrir des relations invariantes entre les variables d'un programme: Cooper[1971], Caplain[1975], Elspas[1974], Germain & Wegbreit[1975], Katz & Manna[1976], Wegbreit[1974], Wegbreit[1977].

Signalons pour terminer que l'application à la découverte automatique d'un intervalle de valeurs des variables numériques d'un programme a été implémentée par un élève de J.Cohen à l'Université Brandeis à la suite de Cousot & Cousot[1976], Curry[1977] applique l'idée d'interprétation abstraite de Cousot & Cousot[1977] à un langage de programmation graphique.

CHAPITRE 6

ANALYSE SEMANTIQUE DES PROCEDURES RECURSIVES

6. ANALYSE SEMANTIQUE DES PROCEDURES RECURSIVES

6. ANALYSE SEMANTIQUE DES PROCEDURES RECURSIVES

Nous considérons maintenant un langage de programmation plus général que précédemment avec affectation, instruction conditionnelle, instruction de branchement inconditionnel, structure de bloc et procédures éventuellement récursives (avec passage de paramètres par valeur-résultat). Cette étude est nécessaire pour tenir compte de traits de langage très largement utilisés dans les langages de programmation. Elle a également l'avantage de montrer que la démarche que nous avons suivie, dans les chapitres précédents, pour élaborer une méthode d'analyse sémantique des programmes est tout à fait générale. Nous reprenons le plan de l'étude des programmes séquentiels itératifs, à ceci près qu'au lieu de considérer des systèmes d'équations de la forme $X=F(X)$, nous devons considérer des systèmes de la forme $f(X) = F(f)(X)$. En effet, nous avons observé au chapitre 3 que nous pouvions analyser un programme séquentiel itératif π en associant à chaque point α du programme un prédicat $P_\alpha(\varphi)$ qui dépend implicitement d'une spécification d'entrée φ. Ces prédicats étaient obtenus comme solution d'un système d'équations $P(\varphi)=G(P)(\varphi)$. En observant que le système d'équations peut s'écrire sous la forme $P(\varphi)=F_\pi(\varphi)(P(\varphi))$ et en analysant le programme pour une spécification donnée φ, il n'était pas nécessaire de considérer un système d'équations fonctionnelles puisque nous pouvions poser $X_\alpha=P_\alpha(\varphi)$ et résoudre $X=F(X)$ avec $F=F_\pi(\varphi)$. Dans le cas d'une procédure récursive le système d'équations $P(\varphi)=G(P)(\varphi)$ s'écrit sous la forme $P(\varphi)=H(P(g(\varphi)))$ puisque pour chaque appel récursif de la procédure, il y a une spécification d'entrée différente qui est fonction de la spécification d'entrée correspondant à l'appel principal. Pour la résolution exacte de $P(\varphi)=G(P)(\varphi)$, la simplification consistant à ne considérer que la seule spécification d'entrée correspondant à un appel principal donné, ne peut donc pas être retenue puisque ceci conduit à considérer chacune des spécifications d'entrée correspondant aux diverses invocations récursives de la procédure, c'est-à-dire un nombre généralement infini d'équations. Il

est donc nécessaire d'associer en chaque point de la procédure une fonction (ou transformateur) de prédicats, ce qui ne pose pas de nouvelle difficulté théorique puisqu'on peut résoudre $P(\varphi)=G(P)(\varphi)$ en utilisant les résultats obtenus dans les chapitres précédents (en posant $P=F(P)$ avec $F=\lambda P.\{\lambda\varphi.[G(P)(\varphi)]\}$). Toutefois dans le cas d'une analyse approchée, il est possible de spécialiser les méthodes automatiques de résolution approchée au cas de systèmes d'équations fonctionnelles tout en évitant la difficulté d'avoir à représenter en machine des fonctions. Il suffit de reprendre l'idée de spécifications d'entrée données et celle d'approximation pour éviter d'avoir à considérer un nombre infini de spécifications d'entrée.

6.1 SEMANTIQUE DEDUCTIVE EN AVANT DES PROCEDURES RECURSIVES

Comme nous considérons maintenant un langage à structure de bloc, chaque variable du programme n'est pas visible en chaque point du programme. Il faut donc définir en chaque point du programme un environnement $(x_1:U_1,\ldots,x_n:U_n)$, en bref $(\bar{x}:\bar{U})$, qui donne les variables du programme visibles en ce point et pour chacune d'elles le domaine de ses valeurs. Cet environnement est défini statiquement par la syntaxe du langage.

Soit f une procédure comportant α points programme $a_1, \ldots, a_\alpha$ et dont les paramètres passés par valeur $\bar{v}=(v_1,\ldots,v_p)$ et résultat $\bar{r}=(r_1,\ldots,r_q)$ sont à valeur dans $\bar{T}=(T_1,\ldots,T_p)$ et $\bar{T}'=(T'_1,\ldots,T'_q)$. A chaque point a_i de la procédure f auquel est associé un transformateur de prédicat $\phi_i\in((\bar{T}\to\mathcal{B})\to(\bar{U}\to\mathcal{B}))$ où $\mathcal{B}=\{\underline{vrai}, \underline{faux}\}$ qui est obtenu comme la plus petite solution d'un système d'équations sémantiques en avant associé à la procédure f. Soit $\varphi\in(\bar{T}\to\mathcal{B})$ une spécification d'entrée de la procédure f, alors l'ensemble des valeurs possibles des variables $\bar{x}$ au point a_i lors d'une exécution de f, (appelée avec des paramètres d'entrée dont les valeurs $\bar{v}$ sont telles que $\varphi(\bar{v})$ est vrai), est caractérisé par $\phi_i(\varphi)$.

Nous donnons maintenant les règles de construction du système d'équations en avant associé à une procédure. Les règles pour l'affectation, les tests et les branchements inconditionnels sont les mêmes qu'au chapitre 3 mise à part la prise en compte de la portée des identificateurs résultant

de la structure de bloc. La principale difficulté est constituée par les règles correspondant aux déclarations et appels de procédure.

Affectation :

Soient $(\bar{x}:\bar{U})$ et ϕ_i les environnement et transformateur de prédicats associés au point a_i d'une procédure, avant l'affectation $\bar{x}:=e(\bar{x})$ où $e\epsilon(\bar{U}\to\bar{U})$. Alors l'environnement correspondant au point a_j après l'affectation est $(\bar{x}:\bar{U})$ et $\phi_j=\underline{affectation}(e)\circ\phi_i$. En résumé, on écrira :

$<\phi_i\ ,\ (\bar{x}:\bar{U})>$
$\bar{x}:=e(\bar{x})$;
$<\phi_j=\underline{affectation}(e)\circ\phi_i\ ,\ (\bar{x}:\bar{U})>$

Instruction conditionnelle :

$<\phi_i\ ,\ (\bar{x}:\bar{U})>$
si $p(\bar{x})$ **alors** (où $p\epsilon(\bar{U}\to B)$)
  $<\phi_j=\underline{test}(p)\circ\phi_i\ ,\ (\bar{x}:\bar{U})>$
  ...
sinon
  $<\phi_k=\underline{test}(\underline{non}(p))\circ\phi_i\ ,\ (\bar{x}:\bar{U})>$
  ...
finsi;

Bloc :

A la différence d'ALGOL 60 (Naur [1963]) nous considérons qu'une seule déclaration est applicable à un identificateur en chaque point d'un programme, c'est-à-dire que comme en LIS un identificateur I déclaré dans un bloc A ne peut pas être redéclaré dans un bloc B intérieur à A avec l'effet qu'à l'intérieur de B la déclaration de I en B masque la déclaration de I en A. Ceci n'est pas une restriction sémantique puisqu'il est toujours possible de modifier syntaxiquement les identificateurs du bloc intérieur. Ce trait de langage a d'ailleurs été adopté dans quelques langages de programmation (Ichbiah et al. [1974], Lampson et al. [1977]).

$\langle \phi_i \ , \ (\bar{x}:\bar{U}) \rangle$

début

$y_1:t_1, \ \ldots, \ y_m:t_m;$

$\langle \phi_j = \underline{\text{début}}(\bar{U},\bar{T}) \circ \phi_i \ , \ (\bar{x}:\bar{U},\bar{y}:\bar{T}) \rangle$

...

$\langle \phi_k \ , \ (\bar{x}:\bar{U},\bar{y}:\bar{T}) \rangle$

fin;

$\langle \phi_\ell = \underline{\text{fin}}(\bar{U},\bar{T}) \circ \phi_k \ , \ (\bar{x}:\bar{U}) \rangle$

où

- les identificateurs $\bar{y}=y_1,\ldots,y_m$ sont syntaxiquement différents des $\bar{x}=x_1, \ \ldots, \ x_n$.

- $\bar{T}=T_1 x \ldots . x T_m$ et les $T_1, \ \ldots, \ T_m$ sont des domaines de valeurs des variables de type $t_1, \ \ldots, \ t_m$.

- $\underline{\text{début}}(\bar{U},\bar{T}) = \underline{\text{début}}((U_1 x \ldots x U_n),(T_1 x \ldots x T_m))$

  $\in (\bar{U} \to \mathcal{B}) \to (\bar{U} x \bar{T} \to \mathcal{B})$

  $\in ((U_1 x \ldots x U_n \to \mathcal{B}) \to (U_1 x \ldots x U_n x T_1 x \ldots x T_m \to \mathcal{B}))$

  $= \lambda P.\{\lambda(\bar{x},\bar{y}).[P(\bar{x}) \ \underline{\text{et}} \ \bar{y}=\bar{\Omega}]\}$

  $= \lambda P.\{\lambda(x_1,\ldots,x_n,y_1,\ldots,y_m).[P(x_1,\ldots,x_n) \ \underline{\text{et}} \ (\overset{m}{\underset{j=1}{ET}}(y_j=\Omega_j))]\}$

  où Ω_j est la valeur "non-initialisé" dans T_j.

- $\underline{\text{fin}}(\bar{U},\bar{T})$

  $\in ((\bar{U} x \bar{T} \to \mathcal{B}) \to (\bar{U} \to \mathcal{B}))$

  $= \lambda P.\{\lambda(\bar{x}).[\exists \bar{v} \in \bar{T} : P(\bar{x},\bar{v})]\}$

  $= \lambda P.\{\lambda(x_1,\ldots,x_n).[\exists (v_1,\ldots,v_m) \in T_1 x \ldots x T_m : P(x_1,\ldots,x_n,v_1,\ldots,v_m)]\}$

On exprime simplement que dans le bloc, les variables $x_1, \ \ldots, \ x_n$ et $y_1, \ \ldots, \ y_m$ sont visibles. A l'entrée du bloc, on sait que les variables $x_1, \ \ldots, \ x_m$ ont même valeur qu'à l'extérieur du bloc et que les $y_1, \ \ldots, \ y_m$ ne sont pas initialisés. A la sortie du bloc, les variables $y_1,\ldots,y_m$ locales au bloc sont éliminées.

Soient P, i, n, $i_1, \ \ldots, \ i_q$ tels que $P \in (U_1 x \ldots x U_n \to \mathcal{B})$, $(1 \le i,q \le n)$, $(\forall k \in [1,q], \ (1 \le i_k \le n))$ et $(\forall k,\ell \in [1,q], \ (k \ne \ell) \Rightarrow (i_k \ne i_\ell))$. Dans la suite on utilisera les notations suivantes :

$$\bar{\sigma}_i^n(P) = \lambda(v_1,\ldots,v_{i-1},v_{i+1},\ldots,v_n).[\exists a\in U_i : P(v_1,\ldots,v_{i-1},a,v_{i+1},\ldots,v_n)]$$

$$\bar{\sigma}^n_{i_1,\ldots,i_q} = \bar{\sigma}^{n-q+1}_{i_1}\circ\bar{\sigma}^{n-q+2}_{i_2}\circ\ldots\circ\bar{\sigma}^{n-1}_{i_{q-1}}\circ\bar{\sigma}^{n}_{i_q}$$

(noter que : $\underline{\text{fin}}((U_1\times\ldots\times U_n),(T_1\times\ldots\times T_m)) = \bar{\sigma}^{n+m}_{n+1,\ldots,n+m}$)

$$\sigma_i^n(P) = \lambda x.[\exists(a_1,\ldots,a_{i-1},a_{i+1},\ldots,a_n)\in(U_1\times\ldots\times U_{i-1}\times U_{i+1}\times\ldots\times U_n) : P(a_1,\ldots,a_{i-1},x,a_{i+1},\ldots,a_n)]$$

$$\sigma^n_{i_1,\ldots,i_q}(P) = \lambda(x_1,\ldots,x_q).[(\forall k\in[1,n]-\{i_1,\ldots,i_q\},\ \exists a_k\in U_k) : P(v_1,\ldots,v_n)]$$

où $\forall \ell\in[1,n]$, v_ℓ = $\underline{\text{si}}$ $(\exists j\in[1,q]: \ell=i_j)$ $\underline{\text{alors}}$ x_j $\underline{\text{sinon}}$ a_ℓ $\underline{\text{finsi}}$;

Etiquette et branchements inconditionnels :

Nous considérons que les valeurs des étiquettes sont statiquement déterminées et que les branchements inconditionnels sont à l'intérieur d'une même procédure. (Les sauts à l'extérieur d'une procédure qui amèneraient à considérer des équations plus complexes sont donc exclus).

Le transformateur de prédicats ϕ_i associé au point de programme a_i suivant une étiquette L est la disjonction des transformateurs de prédicats ϕ_j associés aux points de programme a_j précédant une instruction $\underline{\text{allera}}$ L ou précédant L. Dans le cas d'un branchement inconditionnel hors d'un bloc, la sortie du bloc doit être considérée. Comme nous voulons éviter un lourd formalisme syntaxique pour décrire cette règle, un exemple suffira :

```
        <(x̄:ū)>
        ...
{1}
        allera L;
        ...
{2}
   L:
{3}
        ...
         début
             y1:t1;
             ...
             début
                 y2:t2;
                 ...
{4}
                 allera L;
                 ...
             fin;
             ...
{5}
             allera L;
             ...
         fin;
        ...
{6}
        allera L;
```

On a :

$$\phi_3 = [\phi_1] \text{ ou } [\phi_2] \text{ ou } [\text{fin}((\bar{u}\times T_1),(T_2))\circ\text{fin}((\bar{u}),(T_1))\circ\phi_4]$$

$$\text{ou } [\text{fin}((\bar{u}),(T_1))\circ\phi_5] \text{ ou } [\phi_6]$$

Corps d'une procédure :

Nous considérons un mécanisme de passage de paramètres par valeur (la valeur du paramètre effectif est recopiée dans le paramètre formel avant l'appel de procédure) ou par résultat (la valeur du paramètre formel est recopiée dans le paramètre effectif après l'appel de la procédure). Pour simplifier les règles de construction du système d'équations sémantiques en avant associées à une procédure :

- nous considérons que dans le passage de paramètres par valeur, le paramètre effectif est une variable. (On peut obtenir la règle correspondant au cas général d'une expression en considérant que l'appel f(e) est équivalent à début z:t; z:=e; f(z); fin;).

- Nous ne considérerons pas le passage de paramètre par valeur-résultat (puisque l'appel f(x) est équivalent à début z:t; f'(x,z); x:=z fin où le corps de la procédure f' est le même que celui de f sauf qu'il se termine par l'affectation de x à z).

- Nous considérerons aussi qu'aucune variable globale n'est visible dans une procédure (puisque le mécanisme de passage de paramètres peut être utilisé pour accéder à, ou faire un effet de bord sur une variable globale (quand les variables globales accessibles dans le corps de la procédure sont également accessibles en chacun des points d'appel)).

- Nous omettons les fonctions qui sont une simple facilité d'écriture que l'on peut toujours remplacer par une procédure en introduisant un paramètre résultat supplémentaire.

Toutes les restrictions précédentes ne concernent véritablement que des facilités d'écriture et la seule véritable restriction sémantique est la suivante :

- Nous supposerons que l'on peut toujours associer statiquement un corps de procédure à un appel de procédure (ce qui exclut le passage de fonctions ou procédures en paramètre de procédure).

Nous pouvons maintenant énoncer la règle de construction des équations associées au corps d'une procédure :

$\underline{\text{procédure}}\ f(\bar{v}:\bar{t}\ \underline{\text{valeur}};\ \bar{r}:\bar{t}'\ \underline{\text{résultat}}) =$
$\quad <\phi_i = \underline{\text{entrée}}(\overline{T},\overline{T}')\ ,\ (\bar{v}':\overline{T},\bar{v}:\overline{T},\bar{r}:\overline{T}')>$
$\quad \ldots$
$\quad <\phi_j>$
$\underline{\text{fin-proc}};$
$<\phi_k = \underline{\text{sortie}}(\overline{T},\overline{T}')\circ\phi_j>$

où

- Les variables $\bar{v}'$ n'apparaissent pas dans le corps de la procédure,

- $\overline{T}$ et $\overline{T}'$ sont les domaines de valeurs des variables $\bar{v}$ et $\bar{r}$ de types $\bar{t}$ et $\bar{t}'$,

- $\underline{\text{entrée}}(\overline{T},\overline{T}')$
  $\in ((\overline{T}\to B) \to (\overline{T}\times\overline{T}'\times\overline{T}\to B))$
  $= \lambda P.\{\lambda(\bar{v}',\bar{r},\bar{v}).[P(\bar{v}')\ \underline{\text{et}}\ (\bar{v}=\bar{v}')\ \underline{\text{et}}\ (\bar{r}=\Omega)]\}$

- $\underline{\text{sortie}}(\overline{T},\overline{T}')$
  $\in ((\overline{T}\times\overline{T}'\times\overline{T}\to B) \to (\overline{T}\times\overline{T}'\to B))$
  $= \lambda P.\{\lambda(\bar{v}',\bar{r}).[\exists\bar{v}\in\overline{T} : P(\bar{v}',\bar{v},\bar{r})]\}$

  On notera que si $\overline{T}=T_1\times\ldots\times T_p$ et $\overline{T}'=T_{p+1}\times\ldots\times T_q$ alors

  $\underline{\text{sortie}}(\overline{T},\overline{T}')=\sigma^{p+q}_{1,\ldots,q}$

Intuitivement, une spécification des paramètres $\bar{v}$ passés par valeur est connue tandis que les paramètres $\bar{r}$ passés par résultat ne sont pas initialisés. La valeur initiale des paramètres d'entrée $\bar{v}$ est recopiée dans les variables auxiliaires $\bar{v}'$ de sorte qu'à la sortie de la procédure, on connait un prédicat, sur la valeur initiale des paramètres d'entrée et sur les valeurs finales des paramètres d'entrée et de sortie, qui exprime la condition de terminaison de la procédure et la valeur finale des paramètres. Comme la valeur finale des paramètres d'entrée n'est d'aucun intérêt dans le contexte d'appel, les variables $\bar{v}$ sont éliminées à la sortie de la procédure.

Appel de procédure :

Pour éviter d'avoir à choisir un ordre de passage des paramètres résultat, nous supposerons que tous les paramètres effectifs sont des variables syntaxiquement distinctes.

Soit $P(\bar{x},\bar{v},\bar{r})$ un prédicat caractérisant le domaine des variables $\bar{x}$, $\bar{v}$ et $\bar{r}$ avant l'appel de procédure $f(\bar{v},\bar{r})$ qui s'effectue dans l'environnement syntaxique $(\bar{x}:\bar{U},\bar{v}:\bar{T},\bar{r}:\bar{T}')$. Les variables $\bar{x}$ n'interviennent pas dans l'appel et les paramètres $\bar{v}$ passés par valeur ne sont pas modifiés, de sorte que $Q(\bar{x},\bar{v}) = \{\exists\bar{r}\epsilon\bar{T}' : P(\bar{x},\bar{v},\bar{r})\}$ est vrai après l'appel. La spécification des paramètres d'entrée $\bar{v}$ avant l'appel est $\varphi=\lambda\bar{v}.\{\exists\bar{x}\epsilon\bar{U},\exists\bar{r}\epsilon\bar{T}' : P(\bar{x},\bar{v},\bar{r})\}$. Soit $\bar{f}$ le transformateur de prédicat associé à la procédure f, on a vu que $\bar{f}(\varphi)(\bar{v},\bar{r})$ exprime une condition sur les valeurs des paramètres d'entrée $\bar{v}$ pour que l'exécution de f se termine et caractérise le domaine des résultats $\bar{r}$ de la procédure. On a donc :

$$\begin{array}{l}<P(\bar{x},\bar{v},\bar{r}) \ , \ (\bar{x}:\bar{U},\bar{v}:\bar{T},\bar{r}:\bar{T}')> \\ f(\bar{v};\bar{r}); \\ <(\bar{f}(\varphi)(\bar{v},\bar{r}) \ \underline{\text{et}} \ Q(\bar{x},\bar{v})) \ , \ (\bar{x}:\bar{U},\bar{v}:\bar{T},\bar{r}:\bar{T}')>\end{array}$$

où

- $\varphi = \lambda\bar{v}.\{\exists\bar{x}\epsilon\bar{U},\exists\bar{r}\epsilon\bar{T}' : P(\bar{x},\bar{v},\bar{r})\}$
- $Q = \lambda(\bar{x},\bar{v}).\{\exists\bar{r}\epsilon\bar{T}' : P(\bar{x},\bar{v},\bar{r})\}$

Pour plus de précision, nous pouvons énoncer cette règle en détail comme suit :

$$\begin{array}{l}<\phi_i,(x_1:U_1,\ldots,x_n:U_n)> \\ f(x_{i_1},\ldots,x_{i_p};x_{i_{p+1}},\ldots,x_{i_q}); \\ <\phi_j=\underline{\text{appel}}(f,(U_1\times\ldots\times U_n),(i_1,\ldots,i_p),(i_{p+1},\ldots,i_q))\circ\phi_i \ , \\ \qquad\qquad (x_1:U_1,\ldots,x_n:U_n)>\end{array}$$

où

- $\underline{appel}(f,(U_1 x \ldots x U_n),(i_1,\ldots,i_p),(i_{p+1},\ldots,i_q))$

$\in ((U_1 x \ldots x U_n \to B) \to (U_1 x \ldots x U_n \to B))$

$= \lambda P.\{\lambda(x_1,\ldots,x_n).[\phi_k(\sigma^n_{i_1,\ldots,i_p}(P))(x_{i_1},\ldots,x_{i_p},x_{i_{p+1}},\ldots,x_{i_q})$

$\underline{et}\ \Sigma^n_{i_{p+1},\ldots,i_q}(\bar{\sigma}^n_{i_{p+1},\ldots,i_q}(P))(x_1,\ldots,x_n)]\}$

où

- ϕ_k est associé au point de programme suivant le $\underline{fin\text{-}proc}$ du corps de la procédure f,

- Si $(r \leq n)$, $(\forall k \in [1,r],\ 1 \leq i_k \leq n)$, $(\forall k,\ell \in [1,r],\ (k \neq \ell) \Rightarrow (i_k \neq i_\ell))$ et $P \in (U_1 x \ldots x U_{n-r} \to B)$ alors,

$\Sigma^n_{i_1,\ldots,i_r}(P) = \lambda(x_1,\ldots,x_n).[P(v_{j_1},\ldots,v_{j_{n-r}})]$ où $\forall k \in [1,n-r]$,

$j_k = min\{i:\ (i > j_{k-1})\ \underline{et}\ (\forall \ell \in [1,r],\ i \neq i_\ell)\}$ avec $j_0 = 0$.

Exemple 6.1.0.1 :

Nous illustrons l'association d'un système d'équations sémantiques en avant à une procédure récursive sur l'exemple classique de la factorielle :

```
{1}   procédure f(x: entier valeur; y: entier résultat) =
{2}       si x=0 alors
{3}           y:=1;
{4}       sinon
{5}           début z: entier;
{6}               z:=x-1;
{7}               f(z;y);
{8}               y:=x*y;
{9}           fin;
{10}      finsi;
{11}  fin-proc;
```

le système d'équations sémantiques en avant associé à cette procédure est :

$\phi_1 \in$ (entier$\to$B) $\to$ (entier$^3\to$B)

= entrée[(entier),(entier)]

= $\lambda P.\{\lambda(a,y,x).[P(a)$ et $(x=a)$ et $(y=\Omega)]\}$

$\phi_2 \in$ (entier$\to$B) $\to$ (entier$^3\to$B)

= test$(\lambda(a,y,x).(x=0))\circ\phi_1$

= $\lambda P.\{\lambda(a,y,x).[\phi_1(P)(a,y,x)$ et $(x=0)]\}$

$\phi_3 \in$ (entier$\to$B) $\to$ (entier$^3\to$B)

= affectation$(\lambda(a,y,x).(a,1,x))\circ\phi_2$

= $\lambda P.\{\lambda(a,y,x).[\exists m : \phi_2(P)(a,m,x)$ et $(y=1)]\}$

$\phi_4 \in$ (entier$\to$B) $\to$ (entier$^3\to$B)

= test$(\lambda(a,y,x).(x\neq 0))\circ\phi_1$

= $\lambda P.\{\lambda(a,y,x).[\phi_1(P)(a,y,x)$ et $(x\neq 0)]$

$\phi_5 \in$ (entier$\to$B) $\to$ entier$^4\to$B)

= début(entier^3, entier)$\circ\phi_4$

= $\lambda P.\{\lambda(a,y,x,z).[\phi_4(P)(a,y,x)$ et $(z=\Omega)]\}$

$\phi_6 \in$ (entier$\to$B) $\to$ (entier$^4\to$B)

= affectation$(\lambda(a,y,x,z).(a,y,x,x-1))\circ\phi_5$

= $\lambda P.\{\lambda(a,y,x,z\ ).[\exists m : \phi_5(P)(a,y,x,m)$ et $(z=x-1)]\}$

$$\phi_7 \in (\underline{entier}\to\mathcal{B}) \to (\underline{entier}^4\to\mathcal{B})$$
$$= \underline{appel}(f,\underline{entier}^4,(4),(2))\circ\phi_6$$
$$= \lambda P.\{\lambda(a,y,x,z).[\phi_{11}(\sigma_4^4(P))(z,y) \ \underline{et}\ \overline{\sigma}_2^4(P)(a,x,z)]\}\circ\phi_6$$
$$= \lambda P.\{\lambda(a,y,x,z).[\phi_{11}(\sigma_4^4(\phi_6(P)))(z,y) \ \underline{et}\ \overline{\sigma}_2^4(\phi_6(P))(a,x,z)]\}$$
$$\phi_8 \in (\underline{entier}\to\mathcal{B}) \to (\underline{entier}^4\to\mathcal{B})$$
$$= \underline{affectation}(\lambda(a,y,x,z).(a,x*y\ ,x,z))\circ\phi_7$$
$$= \lambda P.\{\lambda(a,y,x,z).[\exists n : \phi_7(P)(a,n,x,z) \ \underline{et}\ (y=x*n)]\}$$
$$\phi_9 \in (\underline{entier}\to\mathcal{B}) \to (\underline{entier}^4\to\mathcal{B})$$
$$= \underline{fin}(\underline{entier}^3, \underline{entier})\circ\phi_8$$
$$= \overline{\sigma}_4^4\circ\phi_8$$
$$\phi_{10} \in (\underline{entier}\to\mathcal{B}) \to (\underline{entier}^3\to\mathcal{B})$$
$$= \phi_3 \ \underline{ou}\ \phi_9$$
$$\phi_{11} \in (\underline{entier}\to\mathcal{B}) \to (\underline{entier}^2\to\mathcal{B})$$
$$= \underline{sortie}[(\underline{entier}),(\underline{entier})]\circ\phi_{10}$$
$$= \overline{\sigma}_3^3\circ\phi_{10}$$

Fin de l'exemple

Les règles de construction du système d'équations sémantiques en avant associé à un programme comportant des procédures récursives peuvent être justifiées par rapport à une sémantique opérationnelle ou dénotationnelle. Mise à part la complexité introduite par la récursivité la démarche est fondamentalement la même qu'au chapitre 3. En particulier la plus petite solution (pour l'implication) du système d'équations

$$\{\ \phi = \lambda\psi.[\![\ \lambda P.\{F_\pi(\psi)(P)\}]\!]\ [\![\phi]\!]$$

associé au programme π s'obtient par le théorème 2.7.0.1 puisque $\lambda\psi.\ [\![\lambda P.\{F_\pi(\psi)(P)\}]\!]$ est un morphisme complet pour la disjonction.

Exemple 6.1.0.2.

Nous résolvons le système d'équations associé à la procédure factorielle de l'exemple 6.1.0.1. Pour les calculs à la main il est plus facile de simplifier le système d'équations, par exemple en éliminant $\phi_1, \ldots, \phi_{10}$ ce qui donne :

$$\begin{cases} \phi_{11} = \lambda\psi.[\![\ \lambda P.\{\lambda(a,y).[(P(a)\ \underline{et}\ (a=0)\ \underline{et}\ (y=1)) \\ \qquad\qquad \underline{ou}\ (\exists n : \psi(\lambda z.[P(z+1)\ \underline{et}\ (z+1 \neq 0)])(a-1,n) \\ \qquad\qquad \underline{et}\ P(a)\ \underline{et}\ (a\neq 0)\ \underline{et}\ (y=a*n)]\}]\!] [\![\phi_{11}]\!] \end{cases}$$

Résolvant itérativement en partant de l'infimum de $((\underline{entier}\to\mathcal{B})\to(\underline{entier}^2\to\mathcal{B}))$, nous obtenons les premiers termes :

$$[\phi_{11}^0 = \lambda P.\{\lambda(a,y).[\underline{faux}]\}$$
$$[\phi_{11}^1 = \lambda P.\{\lambda(a,y).[P(a)\ \underline{et}\ (a=0)\ \underline{et}\ (y=a!)]\}$$
$$[\phi_{11}^2 = \lambda P.\{\lambda(a,y).[P(a)\ \underline{et}\ (0\leq a\leq 1)\ \underline{et}\ (y=a!)]\}$$

L'itération est infinie de sorte que pour passer à la limite nous inventons le terme général ϕ_{11}^k, montrons qu'il est correct par induction finie, et passons à la limite en faisant tendre k vers l'infini. Le théorème 2.7.0.1 montre que cette limite est la plus petite solution (pour l'implication) de l'équation fonctionnelle. Pour le pas d'induction supposons que nous ayons :

$$[\phi_{11}^k = \lambda P.\{\lambda(a,y).[P(a)\ \underline{et}\ (0\leq a\leq (k-1))\ \underline{et}\ (y=a!)]\}$$

alors il vient :

- $\phi_{11}^k\ (\lambda z.[P(z+1)\ \underline{et}\ (z+1 \neq 0)]) =$
  $\lambda(a,y).[P(a+1)\ \underline{et}\ (a+1 \neq 0)\ \underline{et}\ (0\leq a\leq (k-1))\ \underline{et}\ (y=a!)]$
- $\phi_{11}^k\ (\lambda z.[P(z+1)\ \underline{et}\ (z+1 \neq 0)])(a-1,n) = \{P(a)\ \underline{et}\ (1\leq a\leq k)\ \underline{et}\ (n=(a-1)!)\}$
- $(\exists n: \phi_{11}\ (\lambda z.[P(z+1)\ \underline{et}\ (z+1 \neq 0)])(a-1,n)\ \underline{et}\ P(a)\ \underline{et}\ (a\neq 0)\ \underline{et}\ (y=a*n)) =$
  $\{P(a)\ \underline{et}\ (1\leq a\leq k)\ \underline{et}\ (y=a!)\}$

$$[\phi_{11}^{k+1} = \lambda P.\{\lambda(a,y).[P(a)\ \underline{et}\ (0\leq a\leq k)\ \underline{et}\ (y=a!)]\}$$

Passant à la limite nous obtenons :

$$\phi_{11}^{\omega} = \underset{k\in\omega}{OU}\ (\lambda P.\{\lambda(a,y).[P(a)\ \underline{et}\ (0\leq a\leq k)\ \underline{et}\ (y=a!)]\})$$
$$[\phi_{11}^{\omega} = \lambda P.\{\lambda(a,y).[P(a)\ \underline{et}\ (0\leq a)\ \underline{et}\ (y=a!)]\}$$

Remarquons en particulier que la factorielle, telle que nous l'avons écrite ne se termine pas pour des valeurs négatives de son paramètre d'entrée.
Fin de l'exemple.

Exemple 6.1.0.3.
La fonction 91 de MacCarthy

f(x) = si x>100 alors x-10 sinon f(f(x+11)) finsi

qui calcule :

si x>100 alors x-10 sinon 91 finsi

nous permet d'illustrer le cas d'appels récursifs imbriqués. Dans notre langage elle s'écrit :

```
{1}  procédure f(x: entier valeur; y: entier résultat) =
{2}      si x>100 alors
{3}          y:=x-10;
{4}      sinon
{5}          début z: entier;
{6}              z:=x+11;
{7}              début t: entier;
{8}                  f(z;t);
{9}                  f(t;y);
{10}             fin;
{11}         fin;
{12}     finsi;
{13} fin-proc;
```

Le système d'équations sémantiques en avant associé à cette procédure est le suivant :

$$\phi_1 \in (\underline{\text{entier}} \to \mathcal{B}) \to (\underline{\text{entier}}^3 \to \mathcal{B})$$
$$= \underline{\text{entrée}}[(\underline{\text{entier}}), (\underline{\text{entier}})]$$
$$= \lambda P.\{\lambda(a,y,x).[P(a) \ \underline{\text{et}} \ (x=a) \ \underline{\text{et}} \ (y=\Omega)]\}$$

$\phi_2 \in (\underline{entier} \to \mathcal{B}) \to (\underline{entier}^3 \to \mathcal{B})$

$= \underline{test}(\lambda(a,y,x).(x>100)) \circ \phi_1$

$= \lambda P.\{\lambda(a,y,x).[\phi_1(P)(a,y,x) \ \underline{et} \ (x>100)]\}$

$\phi_3 \in (\underline{entier} \to \mathcal{B}) \to (\underline{entier}^3 \to \mathcal{B})$

$= \underline{affectation}(\lambda(a,y,x).(a,x-10,x)) \circ \phi_2$

$= \lambda P.\{\lambda(a,y,x).[\exists m : \phi_2(P)(a,m,x) \ \underline{et} \ (y=x-10)]\}$

$\phi_4 \in (\underline{entier} \to \mathcal{B}) \to (\underline{entier}^3 \to \mathcal{B})$

$= \underline{test}(\lambda(a,y,x).(x\leq 100)) \circ \phi_1$

$= \lambda P.\{\lambda(a,y,x).[\phi_1(P)(a,y,x) \ \underline{et} \ (x\leq 100)]\}$

$\phi_5 \in (\underline{entier} \to \mathcal{B}) \to (\underline{entier}^4 \to \mathcal{B})$

$= \underline{début}((\underline{entier}^3),(\underline{entier})) \circ \phi_4$

$= \lambda P.\{\lambda(a,y,x,z).[\phi_4(P)(a,y,x) \ \underline{et} \ (z=\Omega)]\}$

$\phi_6 \in (\underline{entier} \to \mathcal{B}) \to (\underline{entier}^4 \to \mathcal{B})$

$= \underline{affectation}(\lambda(a,y,x,z).(a,y,x,x+11)) \circ \phi_5$

$= \lambda P.\{\lambda(a,y,x,z).[\exists m : \phi_5(P)(a,y,x,m) \ \underline{et} \ (z=x+11)]\}$

$\phi_7 \in (\underline{entier} \to \mathcal{B}) \to (\underline{entier}^5 \to \mathcal{B})$

$= \underline{début}((\underline{entier}^4),(\underline{entier})) \circ \phi_6$

$= \lambda P.\{\lambda(a,y,x,z,t).[\phi_6(P)(a,y,x,z) \ \underline{et} \ (t=\Omega)]\}$

$\phi_8 \in (\underline{entier} \to \mathcal{B}) \to (\underline{entier}^5 \to \mathcal{B})$

$= \underline{appel}(f,\underline{entier}^5,(4),(5)) \circ \phi_7$

$= \lambda P.\{\lambda(a,y,x,z,t).[\phi_{13}(\sigma^5_4(\phi_7(P)))(z,t) \ \underline{et} \ \overline{\sigma}^5_5(\phi_7(P))(a,y,x,z)]\}$

$\phi_9 \in (\underline{entier} \to \mathcal{B}) \to (\underline{entier}^5 \to \mathcal{B})$

$= \underline{appel}(f,\underline{entier},(5),(2)) \circ \phi_8$

$= \lambda P.\{\lambda(a,y,x,z,t).[\phi_{13}(\sigma^5_5(\phi_8(P)))(t,y) \ \underline{et} \ \overline{\sigma}^5_2(\phi_8(P))(a,x,z,t)]\}$

$\phi_{10} \in (\underline{entier} \to \mathcal{B}) \to (\underline{entier}^4 \to \mathcal{B})$

$= \underline{fin}((\underline{entier}^4),(\underline{entier})) \circ \phi_9$

$= \overline{\sigma}^5_5 \circ \phi_9$

$\phi_{11} \in (\underline{entier} \to \mathcal{B}) \to (\underline{entier}^3 \to \mathcal{B})$

$= \underline{fin}((\underline{entier}^3),(\underline{entier})) \circ \phi_{10}$

$= \overline{\sigma}^4_4 \circ \phi_{10}$

$\phi_{12} \in (\underline{entier} \to \mathcal{B}) \to (\underline{entier}^3 \to \mathcal{B})$

$= \phi_3 \ \underline{ou} \ \phi_{11}$

$\phi_{13} \in (\underline{entier} \to \mathcal{B}) \to (\underline{entier}^2 \to \mathcal{B})$

$= \underline{sortie}[(\underline{entier}),(\underline{entier})] \circ \phi_{12}$

$= \overline{\sigma}^3_3 \circ \phi_{12}$

Ce système d'équations se simplifie comme suit :

$$\begin{cases} \phi_8 = \lambda P.\{\lambda(a,y,x,z,t).[\phi_{13}(\lambda z.[P(z-11) \text{ et } (z\leq 111)])(z,t) \text{ et } P(a) \text{ et } \\ \qquad\qquad (x=a) \text{ et } (y=\Omega) \text{ et } (x\leq 100) \text{ et } (z=x+11)]\} \\ \phi_9 = \lambda P.\{\lambda(a,y,x,z,t).[\phi_{13}(\sigma_5^5(P))(t,y) \text{ et } \overline{\sigma}_2^5(P)(a,x,z,t)]\}\circ\phi_8 \\ \phi_{13} = \lambda P.\{\lambda(a,y).[P(a) \text{ et } (a>100) \text{ et } (y=a-10)]\}) \text{ ou } (\sigma_{1,2}^5 \circ \phi_9) \end{cases}$$

ce qui peut s'écrire :

$$\begin{cases} \phi = \lambda P.\{\lambda(a,y).[P(a) \text{ et } (a>100) \text{ et } (y=a-10)] \text{ ou } \\ \qquad\qquad \sigma_{1,2}^5(\psi(\lambda(a,y,x,z,t).[\phi(\lambda z.[P(z-11) \text{ et } (z\leq 111)])(z,t) \\ \qquad\qquad\qquad\qquad \text{et } P(a) \text{ et } (a\leq 100) \text{ et } (x=a) \\ \qquad\qquad\qquad\qquad \text{et } (y=\Omega) \text{ et } (z=x+11)]))\} \\ \psi = \lambda P.\{\lambda(a,y,x,z,t).[\phi(\sigma_5^5(P))(t,y) \text{ et } \overline{\sigma}_2^5(P)(a,x,z,t)]\} \end{cases}$$

Le plus petit point fixe est calculé par approximations successives en partant de l'infimum :

$$\begin{cases} \phi^0 = \lambda P.\{\lambda(a,y).[\underline{faux}]\} \\ \psi^0 = \lambda P.\{\lambda(a,y,x,z,t).[\underline{faux}]\} \\ \phi^1 = \lambda P.\{\lambda(a,y).[P(a) \text{ et } (a>100) \text{ et } (y=a-10)]\} \\ \phi^2 = \lambda P.\{\lambda(a,y).[P(a) \text{ et } (((a>100) \text{ et } (y=a-10)) \text{ ou } ((a=100) \\ \qquad\qquad \text{et } (y=91)))]\} \end{cases}$$

Avec un peu d'imagination on envisage le terme général de la séquence comme suit :

$$\begin{cases} \phi^k = \lambda P.\{\lambda(a,y).[P(a) \text{ et } \{((a>100) \text{ et } (y=a-10)) \text{ ou } ((k<11) \\ \qquad\qquad \text{et } (102-k\leq a\leq 100) \text{ et } (y=91)) \text{ ou } ((k\geq 11) \\ \qquad\qquad \text{et } (91-11*(k-11)\leq a\leq 100) \text{ et } (y=91))\}]\} \\ \psi^k = \lambda P.\{\lambda(a,y,x,z,t).[\phi^k(\sigma_5^5(P))(t,y) \text{ et } \overline{\sigma}_2^5(P)(a,x,z,t)]\} \end{cases}$$

L'étape d'induction ($k\geq 2$) montre que cette hypothèse est correcte :

$$\phi^k(\lambda z.[P(z-11) \text{ et } (z\leq 111)])(z,t) = [P(z-11) \text{ et } (z\leq 111) \text{ et } \{((z>100) \text{ et } (t=z-10)) \text{ ou } ((k\leq 11) \text{ et } (102-k\leq z\leq 100) \text{ et } (t=91) \text{ ou } ((k\geq 11) \text{ et } (91-11*(k-11)\leq z\leq 100) \text{ et } (t=91))\}]$$

Posons

$$Q = \lambda(a,y,x,z,t).[\phi^k(\lambda z.[P(z-11) \text{ et } (z\leq 111)])(z,t) \text{ et } P(a) \text{ et } (a\leq 100) \text{ et } (x=a) \text{ et } (y=\Omega) \text{ et } (z=x+11)]$$

Après substitutions et simplifications nous obtenons :

$$Q = \lambda(a,y,x,z,t).[P(a) \text{ et } (x=a) \text{ et } (y=\Omega) \text{ et } (z=a+11) \text{ et } \{((90\leq a\leq 100) \text{ et } (t=a+1)) \text{ ou } ((k\leq 11) \text{ et } (91-k\leq a\leq 89) \text{ et } (t=91)) \text{ ou } ((k\geq 11) \text{ et } (91-11*((k+1)-11)\leq a\leq 89) \text{ et } (t=91))\}]$$

Il vient : $\sigma_5^5(Q) = \lambda t.[((91\leq t\leq 100) \text{ et } P(t-1)) \text{ ou } (t=91)]$, ce qui permet d'évaluer

$$\phi^k(\sigma_5^5(Q))(t,y) = ((y=91) \text{ et } (\{P(t-1) \text{ et } \{(t=101) \text{ ou } ((k\leq 11) \text{ et } (102-k\leq t\leq 100)) \text{ ou } ((k\geq 11) \text{ et } (91\leq t\leq 100))\}\} \text{ ou } \{(k\geq 11) \text{ et } (t=91)\}))$$

comme

$$\sigma_{1,2}^5(\psi^k(Q)) = \lambda(a,y).[P(a) \text{ et } (y=91) \text{ et } \{(a=100) \text{ ou } ((k\leq 11) \text{ et } (102-(k+1)\leq a\leq 99)) \text{ ou } ((k\geq 11) \text{ et } (90\leq a\leq 99)) \text{ ou } ((k\geq 11) \text{ et } (a=90)) \text{ ou } ((k=11) \text{ et } (91-k\leq a\leq 89)) \text{ ou } ((k\geq 11) \text{ et } (91-11*(k+1)-11)\leq a\leq 89))\}]$$

nous obtenons :

$$\left[\phi^{k+1} = \lambda P.\{\lambda(a,y).[P(a) \text{ et } \{((a>100) \text{ et } (y=a-10)) \text{ ou } ((k+1\leq 11) \text{ et } (102-(k+1)\leq a\leq 100) \text{ et } (y=91)) \text{ ou } ((k+1\geq 11) \text{ et } (91-11*((k+1)-11)\leq a\leq 100) \text{ et } (y=91))\}]\right\}$$

Passant à la limite, le plus petit point fixe est :

$$\left[\phi^{\omega} = \lambda P.\{\lambda(a,y).[P(a) \text{ et } \{((a>100) \text{ et } (y=a-10)) \text{ ou } ((a\leq 100) \text{ et } (y=91))\}]\right\}$$

Fin de l'exemple.

Exemple 6.1.0.4 :

Ce dernier exemple (calcul des puissances de deux) illustre l'emploi du théorème 4.1.1.0.2 pour calculer à la main une approximation supérieure du plus petit point fixe du système d'équations associé à une procédure ce qui permet de donner une preuve de correction partielle.

```
{1}   procédure f(x: entier valeur; y: entier résultat) =
{2}       y:=1;
{3}   L:
{4}       si x>0 alors
{5}           x:=x-1;
{6}           début z: entier;
{7}               f(x;z);
{8}               y:=y+z;
                  allera L;
              fin;
          finsi;
{9}
{10}  fin-proc;
```

Le système d'équations sémantiques en avant associé à cette procédure est le suivant :

$$\phi_1 \in (\underline{\text{entier}} \to \mathcal{B}) \to (\underline{\text{entier}}^3 \to \mathcal{B})$$
$$= \underline{\text{entrée}}((\underline{\text{entier}}),(\underline{\text{entier}}))$$
$$= \lambda P.\{\lambda(a,y,x).[P(a)\ \underline{\text{et}}\ (x=a)\ \underline{\text{et}}\ (y=\Omega)]\}$$

$$\phi_2 \in (\underline{\text{entier}} \to \mathcal{B}) \to (\underline{\text{entier}}^3 \to \mathcal{B})$$
$$= \underline{\text{affectation}}(\lambda(a,y,x).(a,1,x)) \circ \phi_1$$
$$= \lambda P.\{\lambda(a,y,x).[\exists m : \phi_1(P)(a,m,x)\ \underline{\text{et}}\ (y=1)]\}$$

$$\phi_3 \in (\underline{\text{entier}} \to \mathcal{B}) \to (\underline{\text{entier}}^3 \to \mathcal{B})$$
$$= \phi_2\ \underline{\text{ou}}\ \underline{\text{fin}}((\underline{\text{entier}}^4),(\underline{\text{entier}})) \circ \phi_8$$
$$= \phi_2\ \underline{\text{ou}}\ \overline{\sigma}^4_4 \circ \phi_8$$

$$\phi_4 \in (\underline{\text{entier}} \to \mathcal{B}) \to (\underline{\text{entier}}^3 \to \mathcal{B})$$
$$= \underline{\text{test}}(\lambda(a,y,x).(x>0)) \circ \phi_3$$
$$= \lambda P.\{\lambda(a,y,x).[\phi_3(P)(a,y,x)\ \underline{\text{et}}\ (x>0)]\}$$

$$\phi_5 \in (\underline{\text{entier}} \to \mathcal{B}) \to (\underline{\text{entier}}^3 \to \mathcal{B})$$
$$= \underline{\text{affectation}}(\lambda(a,y,x).[a,y,x-1]) \circ \phi_4$$
$$= \lambda P.\{\lambda(a,y,x).[\phi_4(P)(a,y,x+1)]\}$$

$$\phi_6 \in (\underline{\text{entier}} \to \mathcal{B}) \to (\underline{\text{entier}}^4 \to \mathcal{B})$$
$$= \underline{\text{début}}((\underline{\text{entier}}^3),(\underline{\text{entier}})) \circ \phi_5$$
$$= \lambda P.\{\lambda(a,y,x,z).[\phi_5(P)(a,y,x)\ \underline{\text{et}}\ (z=\Omega)]\}$$

$$\phi_7 \in (\underline{entier}\to\mathcal{B})\to(\underline{entier}^4\to\mathcal{B})$$
$$= \underline{appel}(f,\underline{entier}^4,3,4)\circ\phi_6$$
$$= \lambda P.\{\lambda(a,y,x,z).[\phi_{10}(\sigma_3^4(\phi_6(P)))(x,z) \ \underline{et}\ \overline{\sigma}_4^4(\phi_6(P))(a,y,x)]\}$$
$$\phi_8 \in (\underline{entier}\to\mathcal{B})\to(\underline{entier}^4\to\mathcal{B})$$
$$= \underline{affectation}(\lambda(a,y,x,z).(a,y+z,x,z))\circ\phi_7$$
$$= \lambda P.\{\lambda(a,y,x,z).[\exists m : \phi_7(P)(a,m,x,z) \ \underline{et}\ (y=m+z)]\}$$
$$\phi_9 \in (\underline{entier}\to\mathcal{B})\to(\underline{entier}^3\to\mathcal{B})$$
$$= \underline{test}(\lambda(a,y,x).(x\leq 0))\circ\phi_3$$
$$= \lambda P.\{\lambda(a,y,x).[\phi_3(P)(a,y,x) \ \underline{et}\ (x\leq 0)]\}$$
$$\phi_{10} \in (\underline{entier}\to\mathcal{B})\to(\underline{entier}^2\to\mathcal{B})$$
$$= \underline{sortie}((\underline{entier}),(\underline{entier}))\circ\phi_9$$
$$= \overline{\sigma}_3^3\circ\phi_9$$

Ce système d'équations peut être substantiellement simplifié en :

$$\begin{cases} \phi_3 = F_1(\phi_3,\phi_{10}) \\ \quad = \lambda P.\{\lambda(a,y,x).[(P(a) \ \underline{et}\ (x=a) \ \underline{et}\ (y=1)) \ \underline{ou} \\ \qquad\qquad (\exists m: \phi_{10}(\sigma_3^3(\lambda(a,y,x).[\phi_3(P)(a,y,x+1) \\ \qquad\qquad \underline{et}\ (x\geq 0)]))(x,y-m) \ \underline{et}\ \phi_3(P)(a,m,x+1) \ \underline{et}\ (x\geq 0))]]\} \\ \phi_{10} = F_2(\phi_3,\phi_{10}) \\ \quad = \lambda P.\{\sigma_3^3(\lambda(a,y,x).[\phi_3(P)(a,y,x) \ \underline{et}\ (x\leq 0)])\} \end{cases}$$

Résolvant itérativement à partir de l'infimum :

$$\begin{cases} \phi_3^0 = \lambda P.\{\lambda(a,y,x).[\underline{faux}]\} \\ \phi_{10}^0 = \lambda P.\{\lambda(a,y).[\underline{faux}]\} \end{cases}$$

en utilisant la stratégie chaotique de Gauss-Seidel :

$$\begin{cases} \phi_3^{k+1} = F_1(\phi_3^k,\phi_{10}^k) \\ \phi_{10}^{k+1} = F_2(\phi_3^{k+1},\phi_{10}^k) \end{cases}$$

nous obtenons les premiers termes :

$$\begin{cases} \phi_3^1 = \lambda P.\{\lambda(a,y,x).[P(a)\underline{et}(x=a)\underline{et}(y=1)]\} \\ \phi_{10}^1 = \lambda P.\{\lambda(a,y).[P(a)\underline{et}(a\leq 0)\underline{et}(y=1)]\} \end{cases}$$

$$\begin{cases} \phi_3^2 = \lambda P.\{\lambda(a,y,x).([P(a)\underline{et}(x=a)\underline{et}(y=1)]\ \underline{ou}\ [P(a)\underline{et}(x=0)\underline{et}(a=1)\underline{et}(y=2)])\} \\ \phi_{10}^2 = \lambda P.\{\lambda(a,y).([P(a)\underline{et}(a\leq 0)\underline{et}(y=1)]\ \underline{ou}\ [P(a)\underline{et}(a=1)\underline{et}(y=2)])\} \end{cases}$$

$$\begin{cases} \phi_3^3 = \lambda P.\{\lambda(a,y,x).([P(a)\underline{et}(x=a)\underline{et}(y=1)]\ \underline{ou}\ [P(a)\underline{et}(x=0)\underline{et}(a=1)\underline{et}(y=2)] \\ \qquad\qquad \underline{ou}\ [P(a)\underline{et}(x=1)\underline{et}(a=2)\underline{et}(y=3)])\} \\ \phi_{10}^3 = \lambda P.\{\lambda(a,y).([P(a)\underline{et}(a\leq 0)\underline{et}(y=1)]\ \underline{ou}\ [P(a)\underline{et}(a=1)\underline{et}(y=2)])\} \end{cases}$$

$$\begin{cases} \phi_3^4 = \lambda P.\{\lambda(a,y,x).([P(a)\underline{et}(x=a)\underline{et}(y=1)]\ \underline{ou}\ [P(a)\underline{et}(x=0)\underline{et}(a=1)\underline{et}(y=2)] \\ \qquad\qquad \underline{ou}\ [P(a)\underline{et}(x=0)\underline{et}(a=2)\underline{et}(y=4)] \\ \qquad\qquad \underline{ou}\ [P(a)\underline{et}(x=1)\underline{et}(a=2)\underline{et}(y=3)])\} \\ \phi_{10}^4 = \lambda P.\{\lambda(a,y).([P(a)\underline{et}(a\leq 0)\underline{et}(y=1)]\ \underline{ou}\ [P(a)\underline{et}(a=1)\underline{et}(y=2)] \\ \qquad\qquad \underline{ou}\ [P(a)\underline{et}(a=2)\underline{et}(y=4)])\} \end{cases}$$

$$\begin{cases} \phi_3^5 = \lambda P.\{\lambda(a,y,x).([P(a)\underline{et}(x=a)\underline{et}(y=1)]\ \underline{ou}\ [P(a)\underline{et}(x=0)\underline{et}(a=1)\underline{et}(y=2)] \\ \qquad\qquad \underline{ou}\ [P(a)\underline{et}(x=0)\underline{et}(a=2)\underline{et}(y=4)]\ \underline{ou}\ [P(a)\underline{et}(x=1)\underline{et}(a=2))) \\ \qquad\qquad \underline{et}(y=3)]\ \underline{ou}\ [P(a)\underline{et}(x=2)\underline{et}(a=3)\underline{et}(y=5)])\} \\ \phi_{10}^5 = \lambda P.\{\lambda(a,y).([P(a)\underline{et}(a\leq 0)\underline{et}(y=1)]\ \underline{ou}\ [P(a)\underline{et}(a=1)\underline{et}(y=2)] \\ \qquad\qquad \underline{ou}\ [P(a)\underline{et}(a=2)\underline{et}(y=4)])\} \end{cases}$$

Avec un peu d'intuition le calcul de ces premiers termes permet d'envisager la forme suivante pour le terme général de l'itération :

$$\phi_3^k = \lambda P.\{\lambda(a,y,x).([P(a)\underline{et}(x=a)\underline{et}(y=1)]\ \underline{ou}\ [P(a)\underline{et}(0\leq x<a)\underline{et}(y=1+\sum_{i=x}^{a-1} 2^i)\underline{et}((a*(a+1))/2-x<k)])\}$$

Nous en déduisons :

$$\phi_{10}^k = \lambda P.\{\lambda(a,y).([P(a)\underline{et}(a\leq 0)\underline{et}(y=1)]\ \underline{ou}\ [P(a)\underline{et}(0<a)\underline{et}(y=2^a)\underline{et}((a*(a+1))/2<k)])\}$$

Il est aisé de vérifier que les premiers termes sont de la forme ci-dessus. Pour le pas d'induction il faut montrer que $\phi_3^{k+1} = F_1(\phi_3^k, \phi_{10}^k)$. Les calculs étant complexes nous nous contenterons de vérifier que $F_1(\phi_3^k, \phi_{10}^k) \Rightarrow \phi_3^{k+1}$. Comme il est évident que $\phi_3^k \Rightarrow \phi_3^{k+1}$, on aura

$(\phi_3^k \underline{ou} F_1(\phi_3^k,\phi_{10}^k)) \Rightarrow \phi_3^{k+1}$ ce qui montre que ϕ_3^k est le terme général d'une itération croissante approchée supérieurement (4.1.1.0.1).

$$\phi_{10}^k(\sigma_3^3(\lambda(a,y,x).[\phi_3^k(P)(a,y,x+1)\underline{et}(x\geq 0)]))(x,y-m)$$
$$\Rightarrow \phi_{10}^k(\lambda x.x\geq 0)(x,y-m)$$
$$\Rightarrow [(x=0)\underline{et}(y-m=1)]\ \underline{ou}\ [(0<x)\underline{et}(y-m=2^x)\underline{et}((x*(x+1))/2<k)]$$

$$\phi_3^k(P)(a,m,x+1)\ \underline{et}\ (x\geq 0)$$
$$= [P(a)\underline{et}(x+1=a)\underline{et}(m=1)\underline{et}(x\geq 0)]\ \underline{ou}$$
$$[P(a)\underline{et}(0\leq x<a-1)\underline{et}(m=1+\sum_{i=x+1}^{a-1}2^i)\underline{et}(((a*(a+1))/2-x)<k+1)]$$

donc

$$F_1(\phi_3^k,\phi_{10}^k) \Rightarrow \lambda P.\{\lambda(a,y,x).([P(a)\underline{et}(x=a)\underline{et}(y=1)]$$
$$\underline{ou}\ [P(a)\underline{et}(x=0)\underline{et}(a=1)\underline{et}(y=2)]$$
$$\underline{ou}\ [P(a)\underline{et}(x=0<a-1)\underline{et}(y=1+\sum_{i=x}^{a-1}2^i)$$
$$\underline{et}(((a*(a+1))/2-x)<k+1)]$$
$$\underline{ou}\ [P(a)\underline{et}(0<x=a-1)\underline{et}(y=1+2^{a-1})$$
$$\underline{et}((a*(a-1))/2<k)]$$
$$\underline{ou}\ [P(a)\underline{et}(0<x<a-1)\underline{et}(y=1+\sum_{i=x}^{a-1}2^i)$$
$$\underline{et}(((a*(a+1))/2-x)<k+1)]\}$$
$$\Rightarrow \lambda P.\{\lambda(a,y,x).([P(a)\underline{et}(x=a)\underline{et}(y=1)]$$
$$\underline{ou}\ [P(a)\underline{et}(0\leq x<a)\underline{et}(y=1+\sum_{i=x}^{a-1}2^i)$$
$$\underline{et}(((a*(a+1))/2-x)<k+1)])\}$$
$$\Rightarrow \phi_3^{k+1}$$

Le terme de rang ω de l'itération chaotique croissante approchée supérieurement est donc $\phi_3^\omega = \bigcup_{k<\omega}\phi_3^k$ soit :

$$\left[\begin{array}{l}\phi_3^\omega = \lambda P.\{\lambda(a,y,x).[P(a)\ \underline{et}\ (((x=a)\underline{et}(y=1))\underline{ou}((0\leq x<a)\underline{et}(y=1+\sum_{i=x}^{a-1}2^i)))]\}\\ \quad \text{nous en déduisons :}\\ \phi_{10}^\omega = \lambda P.\{\lambda(a,y).[P(a)\ \underline{et}\ (((a\leq 0)\underline{et}(y=1))\underline{ou}((a>0)\underline{et}(y=1+\sum_{i=0}^{a-1}2^i)))]\}\\ \quad = \lambda P.\{\lambda(a,y).[P(a)\ \underline{et}\ (((a<0)\underline{et}(y=1))\underline{ou}((a\geq 0)\underline{et}(y=?^a)))]\}\end{array}\right.$$

En abrégeant le système d'équations en $\phi=F(\phi)$ nous avons :

$$\phi^\omega = \underset{k<\omega}{\underline{\bigcup}}\phi^k = \underset{k<\omega}{\underline{\bigcup}}\phi^{k+1} \Leftarrow \underset{k<\omega}{\underline{\bigcup}}F(\phi^k) = F(\underset{k<\omega}{\underline{\bigcup}}\phi^k) = F(\phi^\omega)$$

car nous avons montré ($\forall k<\omega$, $F(\phi^k) \Rightarrow \phi^{k+1}$) et F est un morphisme complet pour la disjonction. Comme ϕ^ω est un post-point fixe de F, c'est la limite de l'itération (4.1.1.0.1.(c)) et le théorème 4.1.1.0.2 montre que $lfp(F) \Rightarrow \phi^\omega$. Nous en déduisons donc que la procédure f est partiellement correcte : si x est un entier positif ou nul et f(x,y) se termine alors $y=2^x$. La preuve de terminaison est évidente par récurrence finie.
Fin de l'exemple.

6.2 METHODES CONSTRUCTIVES D'APPROXIMATION DE SOLUTIONS D'UN SYSTEME D'EQUATIONS FONCTIONNELLES

Les techniques développées au chapitre 4 sont directement applicables pour approcher le plus petit point fixe du système d'équations sémantiques en avant :

$$\phi=F(\phi) \quad \text{où } \phi \in (\prod_{i=1}^{n} (((U^m \to B) \to (U^{m_i} \to B)))$$

associé à une procédure récursive (ayant m paramètres d'entrée à valeurs dans U et comportant n points-programmes). Par exemple pour une approximation supérieure nous définirons une image approchée supérieurement L_k de $(U^k \to B)$ pour tout $k \geq 0$ (définition 4.2.7.0.6):

$$L_k \quad \overline{\triangleright}(@_k, \gamma_k) \quad (U^k \to B)$$

de sorte que le paragraphe 4.2.8 nous indique comment construire un système d'équations approché en définissant à l'aide du théorème 4.2.8.0.3 :

$$(L_m \to L_{m_i}) \quad \overline{\triangleright}(@_{m_i,m}, \gamma_{m_i,m}) \quad (((U^m \to B) \to (U^{m_i} \to B))$$

avec

$$@_{m_i,m} = \lambda\phi.(@_{m_i} \circ \phi \circ \gamma_m)$$

$$\gamma_{m_i,m} = \lambda\phi.(\gamma_{m_i} \circ \phi \circ @_m)$$

et

$$\prod_{i=1}^{n} (L_m \to L_{m_i}) \quad \overline{\triangleright}(\overline{@}_n, \overline{\gamma}_n) \quad (\prod_{i=1}^{n} (((U^m \to B) \to (U^{m_i} \to B)))$$

avec

$$\overline{@}_n = \lambda(\phi_1, \ldots, \phi_n).(@_{m_1,m}(\phi_1), \ldots, @_{m_n,m}(\phi_n))$$

$$\overline{\gamma}_n = \lambda(\phi_1, \ldots, \phi_n).(\gamma_{m_1,m}(\phi_1), \ldots, \gamma_{m_n,m}(\phi_n))$$

ce qui donne un système d'équations approché de la forme :

$$\phi = F(\phi) \quad \text{où} \quad \phi \in \left(\prod_{i=1}^{n} (L_m \to L_{m_i}) \right)$$

où les règles de construction de $\overline{F}$ assurent que :

$$\overline{@}_n(F) \sqsubseteq \overline{F}$$

avec

$$\overline{@}_n = \lambda F.\overline{@}_n \circ F \circ \overline{\gamma}_n$$

$$\overline{\gamma}_n = \lambda F.\overline{\gamma}_n \circ F \circ \overline{@}_n$$

Ayant calculé $lfp(\overline{F})$ en utilisant le théorème 2.9.1.0.1 ou une approximation supérieure $\overline{\phi}$ de $lfp(\overline{F})$ en utilisant les techniques d'approximation de points fixes basées sur une accélération de la convergence par extrapolation du paragraphe 4.1 le théorème 4.2.8.0.4 assure que :

$$lfp(F) \Rightarrow \overline{\gamma}_n(\overline{\phi})$$

Soit par exemple à faire une analyse approchée du signe de la fonction :
f(x) = si x≥1000 alors x sinon -f(-2*x) finsi;
en utilisant l'approximation définie au paragraphe 5.3. La fonction qui au signe de x associe le signe de f(x) est le plus petit point fixe de la fonctionnelle :

$$F = \lambda\phi.\{\lambda P.[(P \sqcap \ddagger) \sqcup -\phi(-P)]\}$$

résolvant itérativement nous obtenons :

$$\phi^0 = \lambda P.\perp$$

$$\phi^1 = F(\phi^0) = \lambda P.\{(P \sqcap \ddagger) \sqcup -\perp\} = \lambda P.(P \sqcap \ddagger)$$

$$\begin{aligned}\phi^2 = F(\phi^1) &= \lambda P.\{(P \sqcap \ddagger) \sqcup -((-P) \sqcap \ddagger)\} \\ &= \lambda P.\{(P \sqcap \ddagger) \sqcup (P \sqcap \div)\} \text{ car } (-P) \sqcap \ddagger = -(P \sqcap \div) \text{ et } --P = P \\ &= \lambda P.\{(P \sqcap (\ddagger \sqcup \div))\} \\ &= \lambda P.\{(P \sqcap \top)\} \\ &= \lambda P.P\end{aligned}$$

$$\begin{aligned}\phi^3 = F(\phi^2) &= \lambda P.\{(P \sqcap \ddagger) \sqcup --P\} \\ &= \lambda P.P\end{aligned}$$

Nous venons donc de montrer que le signe de f(x) est le même que le signe de x. Notre démarche est exactement la même qu'au chapitre 5, à la seule différence que les inconnues ϕ_i du système d'équations $\phi=\overline{F}(\phi)$ sont des fonctions $(\phi_i \in (L_m \to L_{m_i}))$ tandis qu'au chapitre 5 nous considérions un

système d'équations $X=\overline{F}(X)$ où $X_i \in L_k$. Ainsi dans l'exemple ci-dessus les calculs portaient sur des fonctions représentées par des lambda-expressions symboliques et non pas sur les éléments du treillis $L_1 = \{\perp, 0, \ddagger, \doteq, \top\}$. En machine cette représentation des fonctions n'est pas pratique car le calcul symbolique est difficilement automatisable et l'on utilisera plus volontiers une représentation des éléments de $L_1 \to L_1$ par une table à cinq entrées. Toutefois dans le cas général la cardinalité de L_1 est grande ou infinie et aucune représentation finie des éléments de $L_1 \to L_1$ n'est commode en machine. Toutefois en pratique il n'est pas nécessaire de connaître la fonction $lfp(F)$ mais beaucoup plus simplement le prédicat $lfp(F)(P)$ pour un certain nombre de conditions d'entrée P. Ceci revient à dire que nous nous contenterons de calculer l'approximation supérieure $\overline{\phi} \in (L_1 \to L_1)$ de $\overline{\mathcal{C}}_1(lfp(F))$ pour un nombre fini d'arguments ce qui permettra de représenter $\overline{\phi}$ par une table. Parmi les idées possibles nous pouvons pour tout x de L_1 envisager d'approcher $\phi(x)$ par $\phi(\top)$ car $x \sqsubseteq \top$ implique par monotonie $\phi(x) \sqsubseteq \phi(\top)$. Mieux encore, nous pouvons calculer $\overline{\phi}(P)$ où P est donné par les divers contextes d'appel. Plutôt que de faire l'analyse de la procédure à la déclaration ceci revient à faire l'analyse pour chaque appel. Par exemple pour calculer $lfp(\overline{F})(\ddagger)$ nous aurons :

$$\phi^0(P) = \perp \qquad \forall P = \{\perp, 0, \ddagger, \doteq, \top\}$$
$$\phi^1(\ddagger) = (\ddagger \sqcap \ddagger) \sqcup \phi^0(\doteq) = \ddagger \sqcup \phi^0(\doteq) = \ddagger \sqcup \perp = \ddagger$$
$$\phi^1(\doteq) = (\doteq \sqcap \ddagger) \sqcup \phi^1(\doteq) = 0 \sqcup \phi^1(\ddagger) = 0 \sqcup \ddagger = 0 \sqcup \doteq = \doteq$$
$$\phi^2(\ddagger) = (\ddagger \sqcap \ddagger) \sqcup \phi^1(\doteq) = \ddagger \sqcup \phi^1(\doteq) = \ddagger \sqcup \doteq = \ddagger$$
$$\phi^2(\doteq) = (\doteq \sqcap \ddagger) \sqcup \phi^2(\doteq) = 0 \sqcup \phi^2(\ddagger) = 0 \sqcup \ddagger = 0 \sqcup \doteq = \doteq$$

Le calcul de $lfp(\overline{F})(\ddagger)$ ne nécessite pas la connaissance complète de la fonction $lfp(\overline{F})$ mais simplement celle de $lfp(\overline{F})(\doteq)$ et $lfp(\overline{F})(\ddagger)$. Nous formalisons maintenant les idées qui viennent d'être présentées (6.2.1) et résolvons ensuite (6.2.2) les problèmes posés par des espaces de cardinalité infinie en s'inspirant de 4.1.

6.2.1 Résolution d'un système d'équations fonctionnelles à point fixe dans un espace fini par itération chaotique

DEFINITION 6.2.1.0.1 *Itération chaotique finie pour un système d'équations fonctionnelles à point fixe*

Soient pour i=1,...,n les treillis complets D_i, D'_i, $L=\prod_{i=1}^{n}(D_i \to D'_i)$ et $F \in \mathit{conts}(L \to L)$. Nous considérons le système d'équations fonctionnelles $\phi=F(\phi)$ qui se détaille en :

$$\left\{\begin{array}{l} \phi_i = F_i(\phi) \quad \text{où} \quad F_i \in (L \to (D_i \to D'_i)) \\ \quad = \lambda(\psi_1,\ldots,\psi_n).\{\lambda P.[f_i(P,\psi_1,\ldots,\psi_n)]\}(\phi_1,\ldots,\phi_n) \\ i = 1,\ldots,n \end{array}\right.$$

- Un *index* est un vecteur $(J_1,\ldots,J_n)$ tel que $\{\forall i \in [1,n],\ J_i \subseteq D_i\}$
- Etant donné un index $J \subseteq \prod_{i=1}^{n} D_i$ nous définissons F_J par :

$$\forall \phi \in L,\ F_J(\phi)=\psi$$

avec

$$\forall i \in [1,n],\ \psi_i=\lambda X.\ \underline{\text{si}}\ X \in J_i\ \underline{\text{alors}}\ F_i(\phi)(X)\ \underline{\text{sinon}}\ \phi_i(X)\ \underline{\text{finsi}};$$

- Soient $\phi \in L$, $Y \in D_j$, nous dirons que l'*évaluation de* $F_j(\phi)(Y)$ *nécessite l'évaluation de* $\phi_i(X)$ ce que nous noterons $F_j(\phi)(Y) \bullet\!\!\to \phi_i(X)$ si et seulement si F_j s'écrit sous la forme $\lambda\psi.\{\lambda P.[\ldots\psi_i(f(P,\psi))\ldots]\}$ et $f(Y,\phi)=X$.
- Soit $V \subseteq \prod_{i=1}^{n} D_i$, nous noterons F-*fermeture*(ϕ,V) le plus petit vecteur (pour l'inclusion $\subseteq$) $\overline{V} \subseteq \prod_{i=1}^{n} D_i$ tel que :

$$\{\forall i \in [1,n],\ \overline{V}_i = V_i \cup \{X \in D_i : \{\exists k \in [1,n],\ \exists Y \in \overline{V}_k : F_k(\phi)(Y) \bullet\!\!\to \phi_i(X)\}\}$$

- *Une itération chaotique finie partant de* $\phi^0 \in L$, *pour* $V \subseteq \prod_{i=1}^{n} D_i$ *et définie par* F *et la suite admissible d'index* J^0, J^1, ..., J^k, ... est une suite ϕ^0, ϕ^1, ..., ϕ^k, ... d'éléments de L telle que :

$$.\{\forall k \geq 1,\ \phi^k = F_{J^{k-1}}(\phi^{k-1})\}$$

$$.\{(\forall i \in [1,n]),(\forall X \in V_i),(\forall k \geq 0),\ (\exists \ell \geq 1) : (X \in J_i^{k+\ell})\ \underline{\text{et}}\ (\forall_j \in [1,n],\ \overline{W}_j \subseteq \bigcup_{p=0}^{\ell-1} J_j^{k+p}),$$

où $(W_i=\{X\})$, $(\forall_j \in [1,n] : (j \neq i),(W_j=\emptyset))$, $(\overline{W}=F\text{-}\mathit{fermeture}(\phi^{k+1},W))\}$

LEMME 6.2.1.0.2

Une itération chaotique finie pour la fonctionnelle F partant de $\phi^0 \epsilon prefp(F)$ est une chaîne ascendante :
$\{\forall k \geq 0,\ \phi^k \subseteq \phi^{k+1} \subseteq F(\phi^k) \subseteq luis(F)(\phi^0)\}$.

Preuve : Comme $\phi^0 \subseteq F(\phi^0) \subseteq luis(F)(\phi^0)$ pour tout i=1,...,n et $X \epsilon D_i$ nous avons $\phi_i^0(X) \subseteq F_i(\phi^0)(X) \subseteq (luis(F)(\phi^0))_i(X)$. Si $X \epsilon J_i^0$ alors $\phi_i^0(X) \subseteq \phi_i^1(X) = F_i(\phi^0)(X)$ sinon $X \notin J^0$ auquel cas $\phi_i^0(X) = \phi_i^1(X) \subseteq F_i(\phi^0)(X)$.

Pour le pas d'induction supposons que pour k>0 nous ayons $\phi^{k-1} \subseteq \phi^k \subseteq F(\phi^{k-1}) \subseteq luis(F)(\phi^0)$. Pour tout i=1,...,n et tout $X \epsilon D_i$ nous avons soit $X \epsilon J_i^{k-1}$ auquel cas $\phi_i^k(X) = F_i(\phi^{k-1})(X) \subseteq F_i(\phi^k)(X) \subseteq (luis(F)(\phi^0))_i(X)$ car F_i est monotone ou $X \notin J^{k-1}$ et dans ce cas $\phi_i^k(X) = \phi_i^{k-1}(X) \subseteq F_i(\phi_i^{k-1})(X) \subseteq F_i(\phi^k)(X) \subseteq (luis(F)(\phi^0))_i(X)$ ce qui montre que $\phi_i^k \subseteq F_i(\phi^k) \subseteq (luis(F)(\phi^0))_i$. Maintenant pour tout i=1,...,n et tout X de D_i nous avons soit $X \epsilon J_i^k$ et $\phi_i^k(X) \subseteq F_i(\phi^k)(X) = \phi_i^{k+1}(X)$ ou $X \notin J_i^k$ et $\phi_i^k(X) = \phi_i^{k+1}(X) \subseteq F_i(\phi^k)(X)$ ce qui montre que $\phi^k \subseteq \phi^{k+1} \subseteq F(\phi^k) \subseteq luis(F)(\phi^0)$.
Par récurrence finie sur k le lemme est démontré.
Fin de la preuve.

THEOREME 6.2.1.0.3

Soit $\phi^0 \epsilon prefp(F)$ tel que $\phi^0 \subseteq lfp(F)$ alors une itération chaotique finie pour la fonctionnelle F partant de ϕ^0 pour $V \subseteq \prod_{i=1}^{n} D_i$ et stationnaire après ε pas est telle que :

$$\{(\forall i \epsilon [1,n]),\ (\forall X \epsilon V_i),\ \phi_i^{\varepsilon}(X) = (lfp(F))_i(X)\}$$

Preuve : Posons $W^{\varepsilon} = F\text{-}fermeture(\phi^{\varepsilon}, V)$. Soient $i \epsilon [1,n]$ et $X \epsilon V_i$. Par définition d'une suite admissible d'index $\exists \ell \geq 1$ tel que $X \epsilon J_i^{\varepsilon+\ell}$ et donc $\phi_i^{\varepsilon+\ell+1}(X) = F_i(\phi^{\varepsilon+\ell})(X)$ et comme $\phi_i^{\varepsilon+\ell+1} = \phi_i^{\varepsilon+\ell} = \phi_i^{\varepsilon}$ nous avons $\phi_i^{\varepsilon} = F_i(\phi_i^{\varepsilon})$. Soit maintenant $X \epsilon (W_i^{\varepsilon} - V_i)$ alors $\exists k \epsilon [1,n]$, $\exists Y \epsilon V_k$ tels que $F_k(\phi^{\varepsilon})(Y) \mapsto \phi_j^{\varepsilon}(Z)$ et $F_j(\phi^{\varepsilon})(Z) \mapsto \ldots \mapsto \phi_i^{\varepsilon}(X)$ ce qui implique $\exists \ell \geq 1$ tel que $Y \epsilon J_k^{\varepsilon+\ell}$ et $X \epsilon F\text{-}fermeture(\phi^{\varepsilon+1}, (\emptyset, \emptyset, \ldots, \{Y\}, \ldots, \emptyset))_i$ où $\{Y\}$ est en kième position. Par conséquent $\exists p \epsilon [0, \ell-1]$ tel que $X \epsilon J_i^{\varepsilon+p}$ auquel cas $\phi_i^{\varepsilon+p+1}(X) = F_i(\phi^{\varepsilon+p})(X)$ soit $\phi_i^{\varepsilon}(X) = F_i(\phi^{\varepsilon})(X)$. Comme $V_i \subseteq W_i^{\varepsilon}$ nous avons donc

montré que $\{\forall X \in W_i^\varepsilon,\ \phi_i^\varepsilon(X) = F_i(\phi^\varepsilon)(X)\}$.

Soit $\psi^0 \in L$ tel que pour tout $i=1,\ldots,n$ et tout $X \in D_i$ nous ayons $\psi_i^0(X) =$ <u>si</u> $X \in W_i$ <u>alors</u> $\phi_i^\varepsilon(X)$ <u>sinon</u> $\perp$ <u>finsi</u>. Alors $\psi^0 \in prefp(F)$ et d'après 6.2.1.0.2 $\psi^0 \sqsubseteq luis(F)(\phi^0) = lfp(F)$ et d'après le théorème 2.7.0.1 nous avons $lfp(F) = \bigsqcup_{k\in\omega} \psi^k$ où $\psi^k = F(\psi^{k-1})$. Montrons que pour tout $k \in \omega$ on a $F\text{-}fermeture(\psi^0, W^\varepsilon) = F\text{-}fermeture(\psi^k, W^\varepsilon) = W^\varepsilon$ et $\forall i \in [1,n],\ \forall X \in W_i^\varepsilon$, $\psi_i^k(X) = \psi_i^0(X)$. Pour $k=0$ nous avons $F\text{-}fermeture(\psi^0, W^\varepsilon) = F\text{-}fermeture(\phi^0, W^\varepsilon) = F\text{-}fermeture(\phi^0, V) = W^\varepsilon$. Supposons le lemme vrai jusqu'à k. On a $\psi_i^{k+1}(X) = F_i(\psi^k)(X)$. Pour tout $Z \in D_j$ tel que $F_i(\psi^k)(X) \mapsto \psi_j^k(Z)$ nous savons par hypothèse d'induction que $Z \in W_j^\varepsilon$ et donc $\psi_j^k(Z) = \psi_j^0(Z)$ ce qui implique $\psi_j^k(Z) = \psi_j^0(Z)$ et $\psi_i^{k+1}(X) = F_i(\psi^k)(X) = F_i(\psi^0)(X) = \psi_i^0(X)$. De plus $F\text{-}fermeture(\psi^{k+1}, W^\varepsilon) = W^\varepsilon$ car pour tout $Y \in W_j^\varepsilon$, $F_j(\psi^{k+1})(Y) \mapsto \phi_i(X)$ implique que $F_j(\psi^0)(Y) \mapsto \phi_i(X)$ et donc $X \in F\text{-}fermeture(\psi^0, W^\varepsilon) = W^\varepsilon$. Par récurrence sur k nous avons $\forall X \in W_i^\varepsilon,\ \psi_i^k(X) = \phi_i^\varepsilon(X)$ et donc $(lfp(F))_i(X) = \bigsqcup_{k\in\omega} \psi_i^k(X) = \phi_i^\varepsilon(X)$.

Fin de la preuve

Dans le cas où les D_i', $i=1,\ldots,n$ sont des treillis satisfaisant la condition de chaîne ascendante alors l'itération chaotique est stationnaire mais il se peut que pour satisfaire à la définition 6.2.1.0.1 toute suite admissible d'index ait à contenir un index avec une composante infinie. C'est par exemple le cas de l'équation $\phi = \lambda\psi.\{\lambda X.[\psi(X+1)]\}(\phi)$ où $\phi \in (L \to L)$ avec $L = Z \cup \{\perp, \top\}$ ordonné par $\perp \sqsubseteq \perp \sqsubseteq X \sqsubseteq X \sqsubseteq \top \sqsubseteq \top$ pour tout $X \in Z$ et $\perp+1 = \perp$ et $\top+1 = \top$. En pratique, la définition 6.2.1.0.1 n'est donc applicable que pour des index à composantes finies, ce qui par exemple est le cas si les $D_1, \ldots, D_n$ sont des treillis finis.

Nous abordons maintenant le cas général en reprenant les algorithmes d'approximation de points fixes basés sur une accélération de la convergence par extrapolation du paragraphe 4.1 qui sont adaptés au cas de fonctionnelles.

6.2.2 Itération chaotique croissante avec élargissement supérieur pour approcher la solution d'un système d'équations fonctionnelles

DEFINITION 6.2.2.0.1 *Itération chaotique croissante avec élargissement supérieur pour un système d'équations fonctionnelles à point fixe*

Soient pour i=1,...,n les treillis complets D_i, D'_i, $L=\prod_{i=1}^{n}(D_i\to D'_i)$ et $F\in mon(L\to L)$. Soient $\widetilde{F}\in(L\to L)$ telle que $F\sqsubseteq\widetilde{F}$ et pour i=1,...,n les élargissements supérieurs $\overline{\nabla}_i\in(D'_i\times D'_i\to D'_i)$ satisfaisant la définition 4.1.2.0.4.

- Etant donné un index $J\subseteq\prod_{i=1}^{n}D_i$ nous définissons $\widetilde{F}_J$ par $\forall\phi\in L$, $\widetilde{F}_J(\phi)=\psi$ avec $\forall i\in[1,n]$, $\psi_i=\lambda X.$ *si* $X\in J_i$ *alors* $\phi_i(X)\overline{\nabla}_i\widetilde{F}_i(\phi)(X)$ *sinon* $\phi_i(X)$ *finsi*.

- Une *itération chaotique croissante partant de* $\phi^0\in L$, *pour* $V\subseteq\prod_{i=1}^{n}D_i$ *et définie par* $\widetilde{F}$ *et la suite admissible d'index* $J^0, J^1, \ldots, J^k,\ldots$ est une suite $\phi^0, \phi^1, \ldots, \phi^k,\ldots$ d'éléments de L telle que :

$$\{\forall k\geq 1,\ \phi^k=\widetilde{F}_{J^{k-1}}(\phi^{k-1})\}$$

$$\{\{(\forall i\in[1,n]),(\forall X\in V_i),(\forall k\geq 0),(\exists \ell\geq 1)\ :\ (X\in J_i^{k+\ell})\ \underline{et}\ (\forall_j\in[1,n],\ \overline{W}_j\subseteq\bigcup_{p=0}^{\ell-1}J_j^{k+p}),$$

où $(W_i=\{X\})$, $(\forall_j\in[1,n]\ :\ (j\neq i),(W_j=\emptyset))$, $(\overline{W}=\widetilde{F}\text{-}fermeture(\phi^{k+1},W))\}\}$.

THEOREME 6.2.2.0.2

Une itération chaotique croissante partant de $\phi^0\in L$, pour $V\subseteq\prod_{i=1}^{n}D_i$ et définie par F et $\widetilde{F}$ est une chaîne ascendante stationnaire dont la limite ϕ^ε est telle que :

$$\{(\forall i\in[1,n]),\ (\forall X\in V_i),\ (lfp(F))_i(X)\sqsubseteq\phi_i^\varepsilon(X)\}$$

Preuve : D'après 4.1.2.0.4.(a) nous avons $\forall X\in D_i$, $\phi_i(X)\sqsubseteq\phi_i(X)\overline{\nabla}_i\widetilde{F}_i(\phi)(X)$ ce qui montre que $\widetilde{F}_J$ est extensive pour tout $k\geq 0$ et par conséquent la suite $\phi^0, \phi^1, \ldots, \phi^k,\ldots$ est une chaîne ascendante. Comme X apparaît une infinité de fois dans la suite $J^0, \ldots, J^k,\ldots$ il existe une suite $i_0, i_1, \ldots, i_k$ telle que $\forall j\in[1,n]$, $\phi_j^{i_{k+1}}(X)=\phi_j^{i_k}(X)\overline{\nabla}_j C_j$ où $C_j\in D'_j$ ce qui montre d'après 4.1.2.0.4(b) que la suite $\phi^0, \ldots, \phi^k,\ldots$ est stationnaire.

Posons $W^\varepsilon=\widetilde{F}\text{-}fermeture(\phi^\varepsilon,V)$. Pour tout i=1,...,n et tout $X\in V_i$ nous avons $\exists\ell\geq 1$ tel que $X\in J_i^{\varepsilon+\ell}$ et donc $\phi_i^{\varepsilon+\ell+1}(X)=\phi_i^{\varepsilon+\ell}(X)\overline{\nabla}_i\widetilde{F}_i(\phi^{\varepsilon+\ell})(X)$ soit $\phi_i^\varepsilon(X)=\phi_i^\varepsilon(X)\overline{\nabla}_i\widetilde{F}_i(\phi^\varepsilon)(X)\sqsupseteq\phi_i^\varepsilon(X)\sqcup\widetilde{F}_i(\phi^\varepsilon)(X)\sqsupseteq\phi_i^\varepsilon(X)$ et donc

$\tilde{F}_i(\phi^\varepsilon)(X) \subseteq \phi_i^\varepsilon(X)$. Supposons maintenant que $X\in(W_i^\varepsilon - V_i)$ alors $\exists k\in[1,n]$, $\exists Y\in V_k$ tel que $F_k(\phi^\varepsilon)(Y) \mapsto \phi_j^\varepsilon(Z)$ et $F_j(\phi^\varepsilon)(Z) \mapsto \ldots \mapsto \phi_i^\varepsilon(X)$ ce qui implique $\exists \ell\geq 1$: $Y\in J_k^{\varepsilon+\ell}$ et $X\in F\text{-}\textit{fermeture}(\phi^{\varepsilon+1},(\emptyset,\emptyset,\ldots,\{Y\},\ldots,\emptyset))_i$ où $\{Y\}$ est en kième position. Par conséquent $\exists p\in[0,\ell-1]$ tel que $X\in J_i^{\varepsilon+p}$ et donc $\phi_i^{k+p+1}(X) = \phi_i^{\varepsilon+p}(X)\overline{\nabla}_i\tilde{F}_i(\phi^\varepsilon)(X)$ ce qui implique comme précédemment que $\tilde{F}_i(\phi^\varepsilon)(X) \subseteq \phi_i^\varepsilon(X)$.

Définissons ψ telle que $\forall i\in[1,n], \psi_i=\lambda X.$ <u>si</u> $X\in W_i^\varepsilon$ <u>alors</u> $\phi_i^\varepsilon(X)$ <u>sinon</u> τ <u>fi</u>; ψ est un post-point fixe de $\tilde{F}$ et comme $F\sqsubseteq\tilde{F}$, ψ est un post-point fixe de F de sorte que le théorème 2.5.5.0.1 implique $lfp(F)\sqsubseteq\psi$ ce qui montre $\forall i\in[1,n]$, $\forall X\in V_i\subseteq W_i^\varepsilon$, $(lfp(F))_i(X) \subseteq \phi_i(X)$.
Fin de la preuve.

6.3 EXEMPLES D'ANALYSE SEMANTIQUE APPROCHEE EN AVANT DES PROCEDURES RECURSIVES.

Ce paragraphe reprend quelques applications décrites au chapitre 5 pour illustrer l'approximation supérieure de la plus petite solution d'un système d'équations sémantiques fonctionnelles en avant associé à une procédure récursive.

6.3.1 Cas d'un espace de propriétés approchées fini

6.3.1.1 Signe des variables d'une procédure

Exemple 6.3.1.1.0.1

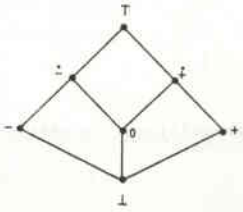
Considérons la procédure suivante :

```
{1}   procédure f(x: entier valeur; y: entier résultat)=
{2}       si x≥1000 alors
{3}           y:=x;
{4}       sinon
{5}           début z: entier;
{6}               z:=-2*x;
{7}               f(z;y);
{8}               y:=-y;
{9}           fin;
{10}      finsi;
{11}  fin-proc;
```

Le système d'équations sémantiques en avant qui lui est associé est le suivant :

$$
\begin{cases}
\phi_1 = \lambda P.\{\lambda(a,y,x).[P(a) \text{ et } (x=a) \text{ et } (y=\Omega)]\} \\
\phi_2 = \text{test}(\lambda(a,y,x).(x\geq 1000))\circ\phi_1 \\
\phi_3 = \text{affectation}(\lambda(a,y,x).(a,x,x))\circ\phi_2 \\
\phi_4 = \text{test}(\lambda(a,y,x).(x<1000))\circ\phi_3 \\
\phi_5 = \lambda P.\{\lambda(a,y,x,z).[P(a,y,x) \text{ et } (z=\Omega)]\} \\
\phi_6 = \text{affectation}(\lambda(a,y,x,z).(a,y,x,-2*x))\circ\phi_5 \\
\phi_7 = \lambda P.\{\lambda(a,y,x,z).[\phi_{11}(\sigma_4^4(P))(z,y) \text{ et } \bar{\sigma}_2^4(P)(a,x,z)]\}\circ\phi_6 \\
\phi_8 = \text{affectation}(\lambda(a,y,x,z).(a,-y,x,z))\circ\phi_7 \\
\phi_9 = \bar{\sigma}_4^4\circ\phi_8 \\
\phi_{10} = \phi_3 \text{ ou } \phi_9 \\
\phi_{11} = \sigma_{1,2}^3\circ\phi_{10}
\end{cases}
$$

Choisissant la fermeture définie au paragraphe 5.2.1 un prédicat P portant sur n variables est approché par un vecteur de n éléments du treillis :



Le système d'équations approchées correspondant est le suivant (nous noterons $P = (P(1),\ldots,P(n))$ quand $P \in L^n$) :

$$\phi_1 \in (L^1 \to L^3)$$
$$= \lambda P.(P(1),\perp,P(1))$$
$$\phi_2 \in (L^1 \to L^3)$$
$$= \lambda P.\{\phi_1(P) \sqcap (\top,\top,+)\}$$
$$\phi_3 \in (L^1 \to L^3)$$
$$= \lambda P.\{\phi_2(P)(y \leftarrow x)\}$$
$$\phi_4 \in (L^1 \to L^3)$$
$$= \phi_3$$
$$\phi_5 \in (L^1 \to L^4)$$
$$= \lambda P.(P(1),P(2),P(3),\perp) \circ \phi_4$$
$$\phi_6 \in (L^1 \to L^4)$$
$$= \lambda P.\{\phi_5(P)(z \leftarrow x)\}$$
$$\phi_7 \in (L^1 \to L^4)$$
$$= \lambda P.\{(\top,\phi_{11}(P(4))(2),\top,\phi_{11}(P(4))(1)) \sqcap (P(1),\top,P(3),P(4))\} \circ \phi_6$$
$$\phi_8 \in (L^1 \to L^4)$$
$$= \lambda P.\{\phi_7(P)(y \leftarrow -y)\}$$
$$\phi_9 \in (L^1 \to L^3)$$
$$= \lambda P.(P(1),P(2),P(3)) \circ \phi_8$$
$$\phi_{10} \in (L^1 \to L^3)$$
$$= \lambda P.\{\phi_3(P) \sqcup \phi_9(P)\}$$
$$\phi_{11} \in (L^1 \to L^2)$$
$$= \lambda P.(P(1),P(2)) \circ \phi_{10}$$

Ce système d'équations peut être simplifié comme suit :

$$\phi_{11} \in (L^1 \to L^2)$$
$$= \lambda P.(P(1),\{(P(1) \sqcap +) \sqcup -\phi_{11}(-P(1))(1)\})$$

ou plus simplement :

$$\begin{cases} \phi_{11} \in (L \to L) \\ \quad = \lambda x.\{(x \sqcap +) \sqcup -\phi_{11}(-x)\} \end{cases}$$

Soit à calculer $\phi_{11}(\dot{+})$ en utilisant la définition 6.2.1.0.1. Nous avons :

Pas 0:

$$\phi_{11}^0 = \lambda x.\perp$$

Pas 1: $J_0 = (\{\dot{+}\})$

$$\phi_{11}^1(\dot{+}) = (\dot{+} \sqcap +) \sqcup -\phi_{11}^0(-(\dot{+})) = + \sqcup \perp = +$$

Pas 2: $J_1 = (\{\dot{-}\})$

$$\phi_{11}^2(\dot{-}) = (\dot{-} \sqcap +) \sqcup -\phi_{11}^1(-(\dot{-})) = \perp \sqcup -\phi_{11}^1(\dot{+}) = -$$

Pas 3: $J_2 = (\{\dot{+}\})$

$$\phi_{11}^3(\dot{+}) = (\dot{+} \sqcap +) \sqcup -\phi_{11}^2(-\dot{+}) = + \sqcup -\phi_{11}^2(\dot{-}) = + \sqcup -(-) = + \sqcup + = +$$

Pas 4: $J_3 = (\{\dot{-}\})$

$$\phi_{11}^4(\dot{-}) = (\dot{-} \sqcap +) \sqcup -\phi_{11}^3(-(\dot{-})) = \perp \sqcup -\phi_{11}^3(\dot{+}) = -$$

Le calcul converge et donc $\phi_{11}(\dot{+}) = +$ et $\phi_{11}(\dot{-}) = -$.

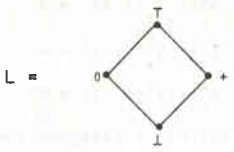
Fin de l'exemple.

Exemple 6.3.1.1.0.2

Un exemple plus complexe est donné par la fonction d'Ackermann définie sur les entiers naturels par :

```
f(x,y) = si x=0 alors
             y+1
         sinonsi y=0 alors
             f(x-1;1)
         sinon
             f(x-1;f(x,y-1))
         finsi;
```

En choisissant l'espace de propriétés approchées :



il faut résoudre :

$$\begin{cases} \phi_1 = \lambda(x,y).[\mathit{incr}(y) \sqcup \phi_1(\mathit{decr}(x \sqcap +),+) \sqcup \phi_2(x \sqcap +, \phi_1(x \sqcap +, \mathit{decr}(y \sqcap +)))] \\ \phi_2 = \lambda(x,t).[\phi_1(\mathit{decr}(x),t)] \end{cases}$$

où

$\mathit{incr} = \lambda x.\underline{\text{cas}}\ x\ \underline{\text{dans}}\ \bot\to\bot\ ;\ 0\to +\ ;\ +\to +\ ;\ \top\to +\ \underline{\text{fincas}}$;

$\mathit{decr} = \lambda x.\underline{\text{cas}}\ x\ \underline{\text{dans}}\ \bot\to\bot\ ;\ 0\to\bot\ ;\ +\to\top\ ;\ \top\to\top\ \underline{\text{fincas}}$;

La valeur de $\phi_1(\top,\top)$ peut être calculée par une itération chaotique finie pour $V=(\{(\top,\top)\},\emptyset)$ et partant de :

$$\begin{cases} \phi_1^0 = \lambda(x,y).\bot \\ \phi_2^0 = \lambda(x,y).\bot \end{cases}$$

Pas 1: $J_0 = (\{(\top,\top),(+,\top),(\top,+)\},\emptyset)$

$\phi_1^1(\top,\top) = + \sqcup \phi_1^0(\top,+) \sqcup \phi_2^0(+,\phi_1^0(+,\top)) = +$

$\phi_1^1(+,\top) = + \sqcup \phi_1^0(\top,+) \sqcup \phi_2^0(+,\phi_1^0(+,\top)) = +$

$\phi_1^1(\top,+) = + \sqcup \phi_1^0(\top,+) \sqcup \phi_2^0(+,\phi_1^0(+,\top)) = +$

et $F\text{-}\mathit{fermeture}(\phi^1,(\{(\top,\top)\},\emptyset)) = (\{(\top,\top),(\top,+),(+,\top)\},\{(+,\bot)\})$

Pas 2: $J_1 = (\emptyset,\{(+,+)\})$

$\phi_2^2(+,+) = \phi_1^1(\top,+) = +$

Pas 3: $J_2 = (\{(\tau,\tau),(+,\tau),(\tau,+)\},\emptyset)$

$\phi_1^3(\tau,\tau) = + \sqcup \phi_1^2(\tau,+) \sqcup \phi_2^2(+,\phi_1^2(+,\tau))$

$= + \sqcup \phi_1^1(\tau,+) \sqcup \phi_2^2(+,\phi_1^1(+,\tau)) = +$

$\phi_1^3(+,\tau) = + \sqcup \phi_1^2(\tau,+) \sqcup \phi_2^2(+,\phi_1^2(+,\tau)) = +$

$\phi_1^3(\tau,+) = + \sqcup \phi_1^1(\tau,+) \sqcup \phi_2^2(+,\phi_1^1(+,\tau)) = +$

et F-*fermeture*$(\phi^3,(\{(\tau,\tau)\},\emptyset)) = (\{(\tau,\tau),(\tau,+),(+,\tau)\},\{(+,+)\})$

Pas 4: $J_3 = (\emptyset,\{(+,+)\})$

$\phi_2^4(+,+) = \phi_1^3(\tau,+) = +$

Pas 5: $J_4 = (\{(\tau,\tau)\})$

$\phi_1^5(\tau,\tau) = + \sqcup \phi_1^4(\tau,+) \sqcup \phi_2^4(+,\phi_1^4(+,\tau))$

$= + \sqcup \phi_1^3(\tau,+) \sqcup \phi_2^4(+,\phi_1^3(+,\tau)) = +$

et F-*fermeture*$(\phi^5,(\{(\tau,\tau)\},\emptyset)) = (\{(\tau,\tau),(\tau,+),(+,\tau)\},\{(+,+)\})$

Nous avons démontré automatiquement que si la fonction d'Ackermann est appelée avec des arguments entiers naturels alors il en est de même dans les appels récursifs subséquents et le résultat est un entier strictement positif.

Fin de l'exemple.

Remarque 6.3.1.1.0.3 *Détermination d'une suite admissible d'index*

En pratique la suite admissible d'index est déterminée en cours de calcul, par exemple en utilisant l'algorithme suivant :

A chaque pas k, on évalue $\phi_i(X)$ pour tous les $X \in V_i$ et quand $F_i(X,\phi) \overset{*}{\mapsto} \phi_j(Y)$, on détermine la valeur Z de $\phi_j(Y)$ comme suit :

- Si $\phi_j(Y)$ a déjà été évalué au pas k, Z est la valeur correspondante ;
- Sinon si $\phi_j(Y)$ est en cours d'évaluation (c'est-à-dire $F_j(Y,\phi) \overset{*}{\mapsto} \phi_j(Y)$) alors si k>1 la valeur de Z est celle de $\phi_j(Y)$ au pas k-1 sinon k=1 et Z est l'infimum $\perp$ de D_j ;
- Sinon Z est la valeur de $F_j(Y,\phi)$.

Fin de la remarque.

6.3.1.2 Pointeurs nuls et pointeurs non nuls

Considérons la procédure *inverser*(L,nil,L') qui donne une copie L' de l'image inverse d'une liste linéaire chaînée L :

```
type noeud = enregistrement
                val : entier;
                suiv: ↑noeud;
             fin;

procédure inverser(x,y: ↑noeud valeur; z: ↑noeud résultat)=
    si x=nil alors
        z:=y;
    sinon
        début t: ↑noeud;
              t:=allouer(noeud);
              t.val := x.val;
              t.suiv := y;
              inverser(x.suiv,t;z);
        fin;
    finsi;
fin-proc;
```

Reprenant 5.6.1.1, le système d'équations approchées associé à *inverser* est :

$$\phi_1 = \lambda(x,y).[\underline{cas}\ x\ \underline{dans}\ \bot \to \bot;\ nil \to y;\ \neg nil \to \sigma_3^{*3}(\phi_2(\neg nil,y,\bot));$$
$$\top \to y \sqcup \sigma_3^{*3}(\phi_2(\top,y,\bot))\ \underline{fincas};$$
$$\phi_2 = \lambda(x,y,z).[\overline{\sigma}_4^{*4}(\phi_3(x,y,z,\neg nil))]$$
$$\phi_3 = \lambda(x,y,z,t).[(x,y,\phi_1(suiv(x),t),t)]$$

où

$$\sigma_i^{*n}\{(x_1,\ldots,x_i,\ldots,x_n)\} = x_i$$
$$\overline{\sigma}_i^{*n}\{(x_1,\ldots,x_i,\ldots,x_n)\} = (x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n)$$
$$suiv = \lambda p.\underline{cas}\ p\ \underline{dans}\ \bot \to \bot;\ nil \to \bot;\ \neg nil \to \top;\ \top \to \top\ \underline{fincas};$$

Evaluant $\phi_1(\neg nil,nil)$ en déterminant une suite admissible d'index par l'algorithme 6.3.1.1.0.3 nous obtenons :

Pas 1:

$$\phi_1(\neg nil, nil) = \sigma_3^{*3}(\phi_2(\neg nil, nil, \bot))$$

$$= \sigma_3^{*3}(\bar{\sigma}_4^{*4}(\phi_3(\neg nil, nil, \bot, \neg nil)))$$

$$= \sigma_3^{*3}(\bar{\sigma}_4^{*4}(\neg nil, nil, \phi_1(\top, \neg nil), \neg nil))$$

$$= \sigma_3^{*3}(\bar{\sigma}_4^{*4}(\neg nil, nil, (\neg nil \sqcup \sigma_3^{*3}(\phi_2(\top, \neg nil, \bot))), \neg nil))$$

$$= \sigma_3^{*3}(\bar{\sigma}_4^{*4}(\neg nil, nil, (\neg nil \sqcup \sigma_3^{*3}(\bar{\sigma}_4^{*4}(\phi_3(\top, \neg nil, \bot, \neg nil)))), \neg nil))$$

$$= \sigma_3^{*3}(\bar{\sigma}_4^{*4}(\neg nil, nil, (\neg nil \sqcup \sigma_3^{*3}(\bar{\sigma}_4^{*4}(\top, \neg nil, \phi_1(\top, \neg nil), \neg nil))), \neg nil))$$

Comme $\phi_1(\top, \neg nil)$ est en cours d'évaluation, il est approché par $\bot$:

$$= \sigma_3^{*3}(\bar{\sigma}_4^{*4}(\neg nil, nil, (\neg nil \sqcup \sigma_3^{*3}(\bar{\sigma}_4^{*4}(\top, \neg nil, \bot, \neg nil))), \neg nil))$$

$$= \neg nil$$

Pas 2:

$$\phi_1(\neg nil, nil) = \sigma_3^{*3}(\bar{\sigma}_4^{*4}(\neg nil, nil, \phi_1(\top, \neg nil), \neg nil))$$

$$= \sigma_3^{*3}(\bar{\sigma}_4^{*4}(\neg nil, nil, (\neg nil \sqcup \sigma_3^{*3}(\bar{\sigma}_4^{*4}(\top, \neg nil, \phi_1(\top, \neg nil), \neg nil))), \neg nil))$$

Comme $\phi_1(\top, \neg nil)$ est en cours d'évaluation il est approché par la valeur $\neg nil$ obtenue au pas précédent :

$$= \sigma_3^{*3}(\bar{\sigma}_4^{*4}(\neg nil, nil, (\neg nil \sqcup \sigma_3^{*3}(\bar{\sigma}_4^{*4}(\top, \neg nil, \neg nil, \neg nil))), \neg nil))$$

$$= \neg nil$$

Noter que les calculs peuvent être réordonnés pour correspondre à la définition 6.2.1.0.1.

En ce qui concerne l'exemple proprement dit nous avons automatiquement découvert que *inverser*(L,nil,L') retourne L' différent de nil quand L n'est pas nil.

6.3.1.3 Pointeurs repérant des enregistrements distincts

Reprenons l'exemple de la procédure *inverser:*

```
{1}  procédure inverser(x,y: ↑noeud valeur; z: ↑noeud résultat)=
{2}      si x=nil alors
{3}          z:=y;
{4}      sinon
{5}          début t: ↑noeud;
{6}              t:=allouer(noeud); t.val:=x.val;
{7}              t.suiv:=y;
{8}              inverser(x.suiv,t;z);
{9}          fin;
{10}     finsi;
{11} fin-proc;
```

en lui appliquant l'analyse approchée du paragraphe 5.6.1.2. L'image approchée d'un prédicat portant sur n variables de type pointeurs $x_1,\ldots,x_n$ est une application de la forme $\lambda\{x_1,\ldots,x_n\}.P$ où P est une partition de $\{x_1,\ldots,x_n\}$. Rappelons la convention que si x_i et y_j sont dans des parties distinctes, elles ne permettent pas de référencer, même indirectement, le même enregistrement. Nous noterons :

$$\{/X_1,\ldots,X_n/\ldots/Y_1,\ldots,Y_m/\} + \{Z\} = \{/X_1,\ldots,X_n/\ldots/Y_1,\ldots,Y_m/Z/\}$$

$$\{/X,X_1,\ldots,X_n/\ldots/Y_1,\ldots,Y_m/\} - \{X\} = \{/X_1,\ldots,X_n/\ldots/Y_1,\ldots,Y_m/\}$$

Le système d'équations approché associé à *inverser* est alors :

$$\begin{cases}
\phi_1 = \lambda C.(\lambda(a,b,z,x,y).[[C(x,y)+\{a,b,z\}) \sqcup \{/a,x/b,y/z/\}]) \\
\phi_2 = \lambda C.(\lambda(a,b,z,x,y).[\varepsilon(x,C(a,b,z,x,y))]) \circ \phi_1 \\
\phi_3 = \lambda C.(\lambda(a,b,z,x,y).[\varepsilon(z,C(a,b,z,x,y)) \sqcup \{/a/b/x/y,z/\}]) \circ \phi_2 \\
\phi_4 = \phi_1 \\
\phi_5 = \lambda C.(\lambda(a,b,z,x,y,t).[C(a,b,x,y,z)+\{t\}]) \circ \phi_4 \\
\phi_6 = \lambda C.(\lambda(a,b,z,x,y,t).[\varepsilon(t,C(a,b,z,x,y,t))]) \circ \phi_5 \\
\phi_7 = \lambda C.(\lambda(a,b,z,x,y,t).[C(a,b,z,x,y,t) \sqcup \{/a/b/z/x/t,y/\}]) \circ \phi_6 \\
\phi_8 = \lambda C.(\lambda(a,b,z,x,y,t).[[\phi_{11}\ (\lambda(x,t).[C(a,b,z,x,y,t)-\{a,b,z,y\}])(x,t,z) \\
\qquad +\{a,b,y\}] \sqcup \varepsilon(z,C(a,b,z,x,y,t))]) \circ \phi_7 \\
\phi_9 = \lambda C.(\lambda(a,b,z,x,y).[C(a,b,z,x,y,t)-\{t\}]) \circ \phi_8 \\
\phi_{10} = \phi_3 \sqcup \phi_9 \\
\phi_{11} = \lambda C.(\lambda(a,b,z).[C(a,b,z,x,y)-\{x,y\}]) \circ \phi_{10}
\end{cases}$$

Résolvons ce système d'équations pour la spécification C=λ(x,y).{/x/y/} correspondant à l'appel *inverser*(L,nil,L').

Pas 1:

$\phi_1(C) = \lambda(a,b,z,x,y).[[\{/x/y/\}+\{a,b,z\}) \sqcup \{/a,x/b,y/z/\}]$
$= \lambda(a,b,z,x,y).\{/a,x/b,y/z/\}$

$\phi_2(C) = \lambda(a,b,z,x,y).[\varepsilon(x,\{/a,x/b,y/z/\})]$
$= \lambda(a,b,z,x,y).\{/a/x/b,y/z/\}$

$\phi_3(C) = \lambda(a,b,z,x,y).[\varepsilon(z,\{/a/x/b,y/z/\}) \sqcup \{/a/b/x/y,z/\}]$
$= \lambda(a,b,z,x,y).\{/a/x/b,y,z/\}$

$\phi_4(C) = \lambda(a,b,z,x,y).\{/a,x/b,y/z/\}$

$\phi_5(C) = \phi_6(C)$
$= \lambda(a,b,z,x,y,t).\{/a,x/b,y/z/t/\}$

$\phi_7(C) = \lambda(a,b,z,x,y,t).(\{/a,x/b,y/z/t/\} \sqcup \{/a/b/z/x/t,y/\})$
$= \lambda(a,b,z,x,y,t).\{/a,x/b,y,t/z/\}$

Comme $\lambda(x,t).[\phi_7(C)(a,b,z,x,y,t)-\{a,b,z,y\}]$
$= \lambda(x,t).[/a,x/b,y,t/z/-\{a,b,z,y\}]$
$= \lambda(x,t).\{/x/t/\} = C$

et la première approximation de $\phi_{11}(C)$ étant l'infimum $\lambda(a,b,z).[\{/a/b/z/\}]$ nous avons :

$\phi_8(C) = \lambda(a,b,z,x,y,t).[(\lambda(a,b,z).[\{/a/b/z/\}](x,t,z)+\{a,b,y\}) \sqcup \varepsilon(z,\{/a,x/b,y,t/z/\})]$

$= \lambda(a,b,z,x,y,t).\{/a,x/b,y,t/z/\}$

$\phi_9(C) = \lambda(a,b,z,x,y).[\{/a,x/b,y,t/z/\}-\{t\}]$

$= \lambda(a,b,z,x,y).\{/a,x/b,y/z/\}$

$\phi_{10}(C) = \phi_3(C) \sqcup \phi_9(C)$

$= \lambda(a,b,z,x,y).[\{/a/x/b,y,z/\} \sqcup \{/a,x/b,y/z/\}]$

$= \lambda(a,b,z,x,y).\{/a,x/b,y,z/\}$

$\phi_{11}(C) = \lambda(a,b,z).[\{/a,x/b,y,z/\}-\{x,y\}]$

$\lambda(a,b,z).\{/a/b,z/\}$

Pas 2:

$\phi_1(C) = \phi_4(C) = \lambda(a,b,z,x,y).\{/a,x/b,y/z/\}$

$\phi_2(C) = \lambda(a,b,z,x,y).\{/a/x/b,y/z/\}$

$\phi_3(C) = \lambda(a,b,z,x,y).\{/a/x/b,y,z/\}$

$\phi_5(C) = \phi_6(C) = \lambda(a,b,z,x,y,t).\{/a,x/b,y/z/t/\}$

$\phi_7(C) = \lambda(a,b,z,x,y,t).\{/a,x/b,y,t/z/\}$

A nouveau $\lambda(x,t).[\phi_7(C)(a,b,z,x,y,t)-\{a,b,z,y\}]$

$= \lambda(x,t).\{/x/t/\} = C$

et la valeur de $\phi_{11}(C)$ étant $\lambda(a,b,z).\{/a/b,z/\}$ au pas précédent nous avons :

$\phi_8(C) = \lambda(a,b,z,x,y,t).\{/a,x/b,y,t/z/\}$

$\phi_9(C) = \lambda(a,b,z,x,y).\{/a,x/b,y/z/\}$

$\phi_{10}(C) = \lambda(a,b,z,x,y).\{/a,x/b,y,z/\}$

$\phi_{11}(C) = \lambda(a,b,z).\{/a/b,z/\}$

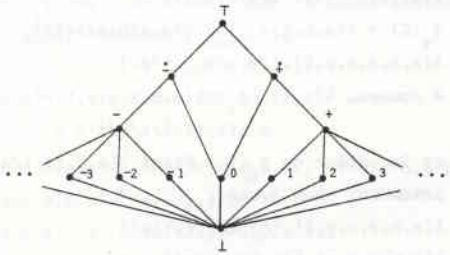
Nous avons donc découvert automatiquement qu'après l'appel *inverser*(L,nil;L'), les références L et L' ne peuvent pas repérer même indirectement le même enregistrement. Des informations similaires sont disponibles en chaque point de la procédure pour la spécification d'entrée $\lambda(x,y).\{/x/y/\}$.

6.3.2 Cas d'un espace de propriétés approchées satisfaisant la condition de chaîne ascendante

Considérons la procédure :

```
procédure factorielle(x,y,z: entier valeur; f: entier résultat)=
     si x=y alors
          f:=z;
     sinon
          factorielle(x,y+1,z*(y+1);f);
     finsi;
fin-proc;
```

telle que *factorielle*(n,0,1;f) délivre f=n! quand n≥0. Nous proposons de l'analyser sur l'espace de propriétés approchées :



Ignorant le test (x=y) il faut résoudre :

$$\phi = F(\phi) = \lambda(x,y,z).[z \sqcup \phi(x,y+1,z*(y+1))]$$

Il est clair que $\phi(7,0,1) \mapsto \phi(7,1,1) \mapsto \phi(7,2,4) \mapsto \ldots \mapsto \phi(7,k,k!) \mapsto \ldots$ de sorte que toute suite admissible d'index doit contenir un index avec une composante de cardinalité infinie.
Nous proposons donc d'approcher F par $\tilde{F}$ tel que $F \sqsubseteq \tilde{F}$ de sorte que $lfp(F) \sqsubseteq lfp(\tilde{F})$ (th. 4.3.0.1). En choisissant :

$$\tilde{\phi} = \tilde{F}(\tilde{\phi}) = \lambda(x,y,z).[z \sqcup \tilde{\phi}(x \sqcup x, y \sqcup (y+1), z \sqcup (z*(y+1))]$$

chaque fois que $\tilde{\phi}(x,y,z) \mapsto \tilde{\phi}(x',y',z')$ nous avons $(x,y,z) \sqsubseteq (x',y',z')$. Comme le treillis des propriétés approchées satisfait la condition de chaîne ascendante une telle dérivation est nécessairement finie.

$$\begin{aligned}
\tilde{\phi}(7,0,1) &= 1 \sqcup \tilde{\phi}(7,0 \sqcup 1,1 \sqcup 1) = 1 \sqcup \tilde{\phi}(7,\ddagger,1) \\
&= 1 \sqcup (1 \sqcup \tilde{\phi}(7,\ddagger,+)) \\
&= 1 \sqcup (1 \sqcup (+ \sqcup \tilde{\phi}(7,\ddagger,+))) = 1 \sqcup (1 \sqcup (+ \sqcup \perp)) = + \\
\tilde{\phi}(7,0,1) &= 1 \sqcup (1 \sqcup \tilde{\phi}(7,\ddagger,+)) \\
&= 1 \sqcup (1 \sqcup (+ \sqcup \tilde{\phi}(7,\ddagger,+))) = 1 \sqcup (1 \sqcup (+ \sqcup +)) = +
\end{aligned}$$

et plus généralement $\tilde{\phi}(\tau,0,1)=+$ ce qui montre que *factorielle*(n,0,1;f) délivre un entier f strictement positif quand elle termine.

6.3.3 Cas général d'un espace de propriétés approchées infini ne satisfaisant pas la condition de chaîne ascendante

Comme exemple d'espace de propriétés approchées ne satisfaisant pas la condition de chaîne ascendante, nous considérons le treillis des intervalles d'entiers du paragraphe 5.7.1. Soit à résoudre l'équation fonctionnelle :

$$\left\{ \phi_1 = \lambda x.\{\phi_1(x+[1,1])\} \right.$$

Elle illustre le problème déjà rencontré au paragraphe précédent d'un domaine de paramètres non convergent, par exemple
$\phi_1([0,255]) \mapsto \phi_1([1,256]) \mapsto \phi_1([2,257]) \mapsto \ldots$ que nous traitons en approchant supérieurement ϕ_1 par $\tilde{\phi}_1$ défini par :

$$\left\{ \tilde{\phi}_1 = \lambda x.\{\phi_1(x \,\overline{\nabla}\, (x+[1,1]))\} \right.$$

que nous résolvons en utilisant le théorème 6.2.1.0.3. Nous obtenons :

Pas 1:

$$\tilde{\phi}_1^1([0,255]) = \tilde{\phi}_1([0,255] \,\overline{\nabla}\, [1,256])$$
$$= \tilde{\phi}_1^1([0,+\infty])$$
$$= \tilde{\phi}_1([0,+\infty] \,\overline{\nabla}\, [2,+\infty]) = \tilde{\phi}_1^0([0,+\infty]) = \perp$$

Pas 2:

$$\tilde{\phi}_1^2([0,255]) = \tilde{\phi}_1^2([0,+\infty]) = \tilde{\phi}_1^1([0,+\infty]) = \perp$$

Considérons maintenant l'équation fonctionnelle :

$$\left\{ \phi_2 = \lambda x.\{[0,0] \sqcup ([1,1]+\phi_2(x))\} \right.$$

qui illustre le problème déjà rencontré au paragraphe 5.7.3 d'une itération non convergente. En effet le calcul de $\phi_2([0,255])$ consiste à résoudre $x=[0,0] \sqcup ([1,1]+x)$ où $x=\phi_2([0,255])$. Nous traitons ce problème en résolvant :

$$\left\{ \tilde{\phi}_2 = \lambda x.\{\tilde{\phi}_2(x) \,\overline{\nabla}\, ([0,0] \sqcup ([1,1]+\tilde{\phi}_2(x)))\} \right.$$

ce qui donne :

Pas 1:

$$\tilde{\phi}_2^1([0,255]) = \tilde{\phi}_2^0([0,255]) \,\overline{\nabla}\, ([0,0] \sqcup ([1,1]+\tilde{\phi}_2^0([0,255])))$$
$$= \perp \,\nabla\, ([0,0] \sqcup \perp) = [0,0]$$

Pas 2:

$$\tilde{\phi}_2^2([0,255]) = \tilde{\phi}_2^1([0,255]) \,\overline{\nabla}\, ([0,0] \sqcup ([1,1]+\tilde{\phi}_2^1([0,255])))$$
$$= [0,0] \,\overline{\nabla}\, ([0,0] \sqcup [1,1]) = [0,+\infty]$$

Pas 3:

$$\tilde{\phi}_2^3([0,255]) = \tilde{\phi}_2^2([0,255]) \,\overline{\nabla}\, ([0,0] \sqcup ([1,1]+\tilde{\phi}_2^2([0,255])))$$
$$= [0,+\infty] \,\overline{\nabla}\, ([0,0] \sqcup [1,+\infty]) = [0,+\infty]$$
$$\tilde{\phi}_2([0,255]) \subseteq \tilde{\phi}_2^2([0,255]) = [0,+\infty].$$

Dans le cas général les deux phénomènes se rencontrent en même temps, ils se résolvent comme précédemment dans le cadre du théorème 6.2.2.0.2. Par exemple l'analyse de la fonction 91 de MacCarthy (6.1.0.3) consiste à

résoudre :

$$\left\{ \phi_1 = \lambda x.[((x \sqcap [101,+\infty]) - [10,10]) \sqcup \phi_1(\phi_1((x \sqcap [-\infty,100]) + [11,11]))] \right.$$

que l'on approche supérieurement par :

$$\left\{ \tilde{\phi}_1 = \lambda x.[\tilde{\phi}_1(x) \overline{\nabla} \{((x \sqcap [101,+\infty]) - [10,10]) \sqcup \tilde{\phi}_1(x \overline{\nabla} \tilde{\phi}_1(x \overline{\nabla} ((x \sqcap [-\infty,100]) + [11,11])))\}] \right.$$

et permet de découvrir que la fonction 91 délivre un résultat supérieur ou égal à 91 :

$$\left[\tilde{\phi}_1([-\infty,+\infty]) = [91,+\infty] \right.$$

6.4 NOTES BIBLIOGRAPHIQUES

Les résultats de ce chapitre améliorent Cousot & Cousot[1977d] particulièrement en ce qui concerne le traitement des branchements inconditionnels (un système d'équations est maintenant associé à un programme en utilisant la notion de point programme et non plus par induction sur la structure syntaxique du programme). L'exemple 6.3.1.3 est également traité plus rigoureusement.

Au paragraphe 6.1 l'emploi de variables auxiliaires (permettant de mémoriser les valeurs initiales) pour exprimer les assertions intermédiaires dans les procédures récursives nous semble à postériori indispensable comme il l'est dans les règles de preuves de procédures introduites par Hoare[1971] et généralisées par Igarashi et al.[1975], Ernst[1977], Apt & de Bakker[1977], Guttag et al.[1977] pour que le système axiomatique soit complet (de Bakker & Meertens[1975], Cook[1975], Gorelick[1975], Apt & Meertens[1977], Apt et al.[1977], Clarke[1977]). Ceci permet aussi de définir le sens d'une procédure récursive indépendamment d'un appel particulier. Noter que pour définir la sémantique des procédures nous n'avons pas utilisé la technique des substitutions syntaxiques (qui semble de puissance limitée, de Bakker[1977d]) et que pour autant nous n'avons pas eu recours au concept de "continuation" comme dans Milne[1977]. Ceci est

possible tant qu'un corps de procédure peut être associé statiquement à tout nom de procédure et exclut l'appel par nom, le passage de procédures ou fonctions en paramètres, les coroutines etc.... Un autre élément de comparaison avec Milne[1977] est que nous associons un système d'équations non pas à un langage mais à un programme ce qui est utile pour raisonner sur les techniques d'approximation.

L'analyse automatique de propriétés sémantiques de procédures récursives a été très peu étudiée. Nous pouvons citer Sintzoff[1972] qui étudie la vérification manuelle de propriétés à l'aide d'une exécution symbolique, Wegbreit[1975] qui traite les procédures en faisant une expansion du corps de la procédure à chaque appel et Karr[1975] qui propose une exécution symbolique sur le programme itératif correspondant à la compilation de la procédure à l'aide d'une pile de récursivité.

Dans le cas particulier des techniques booléennes classiques d'optimisation de programmes on peut citer Spillman[1971], Allen[1974], Lomet[1975], Rosen[1975] et Barth[1977].

7. CONCLUSIONS

Des éléments de conclusion ayant été donnés dans chaque chapitre, nous discutons maintenant quelques axes de travail.

L'approche du point fixe à l'étude du comportement des systèmes dynamiques discrets (paragraphe 3.1) ainsi que l'étude de méthodes d'approximation de points fixes (chapitre 4) font bien apparaître que les techniques d'analyse sémantique des programmes peuvent être étudiées indépendamment des langages utilisés pour la programmation. Toutefois, nous avons consacré la plus grande partie de notre étude au cas des systèmes dynamiques discrets déterministes parce que les programmes déterministes sont les plus fréquents dans les applications en Informatique. Le cas des programmes non-déterministes et parallèles, qui méritent également d'être étudiés, est apparemment plus complexe. Les systèmes dynamiques discrets offrent certainement un cadre suffisamment abstrait pour permettre une étude indépendante des problèmes de langages (qui en général, compliquent plutôt qu'ils ne facilitent la compréhension des problèmes de programmation non-déterministe).

Les problèmes de langages n'interviennent réellement que lorsqu'il s'agit de faire l'analyse sémantique de programmes écrits dans un langage particulier. Sur ce point, notre travail doit être complété pour tenir compte des problèmes posés d'une part, par les structures d'information complexes et d'autre part, par les structures de contrôle dynamiques (qui ne permettent pas un partitionnement statique de l'ensemble des états). Il est quelquefois difficile de déduire un sémantique déductive d'une sémantique de plus bas niveau ou informelle. Ce travail pourrait être entrepris pour les langages de programmation existants qui sont les plus couramment utilisés. Il semble en effet, qu'il ne soit pas possible de faire des raisonnements rigoureux sur les programmes écrits dans un langage s'il n'est pas possible de définir la sémantique déductive de ce langage.

Le chapitre 4 définit potentiellement, grâce aux notions de fermeture et d'extrapolation, toutes les analyses approchées et éventuellement automati-

sables qu'on peut envisager pour les programmes. Toutefois, ce point de vue est théorique et dans les applications pratiques il reste un travail important qui est nécessaire pour trouver un bon niveau d'approximation, c'est-à-dire offrant un bon rapport coût / précision de l'analyse. Il est certain que de nombreuses applications peuvent être développées pour les langages classiques, comme Algol 68 qui réunit la plupart des difficultés qu'on peut rencontrer avec les autres langages de programmation.

Il est peut-être plutôt préférable, d'envisager de nouveaux langages ou traits de langages orientés vers la résolution des problèmes d'analyse sémantique exacte ou approchée des programmes. Par exemple, notre étude fait ressortir que les déclarations locales sont plus utiles que les déclarations globales, que l'analyse des propriétés des objets manipulés par un programme dépend des opérations éffectuées sur ces objets (point de vue des 'types abstraits') mais aussi et surtout du contexte dans lequel ces opérations sont effectuées, que l'extension d'un langage de programmation devrait s'accompagner des informations nécessaires à l'analyse des traits de langage en extension, que le type des objets peut être analysé avec plus ou moins de finesse et qu'il existe une hiérarchie entre les notions de type et d'assertions. Ces idées, qu'il est nécessaire d'approfondir, pourraient guider la conception d'un langage de programmation, ce qui constituerait un point de vue souvent négligé dans le développement des langages de très haut niveau.

Dans les chapitres 2 et 4, nous avons étudié la construction et l'approximation de points fixes d'opérateurs monotones sur un treillis d'un point de vue purement algébrique. Il n'empêche que l'analogie avec les méthodes d'analyse numérique nous a été utile et peut certainement encore être exploitée avec succès. Il est possible d'améliorer l'éfficacité des méthodes itératives et d'approfondir notre notion d'extrapolation. De nouvelles méthodes (pas nécessairement itératives) de résolution des équations sémantiques approchées peuvent être imaginées. Notre hypothèse d'équations monotones sur un treillis complet convenait bien à nos problèmes. Elle est un peu forte pour certains problèmes qui font intervenir une négation non monotone. Il faut donc réfléchir à des hypothèses plus faibles que la monotonie. Il est certain qu'il faut également envisager des hypothèses plus fortes, car nous avons constaté dans les applications que certaines hypothèses spécifiques à des applications particulières, sont pourtant utiles pour imaginer des méthodes de résolution.

B. BIBLIOGRAPHIE

ABIAN S. & BROWN A.B. [1971], *A theorem on partially ordered sets with applications to fixed point theorems,* Canad. J. Math•• 13(1961), 78-82.

ABTROUN A. [1977]. *Recherche d'une permutation optimale des variables dans la de Gauss-Seidel,* ThËse de 3ème cycle. Universite Scientifique et Medicale de Grenoble. (Mai 1977).

ACHACHE A. [1969]. *Structure de l'ensemble des fermetures d'un treillis complet,* Portugal. Math•• 28(1969) . 111-119.

AHO A.V. & ULLMAN J.D. [1976]. *Node listings for reducible flow graphs,* J. Computer and Systems Sciences 13, 3(1976). 286-299.

AHO A.V. & ULLMAN J.D. [1977], *Principles of compiler design,* Addison Wesley Pub. Co., (1977).

ALLEN F.E. [1970], *Control flow analysis,* SIGPLAN Notices 5. 7(1970). 1-19.

ALLEN F.E. [1971], *A basis for program optimization,* Proc. IFIP Congress 71, Vol.1. North-Holland Pub. Co.• Amsterdam. (1971). 385-390.

ALLEN F.E. [1974]. *Interprocedural data flow analysis,* Proc. IFIP Congress 74, North-Holland Pub. Co•• Amsterdam. (1974), 398-402.

ALLEN F.E. & COCKE J. [1972]. *Graph theoretic constructs for program flow analysis,* IBM Res. Rep. RC-3923. T.J. Watson Research Center, Yorktown Heights. N.Y.•• U.S.A., (July 1972).

AMANN H. [1976]. *Fixed point equations and non-linear eigenvalue problems in ordered Banach spaces,* SIAM Review. 18(Oct. 1976) . 620-709.

APT K.R. & de BAKKER J.W. [1977]. *Semantics and proof theory of PASCAL procedures,* Tech. Rep•• Stichting Mathematisch Centrum, Amsterdam. (1977) .

APT K.R. & MEERTENS L. [1977], *Completeness with finite systems of intermediate assertions for recursive program schemes,* Report IW-84/77, Mathematisch Centrum. Amsterdam. (1977).

APT K.R•• BERGSTRA J.A. & MEERTENS L.G. [1977]. *Recursive assertions are not enough or are they?* Report IW-92/77. Mathematisch Centrum, Amsterdam. (1977).

BASU S.K. & YEH R.T. *[1975].Strong verification of* programs, Res. Rep. Soft. Eng. and Syst. Lab•• Univ. of Texas at Austin. (June 1975).

BARTH J.M. [1977]. *An interproceduraL data fLow anaLysis aLgorithm,* Proc. of the Fourth ACM Symp. on Principles of Programming Languages. Los Angeles, Calif.• U.S.A•• (Jan. 1977l, 119-131.

BAUOET G. [1976], *Asynchronous iterative methods for muLtiprocessors,* Research Report. Carnegie Mellon Univ•• Pittsburgh. PA•• (Nov. 1976). (à paraitre dans JACM).

BAUER A. & SAAL H. [1974], *Does APL reaLLy need run-time checking,* Software Practice and Experience 4. 2(1974).

BEKIC H. [1969], *DefinabLe operations in generaL aLgebras and the theory of automata and fLowcharts,* Manuscript. IBM Lab•• Vienne, (1969).

BERGE C. [1973], *Graphes et hypergraphes,* Ounod, Paris (1973).

BIRD R. [1976]. *Programs and machines: an introduction to the theory of computation,* Wiley & Sons. London, (1976).

BIRKHOFF G. [1967], *Lattice theory,* AMS Colloquim Publications. XXV, Third edition. Providence, R.I., U.S.A., (1967l.

O. [1977a], *Prograrrming Languages: formaL development of interpreters and compiLers,* Int. Compo Symp., North-Holland Pub. Co., (1977).

BJ(O)RNER O. [1977b], *Programming languages: linguistics and semantics,* Int. Compo Symp., North-Holland Pub. Co•• (1977l.

BOOM H. [1974]. *Optimization analysis of* programs *in languages with pointer variables,* Ph.D. Thesis, Dept. Appl. Math. and Compo Science. University of Waterloo. U.S.A•• (1974).

BOURBAKI N. [1967]. *Théorie des ensembLes,* Livre I. Chap. III. Fas. XX. Ed. Hermann, 2ème édition, Paris, (1967).

BURSTALL R.M. [1969]. *Proving properties of* programs *by structural induction,* Computer Journal 12. (1969). 41-48.

BURSTALL R.M. [1974], *Program proving as hand simulation with a little induction,* Proc. IFIP Congress 74. Software. North-Holland Pub. Co. Amsterdam. (1974). 308-312.

CAPLAIN M [1975], *Finding invariant assertion for proving* programs, Proe. Int. Conf. on Reliable Software. Los Angeles, Calif•• U.S.A•• (April 1975). 165-171.

CARTWRIGHT R. & OPPEN D.C. [1978], *Unrestricted caLLs in Hoare's logic,* Conf. Rec. of the 5th ACM Symp. on Principles of Programming Languages. Tucson. Ariz.• U.S.A., (Jan. 1978). 131-140.

M [1975]. *Itérations chaotiques* sur *un produit d'espaces métriques,* Thèse de 3ème cycle. Lyon. (1975).

CHAZAN O. & MIRANKER W. [1969]. *Chaotic relaxation,* Linear Algebra and its Appl. 2(1969J. 199-222.

CHURCH A. [1951]. *The calculi of lambda-conversion,* Annals of Math. Studies. 6(1951J.

CLARKE E.M. Jr. [1977]. *Program invariants as fixed points,* Proc. 18th Annual Symp. on Foundations of Computer Science. Providence. R.I.• U.S.A.• (Dct.31-Nov.2 1977). 18-29.

CLINT M. & HOARE C.A.R. [1972]. *Program proving jumps and functions,* Acta Informatica. 1(1972J. 214-224.

COCKE J. [1970]. *GLobaL common subexpressions elimination,* SIGPLAN Notices 5. 7(1970), 20-24.

COMTE P. [1976], *Algorithmes de* Thèse de 3ème cycle. Besançon, (1976J.

COOK S.A. [1975], *Axiomatic and interpretative semantics for an ALGOL fragment,* Tech. Rep.79, Dept. of Compo Science, U. of Toronto, Canada (1975J.

COOPER D.C. [1971]. *Programs for mechanical program verification,* Machine Intelligence 6. American Elsevier. New York, (1971J, 43-59.

COUSOT P. [1974], *Definition interprétative et implantation de langages de programmation,* Thèse Oocteur-Ingenieur. Universite Scientifique et Médicale de Grenoble, (Déc. 1974).

COUSOT P. [1976]. *The system implementation language LIS, an introduction,* édité par IRIA. Rocquencourt, (Juin 1976J.

COUSOT P. [1977b]. *A mathematicaL theory of global program analysis,* Présenté au "Panel: Mathematical Theory of Data Flow Analysis". Chairman: ULLMAN J.A. (U.S.A.). Panelists: COUSOT P. (F.J, KENNEDY K. (U.S.A.J. ROSEN B. (U.S.A.), TARJAN R. (U.S.A.). IFIP Congress 1977, Toronto. Canada. (Aug. 1977),

COUSOT P. [1977c], *Analysis of programs properties,* Présenté au "Panel : Use and Benefit of Formal Description Techniques, Report of W.G.2.2.". Chairman: NEUHOLO E. (D.), Panelists : BLUM E. (U.S.A.J; BOEHM B. (1.), COUSOT P. (F.). De BAKKER J. (N.L.), IGARASHI S. (J.J, NIVAT M. (F.), OWICKI S. (U.S.A.)' TENNENT R. (C.O.N.), IFIP Congress 1977. Toronto. Canada, (Aug. 1977).

COUSOT P. [1977d]. *Iterative and approximate methods for compile time analysis of programs,* IBM Seminar. T.J. Watson Research Center. Computer Sciences Dept. Yorktown Heights. N.Y•• U.S.A•• (August 18. 1977)•

COUSOT P. [1977e], *Asynchronous iterative methods for solving a fixed point system of monotone equations in a complete lattice,* Rapport de Recherche n088. Laboratoire IMAG. Grenoble, (Sept. 1977J.

CDUSDT P. [1978]. *Chaotic and asynchronous iterative methods for solving a fixed point system of monotone equations in a complete lattice,* IBM Seminar. T.J. Watson Research Center. Mathematical Sciences Dept.• Yorktown Heights. N.Y•• U.S.A•• (January 26. 1978).

CDUSDT P. g CDUSDT R. *[1975a],V4rification statique de la cohérence dynamique des programmes,* Rapport du contrat IRIA-SESDRI 75-035. (Sept. 1975J.

CDUSDT P. g CDUSDT R. [1975b]. *Static verification of dynamic type properties of variables,* Rapport de Recherche nO 25. Laboratoire IMAG, Grenoble. (Nov. 1975).

CDUSDT P. g CDUSDT R. [1976], *Static determination of dynamic* properties *of programs;.* Proc. 2nd Int. Symp. on Programming. Dunod. Paris. (Avril 1976), 106-130.
[Aussi dans MOL Bulletin 5. Cousot P. (Ed.J. IRIA Rocquencourt. (Sept. 1976), 27-52].

CDUSDT P. g CDUSDT R. [1977a]. *Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints,* Conf. Rec. of the 4th ACM Symp. on Principles of Programming Languages. Los Angeles. Calif•• U.S.A•• (Janv. 1977). 238-252.

CDUSDT P. g CDUSDT R. [1977b]. *Static determination of dynamic properties of generalized type unions,* ACM Conf. on Language Design for Reliable Software. Raleigh. N.C., U.S.A•• (March 1977J. SIGPLAN Notices 12. 3(March 1977J, 77-94.

CDUSDT P. g CDUSDT R. [1977c]. *Fixed point approach to the approximate semantic analysis of programs,* (Juin 1977).

CDUSDT P. g CDUSDT R. *[1977dl.Static determination of dynamic* properties *of recursive procedures,* IFIP W.G.2.2. Working Conf.on Formal Description of Programming Concepts. St-Andrews. N.B.• Canada. North-Holland Pub. Co. (Aug. 1977).

CDUSDT P. &CDUSDT R. [1977e]. *Automatic synthesis of optimal invariant assertions: mathematicaL foundations,* Proc. of the ACM Symp. on Artificial Intellingence & Programming Languages. Rochester. New York. SIGPLAN Notices 12. 8(Aug. 1977J. 1-12.

CDUSDT P. &CDUSDT R. [1977f]. *Constructive versions of Tarski's fixed points theorems,* Rapport de Recherche nO 85. Laboratoire IMAG. Grenoble. (Sept. 1977).

CDUSDT P. &HALBWACHS N. [1978]. *Automatic discovery of linear restraints among variables of a program,* Conf. Rec. of the 5th ACM Symp. on Principles of Programming Languages. Tucson. Ariz.• U.S.A•• (Jan. 1978). 84-97.

CURRY G. [1977]. *Programming by abstract demonstration,* Technical Report n077-D8-D2. Computer Sci. Department. University of Washington, U.S.A., (Aug. 1977).

De BAKKER J.W. [1976]. *Semantics and termination of non deterministic* recursive *programs,* 3rd Int. ColI. Automata. Languages and Programming. University Press. Edinburgh. (1976). 435-477.

De BAKKER J.W. [1977a]. *Topics in semantics,* Lecture Notes for the Advanced Summer School on Math. Foundations of Computer Science. Turku. (June 1977).

De BAKKER J.W. [1977b]. *Recursive programs as predicate transformers,* IFIP Working Conf. on Formal Description of Programming Concepts. St.Andrews. Canada. North-Holland Pub. Co•• (Aug. 1977).

De BAKKER J.W. & MEERTENS L.G. [1975]. *On the of the inductive assertion method,* Journal of Computer and System Sciences 11. 3(1975). 323-357

De BAKKER J.W. & SCOTT o. [1969]. *A theory of programs,* IBM Seminar. Vienna. (1969)•

De MARR R. [1964]. *Common fixed points for isotone mappings,* Colloquium Math.• 13(1964). 45-48.

oEVIoE V. [1964]. *On monotonous mappings of lattices,* Fundamenta. Mathematicae. LIII. (1964). 147-154.

oIJKSTRA E.W. [1968]. *GO TO Statement considered harmful,* Letter to the editor. CACM 11. 3(March 1968).

oIJKSTRA E.W. [1975]. *Guarded commands, non determinacy and formal derivation of programs,* CACM 18. 8(Aug. 1975). 453-457.

oIJKSTRA E.W. [1976]. *A Of programming,* Prentice-Hall. Englewood Cliffs. N.J.. U.S.A.. (1976).

oIJKSTRA E.W. [1977]. *On making more and more* Note EW0622.

oUBREIL-JACoTIN M.L•• LESIEURL. & CRoISoT R. [1963]. *Leçons* sur *la théorie des des structures ordonnées* et *des géométriques,* Gauthier-Villars. Paris. (1963).

oWINGER Ph. [1954]. *On the closure operators of a complete lattice,* Nederl. AKad. Wetench. Proc. Ser. A. 57(1954). 560-563.

oWINGER Ph. [1955]. *The closure operators of the cardinal and ordinal sums and products of partially ordered* sets *and closed lattices,* Nederl. AKad. Wetensch. Proc. Ser. A. 17(1955). 341-351.

EARNEST C. [1974]. *Some topics in code optimization,* JACM 21. 1(1974). 76-102.

ELSPAS B. [1974]. *The semi-automatic generation of inductive assertions for proving program correctness,* Research Rep., SRI. Menlo Park. Calif•• U.S.A., (July 1974).

EL TARAZI M.N. [1976]. *Sur des algorithmes mixtes* par *blocs de type Newton-Relaxation chaotique ‡ retards,* CRAS Paris, t. 283, Série A, (Oct. 1976J, 721-724.

ERNST G.W. [1977], *Rules of inference for procedure calls,* Acta Informatica 8, (1977), 145-152.

FINANCE J.P. [1976], *Une formalisation de la des langages de programmation,* RAIRO 2(AoOt 1976), 10(Oec. 1976).

FLOYD R.W. [1967], *Assigning meaning to programs,* Proc. Symp. in Applied Math., Vo1.19. AMS. Providence, R.I.. U.S.A., (1967),19-32.

GERMAN S. [1978], *Automating proofs of the absence of common runtime* errors, Conf. Rec. of the 5th ACM Symp. on Principles of Programming Languages, Tucson. Ariz., U.S.A., (Jan. 1978), 105-118.

GERMAN S. & WEGBREIT B. [1975], *A synthesizer of inductive assertions,* IEEE Trans. Software Eng., SE-1, 1(March 1975), 68-75.

GILLETT W.O. [1977]. *Iterative global flow techniques for detecting program anomalies,* Ph.D Thesis. UIUCOCS-R-77-848. U. of Illinois at Urbana Chapaign, (Jan. 1977).

GORELICK G.A. [1975]. *A complete axiomatic system for proving assertions about recursive and non-recursive programs,* Tech. Rep. 75. Dept. of Compo Science. U. of Toranto. Canada, (Jan. 1975).

GRAHAM S.L. & WEGMAN M [1976]. *A fast and usually linear algorithm for global flow analysis,* JACM 23, 1(1976). 172-202.

GRATZER G. [1971], *Lattice firstconcepts and distributive lattices,* W.H. Freeman and Co•• San Francisco, Calif. U.S.A., (1971).

GUTTAGJ.V., HORNING J.J. & LONDON R.L. [1977]. *A proof rule for EUCLID procedures,* IFIP W.G.2.2. Working Conf. on Formal Description of Programming Concepts, St-Andrews, N.B.,Canada, North-Holland Pub. Co. (Aug. 1977).

HANTLER S.L. & KING J.C. [1976], *An introduction to proving the correctness of programs,* Compo Surveys 8. 3(Sept. 1976J. 331-353.

HARRISON W. [1977]. *Compiler analysis of the value* ranges *of variables,* IEEE Trans. on Software Engineering, 3(1977), 243-250.

MS. & ULLMAN J.D. [1972], *Flow graph reducibility,* SIAM J. Computing 1. 2(1972J. 188-202.

HECHT MS. & ULLMAN J.D. [1973]. *A simple algorithm for global data flow analysis,* Conf. Rec. of the ACM Symp. on Principles of Programming Languages. Boston, Mass. U.S.A.• (Oct. 1973). 207-217. [Aussi SIAM J. Computing 4,4(1975). 519-532].

HECHT M.S. & ULLMAN J.D. [1974], *Characterizations ofreducible flow graphs,* JACM 21. 3(1974). 367-375.

HECHT M.S. & ULLMAN J.D. [1975], *A simple algorithm for global data flow analysis of programs*, SIAM J. Computing 4, 4(1975), 519-532.

HEHNER E.C.R. [1976], *DO considered OD : a contribution to the programming calculus*, Tech. Rep. CSRG-75, Compo Syst. Res. Group, U. of Toronto. Canada, (Nov. 1976).

HITCHCOCK P. & PARK o. [1973], *Induction rules and proofs of termination*, Proc. Symp. on Automata, Languages and Programming, North-Holland Pub. Co., (1973), 225-251.

HOARE C.A.R. [1962], *Quicksort*, Computer Journal 5, 1(1962), 10-15.

HOARE C.A.R. [1969], *An axiomatic approach to computer programming*, CACM 12, 1o(oct. 1969), 576-580,583.

HOARE C.A.R. [1971], *Procedures and parameters: an axiomatic approach*, Lect. Notes in Math. 188, Springer-Verlag, Berlin, (1971), 102-116.

H. & HOFT M. [1976], *Some fixed point theorems for partially ordered sets*, Canad. J. Math., 5(1976), 992-997.

HOPCROFT J.E. & ULLMAN J.D. [1969]. *Formal languages and their relation to automata*, Addison-Wesley. Reading, Mass., (1969).

ICHBIAH J.D., MOREL E. & RENVOISE C. [1972], *Une directe de tion des systemes de redondance*, Note de programmation CII-oSB/AS/72/o23. (Juin 1972).

ICHBIAH J.D., RISSEN J.P., HELLIARD J.C. & COUSOT P. [1974], *The system implementation language LIS*, ReferenceManual, Rapport CII-4549-E1/EN. (Dec. 1974), Revisé (Jan. 1976).

IGARASHI S., LONDON R.L. & LUCKHAM D.C. [1975], *Automatic program tion. I. A logical basis and its implementation*, Acta 4, (1975). 145-182.

ISEKI K. [1951], *On closure operation in lattice theory*, Nederl. Akad. Wetensch. Proc. Ser. A. 54. Indag. Math.• 13(1951), 318-320.

IVERSON K.E. [1962], *A Programming Language*, J. Wiley & Sons Inc., (1962).

JACQUEMARo C. [1977], *Contribution à d'algorithmes de relaxation à convergence monotone*, Thèse de 3ème cycle, (Mai 1977).

JENSEN J. [1965], *Generation of machine code in ALGOL compilers*, BIT 5. (1965), 235-245.

JENSEN K. & WIRTH N. [1976], *PASCAL user manual and report*, Second edition. Springer-Verlag, Heidelberg, (1976).

JONES N.D. & MUCHNICK S.S. [1976]. *Binding time optimization in programming languages: some thoughts toward the design of an ideaL language*, Conf. Rec. of the 3rd ACM Symp. on Principles of Programming Languages, Atlanta, G.A.• U.S.A.• (Jan. 1976). 77-91.

KAM J.B. & ULLMAN J.D. [1977], *[Ionotone data fLow anaLysis frameworks,* Acta Informatica, 7(1977), 305-317.

KAPLAN M.A. & ULLMAN J.D. [1978], *A generaL scheme for the automatic inference of variable types,* Conf. Rec. of the 5th ACM symp. on Principes of Programming Languages. Tucson. Ariz•• U.S.A•• (Jan. 1978), 60-75.

KARR M. [1975], *Gathering information about* programs, Mass. Compo Associates Inc•• CA-7507-1411, (July 1975),

KARR M. [1976]. *Affine reLationships among variables of a program,* Acta Informatica 6(April 1976). 188-206.

KASVANOV V.N. [1973], *Some* properties *of fully reducible graphs,* Information Processing Letters 2, 4(1974). 113-117.

KATZ S. & MANNA Z. [1976]. *Logical analysis of program,* CACM 19. 4(Avril 1976), 188-206.

KENNEDY K. [1971], *A global flow analysis algorithm,* Int. J. Computer Math. 3. (1971), 5-15.

KENNEDY K. [1972], *Index register allocation in straight line code and simple loops,* dans: Rustin R., Design and Optimization of Compilers, Prentice-Hall, Englewood Cliffs, N.J•• U.S.A•• (1972).

KENNEDY K. [1975], *Node listings applied to dataflow analysis,* Conf. Rec. of the 2nd ACM Symp. on Principles of Programming Languages. Palo Alto. Calif. U.S.A•• (Jan. 1975), 10-21.

KENNEDY K. [1976]. *A comparison of two algorithms for global data flow analysis,* SIAM J. Computing 1, (Mar. 1976). 158-180.

KILOALL G. [1973]. *A unified approach to global program optimization,* Conf. Rec. of the ACM Symp. on Principles of Programming Languages. Boston, Mass. (Oct. 1973), 194-206.

KLEENE S.C. [1952]. *Introduction to metamathematics,* North-Holland. Pub. Co•• Amsterdam. (1952).

KNASTER B. [1928]. *Un* sur *les fonctions d'ensembles,* Ann. Soc. Polon. Math., 5(1928). 133-134.

KNUTH O. [1971], *An empirical study of FORTRAN Programs,* Software Practice and Experience 1. 2(1971), 105-134.

KNUTH O.E. [1973], *The* art *of computer programming, vol. 3, sorting and searching,* Addison-Wesley Pub. Co., Reading. Mass., U.S.A•• (1973).

O. *Structured programming with GOTO statements,* Computing Surveys 6, 4(Oec. 1974).

KOLODNER 1.1. [1968]. *On compLeteness of partially ordered sets and fixpoints theorems for isotone mappings,* Amer. Math. Monthly. 75(1968). 48-49.

KRASNOSEL'SKII M.A. [1964]. *Positive solutions of operator equations,* P. Noordhoff. Groningen. The Netherlands. (1964).

KUHN H.W. & MacKINNON [1975]. *Sandwich method for finding fixpoints,* J. Optimiz. Th. and Applications. 17(1975). 189-204.

LADEGAILLERIE Y. [1973]. sur *un ensemble* RAIRO 1(1973). 35-43.

LAMPSON B.W.. HORNING J.J•• LONDON R.L.• MITCHELL J.G. & POPEK G.J. [1976]. *Report on the programming language EUCLID,* MOL Bulletin 5. Cousot P. (Ed.), (Sept. 1976), 92-172.. SIGPLAN Notices 12. 2(Feb. 1977l.

LANERY E. [1966]. *Recherche d'un minimal d'un convexe,* Thèse de 3ème cycle. Caen. (1966).

LESZCZYLOWSKI J. [1971]. *A theorem on resolving equations in the space of languages,* Bull. Acad. Polon. Sci.• Ser. Sci. Math. Astronom. Phys•• 12(1971). 967-970.

LOMET D.B. [1975]. *Data flow analysis in the presence of procedure calls,* IBM Report RC-5728. T.J. Watson Research Center. Yorktown Heights. N.Y.. U.S.A•• (1975).

LORHO B. [1974]. *De la définition* à *la traduction des langages de* program-*mation* : *des attributs* Thèse d'Etat. Universite Paul Sabatier de Toulouse. (Déc. 1974).

LUCKHAM D. & SUZUKI N. [1976]. *Automatic* program *verification V* : *verification-oriented proof rules for arrays, records and pointeurs,* Report STAN-CS-76-549. Compo Sci. Dept•• Stanford Univ.• Calif.• U.S.A.• (March 1976).

LUONG N.X. [1975]. *Algorithmes de relaxation conduits* par *l'algorithme secondaire,* Thèse de 3ème cycle. Besançon. (1975).

MAHJOUB Z. [1977]. *de chaotiques* sur *les de point fixe* à *grand nombre de variables,* Thèse de Docteur-Ingenieur. Universite Scientifique et Medicale de Grenoble. (Mai 1977l.

MANNA Z. [1974]. *Mathematical theory of computation,* Mac-Graw Hill Book Co.• New York. U.S.A•• (1974).

MANNA Z•• NESS Z. & VUILLEMIN J. [1973]. *Inductive methods for proving properties of programs,* CACM 16. 8(Aug. 1973). 491-502.

MANNA Z. & SHAMIR A. [1977]. *The convergence of functions to fixed points of recursive definitions,* Report STAN-CS-77-614. Stanford Univ.• Calif.• U.S.A•• (May 1977).

MARKOWSKY G. [1976]. *Chain-complete posets and directed* sets *with applications,* Algebra Univ.• 6(1976). 53-58.

MEERTENS L. [1975]. *Mode and meaning,* dans Schuman S. (Ed.). New directions in algorithmic languages 1975. IFIP W.G.2.1.• IRIA Pub•• (1975).

MIELLOU J.C. [1975a), *de relaxation chaotique* à *retards*, RAIRO. Revue Rouge. AFCET RI. (1975). 55-82.

MELLoU J.C. [1975b), *chaotiques* à *retards* ; *de la convergence dans le cas d'espaces partiellement* CRAS Paris. t. 280, Série A. (Jan. 1975). 233-236.

MIELLOU J.C. [1977). *Algorithmes de relaxation* : *de convergence monotone*, Séminaire d'Analyse Numérique nO 278, Laboratoire IMAG. Grenoble, (Juin 1977).

MILNE R. [1977), *Transforming predicate* W.G.2.2. Working Conf. an Formal Description of Programming Concepts, St-Andrews. N.B., Canada. (Aug. 1977).

MILNE R. & STRACHEY C. [1976), *A theory of programming 'language semantics*, Chapman and Hall (Londresl & Wiley [New York), (1976).

MIRANKER W.L. [1977), *Parallel methods for solving equations*, IBM Research Report RC-6545 [# 28250), Mathematical Sciences Dept., T.J. Watson Research Center, Yorktown Heights. N.Y. (May 1977).

MONK O. [1969). *Introduction to set theory*, Int. Series in Pure and Applied Mathematics, Mac-Graw Hill Book Ca., N.Y., U.S.A•• (1969).

MONTEIRO A. [1945), *de de fermeture* par *un seul axiome*, Portugal. Math. 4(1945). 158-160.

MONTEIRO A. & RIBEIRO H. [1942). *de fermeture* et *ses invariants dans les systemes partiellement ordonnés*, Portugal. Math. 3(1942). 171-184.

MOORE E.H. [1910), *Introduction to a form of general analysis*, New Haven Colloquium. (1910).

MOREL E. & RENVOISE C. [1974), *Etude* et *d'un optimiseur global*, Thèse de 3ème cycle. Univ. de Paris VI, (Juin 1974).

MORGADO J. [1960). *Some results on closure operations of partially ordered sets*, Portugal. Math. 19(1960). 101-139.

MORGADO J. [1961). *On the closure operators of the ordinal* sum *of partially ordered sets*, Nederl. Akad. Wetench. Proc. Ser. 1, 23(1961). 546-550.

MORGADO J. [1962a). *Note on complemented closure operators Of complete lattices*, Portugal. Math. 21, 3(1962), 135-142.

MORGADO J. [1962b), *A characterization of the closure operators by means of a single axiom*, Portugal. Math., 21(1962). 155-156.

MORGADO J. [1963). *On the closure operators of the cardinal product of partially ordered sets*, Nederl. Akad. Wetench. Proc: Ser. A. 25(1963), 65-75.

MORGADO J. [1964). *Note on the distributive* lattice *operators of a complete lattice*, Portugal. Math. 23. 1(1964). 11-25.

MORGADO J. [1965a], *Note on the system of closure operators of the ordinal product of partially ordered* sets, Portugal. Math. 24, 4(1965), 189-220.

MORGADO J. [1965b]. *A single axiom for closure operators of partially ordered* sets, Gazeta de Mathematica, 100(1965), 57-58.

MORGADO J. [1966], *Factorization of the lattice of closure operators of a complete lattice,* Portugal. Math. 25. 1(1966), 181-185.

NAUR P. [1963], (Ed.). *"Revised report on the algorithmic language ALGOL 60",* CACM 6, 1CJan. 1963), 1-17.

NAUR P. [1965], *Checking of operand types in ALGOL compilers,* BIT 5. (1965), 151-163.

NAUR P. [1966]. *Proof of algorithms by general* snapshots, BIT 6. (1966). 310-316.

NIVAT M. [1972], *Langages algébriques* sur *la magma libre et semantique des schémas de programmes,* Proc. Call. an Automata. Formal languages and Programming. North-Holland Pub. Ca., Amsterdam, (1972).

ORE O. [1943a], *Some studies on closure relations,* Duke Math. Journal 10, (1943), 761-785.

ORE o. [1943b], *Combinations of closure relations,* Ann. of Math•• 44(1943), 514-533.

OSTROWSKI A. [1955], *Determination mit überwiegender haupt diagonale und die absolute konvergenz von linearen iterations prozessen,* Comm. Math. Helv. 30. (1955). 175-210.

PARK o. [1969], *Fixpoint induction and proofs of program properties,* Machine Intelligence. 5(1969), 59-78.

PAIR C. [1974], *Formalization of the notion of data, information and information* structure, Data base management. North-Holland Pub. Ca., Aisterdam, (1974).

PASINI A. [1974], *Some fixed point theorems of the mappings of partially ordered* sets, Rend. Sem. Mat. Univ. Padova•• 51(1974), 167-177.

PELCZAR A. [1961]. *On the invariant points of a transformation,* Ann. Polan. Math.. 11(1961), 199-202.

PELCZAR A. [1971], *Remarks on commuting mappings in partially ordered spaces,* Zeszyty Nauk. Univ. Jagiello., Prace Mat., Zeszyt, 15(1971), 131-133.

PNUELI A. [1977]. *The temporal logic of programs,* Proc. 18th Annual Symp. an Foundations of Computer Science. Providence, R.I•• U.S.A•• (oct.31-Nov.2 1977). 46-57.

REMY J.L. [1974], *Structure d'information, formalisation des notions d'accès et de modification d'une donnée,* Thèse de 3ème cycle, Univ. de Nancy I, (1974).

RIEF J.H. & LEWIS H.R. [1977]. *Symbolic evaluation and the global value graph,* Conf. Rec. of the 4th ACM Symposium on Principles of Programming Languages. Los Angeles. Calif., U.S.A.• (Jan. 19771. 104-118.

ROBERT F. [1974]. *Contraction en norme vectorielle. Convergence d'iterations chaotiques* pour *des equations de point fixe à plusieurs variables,* Gatlingburgh VI Symp. on Num. Alg., (Dec. 1974). Linear Algebra and its AppL 13, (1976). 19-35.

ROBERT F. [1976a]. *Sur la transformation de Gauss-Seidel,* Seminaire d'Analyse Numerique n0255, Laboratoire IMAG, Grenoble, (Nov. 1976).

ROBERT F. [1976b]. *Convergence locale d'iterations chaotiques non linéaires,* Rapport de Recherche n058. Laboratoire IMAG, Grenoble, (Déc. 1976).

ROCKAFELLAR R.T. [1976], *Monotone operators and the proximal point algorithm,* SIAM J. Control and Optimization, 14(1976), 877-898.

ROSEN B.K. [1975], *Data flow analysis for procedural languages,* IBM Report RC-5211, T.J. Watson Research Center. Yorktown Heights, N.Y•• U.S.A•• (1975).

ROSEN B.K. [1978], *Monoids for rapid data analysis,* Conf. Rec. of the 5th ACM Symp. on Principles of Programming Languages. Tucson. Ariz•• U.S.A•• (Jan. 1978), 47-59.

SCARF H. [1967], *The approximation of fixed points of a continuous mapping,* SIAM J. AppL Math., 15(19671, 1328-1343.

SCHAEFER M. [1973], *A mathematical theory of global program optimization,* Prentice Hall, Englewood Cliffs, N.J., U.S.A•• (1973).

SCHWARTZ J.T. [1973], *On programming: an interim report on the SETL project installment* 1. *Generalities, installment* 2. *The SETL language and examples of its use,* New York Univ., (1973).

SCHWARTZ J.T. [1975], *Automatic data structure choice in a language of very high level,* CACM 18. 12(Oec. 1975), 722-728.

SCOTT O. [1972], *Continuous lattices,* Proc. 1971 Dalhousie Conf. Lect. Notes in Math. 274, Springer-Verlag, New York. 97-136.

SCOTT O. [1976], *Data types* as *lattices,* SIAM J. on computing 5, 3(Sept. 1976), 522-587.

SCOTT O. [1977a], 1976 *AGM Turing award lecture logic and programming languages,* CACM 20. 9(Sept. 1977).

SCOTT O. [1977b], *Retracts,* Notes de cours, Ecole IRIA "Seminaire Avance de Semantique. Sophia-Antipolis. (Oct. 1977).

SCOTT O. & STRACHEY C. [1971]. *Toward a mathematical semantics for computer languages,* Proc. Symp. on computers and Automata. Polytechnic Inst. of Brooklyn. voL21, (1971). 19-46.

SINTZOFF • •[1972]. *Calculating* properties *of* programs *by valuations on models.* Proc. ACM Conf. an Proving Assertions about Programs. SIGPLAN Notices 7, 1(1972). 203-207.

SINTZoFF M. [1975]. *d'assertions* pour *des fonction utilisables ccmne valeurs et affectant des variables extérieures,* Proc. Int. Symp. an Proving and Improving Programs. Arcs et Senans. (Juillet 1975). 11-27.

SINTZoFF M. [1976a]. *Eliminating blind alleys* from *backtrack* programs. 3rd Int. Call. an Automata Languages and Programming. Edinburgh. (July 1976).

SINTZoFF M. [1976b]. *Iterative methods* for *the generation of successful* pro-*grams.* Nates de travail. Lab. MBLE. Bruxelles. (Dec. 1976).

SINTZoFF M. [1977a]. *Inventing* program *construction rules.* IFIP W.G.2.4. Working Conf. an Constructing Quality Software. Novosibirsk. North-Holland Pub. Co., (May 1977).

SINTZoFF M [1977b]. *Some logical construction rules* for *programs.* Nates de travail. CRI Nancy. (1977).

SINTZoFF M. & VAN LAMSWEERoE A. [1975]. *Constructing correct and efficient concurrent* programs. Proc. Int. Conf. an Reliable Software. SIGPLAN Notices. 10(1975). 319-326.

SMITHSON R.E. [1973]. *Fixed points in partially ordered sets.* Pacific J. Math.. 1(1973). 363-367.

SOUTHWELL R.V. [1955]. *Relaxation methods in theoritical physics.* Clarendon Press. Oxford. (1946).

SPILLMAN T.C. [1971]. *exposing side effects in a PL/l optimizing compiler.* Proc. IFIP Congress 71. North-Holland Pub. Ca" Amsterdam. (1971). 376-361.

STEIN P. & ROSENBERG R.L. [1946]. *On the solution of linear simultoneous equations by iterations.* J. London Math. Sac., 23(1946). 111-116.

STOY J. [1977]. *Denotational semantics.* MIT Press. (1977).

STRACHEY C. & WADSWORTH C. [1974], *Continuations: a mathematical semantics* for *full jumps.* Tech. Monograph PRG-11. Oxford U" Camp. Lab., Programming Research Group. (1974).

SUZUKI N. & ISHIHATA K. [1977]. *Implementation of an array bound checker.* Conf. Rec. of the 4th ACM Symp. an Principles of Programming Languages. Las Angeles. Calif•• U.S.A., (Jan. 1977). 132-143.

SZASZ G. [1971]. *des treillis.* ounod. Paris. (1971).

TARJAN R.E. [1974]. *Testing flow graph reducibility.* J. Computer and Systems Sciences 9. 3(1974). 355-365.

TARJAN R.E. [1975]. *Solving path problems on directed graphs.* Research Report STAN-CS-75-526. Computer Sci. Dept•• Stanford U" Calif., U.S.A•• (1975) •

TARJAN R.E. [1976], *Iterative for global flow analysis,* Research Report STAN-CS-75-545, Computer Sci. Dept., Stanford U., Calif., U.S.A., (Feb. 1976).

TARSKI A. [1955], *A lattice theoretical fixpoint theorem and its applications,* Pacific J. Math., 5(1955), 285-310.

TENENBAUM A.M. [1974], *Type determination for very high level* Report NSD-3, Computer Sci. Dept., New York U., U.S.A., (Oct. 1974).

TENNENT R.D. [1976]. *The denotational semantics of programming languages,* CACM 19, 8(1976), 437-453.

TODD M. [1976], *The computation of fixed points and applications,* Lecture Notes in Economics and Math. Systems 114, Springer-Verlag, Berlin, (1976) •

TRAUB J.F. [1964], *Iterative methods for solutions of equations,* Prentice Hall, (1964).

ULLMAN J.D. [1974], *Past algorithms for the of common subespressions,* Acta Informatica 2, 3(1974), 191-213.

ULLMAN J.D. *Data flow analysis,* Proc. 2nd USA-Japan Computer Conf., AFIPS Press, Montwale, N.J. (1975), 335-342.

URSCHLER G. [1974], *Complete redundant expression elimination in flow diagrams,* IBM Research Report RC-4965, T.J. Watson Research Center, Yorktown Heights, N.Y. (1974).

VAN LAMSWEERDE A. [1977], *From verifying termination to guaranteeing it : a case study, IFIP* Working Conf. on Formal Description of Programming concepts, St-Andrews, N.B., Canada, North-Holland Pub. Co., (Aug. 1977).

VAN LAMSWEERDE A. g SINTZDFF M. [1976], *Formal derivation of strongly correct parallel programs,* kapport R338, Lab. MBLE, Bruxelles, (1976).

VUILLEMIN J.E. *Proof techniques for recursive* programs, Ph.D. Thesis, STAN-CS-73-393, Stanford U., Calif. (Oct. 1973).

WARD M. [1942], *The closure operators of a lattice,* Annals Math., 43(1942), 191-196.

WARD L.E. Jr.[1957], *Completeness in semi-lattices,* Canad. J. Math., 9(1957), 578-582.

WEGBREIT B. [1974], *The synthesis of loop predicates,* CACM 17, 2(Feb. 1974), 102-112.

WEGBREIT B. [1975], *Property extraction in well-founded property sets,* IEEE Trans. on Soft. Eng., SE-1, 3(Sept. 1975), 270-285.

WEGBREIT B. [1977], *Complexity of synthesizing inductive assertion,* JACM 24, 3(July 1977), 504-512.

WELSH J. [1977]. *Economic range checking in PASCAL,* Dept. of Camp. Science. Queen's University, Blefast, Northern Ireland. (Oct. 1977J.

WIRTH N. [1971]. *Programm development by stepwise refinement,* CACM 14. 4(April 1971), 221-227.

WIRTH N. [1976]. *Programming languages: what to demand and how to assess them,* Symp. on Software Engineering. Belfast. (April 1976). [Aussi Rep. 17. Eidgen5ssische Technische Hochschule ZUrich, InstitUt fur Informatik] •

WOLK E.S. [1957], *Dedekind completeness and a fixed point theorem,* Canad. J. Math., 9(1957). 400-405.

WONG J.S.W. [1967], *Common fixed point of commuting monotone mappings,* Canad. J. Math.. 19(19671. 617-620.

dernière page de la thèse

AUTORISATION DE SOUTENANCE

VU les dispositions de l'article 5 de l'arrêté du 16 Avril 1974,

VU les rapports de M. JORRAND Philippe

M. FAIR Claude

M. SINTZOFF Michel

Mr COUSOT Patrick est autorisé

à présenter une thèse en soutenance pour l'obtention du grade de

DOCTEUR D'ETAT ES SCIENCES.

Fait à GRENOBLE, le 20 mars 1978

Le Président de l'U.S.M.G.

Le Président de l'I.N.P.G.

Ph. TRAYNARD
Président
de l'Institut National Polytechnique
P.O. le Vice-Président