Généralisations et méthodes correctes pour l'induction mathématique

Pascal Urso

UNIVERSITÉ DE NICE – SOPHIA ANTIPOLIS – UFR SCIENCES

École Doctorale STIC
Département d'Informatique

THÈSE

Présentée pour obtenir le titre de

Docteur en Sciences
de l'Université de Nice – Sophia Antipolis

Spécialité : **Informatique**

par

Pascal Urso

Généralisations et méthodes correctes pour l'induction mathématique

Soutenue publiquement le 29 Mars 2002 devant le jury composé de :

Président	Alain Colmerauer
Directeur	Emmanuel Kounalis
Rapporteurs	Jürgen Giesl
	Michael Rusinovitch
Examinateurs	Zohar Manna
	Michel Rueher
	Robert de Simone

ESSI, Amphi Est à 15h

Remerciements

Je souhaite remercier les personnes qui m'ont apporté leur soutient tout au long de mes années de doctorat.

Tout d'abord Emmanuel Kounalis qui a dirigé cette thèse fut remarquable dans son aide, ses conseils et sa disponibilité. Nos bouillonnantes discutions, parfois fort bruyantes, m'ont orienté sur des sujets de recherche passionnants. Il fut à la fois présent, encourageant et compréhensif. Je dois dire que se fut un réel plaisir de travailler avec lui et ceci depuis mon stage de DEA.

Alain Colmerauer, bien que prévenu peu de jours avant, m'a fait l'honneur de venir présider mon jury.

Jürgen Giesl et Michael Rusinovitch qui ont accepté d'être les rapporteurs de ce manuscrit, m'ont reçu fort amicalement dans leurs laboratoires respectifs. Ils m'ont aussi communiqué bien des remarques précises et pertinentes; remarques qui m'ont permis de consolider et de clarifier plusieurs points.

Zohar Manna et Robert de Simone me font l'honneur de participer à mon jury.

Toute l'équipe COPRIN du laboratoire I3S et de l'INRIA m'a accueilli chaleureusement. Particulièrement Michel Rueher qui me fait aussi l'honneur de participer au jury de cette thèse, m'a donné de précieux conseils dans mon travail de recherche et d'enseignement.

Les personnels de l'UNSA, de l'I3S et de l'INRIA, non pas parce qu'il faut les remercier, mais parce qu'ils furent présents et aimables lorsque j'en avais besoin.

Pour finir je remercie ceux qui m'ont apporté leur affection jour après jour; tous mes amis, mes parents, ma sœur et sa jeune famille et bien sûr ma chère Céline qui a eu la patience me supporter si bien jusqu'ici.

Table des matières

Présentation du manuscrit

Ce manuscrit peut être découpé en trois parties. Les trois premiers chapitres contiennent les définitions de notre domaine de recherche (la preuve par induction) ainsi qu'une introduction à nos méthodes. Les trois suivants abritent le contenu théorique de nos travaux : définitions, algorithmes, et théorèmes. Les trois derniers contiennent des travaux annexes, une implantation logicielle et pour finir une conclusion.

Concepts Préliminaires

Le chapitre 1 contient les définitions formant les bases de la réécriture et de la preuve par induction. On y trouvera aussi un « état de l'art » sur les approches formelles visant à améliorer les capacités des systèmes de preuve ainsi que sur les logiciels implantant ces méthodes.

Le chapitre 2 contient les premières définitions nécessaires à la lecture de nos travaux, c'est-à-dire le cadre formel dans lequel nous situons nos contributions principales.

Dans le chapitre 3, nous donnerons un aperçu de nos apports principaux. Nos propres définitions y seront expliquées informellement à travers quelques exemples non triviaux. Toutes les notions vues dans ce chapitre seront définies ou prouvées formellement dans les trois chapitres suivants.

Contributions Principales

Le chapitre 4 contient nos définitions initiales et les lemmes qui leur sont associés. Ces lemmes permettent de construire la preuve du théorème central de nos méthodes.

Dans le chapitre 5, nous montrerons comment obtenir des généralisations correctes de conjectures. Nous décrirons l'algorithme calculant ces généralisations et nous prouvons sa correction à partir des définitions du chapitre précédent.

Le chapitre 6 contient une nouvelle méthode de preuve par induction. Cette méthode s'appuie sur les définitions du chapitre 4 et sur des notions informellement décrites dans le chapitre 3. Après avoir défini celles-ci, nous donnerons l'algorithme de cette méthode et les théorèmes établissant sa correction.

Travaux Annexes

Dans le chapitre 7 , nous décrirons un heuristique de généralisation pour les systèmes d'équations conditionnelles. Cet heuristique sera notamment appliqué sur la preuve d'une contrainte d'intégrité de base de données formalisée par le calcul de situation.

Le chapitre 8 contient une démonstration de notre logiciel de preuve par induction "NICE" (*Nice Induction for Conditional Equations*) qui utilise les méthodes de cette thèse. Ce logiciel est muni d'une interface graphique et est écrit en JAVA.

Dans le dernier chapitre, nous conclurons cette thèse par une brève analyse de nos résultats et par des propositions pour de futures recherches dans le domaine.

Première partie

Concepts Préliminaires

Chapitre 1

Concepts de base

Dans ce chapitre nous décrirons les cadres de notre domaine de recherche, la preuve par induction.

Nous supposons connues les notions essentielles de la logique équationnelle et des systèmes de réécriture (cf., pour plus de détails, [DJ90]). Cependant, nous introduisons les définitions permettant de suivre la description de nos travaux. Une fois introduites ces notions, nous proposerons nos conventions de notation.

Nous établirons aussi un « état de l'art ». Cet état de l'art est partagé en trois parties. En premier lieu, nous étudierons les méthodes formelles habituellement utilisées pour prouver des théorèmes inductifs. Ensuite, nous exposerons brièvement les principaux systèmes logiciels de preuve. Enfin, nous exposerons les différents travaux visant la spéculation heuristique de lemmes ou de généralisations.

1.1 Logique équationnelle

Dans cette section, nous introduirons les concepts à la base de la notion de théorème inductif.

Dans la logique équationnelle, les axiomes et les conjectures sont représentés par un ensemble d'équations implicitement universellement quantifiées. Ces équations sont

formées de *termes* et ces termes sont engendrés par une algèbre. Nous décrierons donc cette « algèbre des termes » ainsi que des notions relatives : celles de position et de substitution.

Pour un ensemble d'axiomes donné, un théorème (c'est-à-dire une formule « valide ») va être classifié « déductif » ou « inductif » selon sa classe de modèles. Nous décrierons donc ces deux classes de théorèmes ainsi que les modèles que nous devons considérer dans le cas inductif.

Nous étudierons aussi les différentes classes de systèmes de réécriture nous intéressant ainsi que leurs propriétés.

1.1.1 Algèbre des termes

Les termes sont typés et une théorie peut appréhender plusieurs types. Ces termes sont construits à l'aide de symboles de fonctions et de variables. Les symboles de fonctions possèdent une *signature* sur l'ensemble des types, mais les variables ont un typage « libre »[1].

Définition 1.1. L'ensemble des *signatures* Σ est la paire $(\mathcal{S}, \mathcal{F})$ où $\mathcal{S}$ est un ensemble de *type* et $\mathcal{F}$ un vocabulaire fini de *symboles de fonctions*.

Définition 1.2. L'*algèbre des termes*, notée $T(\mathcal{F},\mathcal{X})$, où $\mathcal{X}$ est un ensemble énumérable de *variables* librement typées, est l'ensemble des termes bien typés. Le *type du terme* t est noté $type(t)$ et l'*ensemble des variables* présentes dans t est noté $\mathcal{V}(t)$.

Si le vocabulaire de fonction $\mathcal{F}$ contient au moins une *constante*, l'ensemble des termes sans variable – dit *termes clos* – n'est pas vide.

Définition 1.3. L'algèbre des *termes clos* est dénotée $T(\mathcal{F})$.

Nous pouvons maintenant définir formellement une *équation*.

Définition 1.4. Une *équation* est un élément de $T(\mathcal{F},\mathcal{X}) \times T(\mathcal{F},\mathcal{X})$ et est notée $t = s$

1. C'est-à-dire qu'un même symbole de variable x peut être d'un type T_1 dans une équation, mais d'un autre type T_2 dans une deuxième équation.

pour deux termes t et s.

Remarque : Attention à ne pas confondre le symbole « $=$ » formant une équation et le symbole « $\equiv$ » dénotant l'égalité syntaxique (ou égalité simple) de deux termes.

Positions

Pour pouvoir identifier formellement les sous-termes et les symboles d'un terme, nous utilisons la notion de « position ». Si on représente un terme par un arbre, les positions dans le terme indiquent les différents nœuds de cet arbre. L'ensemble des positions d'un terme est appelé le « domaine » du terme et la position « vide » indique la racine de l'arbre.

Définition 1.5. Pour un terme t, le *domaine* du terme t, noté $dom(t)$, est l'ensemble des positions de t. La *racine* de t est la position notée ε.

Définition 1.6. Pour un terme t et une position $p \in dom(t)$,

- le symbole de t indiqué par p est noté $t(p)$,
- le sous-terme de t indiqué par p est noté t/p,
- le remplacement dans t du sous-terme à la position p par le terme s est noté $t[s]_p$.

La longueur d'une position est la profondeur où se trouve le nœud indiqué par cette position. Les positions sont munies d'une relation d'ordre $\leq$. Nous avons $p \leq q$ si et seulement si p est un préfixe (non-strict) de q.

Définition 1.7. La *longueur* de la position p est noté $|p|$.

Définition 1.8. La relation d'*ordre préfixe* sur les positions est notée $\leq$.

Remarque : Comme convention de notation nous représentons les positions par une série de chiffres entre 1 et 9. Cette notation ne permet pas l'utilisation de symbole de fonction d'arité supérieure à neuf, mais sera suffisante pour notre étude.

• ***Exemple* 1.1.1.** Pour le terme $t = f(x,g(y,z,h(a)))$, nous avons $t(23) = h$, $t/23 = h(a)$ et $t[a]_2 = f(x,a)$.

Substitutions

Une *substitution* associe des variables à des termes d'un type approprié (en accord avec la signature du terme). Si tous ces termes sont clos, la substitution est dite close.

Définition 1.9. Une *substitution* (respectivement une *substitution close*) est une fonction de $\mathcal{X}$ dans $T(\mathcal{F},\mathcal{X})$ (respectivement $T(\mathcal{F})$).

En appliquant une substitution à un terme, on obtient une « instance » de ce terme. Si la substitution est close, l'instance est appelée « instance close ». Une instance d'un terme t selon une substitution σ est le même objet t où toute variable x a été remplacée par son image $\sigma(x)$.

Définition 1.10. Pour un terme t et une substitution (close) σ, l'*instance (close)* $t\sigma$ est le terme t' tel que

$$\forall p \in dom(t).\ \wedge \begin{array}{l} t(p) \in \mathcal{X} \implies t'/p = \sigma(t(p)) \\ t(p) \in \mathcal{F} \implies t'(p) = t(p) \end{array}$$

• ***Exemple* 1.1.2.** Pour le terme $t = g(x,x,h(a))$ tel que $\{g,f,h,a\} \subseteq \mathcal{F}$, et la substitution σ telle que $\sigma(x) = h(y)$, nous avons $t\sigma = g(h(y),h(y),h(a))$.

1.1.2 Déduction et Induction

Une équation e est un *théorème déductif* (dit aussi conséquence déductive) d'un ensemble d'équations $\mathcal{A}$ s'il est valide dans *tous les modèles* de $\mathcal{A}$. Ceci est dénoté

$$\mathcal{A} \models e$$

La propriété suivante nous donne une caractérisation des conséquences déductives.

Propriété 1.1. L'équation $t = s$ est une *conséquence déductive* d'un ensemble d'axiomes $\mathcal{A}$ si et seulement si $t =_{\mathcal{A}} s$ où $=_{\mathcal{A}}$ dénote la plus petite congruence monotone

qui contient $\mathcal{A}$.

Les théorèmes déductifs peuvent aussi être prouvés par *réfutation* en dérivant une contradiction de $\neg e \wedge \mathcal{A}$. Habituellement, $\neg e$ est transformée en une formule universelle $\mathcal{U}$ en introduisant des fonctions skolémisées. Le vocabulaire $\mathcal{F}$ doit alors être étendu.

Le raisonnement par induction, quant à lui, consiste à effectuer des inférences dans des domaines où il existe une relation d'ordre naturel et bien fondée entre les objets.

La notion de *théorème inductif* peut donc être reliée à plusieurs sortes de modèles : modèles « de Herbrand », modèles « initiaux », ou modèles « constructeurs ». Pour de plus amples discussions sur ces différents modèles cf., par exemple, [Pad88].

La « *théorie* » d'un ensemble d'axiomes $\mathcal{A}$ qui nous intéresse est l'ensemble des formules (des équations) qui sont valides dans le *modèle minimal de Herbrand* – dit aussi initial – de $\mathcal{A}$. Tout élément dans le domaine d'un modèle de Herbrand est dénoté par un terme clos construit sur la signature de $\mathcal{A}$. Comme ces termes clos peuvent être aisément ordonnés, l'induction est une technique naturelle pour prouver la validité de formules dans ces modèles. C'est pourquoi nous appelons la *théorie inductive* de $\mathcal{A}$ l'ensemble des équations valides dans le modèle minimal de Herbrand de $\mathcal{A}$:

Définition 1.11. Pour un ensemble d'équations $\mathcal{A}$, une équation $t = s$ est un *théorème inductif* de $\mathcal{A}$ si $t = s$ est valide dans le modèle initial de $\mathcal{A}$. Ceci est dénoté

$$A \models_{ind} t = s \text{ ou } t =_{ind(\mathcal{A})} s$$

• ***Exemple* 1.1.3.** Soit l'ensemble d'équations E définissant l'addition $+$ sur les entiers naturels :

$$E = \begin{cases} 0 + x & = x \\ s(x) + y & = s(x + y) \end{cases} \tag{1.1.1}$$

$x+0 = x$ *n'est pas une conséquence déductive de* E, car il existe des modèles de E dans lesquels $x + 0 = x$ n'est pas valide. Par exemple, prenons un modèle à deux éléments

$\{0, a\}$, dans lequel $s(a)$ est égal à a et $s(0)$ est égal à 0. Pour cette interprétation $a+0$ est égal à 0 et l'équation $x + 0 = x$ n'est pas valide.

Par contre, $x + 0 = x$ *est une conséquence inductive de* E car le modèle initial de E ne contient que des termes construits sur les symboles $\{0, s\}$. C'est-à-dire $\mathbb{N}$, le modèle que l'on « avait en tête » lorsque l'on a écrit cette définition.

Cependant, la notion de théorème inductif est reliée à celle de théorème déductif selon la propriété suivante :

Propriété 1.2. Pour un ensemble d'équations $\mathcal{A}$, une équation $t = s$ est un théorème inductif de $\mathcal{A}$ si et seulement si pour toute substitution σ close, $t\sigma =_{\mathcal{A}} s\sigma$.

Remarque : Si la validité dans *tous* les modèles de Herbrand et dans le modèle initial est identique pour une équation, ce n'est pas le cas, en général, pour une clause (cf. [Pad88] et chapitre 7).

Pour établir la validité de conséquences inductives, les méthodes classiques utilisent soit l'induction « explicite » ([Aub79, BM79, ZKK88, Red90]), soit l'induction « implicite » ([Bac88, BJ97, BKR95, BRH96, Com00, Der97, JK89, Mus80]). Quoiqu'il en soit, les preuves par induction divergent souvent (voir section 1.3).

1.1.3 Réécriture

L'idée de la réécriture est d'imposer une direction dans l'utilisation des équations lors d'une preuve. En fait, un système de réécriture décrit une « relation de réécriture » définie sur les termes.

Définition 1.12. Une *relation de réécriture*, notée $\rightarrow$, est une relation binaire sur $T(\mathcal{F},\mathcal{X}) \times T(\mathcal{F},\mathcal{X})$.

Nous supposons l'existence d'une « *relation d'ordre* » $\succ$ sur les termes. Cette relation binaire est :

– transitive ($t \succ s$ et $s \succ r$ implique $t \succ r$),

– irréflexive (on ne peut avoir $t \succ s$ et $s \succ t$),

– nœthérienne (il n'existe pas de séquence $t_1 \succ t_2 \succ \ldots$),

– monotone ($t \succ s$ implique, pour tout $p \in dom(w)$, $w[t]_p \succ w[s]_p$),

– et stable ($t \succ s$ implique, pour toute substitution σ, $t\sigma \succ s\sigma$).

Cette relation d'ordre est un « ordre complet de simplification » si elle respecte la définition suivante :

Définition 1.13. Une relation d'ordre $\succ$ est un *ordre complet de simplification* si

– pour tout terme $s, t \in T(\mathcal{F})$, soit $s \equiv t$, $s \succ t$, ou bien $t \succ s$,

– pour tout sous-terme strict s de t, $t \succ s$.

Habituellement, la relation d'ordre utilisée pour la réécriture est un ordre *lexicographique* étendant une *précédence* sur les symboles de fonctions.

• ***Exemple* 1.1.4.** Selon la précédence $* > + > s > 0$, nous avons l'ordre lexicographique *succ* tel que :

$$s(0) \succ 0 \ , \ x + 0 \succ x \ , \ x + s(y) \succ s(x + y) \ , \ x * s(y) \succ (x * y) + x$$

Systèmes de réécriture

À l'aide de la relation d'ordre $\succ$, il peut être possible « d'orienter » un ensemble d'axiomes $\mathcal{A}$ en un système de réécriture $\mathcal{R}$.

Définition 1.14. Un *système de réécriture* $\mathcal{R}$ est un ensemble d'équations orientées $\{l \to r\}$ appelées *règles de réécriture*.

Une règle de réécriture $l \to r$ est appliquée à un terme t en trouvant un sous-terme de t qui est une instance l et en le remplaçant par l'instance correspondante de r.

Définition 1.15. Un terme t' est une *simplification selon* $\mathcal{R}$ du terme t, noté $t \to_{\mathcal{R}} t'$ s'il existe une règle $(l \to r) \in \mathcal{R}$ et une substitution σ telles que

$$\exists p \in dom(t) \text{ t.q. } t/p \equiv l\sigma \text{ et } t' \equiv t[r\sigma]_p$$

La relation $\rightarrow_{\mathcal{R}}^{*}$ est la fermeture réflexive et transitive de $\rightarrow_{\mathcal{R}}$. Nous écrivons $a \downarrow_{\mathcal{R}} b$ s'il existe un terme c tel que $a \rightarrow_{\mathcal{R}}^{*} c$ et $b \rightarrow_{\mathcal{R}}^{*} c$.

On obtient la « forme normale » d'un terme en appliquant les règles de $\mathcal{R}$ jusqu'à obtenir un terme qui ne puisse plus être simplifié. La suite de simplifications appliquées à un terme est appelée « *séquence de réécriture* ».

Définition 1.16. La *forme normale selon* $\mathcal{R}$ d'un terme t, noté $t{\downarrow_{\mathcal{R}}}$, est un terme t' tel que $t \rightarrow_{\mathcal{R}}^{*} t'$, et tel que pour toute substitution σ,

$$\forall p \in dom(t'). \neg\exists(l \rightarrow r) \in \mathcal{R} \text{ t.q. } t'/p \equiv l\sigma$$

Remarque : Si la relation de réécriture est nœthérienne, chaque terme possède au moins une forme normale.

Convergence

Les systèmes de réécriture qui nous intéressent particulièrement sont les systèmes « convergents sur les termes clos » . Un système $\mathcal{R}$ est convergent sur les termes clos si tout terme clos possède une et une seule forme normale selon $\mathcal{R}$.

Définition 1.17. Un système de réécriture $\mathcal{R}$ est *convergent sur les termes clos* si pour tout terme a,$b \in T(\mathcal{F})$,

$$\mathcal{R} \models a = b \text{ implique } a \downarrow_{\mathcal{R}} b$$

La propriété ci-dessous caractérise cette classe de systèmes de réécriture. La « terminaison » implique qu'il existe au moins une forme normale, et la « confluence » implique que cette forme normale est unique.

Propriété 1.3. Un système de réécriture est convergent sur les termes clos si et seulement s'il est *terminant* – par rapport à un ordre de réduction $\succ$ – et s'il est *confluent* sur les termes clos – c'est-à-dire qu'il possède la propriété de Church-Rosser sur les termes clos.

L'intérêt d'un tel système est important. En effet, si un ensemble d'équations $\mathcal{A}$ peut être compilé en un système de réécriture convergent sur les termes clos, on peut *décider* de la validité inductive d'une équation en comparant syntaxiquement les formes normales des instances closes des deux côtés de l'équation.

Propriété 1.4. Si un ensemble d'équations $\mathcal{A}$ est compilé en un système de réécriture $\mathcal{R}$ convergent sur les termes clos, pour toute substitution close σ,

$$t\sigma =_{\mathcal{A}} s\sigma \iff t\sigma\downarrow_{\mathcal{R}} \equiv s\sigma\downarrow_{\mathcal{R}}$$

C'est-à-dire

$$\mathcal{A} \models_{ind} t = s \iff \forall \sigma \text{ close. } t\sigma\downarrow_{\mathcal{R}} \equiv s\sigma\downarrow_{\mathcal{R}}$$

Remarque : Le processus de *complétion* transformant une axiomatisation en un système de réécriture convergent n'est pas détaillé dans ce manuscrit. Intuitivement, ce processus est habituellement basé sur un ordre de simplification et applique une série de règles sur l'ensemble des axiomes jusqu'à obtenir un système convergent (cf., par exemple, [DJ90, Bac91]). Néanmoins, ce processus échoue si des axiomes sont impossibles à orienter.

Complétude suffisante

Nous supposons que le vocabulaire des fonctions d'un système de réécriture peut être partagé en deux ensembles :

Définition 1.18. L'ensemble des symboles de fonction, noté $\mathcal{F}_R$, d'un système de réécriture $\mathcal{R}$, peut se partagé en deux ensembles :

- l'ensemble des symboles de *fonction définies*, noté $\mathcal{D}_R$.
- l'ensemble des symboles de *constructeurs libres*, noté $\mathcal{C}_R$.

Les constructeurs d'un système sont appelés « libres » s'il n'existe pas de règle de réécriture dont la racine de la partie droite est un constructeur.

Propriété 1.5. Pour un système de réécriture $\mathcal{R}$, pour tout symbole f de $\mathcal{C}_R$,

$$\forall (l \to r) \in \mathcal{R}.\ l(\varepsilon) \neq f$$

Remarque : Nous avons donc $\mathcal{F}_R = \mathcal{D}_R \cup \mathcal{C}_R$ et, comme les constructeurs sont libres, $\mathcal{D}_R \cap \mathcal{C}_R = \varnothing$.

Quand les formes normales de tous les termes clos sont uniquement construites à l'aide de constructeurs, nous disons que le système est « suffisamment complet ». Autrement dit, *la forme normale de toute instance close n'est composée que de constructeurs.*

Définition 1.19. Un système de réécriture $\mathcal{R}$ est *suffisamment complet* si

$$\forall t \in T(\mathcal{F}_R).\ t{\downarrow_{\mathcal{R}}} \in T(\mathcal{C}_R)$$

Remarque : Cette définition est une limitation par rapport à l'ensemble des systèmes de réécriture mais, cependant naturelle quand on établit les spécifications de manière structurelle ([Gut78]).

Sur la classe des systèmes convergents sur les termes clos et suffisamment complets, on peut définir une méthode inductive « *réfutationnellement complète* » (voir section 1.3.2), dans le sens où toute équation non-valide va être « disprouvée » en un temps fini.

• ***Exemple* 1.1.5.** Le système de réécriture $\mathcal{R}$ définissant l'« arithmétique standard de Péano » est convergent et suffisamment complet :

$$\mathcal{R} = \begin{cases} x + 0 & \to x \\ s(x) + y & \to s(x + y) \\ x * 0 & \to 0 \\ x * s(y) & \to (x * y) + x \end{cases} \tag{1.1.2}$$

1.2 Conventions de notation

Dans cette section, nous nous contentons d'exposer nos conventions de notation. Tout au long de ce manuscrit, nous allons donc adopter les conventions suivantes :

- Un terme quelconque est noté par des lettres minuscules (t, s, ...).
- Une variable est notée par une des lettres x, y, ou z.
- Une position est notée par une série de chiffres dans $\{1,\ldots,9\}$.
- Une substitution est notée par une lettre grecque (σ, θ, ...), et son application à un terme t est notée $t\sigma$.
- La forme normale d'un terme t pour un système de réécriture $\mathcal{R}$ est notée $t{\downarrow_{\mathcal{R}}}$ ou, s'il n'y a pas d'ambiguïté sur le système de réécriture utilisé, $fn(t)$.
- Un terme clos en forme normale est noté par des lettres majuscules (A, T, ...).
- L'ensemble (infini) des formes normales pour un système $\mathcal{R}$ des instances closes d'un terme t est noté ainsi : $\{t\}{\downarrow_{\mathcal{R}}}$.
- Et, enfin, un terme dont la racine est une fonction f dont tous les n arguments sont clos et normalisés sera noté $f(\overrightarrow{A_{1\ldots n}})$.

• ***Exemple* 1.2.1.** Nous pouvons alors avoir :

- Le terme $t = x * \Sigma I(y + z)$.
- Pour le système de réécriture $\mathcal{R} = \{h(g(g(x))) \to h(x);\ h(\diamond) \to \diamond\}$ et la substitution close θ associant y à $\diamond$:
  - $h(g(g(g(\diamond))))\downarrow_{\mathcal{R}} = h(g(\diamond))$
  - $fn(h(y)\theta) = \diamond$
- Le terme clos $T = g(g(g(\diamond)))$
- Le terme $h(T) = h(g(g(g(\diamond))))$ de type $f(\overrightarrow{A_{1\ldots n}})$.

1.3 État de l'art

Dans cette section, nous proposerons un aperçu des principaux travaux dans le domaine des systèmes de preuve automatique par induction. Nous décrierons brièvement les travaux formels et indiquerons dans quels logiciels de preuve ils ont été implantés.

Nous établirons aussi un bref exposé des méthodes heuristiques de découverte de lemmes ou de généralisations. Nous décrivons aussi leurs avantages et leurs limites, peut-être d'une manière pouvant porter à controverse[2].

Quoiqu'il en soit, il faut préciser que les théories incluant l'induction sont soumises à deux grands résultats théoriques qui contraignent toute automatisation :

- Ces théories sont habituellement *incomplètes*, c'est-à-dire qu'elles contiennent des formules valides mais improuvables ([Göd31]). Ceci se traduit par le besoin d'un nombre illimité de règles d'induction distinctes.
- Un nombre arbitraire de formules intermédiaires peuvent être nécessaires à la preuve d'une conjecture ([Kre65]). Ceci se traduit par le besoin d'introduire des généralisations et/ou des lemmes intermédiaires.

Ces deux résultats négatifs introduisent potentiellement un nombre infini de points de branchement dans l'espace de recherche, et ceci à n'importe quelle étape.

Plusieurs approches ont été développées pour automatiser l'induction. Ces approches sont couramment (mais rudement) classifiées en deux catégories : l'induction « *explicite* » et l'induction « *sans induction* » (appelée parfois « *implicite* »). Toujours aussi rudement, la différence clé entre ces deux catégories se fait sur l'utilisation ou non d'axiomes de récurrence explicites.

2. Comme les études portant sur ces différentes techniques ne sont rarement développées par la littérature française nous proposerons une traduction des noms des différents concepts tout en rappelant leur dénomination anglaise.

Malgré tout, que ce soit implicitement ou explicitement, étant donnés une conjecture $t = s$ et un ensemble d'axiomes $\mathcal{A}$[3], effectuer une tentative de preuve par induction consiste typiquement en quatre étapes :

1. Choisir une (ou des) variable(s) (dites « *variables d'induction* ») dans $\mathcal{V}_R(t) \cup \mathcal{V}_R(s)$,
2. Utiliser un « *schéma d'induction* » couvrant toutes les valeurs possibles dans le modèle initial,
3. Tenter de recouvrir les sous-conjectures obtenues en utilisant $\mathcal{A}$ (ou $\mathcal{R}$) et l'« *hypothèse d'induction* » $t = s$ (sous certaines conditions),
4. Recommencer si besoin.

1.3.1 Induction explicite

L'induction explicite fait donc appel à des axiomes du second ordre générés par les constructeurs. Ces axiomes ont la même forme que l'axiome d'induction mathématique de Péano :

$$\forall P.\ \{[P(0) \wedge (\forall x.\ P(x) \implies P(s(x)))] \implies \forall x.\ P(x)\} \tag{1.3.1}$$

Le but de ces axiomes est de développer un schéma d'induction couvrant toutes les valeurs (c'est-à-dire l'infinité des termes générés par les constructeurs) que les variables d'induction peuvent avoir. Il faut ensuite prouver les instances de la conjecture où chacune de ces variables a été remplacée par ces valeurs. Le schéma d'induction est donc utilisé pour réduire un nombre infini de cas en un nombre fini. Par exemple, pour les entiers naturels il s'agit de prouver

1. le cas où la variable d'induction vaut 0 et,
2. en supposant la conjecture valide pour une valeur arbitraire x, le cas où la variable vaut $s(x)$.

3. Transformé ou non en un système de réécriture $\mathcal{R}$

Pour des listes, il s'agira alors du cas où la variable vaut $\varnothing$ et du cas $c.l$ (un élément c en tête suivi de la liste l).

Ce principe a été popularisé en 1969, dans [Bur69], comme le « principe d'induction structurelle ». En 1979, lors du développement d'heuristiques pour choisir les variables d'induction, il a été observé que les définitions des fonctions qui terminent (voir section 1.1.3) pouvaient elles-mêmes suggérer un schéma d'induction ([BM79]). Cette méthode est employée dans le système `Nqthm` [BM88].

En 1987, cette méthode a été étendue pour proposer des schémas d'induction basés sur le concept d'« ensemble couvrant » (*cover set* [ZKK88, Red90]). Cette notion est aussi bien utilisée dans l'induction explicite que dans l'induction implicite (voir plus bas).

Définition 1.20. [Zha88] Pour un ensemble d'équations E et un ordre de simplification $\succ$, un *ensemble couvrant* pour une variable x de type $T \in \S$ est un ensemble de n triplets $\{\langle t_i, H_i, c_i\rangle \mid 1 \leq i \leq n\}$ tel que

- $\{t_i\} \cup H_i$ est un ensemble de termes de type T et c_i un terme booléen,
- et pour tout terme clos g de type T dans $T(\mathcal{C}_R)$, il existe un i et une substitution θ telle que
  - $g =_E t_i\theta$, et $c_i\theta =_E vrai$,
  - et pour tout terme $t' \in H_i$, il existe $g' \in T(\mathcal{C}_R)$ tel que $g \succ g' =_{\mathcal{E}} t'\theta$.

Dans l'induction explicite de [ZKK88], les ensembles couvrants sont utilisés à travers la technique de « l'hypothèse d'induction quantifiée ». Cette notion a été implantée dans le système `RRL` [KZ95] qui contient donc aussi une partie implicite.

Comme autres travaux sur l'induction explicite, nous pouvons citer ceux plus anciens de [Aub79] et le système de preuve `Oyster/CLaM` [BvHHS90].

1.3.2 Induction « sans induction »

L'induction sans induction est aussi appelée « induction implicite » en raison de l'absence d'axiomes de récurrence explicites, bien des méthodes utilisent des schémas d'induction implicite. Cependant, certains auteurs distinguent cette dernière catégorie (voir plus bas).

L'induction sans induction est par essence une « *preuve par consistance* ». Au contraire des preuves par contradiction, la preuve par consistance consiste à montrer la consistance d'un ensemble d'équations constitué de la conjecture à prouver et des axiomes définissant les fonctions apparaissant dans la conjecture ainsi que les fonctions dépendantes. Si à la fin du processus aucun « contre exemple » n'a été généré, alors sous certaines conditions, il peut être affirmé que la conjecture est une conséquence inductive de l'ensemble d'axiomes initial.

Cette méthode a été découverte au tout début des années 1980 par un groupe d'auteurs ([Mus80, Lan81, HH82]) ce qui a généré depuis beaucoup de travaux. Le cœur de la méthode de preuve par consistance est décrit pour le cas purement équationnel dans [Bac91] et pour le cas non-équationnel dans [Com00].

Étant donné un ensemble E d'axiomes et un unique modèle minimal de Herbrand $\mathcal{H}$ de E, l'idée de la preuve par consistance est d'utiliser une axiomatisation (du premier ordre) $\mathcal{A}$ de $\mathcal{H}$ tel que $\phi \cup \mathcal{A} \cup E$ est consistant si et seulement si ϕ est valide dans $\mathcal{H}$.

• ***Exemple* 1.3.1.** Pour les deux équations définissant l'addition sur les entiers naturels (équations de E 1.1.1), le modèle initial $\mathcal{H}$ est $\mathbb{N}$. L'axiomatisation $\mathcal{A}$ peut être

$$\mathcal{A} = \left\{ \begin{array}{c} \forall x.\ 0 \neq s(x) \\ \forall x,y.\ s(x) = s(y) \implies x = y \end{array} \right. \tag{1.3.2}$$

Prouver une conjecture ϕ (par exemple $x + y = y + x$), consiste alors à l'ajouter à $E \cup A$ et essayer d'en dériver une inconsistance.

Pour montrer la consistance (ou l'inconsistance) de $\phi \cup \mathcal{A} \cup E$, la méthode a recours aux techniques de « *saturation* » ([KB73, KM87]).

Les approches de [Mus80, HH82, JK89, Bac91] se distinguent grossièrement par le choix de l'axiomatisation $\mathcal{A}$ et les moyens de détection d'une inconsistance. La formulation de E – un système de réécriture convergent – et les techniques de saturation – la complétion de Knuth-Bendix – ne varient que sur quelques limitations.

Les avantages de cette méthode sont, sous certaines conditions, la complétude réfutationnelle (voir [GS92]) et la possibilité de l'utiliser dans les systèmes de preuve génériques tels que `HOL`, `Nuprl` ou `Isabelle`. Elle ne nécessite donc pas de systèmes de preuve dédiés, contrairement à l'induction explicite et implicite proprement dite. Par contre, la preuve par consistance se révèle relativement peu efficace dans les théories conditionnelles – un ensemble de clauses étant beaucoup plus difficile à saturer qu'un ensemble d'équations.

Induction implicite

Comme nous l'avons évoqué plus haut, la preuve par induction implicite se distingue de la preuve par consistance par l'utilisation de schémas d'induction implicites. Cependant, les techniques de l'induction implicite sont basées sur des idées développées dans le champ de la preuve par consistance.

Chacune de ces méthodes utilise un ensemble de termes (ou un ensemble de paires (contexte, terme)) qui est utilisé pour remplacer des variables d'induction. De telles substitutions produisent de nouvelles conjectures qui peuvent être simplifiées par des instances strictement plus petites que la conjecture originale (l'hypothèse d'induction). La preuve est achevée quand toutes les conjectures générées ont été simplifiées en des théorèmes inductifs connus.

• ***Exemple* 1.3.2.** Soit la définition suivante de l'addition d'entiers relatifs

$$E = \begin{cases} x + 0 &= x \\ s(x) + y &= s(x + y) \\ p(x) + y &= p(x + y) \\ s(p(x)) &= x \\ p(s(x)) &= x \end{cases} \tag{1.3.3}$$

Toutes ces équations peuvent être orientées de gauche à droite pour former un système de réécriture $\mathcal{R}$ terminant. Considérons la conjecture $x+0 = x$. L'ensemble de termes choisi pour substituer la variable x peut être $\{0, s(x), p(x)\}$. On obtient ainsi trois nouvelles conjectures : $0 + 0 = 0$, $s(x) + 0 = s(x)$, et $p(x) + 0 = p(x)$. Ces conjectures peuvent être simplifiées en utilisant $\mathcal{R}$ et l'hypothèse d'induction $x + 0 = x$ où x est « bloquée » (c'est-à-dire ne peut pas être instanciée).

Dans cet exemple, $0 + 0 = 0$ est simplifiée en $0 = 0$ (par le premier axiome). $s(x) + 0 = s(x)$ est simplifié en $s(x + 0) = s(x)$ (par le deuxième axiome) puis en $s(x) = s(x)$ par l'hypothèse d'induction. Similairement, $p(x) + 0 = p(x)$ est simplifié en $p(x) = p(x)$. Ceci complète la preuve.

Nous distinguons deux principaux types d'ensembles permettant une représentation des termes : les ensembles couvrants (voir définition 1.20) et les ensembles tests (voir plus bas). Ces ensembles sont une représentation finie du modèle initial d'une théorie.

Parmi ces méthodes, nous pouvons citer les approches de « l'induction par réécriture » [Red90] et « l'induction par ensemble couvrant » [Zha88]. Ces deux approches, relativement similaires, sont basées sur des ensembles couvrants et une forme du théorème suivant :

Théorème 1.1. [Zha88] *Soit une formule $P(\vec{x})$, et $\left\{\left\langle \vec{t_i}, H_i, c_i \right\rangle \mid 1 \leq i \leq n\right\}$ un ensemble couvrant de $\vec{x}$ selon $\succ$. Si pour tout i, $\left\{P(\vec{t_i})$ **si** $\bigwedge_{\vec{t'} \in H_i} P(\vec{t'}) \wedge c_i\right\}$ est un théorème inductif, alors $P(\vec{x})$ est un théorème inductif.*

Cette méthode est implantée dans le système logiciel de preuve `RRL` [KZ95].

L'approche de « l'induction par ensemble test » [KR90, BKR95] diffère des deux précédentes essentiellement dans le choix des ensembles utilisés pour représenter le modèle initial.

Définition 1.21. [Kou92] Pour un système de réécriture $\mathcal{R}$, un *ensemble test*, noté $TS(\mathcal{R})$, est un ensemble fini de termes dans $T(\mathcal{C}_R,\mathcal{X})$ en forme normale selon $\mathcal{R}$ ayant les propriétés suivantes :

1. pour tout terme s clos en forme normale, il existe un terme t dans $TS(\mathcal{R})$ et une substitution σ close tel que $t\sigma = s$.
2. tout terme non-clos de $TS(\mathcal{R})$ n'a de variable qu'à une profondeur égale ou supérieure à la profondeur de $\mathcal{R}$ moins 1.

La première propriété est aussi une propriété des ensembles couvrants. Avec cette seule propriété, ces ensembles ne peuvent permettre la réfutation de conjecture. La deuxième propriété des ensembles tests permet de le faire ; elle assure que si une instance test[4] ne correspond à aucune partie gauche de $\mathcal{R}$, cela révèle une inconsistance. Cette méthode est *réfutationnellement complète* pour un système de réécriture convergent sur les termes clos et suffisamment complets [KR95]. Elle est implantée dans le système de preuve `SPIKE` [BKR92].

Les différences entre les méthodes se trouvent donc sur le choix des variables d'induction, la définition de l'ensemble de représentation, et sur les conditions d'utilisation de l'hypothèse d'induction.

4. C'est-à-dire l'instance d'une conjecture générée par la substitution d'une variable d'induction par un élément de l'ensemble test.

1.3.3 Heuristiques de découverte de généralisation

Pour une axiomatisation $\mathcal{A}$ donnée, une généralisation d'une formule ϕ est une formule ϕ' telle que

$$\mathcal{A} \models_{ind} \phi' \implies \mathcal{A} \models_{ind} \phi$$

Par exemple $0 + y = 0$ est une généralisation de $0 + sum(x) = 0$[5]. Mais $z + sum(x) = z$ est aussi une généralisation de $0 + sum(x) = 0$ même si cette seconde conjecture n'est pas un théorème inductif[6]. Une généralisation incorrecte de ϕ, telle que $z + sum(x) = z$, est une généralisation ϕ' telle que

$$\mathcal{A} \not\models_{ind} \phi' \wedge \mathcal{A} \models_{ind} \phi$$

Une généralisation peut être obtenue par l'inversion d'une règle d'inférence correcte (cf. [Wal94b]). Par exemple les deux généralisations ci-dessus sont obtenues par inversion de la règle de substitution. En effet, une substitution remplace une variable par un terme et nous avons remplacé un terme par une variable. Mais on peut aussi inverser d'autres règles d'inférence. Par exemple la règle sur les fonctions : $x + y = y + x$ est une généralisation de $sum(x + y) = sum(y + x)$. Mais $-x = x$ est une généralisation incorrecte de $carre(-x) = carre(x)$.

Dans [Wal94b], il est exposé l'ensemble des généralisations pouvant être obtenus par inversion d'une règle d'inférence, y compris les généralisations décrites dans [Aub79]. Pour chaque inversion, il est décrit les risques d'engendrer des généralisations incorrectes. Cette description n'est pas vraiment une méthode heuristique de généralisation. En effet, le nombre de conjectures pouvant être générées de cette manière est par trop grand et le risque d'engendrer des généralisation incorrectes est trop élevé. Même si certains heuristiques simples (tel que la « principe de sous-terme commun ») sont proposés pour réduire l'espace de recherche ou pour filtrer les conjectures générées.

5. Quelque soit la définition de la fonction sum.
6. Sauf si pour tout x, $sum(x)$ vaut 0.

Pour contrôler l'espace de recherche, quelques heuristiques ont été proposés. Les principaux heuristiques spéculant des généralisations ou des lemmes, ont trois principaux points communs :

1. *Ils ne donnent pas de garanties contre d'éventuelles introductions de conjectures non-valides.*
2. Ils sont basés sur l'étude de l'échec d'une tentative de preuve donnée.
3. Ils ne proposent que la généralisation « d'accumulateurs » (c'est-à-dire d'arguments de fonctions sur lesquelles celles-ci ne sont pas définies récursivement[7]).

Ondulation « *Rippling* »

L'heuristique « ondulatoire » (*rippling heuristic*) a été développé à Edimbourg au début des années 1990 pour l'induction explicite afin d'automatiser la preuve de théorèmes ([BSvH+93]). Il est implanté dans le système de preuve `CLAM` ([BvHHS90]). Essentiellement, cet heuristique étudie les différences entre une hypothèse d'induction et sa conclusion. Il « marque » ces différences – typiquement l'ajout de constructeurs ou la disparition de ceux-ci – à l'aide d'annotations et applique des règles de réécriture annotées (dites « règle de vagues » ou *waves-rules*) pour les supprimer.

Le principe de l'heuristique ondulatoire est de réécrire à l'aide de règle de vagues une conclusion d'induction de telle manière qu'un squelette (l'hypothèse d'induction) soit préservé et que les différences soit déplacées à des positions « non gênantes » (par exemple, à la racine du terme). Si cette réécriture fonctionne, il sera possible d'utiliser l'hypothèse d'induction. Basiquement, les différences peuvent être déplacées vers le haut, le bas ou sur d'autres arguments du terme à réécrire.

Ces règles de vagues sont automatiquement générées à partir des définitions de fonctions et de « propriétés logiques » comme la substitution, l'associativité, ou la

7. Par exemple, le premier argument de $+$ dans la définition $\{x + 0 = 0;\ x + s(y) = s(x + y)\}$.

commutativité[8]. En général, une utilisation réussie de l'heuristique requiert de multiples applications de telles règles.

• ***Exemple* 1.3.3.** Comme exemple simplifié, prenons l'axiomatisation E suivante :

$$E = \begin{cases} x + 0 &= x \\ s(x) + y &= s(x + y) \\ double(0) &= 0 \\ double(s(x)) &= s(s(double(x))) \end{cases} \tag{1.3.4}$$

Prenons aussi l'hypothèse d'induction $double(x) = x + x$ dont la conclusion d'induction annotée est $double(s(x)^{\uparrow}) = s(x)^{\uparrow} + s(x)^{\uparrow}$. La notation $x^{\uparrow}$ signifiant que les constructeurs du sous-terme x doivent être déplacés vers le haut du terme. Les règles de vagues utilisées par l'heuristique ondulatoire sont ici :

$$\begin{aligned} double(s(x)^{\uparrow}) &\rightarrow s(s(double(x)))^{\uparrow} \\ s(x)^{\uparrow} + y &\rightarrow s(x + y)^{\uparrow} \\ x + s(y)^{\uparrow} &\rightarrow s(x + y)^{\uparrow} \end{aligned} \tag{1.3.5}$$

La troisième règle est une version commutée de la définition de + mais qui doit cependant être *introduite par l'utilisateur ou proposée par la méthode utilisant la technique ondulatoire.*

L'application des trois règles, dans cet ordre, réécrit la conclusion d'induction en $s(s(double(x))) = s(s(x + x))$ ce qui permet l'utilisation de l'hypothèse d'induction.

Comme nous venons de le voir, l'heuristique ondulatoire n'est pas directement un heuristique de découverte de généralisation, mais un outil permettant l'implantation de telles méthodes.

8. Néanmoins, ces propriétés doivent être spécifiées par l'utilisateur.

Critique de divergence « *Divergence Critic* »

Cette méthode, proposée dans [Wal94a, Wal96], a pour but d'automatiser la découverte de généralisations et de lemmes. Cette méthode fonctionne en deux parties :

1. Une divergence est repérée par une méthode de mise en correspondance de conclusions d'induction successives, cette divergence est représentée par un calcul de différences entre les conclusions,

2. Cette différence est recouverte par l'heuristique d'ondulation pour donner un lemme filtré par plusieurs heuristiques.

La méthode repérant la divergence (« *Difference Matching* » [BW92]) fonctionne schématiquement en analysant la différence entre plusieurs conclusions d'induction successives et en les annotant. Empiriquement, le nombre d'induction à étudier est fixé à trois.

L'annotation des différences permet à l'heuristique d'ondulation de proposer une série de lemmes. Ces lemmes sont ensuite filtrés par un vérificateur de types et un heuristique d'inconsistance (qui n'est pas complet). Finalement, Les lemmes restant qui peuvent être recouverts par d'autres sont éliminés[9].

Les deux étapes principales – identification de la différence et spéculation de lemmes – peuvent être décrites ainsi :

9. Par exemple, $s(x) + s(y) = s(s(x) + y)$ et $0 + s(y) = s(0 + y)$ sont recouvertes par $x + s(y) = s(x + y)$.

1. Pour une série de conclusions d'induction $s_i = t_i$, supposons qu'il existe des contextes G et H du second ordre non triviaux tels que pour tout j, il existe une différence maximale U_j

$$s_j = G(U_j) \text{ et } s_{j+1} = G(H(U_j)^{\uparrow})$$

2. Le critique propose des conjectures de la forme, (où le contexte F est proposé par différents heuristiques)

$$G(H(U_0)^{\uparrow}) = F(G(U_0))^{\uparrow}$$

De tels lemmes fixent la divergence mais la preuve de ceux-ci peut elle-même diverger. Dans [Wal96], il est aussi proposé, dans ce sens, une règle « transversale ». Cette règle propose la généralisation d'un « accumulateur ». Schématiquement, la méthode identifie un accumulateur stable lors de la succession des inductions puis utilise l'heuristique des « termes principaux » [Aub79] et celui dit de « l'égalité ».

Pour la règle transversale, les deux étapes principales peuvent être représentées (avant généralisation) ainsi :

$$s_j = G(U_j,Acc) \text{ et } s_{j+1} = G(H(U_j)^{\uparrow},Acc)$$

qui donne

$$G(H(U_0)^{\uparrow},Acc) = G(U_0,F(Acc)^{\uparrow})$$

Cette méthode a été implantée dans une version du système logiciel de preuve par induction implicite `SPIKE`. Ceci ne fut possible que grâce à un des avantages principaux de cette méthode : une quasi-complète automatisation.

Cependant, les limitations de cette méthode sont intrinsèquement dues aux nombreux heuristiques auxquels elle fait appel. Comme le montre l'algorithme de spéculation lui-même, l'identification des différences (et donc les généralisations) ne peuvent être introduites à plusieurs positions dans les termes. Un nombre d'occurrences d'une

même variable supérieure à deux (par exemple $x+(x+x) = (x+x)+x$) implique quasiment toujours un échec de la méthode. De plus, aucun exemple probant n'est fourni sur des conjectures concernant la multiplication d'entiers (voir exemples de [Wal96]).

Critique de généralisation « *Generalization Critic* »

Ce critique ([IB96a]), comme le dernier, propose la généralisation d'accumulateurs et est basé sur l'heuristique ondulatoire. Ces accumulateurs (*sinks*) sont annotés spécialement.

La méthode est construite sur les notions de « plan de preuve » et de « système tactique de preuve ». Schématiquement, un plan de preuve guide une preuve automatique en choisissant, parmi un ensemble de tactiques, celles qui lui semble nécessaires à l'achèvement de la preuve d'une conjecture donnée. En ce sens, cette méthode est une extension directe de [BSvH$^+$93, IB96b].

Afin d'appréhender plusieurs positions d'accumulateur, le critique utilise des notions d'accumulateurs primaires et secondaires. Bien que ne fonctionnant pas sur les mêmes heuristiques, ceci permet de dépasser l'une des limitations du critique de divergence qui ne peut identifier qu'une position d'accumulation.

Ce critique a été développé pour l'induction explicite et est donc capable de manipuler des formules du second ordre. Enfin, ce critique a été implanté dans le logiciel de preuve `CLAM`.

Toutefois, cette méthode est présenté parfois comme une « organisation de l'espace de recherche », même si certaines généralisations sont obtenues automatiquement (moyennant tout de même l'utilisation de lemmes non-triviaux introduits par l'utilisateur, cf. exemples de [IB96a]). Il est notamment précisé, dans [IB96a], que le critique est une « base pour permettre une meilleure interaction avec l'utilisateur » en lui permettant de sélectionner ou de compléter une généralisation parmi celles proposées

par le critique. Toujours en nous basant sur les exemples de [IB96a], il n'est pas proposé de généralisation automatique de formules concernant la multiplication d'entiers naturels ou contenant plus de deux occurrences d'une même variable dans un même terme. De plus, il n'est pas précisé si toutes les « règles de vagues » utilisées ont été générées automatiquement ou introduites par l'utilisateur.

Découverte de lemme « *Lemma Discovery* »

Cette approche, proposée dans [KS96], n'utilise pas l'heuristique ondulatoire et n'annote donc pas les conjectures (sauf certaines variables). Les accumulateurs ne sont donc pas vus, comme dans les approches précédentes, par des termes annotés mais par des variables qui ne font pas partie des variables d'induction (dites « *variables de non-induction* »). Il n'est pas non plus fait appel à l'étude des règles de réécriture (ou de vagues) mais seulement à l'étude de la tentative de preuve.

Schématiquement, cette approche fonctionne en quatre étapes :

1. Elle « rapproche » une conclusion d'induction de son hypothèse à l'aide d'un ensemble D « d'équations de différence » portant sur les variables de non-induction.
2. Elle calcule un ensemble de « contraintes » à l'aide de D et de la définition des fonctions apparaissant dans D.
3. Elle spécule une instanciation des variables de non-induction de l'hypothèse d'induction pour recouvrir la forme que prennent ces variables dans la conclusion d'induction.
4. Elle applique cette instanciation aux équations de différence qu'elle généralise pour obtenir les lemmes nécessaires, puis elle filtre ces lemmes à l'aide d'un filtre de consistance.

De plus, cette approche part de l'observation que les généralisations habituellement proposées ne sont capables que de prendre en compte des accumulateurs déjà présents dans les termes. Bien que toujours basée sur la généralisation d'accumulateurs, elle se propose d'introduire des accumulateurs s'il n'en existe pas déjà.

• ***Exemple* 1.3.4.** [KS96] Soit l'axiomatisation suivante :

$$E = \begin{cases} ap(x,\varnothing) & = x \\ ap(c.x,y) & = c.ap(x,y) \\ rev(\varnothing) & = \varnothing \\ rev(c.x) & = ap(rev(x),c.\varnothing) \\ R(\varnothing,y) & = y \\ R(c.x,y) & = R(x,c.y) \end{cases} \tag{1.3.6}$$

et l'hypothèse $rev(R(x,\varnothing)) = x$ dont la conclusion d'induction est $rev(R(xc,c.\varnothing)) = c.xc$. L'ensemble différence généré est

$$D1 = \{rev(R(xc,c.\varnothing)) = rev(R(x,\varnothing));\ c.xc = x\}$$

Dans ce cas, l'algorithme de spéculation échoue car il n'y a pas de variable de non-induction. La méthode propose alors d'introduire un « schéma de terme » (*term scheme*) $T(x,z)$ pour obtenir l'équation généralisée

$$rev(R(x,z)) = T(x,z)$$

Une application de l'algorithme sur cette équation donne[10]

$$D2 = \{rev(R(x,c.z)) = rev(R(x,z'));\ T(c.x,z) = T(x,z')\}$$

Les contraintes calculées donnent alors l'instanciation $T(x,z) = ap(rev(z),x)$ (à l'aide d'un lemme introduit par l'utilisateur : $ap(\varnothing,x) = x$).

10. Il n'est nulle part précisé dans [KS96], pourquoi dans ce cas, la variable d'induction x n'est pas substituée (en xc par exemple), contrairement à l'induction effectuée sur l'hypothèse originale. Il est alors beaucoup plus facile d'unifier les différences.

Dans [KS96], il est précisé que tous les exemples de [IB96b] ont été traité par cette approche (sans toutefois préciser comment, ni s'il y a eu recours à des lemmes particuliers). Ici, un exemple concernant la multiplication est proposé, cependant il s'agit ici d'une définition « itérative »[11] et la généralisation ne s'effectue que sur l'argument d'accumulation (le troisième). Enfin, cette méthode a été implantée dans le logiciel de preuve par induction `RRL`.

Analyse

Toutes ces approches se sont révélées efficaces sur de nombreux exemples. Cependant, nous ne proposons pas maintenant une étude comparative de leurs capacités respectives, mais une systématisation de leurs limitations qui sont inhérentes à leurs caractéristiques communes.

Il faut en premier lieu rappeler qu'aucune de ces approches ne peut apporter de garantie contre l'introduction de lemmes ou de généralisations non-valides[12]. Elles sont aussi basées sur un grand nombre d'heuristiques, qui sont autant de points d'amélioration possible, mais autant de « boîtes noires » dont l'efficacité et la correction sont difficiles à vérifier.

Enfin, strictement du point de vue de leur capacité, elles ne proposent que la généralisation d'accumulateurs (c'est-à-dire de variables qui ne sont pas des variables d'induction). Ceci pose un problème pour les conjectures du type $ap(x,ap(x,x)) = ap(ap(x,x),x)$ (c.f. axiomatisation 1.3.6) où la généralisation doit porter à la fois sur le premier argument (qui est une variable d'induction) et sur le dernier, pour donner $ap(y,ap(x,z)) = ap(ap(y,x),z)$.

De plus, nous croyons que leur apparente difficulté à traiter la multiplication est due à l'absence de réel argument d'accumulation dans sa définition et donc à

11. Similaire à $\{mult(x,0,z) = z;\ mult(x,s(y),z) = mult(x,y,z+x)\}$.

12. Sauf à tester les lemmes proposés dans un système de preuve réfutationnellement complet – comme il est aussi proposé dans [Wal96] – ce qui est clairement prohibitif, pour des raisons d'efficacité.

l'impossibilité de trouver un point de divergence précis. Par exemple, la conjecture $x * (x + x) = (x * x) + (x * x)$ doit être généralisée sur le deuxième argument pour donner $x * (y + z) = (x * y) + (x * z)$, or, cet argument n'est pas un accumulateur pour $*$[13].

Un autre désavantage de ces techniques(excepté le critique de divergence) est une faible automatisation. Elles requièrent souvent des interactions avec l'utilisateur que ce soit dans la définition des règles de vagues, dans l'apport de lemmes, ou dans la sélection des généralisations proposées.

13. C'est aussi le cas pour une conjecture du type $mult(x,x+x,0) = mult(x,x,mult(x,x,0))$ devant être généralisée en $mult(x,y+z,0) = mult(x,y,mult(x,z,0))$.

Chapitre 2

Notions préliminaires

Dans ce chapitre, nous introduirons les notions nécessaires à la lecture des chapitres suivants. Nous formaliserons nos définitions de positions et de variables d'induction. Enfin, nous définirons ce que nous appelons les systèmes « monomorphiques » ainsi que leurs propriétés.

Les variables d'induction sont identifiées grâce à l'étude du système de réécriture dans lequel une conjecture donnée doit être validée. Cette étude nous permet d'identifier le ou les arguments à partir desquels une fonction est définie récursivement. À l'aide de cette notion, nous pourrons trouver dans une conjecture quelconque les variables les plus « intéressantes » pour un schéma d'induction.

Les systèmes monomorphiques forment un sous-ensemble des systèmes de réécritures convergents sur les termes clos et complètement définis. Cette classe de systèmes est importante dans le cadre de nos contributions principales. Schématiquement, on peut dire que leur modèle initial est composé des termes ayant une « structure linéaire ».

2.1 Induction

Bien que nos travaux puissent se révéler utiles à toutes les méthodes d'induction (implicite, explicite ou par consistance – voir section 1.3), nous définirons un ensemble de règles d'inférences pour montrer les capacités de nos heuristiques et méthodes. Ces règles d'inférences se placent dans le contexte de l'induction implicite.

Les variables d'induction sont définies à l'aide d'un heuristique attribuant un *poids* à chaque variable apparaissant dans une conjecture. Ce poids est calculé à partir de la définition des fonctions apparaissant dans cette conjecture. La (ou les) variable(s) cumulant les plus grands poids à gauche et à droite de l'équation sont choisies pour effectuer une induction sur cette équation (voir section 1.3).

2.1.1 Règles d'un système de preuve

Notre but n'est pas ici de définir un système de preuve complet, mais juste de proposer les règles minimales d'inférence pour « tester » nos méthodes. Grâce à cet ensemble de règles nous pourrons vérifier que nos généralisations ou nos schémas d'induction permettent bien d'arriver à la preuve d'une conjecture.

Remarque : À partir de maintenant, nous supposerons que toutes nos axiomatisations peuvent être compilées en un système de réécriture convergent sur les termes clos (voir définition 1.17 et section 1.1.3).

Pour un ensemble d'axiomes non-conditionnels, le nombre de règles d'inférence d'une méthode de preuve par induction implicite est réduit. De fait, nous n'introduisons que trois règles pour notre système de preuve « test ». Ces règles inspirées de [KR95][1].

1. La règle GEN dite de « génération » implantant l'induction elle-même.

1. Cependant, nous n'introduirons pas les notions d'induction mutuelle ou de réductibilité, qui, bien que très intéressantes pour l'induction, ne sont pas indispensables pour tester nos méthodes.

2. La règle SUP dite de « suppression » permettant de supprimer les tautologies.

3. La règle INC dite « d'échec » arrêtant la preuve sur une inconsistance.

Ce système de preuve fonctionne simplement en substituant une (ou des) variable(s) d'induction par des éléments de l'ensemble test du système de réécriture pris en compte. Ces éléments doivent être du même type que la variable d'induction.

Définition 2.1. Pour $\mathcal{R}$ un système de réécriture, une *substitution test* remplace les occurrences d'une variable d'induction par un élément de l'ensemble test du même type en renommant les variables.

Ensuite, ces instances sont simplifiées selon le système de réécriture et les instances plus « petites » de l'hypothèse d'induction.

Définition 2.2. Soit une algèbre de termes $T(\mathcal{F},\mathcal{X})$ munie d'un ordre de simplification $\succ$, et une équation $t = s$ telle que $t \succ s$. $(t = s)^{\prec}$ dénote la règle de réécriture $t \to s$ applicable uniquement à l'ensemble des termes $\{u \in T(\mathcal{F},\mathcal{X}) \mid t \succ u\}$.

Définition 2.3. Pour un système de réécriture $\mathcal{R}$ convergent et un ensemble d'équations C, la *génération* est la règle d'inférence[2]

$$\textbf{GEN}: \quad C \cup \{t = s\} \vdash C \cup \{(t\sigma = s\sigma)\downarrow_{R \cup \{(t\sigma = s\sigma)^{\prec}\}} \mid \sigma \text{ une substitution test }\}$$

Les conclusions d'induction simplifiées qui sont des tautologies sont supprimées.

Définition 2.4. Pour un système de réécriture $\mathcal{R}$ convergent et pour un ensemble d'équations C, la *suppression* est la règle d'inférence

$$\textbf{SUP}: \quad C \cup \{t = t\} \vdash C$$

Les conclusions qui montrent une inconsistance arrêtent la tentative de preuve. Pour un système de réécriture convergent sur les termes clos, une équation dont les deux côtés sont des termes clos et dont les formes normales sont syntaxiquement différentes montre une inconsistance.

Définition 2.5. Pour un système de réécriture $\mathcal{R}$ convergent et pour un ensemble

2. Nous adoptons une légère généralisation de la notation $\downarrow_{\mathcal{R}}$: $(t = s)\downarrow_{\mathcal{R}}$ signifiant $t\downarrow_{\mathcal{R}} = s\downarrow_{\mathcal{R}}$.

d'équations C, l'*échec* est la règle d'inférence

$$\textbf{INC}: \quad C \cup \{t = s\} \vdash \square$$

Si t,$s \in T(\mathcal{F}_R)$ *et* $t{\downarrow_{\mathcal{R}}} \neq s{\downarrow_{\mathcal{R}}}$

Remarque : Ce système de preuve test, bien qu'implantant une règle de réfutation et utilisant les ensembles tests, n'est pas censé être réfutationnellement complet.

Ces règles doivent être utilisées sur tout l'ensemble des conjectures par ordre de priorité :

$$\textbf{SUP} \gg \textbf{INC} \gg \textbf{GEN}$$

• ***Exemple* 2.1.1.** Soit l'équation $t = s$

$$0 * x = 0$$

Pour le système de réécriture suivant :

$$\mathcal{R} = \begin{cases} x + s(y) & \to s(x + y) \\ x + 0 & \to x \\ x * s(y) & \to (x * y) + x \\ x * 0 & \to 0 \end{cases} \tag{2.1.1}$$

dont l'ensemble test est $TS(\mathcal{R}) = \{0,s(x)\}$.

Le système $\mathcal{R}$ est orienté grâce à l'ordre lexicographique de simplification $\succ$ fondé sur la précédence $* > + > s > 0$. Les instances de l'hypothèse $t = s$ peuvent donc être orientées de gauche à droite.

Si l'on suppose que x est une variable d'induction (voir section suivante), les substitutions tests σ_0 et σ_s remplacent les occurrences de x respectivement par 0 et

$s(x')$. Appliquons nos règles d'inférence à la conjecture $x * 0 = 0$:

$$\begin{aligned}\{t = s\} &\vdash \left\{(0 * 0 = 0)\downarrow_{\mathcal{R}\cup\{(t\sigma_0 = s\sigma_0)^{\prec}\}};\ (0 * s(x') = 0)\downarrow_{\mathcal{R}\cup\{(t\sigma_s = s\sigma_s)^{\prec}\}}\right\} \\ &= \left\{0 = 0;\ ((0 * x') + 0 = 0)\downarrow_{\mathcal{R}\cup\{(t\sigma_s = s\sigma_s)^{\prec}\}}\right\} \\ &= \left\{0 = 0;\ (0 + 0 = 0)\downarrow_{\mathcal{R}\cup\{(t\sigma_s = s\sigma_s)^{\prec}\}}\right\} = \{0 = 0;\ 0 = 0\} \\ &\vdash \{0 = 0\} \vdash\end{aligned}$$

2.1.2 Variables d'induction

Les « positions d'argument » d'une fonction $f \in \mathcal{F}_R$ sont l'ensemble des positions de longueur 1 qui appartiennent au domaine de tous les termes qui ont pour racine f.

Définition 2.6. Les *positions d'argument* d'un symbole de fonction $f \in \mathcal{F}$ sont les positions de l'ensemble :

$$P_f = \{p \mid |p| = 1 \text{ et } \forall t \in T(\mathcal{F},\mathcal{X}),\ t(\varepsilon) = f \implies p \in dom(t)\}$$

Dans un système de réécriture $\mathcal{R}$, nous décrivons l'« ensemble définition » d'un symbole $f \in \mathcal{D}_R$ comme l'ensemble de règles dont les parties gauches ont pour racine ce symbole f et dont tous les arguments sont des termes constructeurs.

Définition 2.7. Pour $\mathcal{R}$ un système de réécriture, l'*ensemble définition* de $f \in \mathcal{D}_R$ est l'ensemble des règles :

$$\mathcal{R}_f = \{l \to r \in \mathcal{R} \mid l(\varepsilon) = f \implies \forall p \in P_f,\ l(p) \in T(\mathcal{C}_R,\mathcal{X})\}$$

• ***Exemple* 2.1.2.** Soit le système de réécriture

$$\mathcal{R} = \begin{cases} plus(x+1,y) & \to plus(x,y) + 1 \\ plus(1,y) & \to y + 1 \\ plus(plus(x,y),1) & \to plus(x,y) + 1 \end{cases} \tag{2.1.2}$$

Les symboles de fonction $+$ et 1 sont les constructeurs de $\mathcal{R}$ donc, $R_{plus} = \{plus(x+1,y) \to plus(x,y) + 1;\ plus(1,y) \to y + 1\}$.

Parmi les positions d'arguments d'une fonction f, il peut en exister certaines sur lesquelles la fonction est définie récursivement. Dans les parties gauches de l'ensemble définition de f, on trouve alors à ces positions non pas une variable, mais un terme dont la racine est un symbole de constructeur ($\mathcal{C}_R$). Ce terme peut être :

- sans argument (constante) ou
- avec argument(s) (terme constructeur).

On définit les « positions d'induction » d'une fonction comme étant les positions des arguments que le système de réécriture utilise pour définir récursivement la fonction. Ce sont sur ces positions qu'il est le plus naturel d'effectuer un induction car, une induction effectuée sur d'autres positions ne permettent aucune réécriture.

Définition 2.8. Pour $\mathcal{R}$ un système de réécriture et $f \in \mathcal{D}_R$ une fonction les *positions d'induction* de f sont les positions de l'ensemble :

$$PI_{\mathcal{R}}(f) = \{p \in P_f \mid \forall (l \to r) \in \mathcal{R}_f, l(p) \in \mathcal{C}_R\}$$

• ***Exemple* 2.1.3.** Pour toutes les règles $(l \to r) \in \mathcal{R}_{plus}$ nous avons $l(1) \in \{1,+\}$ (ce qui n'est pas le cas pour la position 2), donc $PI_{\mathcal{R}}(plus) = \{1\}$. En effet, une substitution test – la substitution d'une variable par un élément de l'ensemble test $TS(\mathcal{R}) = \{1,x+1\}$ (voir définition 1.21) – appliquée à une variable en premier argument, permet une réécriture alors qu'appliquée à une variable en deuxième argument, est « inutile ». Par exemple pour la substitution test σ,

$$plus(x,y)\sigma\downarrow_{\mathcal{R}} = plus(x'+1,y'+1)\downarrow_{\mathcal{R}} = plus(x',y'+1)+1$$

Cependant, un terme quelconque n'est pas composé d'une unique fonction. Pour pouvoir choisir les meilleures variables sur lesquelles effectuer une induction, nous étudions les « *chemins* » parcourant le terme. Nous appelons « chemins » les positions d'un terme qui sont composées d'une série de positions d'argument (qu'elles soit d'induction ou non).

Il faut donc choisir, parmi toutes les variables d'un terme, celles qui se trouvent en position d'induction « finale »[3], et en priorité parmi celles-ci, celles qui sont sur un chemin finissant sur le maximum de positions d'induction.

Définition 2.9. Pour $\mathcal{R}$ un système de réécriture et t un terme, le *poids d'induction* d'une variable $x \in \mathcal{V}_R(t)$, noté $pi(x,t)$ est calculé récursivement ainsi :

$$
\begin{aligned}
pi(x,t) &= pii(x,t,1) \\
t \equiv x &\implies pii(x,t,i) = i \\
t \neq x \wedge t(\varepsilon) \in \mathcal{F}_R &\implies pii(x,t,i) = \sum_{p \in PI_{\mathcal{R}}(t(\varepsilon))} pii(x,t,i+1) + \sum_{p \in P_{t(\varepsilon)}/PI_{\mathcal{R}}(t(\varepsilon))} pii(x,t,0) \\
t \neq x \wedge t(\varepsilon) \notin \mathcal{F}_R &\implies pii(x,t,i) = 0
\end{aligned}
$$

En fait, si l'on poursuit l'analogie terme/arbre, le poids d'induction d'une variable est calculé ainsi :

1. À chaque feuille de l'arbre (variable), il est attribué un poids. Ce poids est égal à la longueur de la série de positions d'induction qui compose *la fin du chemin* amenant à cette feuille (plus 1 si le chemin n'est composé que de positions d'induction).
2. Le poids d'une variable précise est la somme des poids des feuilles syntaxiquement égales à cette variable.

• ***Exemple* 2.1.4.** Reprenons le système de réécriture 2.1.2, $PI_{\mathcal{R}}(plus) = 1$.

Soit le terme $t \equiv plus(plus(x,y),plus(y,z))$, nous avons les poids d'induction $pi(x,t) = 3$, $pi(y,t) = 1$, et $pi(z,t) = 0$.

Nous pouvons enfin définir les « variables d'induction » qui sont les variables possédant les poids d'induction les plus grands. De plus, nous étendons cette définition sur les équations.

Définition 2.10. Pour un terme t, $x \in \mathcal{V}_R(t)$ est une *variable d'induction* de t si

3. C'est-à-dire celles qui sont à un chemin finissant par une position d'induction.

pour toute variable $y \in \mathcal{V}_R(t)$

$$pi(x,t) \geq pi(y,t)$$

Pour une équation $t = s$, $x \in \mathcal{V}_R(t) \cup \mathcal{V}_R(s)$ est une *variable d'induction* de $t = s$ si pour toute variable $y \in \mathcal{V}_R(t) \cup \mathcal{V}_R(s)$,

$$pi(x,t) + pi(x,s) \geq pi(y,t) + pi(y,s)$$

L'intérêt d'une telle définition est simplement de choisir la meilleure variable pour effectuer une induction. Si le poids d'induction d'une variable ne tient pas en compte pas uniquement les positions d'induction « finales », c'est pour donner la priorité aux variables se trouvant sur un chemin qui est composé de toute *une série de positions d'induction*. Ces variables, si elles existent, sont celles qui impliqueront le plus de réécritures. Or, plus une conclusion d'induction est simplifiée, plus elle se rapprochera de son hypothèse ; comme le montrent les exemples ci-dessous.

Remarque : Cette définition des positions et des variables d'induction diffère de celle proposée dans [KU99a]. Nous avons préféré présenter cette version peu éloignée dans l'esprit, mais qui est plus efficace en terme de rapidité et plus précise dans ses choix.

Dans la définition de [KU99a], il est fait appel à des méthodes d'unification qui sont coûteuses, alors que la définition 2.9 effectue simplement un parcours de l'arbre syntaxique représentant le terme. Il peut même être programmé dans un système de preuve logiciel, un seul parcours de l'arbre pour toutes les variables. Ceci à l'aide d'une simple structure de données de type table associant chaque variable à son poids d'induction en cours de calcul.

Exemples

Prenons le système de réécriture $\mathcal{R}$ suivant :

$$\mathcal{R} = \begin{cases} ap(c.x,y) & \to c.ap(x,y) & ap(\varnothing,y) & \to y \\ r(c.l) & \to ap(r(l),c.\varnothing) & r(\varnothing) & \to \varnothing \\ cross(c.x,d.y) & \to c.(d.cross(x,y)) & cross(\varnothing,\varnothing) & \to \varnothing \\ cross(c.x,\varnothing) & \to c.x & cross(\varnothing,d.y) & \to d.y \end{cases} \tag{2.1.3}$$

Nous noterons en premier lieu que le système de réécriture $\mathcal{R}$ est orienté grâce à la précédence $r > ap > cross > . > \varnothing$.

Voici les positions d'induction pour chaque fonction de $\mathcal{D}_R$:

f	ap	r	$cross$
$PI_{\mathcal{R}}(f)$	{1}	{1}	{1,2}

- ***Exemple* 2.1.5.** $t = s$: $\mathbf{ap}(\mathbf{y},\mathbf{ap}(\mathbf{x},\mathbf{z})) = \mathbf{ap}(\mathbf{ap}(\mathbf{y},\mathbf{x}),\mathbf{z})$

Nous avons les poids d'induction suivants pour chaque variable [4] :

- $pi(y,t) + pi(y,s) = 2 + 3 = 5$,
- $pi(x,t) + pi(x,s) = 1 + 0 = 1$, et
- $pi(z,t) + pi(z,s) = 0 + 0 = 0$.

La variable y est donc la seule variable d'induction de l'équation. Il est ici évident que seule une induction sur y permettra d'appliquer l'hypothèse d'induction. En effet, la conclusion d'induction – générée en substituant y par $c.y'$ – simplifiée est alors :

$$c.ap(y',ap(x,z)) = c.ap(ap(y',x),z)$$

alors qu'appliqué sur x [5], on aurait obtenu $ap(y,c.ap(x',z)) = ap(ap(y,c.x'),z)$. Appliquée sur x et y, on aurait obtenu $c.ap(y',c.ap(x',z)) = c.ap(ap(y',c.x'),z)$. Dans chaque cas, la tentative de preuve aurait divergé car l'hypothèse d'induction n'aurait pas pu être appliquée.

4. Rappelons qu'un bonus de 1 est donné aux variables – ici y – qui sont sur un chemin qui est composé *uniquement* de positions d'induction.

5. En effet, x est en position d'induction « classique » (car terminale) à gauche de l'équation.

• ***Exemple* 2.1.6.** $t = s$: $\mathbf{r(ap(y,r(x))) = ap(x,r(y))}$

Notons tout d'abord que les instances de cette conjecture peuvent uniquement être orientées de gauche à droite car $r(ap(y,r(x))) \succ ap(x,r(y))$ selon la précédence $r > ap$.

Nous avons les poids d'induction suivants pour chaque variable :

- $pi(x,t) + pi(x,s) = 1 + 2 = 3$ et
- $pi(y,t) + pi(y,s) = 3 + 1 = 4$.

La conclusion d'induction – générée en substituant la seule variable d'induction y par $c.y'$ – simplifiée est alors :

$$ap(r(ap(y',r(x))),c.\varnothing) = ap(x,ap(r(y'),c.\varnothing))$$

En appliquant l'hypothèse d'induction $t \to s$ sur le sous-terme $r(ap(y',r(x)))$ à gauche de l'équation, on obtient $t' = s'$:

$$ap(ap(x,r(y')),c.\varnothing) = ap(x,ap(r(y'),c.\varnothing))$$

Nous avons alors les poids d'induction suivants pour chaque variable :

- $pi(x,t) + pi(x,s) = 3 + 2 = 5$ et
- $pi(y',t) + pi(y',s) = 1 + 2 = 3$.

La conclusion d'induction – générée en substituant x par $c'.x'$ – simplifiée est alors :

$$c'.ap(ap(x',r(y')),c.\varnothing) = c'.ap(x',ap(r(y'),c.\varnothing))$$

En appliquant l'hypothèse d'induction $t' \to s'$ sur la partie gauche de l'équation, on obtient :

$$c'.ap(x',ap(r(y'),c.\varnothing)) = c'.ap(x',ap(r(y'),c.\varnothing))$$

Ce qui complète la preuve.

Remarque : Si l'on avait appliqué une induction sur x (ou à x et y à la fois) dans $t = s$, on aurait obtenu $r(ap(y,ap(r(x'),c.\varnothing))) = c.ap(x',r(y))$. Or, on ne peut appliquer l'hypothèse d'induction $t \to s$ sur cette conjecture.

• ***Exemple* 2.1.7.** $t = s$: $\mathbf{r(ap(x,y)) = ap(r(y),r(x))}$

Notons tout d'abord que les instances de cette conjecture peuvent uniquement être orientées de gauche à droite car $r(ap(x,y)) \succ ap(r(y),r(x))$ selon la précédence $r > ap$.

Nous avons les poids d'induction suivants pour chaque variable :

- $pi(x,t) + pi(x,s) = 3 + 1 = 4$ et
- $pi(y,t) + pi(y,s) = 0 + 3 = 3$.

La conclusion d'induction – générée en substituant x par $c.x'$ – simplifiée est alors :

$$ap(r(ap(x',y)),c.\varnothing) = ap(r(y),ap(r(x'),c.\varnothing))$$

En appliquant l'hypothèse d'induction $t \to s$ sur le sous-terme $r(ap(x',y))$ à gauche de l'équation, on obtient :

$$ap(ap(r(y),r(x')),c.\varnothing) = ap(r(y),ap(r(x'),c.\varnothing))$$

Il faut maintenant appliquer l'équation de l'exemple 2.1.5 (orientée de gauche à droite : $ap(ap(x,y),z) \to ap(x,ap(y,z))$) sur la partie gauche pour obtenir :

$$ap(r(y),ap(r(x'),c.\varnothing)) = ap(r(y),ap(r(x'),c.\varnothing))$$

Ce qui complète la preuve.

Remarque : Nous avons fait ici appel à un lemme cependant, une généralisation correcte (voir chapitre 5) telle que $ap(ap(Z,r(x')),W) = ap(Z,ap(r(x'),W))$ suivie d'une induction sur Z suffirait.

De plus, si l'induction avait été appliquée à y (ou à x et y à la fois) dans $t = s$, on aurait obtenu $r(ap(x,c.y')) = ap(ap(r(y'),c.\varnothing),r(x))$ et il aurait été impossible d'utiliser l'hypothèse d'induction $t \to s$ (même orientée $s \to t$).

2.2 Systèmes monomorphiques

Dans cette section nous allons apporter une notion centrale pour nos travaux : les systèmes « monomorphiques ». Après avoir défini cette notion, nous verrons comment vérifier qu'un système de réécriture donné est monomorphique. Enfin nous étudierons les propriétés de ces systèmes, notamment le fait que nous pouvons calculer la « jonction » de deux termes clos en forme normale.

2.2.1 Définition

Les systèmes monomorphiques sont donc un sous-ensemble des systèmes de réécriture. Leur intérêt réside dans le fait que les termes clos en forme normale issus de ces systèmes ont la propriété de pouvoir être décrits de manière régulière sous la forme d'une liste.

Définition 2.11. Un système de réécriture $\mathcal{R}$ convergent sur les termes clos et complètement défini est *monomorphique* si

- il n'existe qu'une seule constante – constructeur sans argument, noté $\perp_T$ – par type T,
- et si tout terme t clos en forme normale :

  $\forall(u,v) \in dom(t)$, tels que $type(t/u) = type(t/v) = type(t)$, nous avons $u \leq v$ ou $v \leq u$

Pour vérifier si un système de réécriture donné est monomorphique, nous devons étudier son ensemble test et définir un ordre sur les types présents dans le système. Un tel ordre sur les types nous permet par exemple, de définir des « listes d'entiers ». Le type *liste* étant supérieur au type *entier*, une liste peut contenir des entiers mais

pas l'inverse.

Définition 2.12. L'ensemble des types d'un système de réécriture $\mathcal{R}$ est

$$\mathcal{T}_R = \{T \mid \exists t \in TS(\mathcal{R}),\, type(t) = T\}$$

Définition 2.13. Pour un système de réécriture $\mathcal{R}$ et deux types $(T,T') \in \mathcal{T}_R$,

$$T \succ T' \text{ si } T \neq T' \text{ et } \exists s \in T(\mathcal{C}_R),\, type(s) = T \wedge \exists p \in dom(s),\, type(s/p) = T'$$

Remarque : L'ordre ainsi définit n'est forcement total.

• ***Exemple* 2.2.1.** Soit le système de réécriture suivant :

$$\mathcal{R} = \begin{cases} x + s(y) & \to s(x + y) \\ x + 0 & \to x \\ sum(c.l) & \to c + sum(l) \\ sum(\varnothing) & \to 0 \\ f(s(x)) & \to g(g(f(x))) \\ f(0) & \to b \end{cases} \tag{2.2.1}$$

Ce système contient trois types.

- Le type T_1 dont les constructeurs sont 0 et s,
- Le type T_2 dont les constructeurs sont $\varnothing$ et .,
- Le type T_3 dont les constructeurs sont g et b.

Nous avons $T_2 \succ T_1$ mais aucune relation entre T_3 et T_1 ou T_2.

Une étude de l'ensemble test du système nous permet de savoir

- s'il existe bien une seule constante par type et
- si les constructeurs ont au plus un argument de leur propre type et aucun argument d'un type supérieur.

Propriété 2.1. Un système de réécriture convergent sur les termes clos $\mathcal{R}$ est monomorphique si la relation $\succ$ est une relation d'ordre strict et si et seulement si[6]

$$\forall T \in \mathcal{T}_R,\ \exists! t \in TS(\mathcal{R}),\ P_{t(\varepsilon)} = \varnothing \wedge type(t) = T$$
$$\text{et } \forall t \in TS(\mathcal{R}),\ \mathcal{V}_R(t) \neq \varnothing \implies \exists! p \in P_{t(\varepsilon)},\ type(t/p) = type(t)$$

Démonstration. • Si $\succ$ est une relation d'ordre strict, on ne peut avoir de terme $t \in T(\mathcal{C}_R)$ et de position $p \in dom(t)$ telle que $type(t/p) \succ type(t)$.

• Soit $\perp$ une constante de $T(\mathcal{F}_R)$. De par la définition 1.21 des ensembles tests, pour tout terme T clos en forme normale, il existe un terme $s \in TS(\mathcal{R})$ et une substitution σ tels que $s\sigma = \perp$. Donc, soit $\perp \in TS(\mathcal{R})$, soit il existe une variable $x \in TS(\mathcal{R})$. De plus, tout terme non-clos de $TS(\mathcal{R})$ n'a de variable qu'à une profondeur égale ou supérieure à la profondeur de $\mathcal{R}$ moins 1. Donc, si la profondeur du système est supérieure à 1[7], il n'y a pas de variable dans $TS(\mathcal{R})$ et toute constante $\perp$ est présente dans $TS(\mathcal{R})$.

• Si $\mathcal{R}$ n'est pas monomorphique il existe un terme $t \in T(\mathcal{C}_R)$ et deux positions u et v tels que $type(t/u) = type(t/v) = type(t) = T$ et ni $u \leq v$ ni $v \leq u$. Soit p le plus grand préfixe commun à u et v, posons $(u',v') \in P_{t(p)}$ telles que $pu' \leq u$ et $pv' \leq v$. D'après la définition 1.21, il existe $s \in TS(\mathcal{R})$ tel que $s\sigma = t/p$. Or, d'après la définition 2.13, comme $\succ$ est une relation d'ordre, $type(t/pu') = type(t/pv') = type(t/p) = T$, ce qui contredit la propriété. □

• ***Exemple* 2.2.2.** Soit le système de réécriture suivant

6. Le quantificateur $\exists!$ signifie « *il existe un et un seul* ».
7. Ce que nous supposerons vrai pour tout système pris en compte.

$$\mathcal{R} = \begin{cases} s(x) < s(y) \to x < y & 0 < 0 \to F & 0 < s(y) \to V & s(x) < 0 \to F \\ eq(s(x),s(y)) \to eq(x,y) & eq(0,0) \to V & eq(0,s(y)) \to F & eq(s(x),0) \to F \\ V \vee x \to V & F \vee x \to x & & \\ V \wedge x \to x & F \wedge x \to F & & \\ x + s(y) \to s(x + y) & x + 0 \to x & & \\ x * s(y) \to (x * y) + x & x * 0 \to 0 & & \end{cases} \tag{2.2.2}$$

Divisons ce système de réécriture en deux ;

- Posons $\mathcal{R}_1 = \mathcal{R}_< \cup \mathcal{R}_{eq} \cup \mathcal{R}_\vee \cup \mathcal{R}_\wedge$, le système $\mathcal{R}_1$ n'est pas monomorphique car il possède 2 constantes V et F du même type ($TS(\mathcal{R}_1) = \{V,F,0,s(x)\}$).
- Posons $\mathcal{R}_2 = \mathcal{R}_+ \cup \mathcal{R}_*$, le système $\mathcal{R}_2$ est monomorphique car $TS(\mathcal{R}_2) = \{0,s(x)\}$ et
  - Il n'y a qu'une seule constante 0.
  - Le constructeur $s(x)$ ne possède qu'un seul argument du même type.

2.2.2 Propriétés

De par la définition des systèmes monomorphiques, nous pouvons identifier la position du seul argument d'un constructeur du même type que le constructeur lui-même.

Définition 2.14. Pour $\mathcal{R}$ un système de réécriture monomorphique et $t \in T(\mathcal{C}_R,\mathcal{X})$ un terme de type T, l'unique position qui vérifie

$$|p| = 1 \text{ et } type(t/p) = T$$

s'appelle la position d'argument *réflexive*, notée $RA(\mathcal{R},t(\varepsilon))$.

Remarque : Comme la position d'argument réflexive est unique, on note sans ambiguïté $c[x]$ un terme dit « terme constructeur » tel que $type(x) = type(c)$ au lieu de $c[x]_{RA(\mathcal{R},c(\varepsilon))}$.

Similairement, dans un terme A clos en forme normale, la constante du même type que A apparaît une seule fois dans l'arbre décrivant le terme (en fait, en fin de la « liste » décrivant le terme).

Propriété 2.2. Pour un système monomorphique $\mathcal{R}$, un terme $A \in T(\mathcal{C}_R)$ de type T, possède la propriété suivante :

$$\exists! p \in dom(A) \text{ tel que } A/p = \bot_T$$

• ***Exemple* 2.2.3.** Soit le système de réécriture monomorphique suivant :

$$\mathcal{R} = \begin{cases} x + s(y) & \to s(x + y) \\ x + 0 & \to x \\ sum(c.l) & \to c + sum(l) \\ sum(\varnothing) & \to 0 \end{cases} \tag{2.2.3}$$

- Le terme $A = 0.(s(0).(s(s(0)).\varnothing))$ contient plusieurs fois la constante 0, mais il ne contient qu'une seule fois la constante $\varnothing$ (à la position 222).
- Le terme $sum(A)\downarrow_{\mathcal{R}} = s(s(s(0)))$ ne contient qu'une seule fois la constante 0 (à la position 111).

2.2.3 Jonction de termes clos en forme normale

Comme la constante n'apparaît qu'une seule fois dans un terme clos en forme normale, nous pouvons « joindre » deux termes clos en forme normale dans un système monomorphique. Il suffit de remplacer la constante du premier terme par le second. Cette « jonction » forme un nouveau terme dont la « liste » qui le décrit commence par le premier terme et finit par le second.

Définition 2.15. Soit $\mathcal{R}$ un système de réécriture monomorphique, la *jonction* de deux termes A et B de même type T en forme normale close pour $\mathcal{R}$ – notée $A \otimes B$ – est le terme :

$$A[B]_p \text{ tel que } A/p = \perp_T$$

Les trois lemmes qui suivent énoncent les propriétés de cet opérateur de jonction.

Lemme 2.1. *Soit $\mathcal{R}$ un système de réécriture monomorphique, pour trois termes A, B, et C de même type clos en forme normale pour $\mathcal{R}$, nous avons :*

- *Si $A \otimes B = A \otimes C$ alors $B = C$*
- *Si $B \otimes A = C \otimes A$ alors $B = C$*

Démonstration. Soit T le type de A (et B et C). Comme $\mathcal{R}$ est monomorphique, d'après la définition 2.11,

- il existe une unique position p telle que $A/p = \perp_T$, donc $A \otimes B = A[B]_p$ et $A \otimes C = A[C]_p$. Comme $A \otimes B = A \otimes C$, nous avons $A[B]_p = A[C]_p$ et $B = C$.
- De même, il existe une unique position q telle que $B/q = \perp_T$, et une unique position r telle que $C/r = \perp_T$, donc $B \otimes A = B[A]_q$ et $C \otimes A = C[A]_r$. Comme $B \otimes A = C \otimes A$, nous avons $B[A]_q = C[A]_r$ et $B = C$.

□

Lemme 2.2. *Soit $\mathcal{R}$ un système de réécriture monomorphique, pour trois termes A, B, et C de même type clos en forme normale pour $\mathcal{R}$, nous avons :*

$$A \otimes (B \otimes C) = (A \otimes B) \otimes C$$

Démonstration. Soit T le type de A (et B et C), comme $\mathcal{R}$ est monomorphique, d'après la définition 2.11, il existe une unique position p telle que $A/p = \perp_T$, et une seule position q telle que $B/q = \perp_T$, donc $A \otimes (B \otimes C) = A[B[C]_q]_p = A[B]_p[C]_{pq}$.

De même, $A \otimes B = A[B]_p$ est un terme clos en forme normale pour $\mathcal{R}$ et il possède une seule position $r = pq$ telle que $(A \otimes B)/r = \perp_T$, donc $(A \otimes B) \otimes C = A[B]_p[C]_r$.

Nous avons alors bien $A \otimes (B \otimes C) = (A \otimes B) \otimes C$ □

Remarque : Comme nous venons de le voir, l'opérateur de jonction est associatif. Nous noterons donc la jonction de trois termes simplement : $A \otimes B \otimes C$.

Lemme 2.3. *Soit $\mathcal{R}$ un système de réécriture monomorphique, pour un terme $t \in T(\mathcal{C}_R,\mathcal{X})$, une position $p = RA(\mathcal{R},t(\varepsilon))$ et deux termes A et B clos en forme normale pour $\mathcal{R}$;*

$$t[A \otimes B]_p = t[A]_p \otimes B$$

Démonstration. Comme $t(\varepsilon) \in \mathcal{C}_R$ et $p = RA(\mathcal{R},t(\varepsilon))$, de par la définition 2.15 de l'opérateur de jonction, nous avons $t[A \otimes B]_p = t[\perp_{type(t)}]_p \otimes A \otimes B = t[A]_p \otimes B$. □

Chapitre 3

Aperçu de nos contributions principales

Dans ce chapitre, nous allons dans un premier temps expliquer les motivations qui sous-tendent notre approche puis illustrer nos principales définitions et méthodes à travers un exemple « non trivial ». Les notions les plus importantes sont celles de « sous-terme libre » et de « partition ». Enfin, nous montrerons comment effectuer des généralisations correctes ainsi qu'une nouvelle méthode pour prouver des théorèmes inductifs.

Les observations suivantes sont à l'origine de nos méthodes pour établir la validité inductive d'une équation :

- Prouver une conjecture $t = s$ par induction consiste à trouver une collection de sous-conjectures $t_1 = s_1 \dots t_n = s_n$. Certaines de ces conjectures peuvent être prouvées directement mais d'autres sont seulement réduites en d'autres conjectures $t_1 = s_1 \dots t_k = s_k$ qui nécessitent d'autres inductions. Or, une tentative de preuve d'un théorème du premier ordre échoue **toujours** si les hypothèses d'induction ne sont pas utilisées pour simplifier les sous-conjectures.
- Un moyen pour que la tentative de preuve réussisse est de « **simplifier** » la conjecture. Par exemple, on peut introduire des lemmes mais aussi généraliser ou

découper cette conjecture en plusieurs parties.

– Finalement, une équation $t = s$ est un théorème inductif – pour un système de réécriture $\mathcal{R}$ convergent sur les termes clos – **si et seulement si** les formes normales des instances closes – $\{t\}\downarrow_{\mathcal{R}}$ et $\{s\}\downarrow_{\mathcal{R}}$ – sont syntaxiquement égales deux à deux.

Si on trouve ce que chaque paire d'éléments de $\{t\}\downarrow_{\mathcal{R}}$ et $\{s\}\downarrow_{\mathcal{R}}$ ont « en commun », nous serons à même de simplifier cette conjecture. Un terme quelconque étant composé de fonctions, nous devons donc étudier le devenir des $f(\overrightarrow{A_{1\ldots n}})$[1] dans la séquence de réécriture amenant à leurs formes normales.

Comme nous le verrons dans ce chapitre, une analyse du système de réécriture $\mathcal{R}$ définissant les fonctions nous permet de savoir – dans certains cas – quelle est l'influence des différents arguments $A_1 \ldots A_n$ sur $f(\overrightarrow{A_{1\ldots n}})\downarrow_{\mathcal{R}}$ et donc sur les formes normales des instances closes de termes quelconques.

3.1 Sous-termes libres dans la construction des formes normales closes

Dans un terme, il peut exister un ensemble de sous-termes dont chacun engendre uniquement une partie précise de chacune des formes normales des instances closes du terme. En fait, ces sous-termes n'influencent, dans la séquence de réécriture amenant à la forme normale d'une instance close du terme, aucune des autres parties. Nous appellerons informellement ces sous-termes les « *sous-termes libres* » d'un terme.

Par exemple pour le système de réécriture

$$\mathcal{R} = \{ap(c.l,L) \to c.ap(l,L);\ ap(\varnothing,L) \to L;\ R(c.l,L) \to R(l,c.L);\ R(\varnothing,L) \to L\}$$

1. Termes dont la racine est une fonction f dont tous les arguments sont normalisés.

et le terme $t = ap(x,R(\mathbf{y},z))$, le sous-terme y n'engendre que la partie *c.b.a.* de la forme normale dans $\mathcal{R}$ de l'instance close suivante :

$$ap(f.g.0,R(\mathbf{a}.\mathbf{b}.\mathbf{c}.\mathbf{0},d.e.0))\downarrow_{\mathcal{R}} = f.g.\mathbf{c}.\mathbf{b}.\mathbf{a}.d.e.0$$

Reconnaître les sous-termes libres d'un terme t et les parties qu'ils engendrent dans $\{t\}\downarrow_{\mathcal{R}}$ nous permet de simplifier une conjecture. En effet, si les deux côtés d'une équation $t = s$ possèdent un sous-terme libre qui engendre une même partie placée au même endroit dans les formes normales des instances closes $\{t\}\downarrow_{\mathcal{R}}$ et $\{s\}\downarrow_{\mathcal{R}}$, on peut, *sans changer la validité de l'équation* :

- « **généraliser** » le sous-terme libre commun par une nouvelle variable (voir section 3.4 et chapitre 5).
- « **partitionner** » les termes pour éliminer les parties communes de la tentative de preuve (voir section 3.5 et chapitre 6).

Il s'agit donc maintenant de pouvoir identifier les sous-termes libres d'un terme. Pour cela, il est nécessaire d'étudier comment sont construites les formes normales des instances closes d'un terme à l'aide d'un système de réécriture. Deux éléments de réponse éclairent cette étude : les systèmes monomorphiques et la définition des fonctions.

3.1.1 Systèmes monomorphiques

Pour que deux parties soient réellement équivalentes pour deux termes, il faut qu'elles soient placées au même endroit dans les formes normales des instances closes des deux termes. Cependant, une instance close en forme normale d'un terme t est un arbre à plusieurs dimensions et il peut exister, dans $\{t\}\downarrow_{\mathcal{R}}$, un nombre indéterminé de chemins différents prolongeant et conduisant à ces parties.

Par contre, une instance close en forme normale d'un terme issu d'un système monomorphique est comparable à une « liste ». Or, dans une liste, deux types de parties

sont aisément identifiables : le *début* et la *fin* – c'est-à-dire les premiers et les derniers éléments de la liste. Ce qui motive notre travail sur les systèmes monomorphiques.

Pour illustrer notre propos, prenons les deux systèmes suivants :

$$\begin{aligned}\mathcal{R} = \{\ & insG(x,a(r,g,d)) \to a(r,insG(x,g),d);\ insG(x,\varnothing) \to x \\ & insD(x,a(r,g,d)) \to a(r,g,insD(x,g));\ insD(x,\varnothing) \to x\ \}\end{aligned} \tag{3.1.1}$$

$$\mathcal{R}' = \{\ ins(e(x),y) \to f(ins(x,y));\ ins(f(x),y) \to e(ins(x,y));\ ins(\varnothing,y) \to y\ \} \tag{3.1.2}$$

Le système $\mathcal{R}'$ est monomorphique – les constructeur e et f n'ayant qu'un argument du même type – mais $\mathcal{R}$ ne l'est pas – le constructeur a représentant un arbre à deux fils.

Dans $\mathcal{R}$, pour les termes $t = insG(x,insD(y,z))$ et $s = insD(y,insG(x,z))$, la partie engendrée par x se trouve bien inséré à gauche dans chaque élément de $\{t\}{\downarrow_{\mathcal{R}}}$ et de $\{s\}{\downarrow_{\mathcal{R}}}$ mais sa position dépend de la forme normale close de z correspondante. Par exemple, pour les instances closes suivantes :

$$\begin{aligned} insG(\mathbf{a}(\mathbf{1},\varnothing,\varnothing),insD(a(2,\varnothing,\varnothing),\varnothing)){\downarrow_{\mathcal{R}}} &= a(2,a(1,\varnothing,\varnothing),\varnothing) \\ insD(a(2,\varnothing,\varnothing),insG(\mathbf{a}(\mathbf{1},\varnothing,\varnothing),\varnothing)){\downarrow_{\mathcal{R}}} &= a(1,\varnothing,a(2,\varnothing,\varnothing)) \end{aligned}$$

Par contre dans $\mathcal{R}'$, pour les termes $t' = ins(z,ins(y,x))$ et $s' = ins(ins(z,y),x)$, la partie engendrée par x se retrouve en queue dans chaque élément de $\{t'\}{\downarrow_{\mathcal{R}'}}$ et de $\{s'\}{\downarrow_{\mathcal{R}'}}$. Et ceci quelque soit les formes normales closes de z et y ; comme nous le prouverons dans le chapitre 4.

3.1.2 Fonctions définies

Un terme quelconque t est composé de fonctions définies, de constructeurs, et de variables alors que les formes normales de ses instances closes $\{t\}{\downarrow_{\mathcal{R}}}$, dans un système $\mathcal{R}$ complètement défini et convergent sur les termes clos, ne sont composées

que de constructeurs. Donc, afin d'appréhender comment les parties issues d'un sous-terme libre sont construites et de les identifier à partir d'un terme quelconque, il faut en premier lieu comprendre la manière dont les fonctions définies manipulent leurs arguments.

Comme nous allons le voir dans la section suivante, certains arguments ne sont pas altérés par la séquence de réécriture, d'autres sont reconstruits à l'identique ou encore utilisés pour créer une série de termes.

3.2 Analyse d'un système de réécriture

Une analyse syntaxique nous permet de comprendre comment une fonction f transforme un argument à une position donnée i. En effet, la manière dont est « copié » l'argument de la partie gauche à la partie droite des règles de réécritures définissant la fonction nous indique quelle est l'influence finale du sous-terme A_i sur la forme normale du terme $f(\overrightarrow{A_{1\ldots n}})$.

Évidemment, chaque fonction manipule différemment ses arguments suivant leurs positions. Nous définirons quatre types de positions d'argument dans lesquelles se trouvent des sous-termes libres : de tête, de queue, contextuelle, et transversale (voir définitions 4.2, 4.3, et 4.4).

Le but final de l'analyse d'un système de réécriture entier est de nous fournir une table dite « *table des arguments libres* » ; cette table nous indique pour chaque fonction les types – si la définition de la fonction s'y prête – de ses positions d'argument. Cette table est pré-compilée et peut être utilisée directement à chaque étape d'une preuve (voir section 4.1).

3.2.1 Arithmétique de Péano

Illustrons maintenant à travers un exemple, la transformation provoquée par un système de réécriture sur différents arguments lors d'une séquence de réécriture amenant à une forme normale. *Cette transformation est induite par la définition des fonctions apparaissant dans le terme ainsi que la position de ces arguments.*

Soit le système de réécriture $\mathcal{R}$ monomorphique – et donc terminant et confluent sur les termes clos – suivant :

$$\begin{array}{l} \{\ (1)\ x + s(y) \to s(x + y);\ (2)\ x + 0 \to x; \\ \quad (3)\ x * s(y) \to (x * y) + x;\ (4)\ x * 0 \to 0\ \} \end{array} \tag{3.2.1}$$

A travers l'étude de $\mathcal{R}$, nous verrons plusieurs types de positions d'arguments qui peuvent contenir des sous-termes libres et que nous appelons « *arguments libres* ».

Position de tête

L'argument de la fonction + à la position 2 est dit « *de tête* ». En effet, dans une séquence de réécriture amenant à la forme normale de $A + B$, le sous-terme clos normalisé B dans cette position va se recomposer, constructeur par constructeur, en tête du terme (règle 1) jusqu'à la constante 0 (règle 2).

Ainsi, $f(\overrightarrow{A_{1\ldots n}})\downarrow_{\mathcal{R}}$, la forme normale close d'un terme dont la racine est la fonction f, va démarrer par une série de constructeurs identique à celle du sous-terme normalisé A_i, si i est la position de tête pour f (voir lemme 4.3). On peut dire que cet argument est libre car il n'engendre que le début de la forme normale.

• ***Exemple* 3.2.1.** $\mathbf{A + s(s(s(0)))}$

$$\to s(A + s(s(0))) \to s(s(A + s(0))) \to s(s(s(A + 0))) \to s(s(s(A)))$$

Position de queue

La position 1 de la fonction + est, quant à elle, dite « *de queue* ». En effet, dans toutes les règles de réécriture – (1) et (2) – définissant +, le sous-terme dans cette position dans la partie droite se retrouve, dans la partie gauche, inchangé et immobile (par rapport au symbole +).

Aucune séquence de réécriture ne saurait donc modifier un sous-terme clos A_i placé dans une position de queue i, et il ne peut se retrouver, à la fin de la séquence de réécriture amenant à $f(\overrightarrow{A_{1\ldots n}})\downarrow_{\mathcal{R}}$, qu'en queue de la forme normale close (voir lemme 4.2 et exemple ci-dessus).

Position contextuelle

La position 2 de la fonction $*$ est dite « *contextuelle* ». Un sous-terme dans cette position d'argument ne forme pas directement une partie de la forme normale, mais il va être utilisé pour placer en tête de la forme normale du terme une série régulière d'éléments. Ces éléments dépendent du contexte dans lequel le sous-terme se trouve.

En effet, dans la partie gauche de la règle (3), le constructeur placé en position 2 n'apparaît pas dans la partie droite. Par contre, le sous-terme $x * y$ qui est égal à la partie gauche « réduite » du constructeur, est placé en position *de queue* dans la partie droite. Donc, le reste de la partie droite – le contexte – se retrouve régulièrement ajouté *au début* des formes normales des instances closes des termes dont la racine est $*$.

Ainsi, $f(\overrightarrow{A_{1\ldots n}})\downarrow_{\mathcal{R}}$, la forme normale d'un terme dont la racine est f va être composée d'une série d'éléments formés par le contexte – A dans le cas de $A * B$ – du sous-terme normalisé en position contextuelle (série d'une longueur équivalente à ce

sous-terme).

• ***Exemple* 3.2.2.** $\mathbf{A} * \mathbf{s}(\mathbf{s}(\mathbf{0}))$

$\rightarrow (A{*}s(s(0)))+A \rightarrow ((A{*}s(0))+A)+A \rightarrow (((A{*}0)+A)+A)+A \rightarrow ((0+A)+A)+A$

Remarque : Un type de position dit « *transversal* » n'apparaît pas dans le système $\mathcal{R}$. Intuitivement, ce type est symétrique au type contextuel dans le sens où la partie gauche « réduite » se retrouve en position de tête dans la partie droite de la règle de réécriture[2].

3.3 Compositions de fonctions

Pour identifier les sous-termes libres d'un terme quelconque, il est nécessaire de comprendre comment chaque fonction manipule ses arguments et comment ces fonctions se combinent entre elles à l'intérieur du terme. Pour illustration, reprenons le système de réécriture $\mathcal{R}$ de la section précédente que nous appliquons à l'équation $t = s$ suivante, considérée comme « difficile » à prouver :

$$x * (x + x) = (x * x) + (x * x)$$

Si l'on applique une substitution close θ quelconque aux deux côtés de l'équation, chaque occurrence de x est substituée par un même terme clos $x\theta$, normalisé de la même façon car $\mathcal{R}$ est convergent sur les termes clos. Pour pouvoir distinguer les différentes occurrences d'un même sous-terme clos normalisé, nous les numérotons ainsi :

$$X1 * (X2 + X3) = (X4 * X5) + (X6 * X7)$$

Dans le sous-terme $X2 + X3$, $X3$ étant en position de tête, sa forme normale va débuter par la partie $X3$. Or, rappelons que la position 2 de la fonction $*$ est

2. Néanmoins, un tel exemple détaillé se trouve page 65

contextuelle, donc la séquence de réécriture conduisant à la forme normale du terme $X1 * (X2 + X3)$ crée une série de $X1$ à partir des constructeurs de la forme normale de $X2 + X3$. La série qui forme le terme commence alors par une sous-série de $X1$ engendrée par la partie engendrée par $X3$ – c'est-à-dire par $X1 * X3$ dans ce cas.

La partie du terme $X1 * (X2 + X3)$ engendrée par $X3$ est donc $X1 * X3$. De même, dans la forme normale de $X2 + X3$, $X2$ se retrouve en fin de sous-terme, et $X1 * X2$ est la partie du terme engendrée par $X2$.

De plus, comme $X5$ et $X7$ sont placés en position contextuelle, ils engendrent les sous-termes $X4 * X5$ et $X6 * X7$ de $(X4 * X5) + (X6 * X7)$. Ces sous-termes étant respectivement placés en position de queue et de tête par rapport à la fonction $+$, ils forment les parties de début et de fin du terme.

Donc, $X2$, $X3$, $X5$, et $X7$ sont aussi des sous-termes libres.

3.3.1 Chemins hauts et bas

Comme nous venons de le voir dans cet exemple, une combinaison de positions d'arguments libres peut elle aussi impliquer la construction d'une partie identifiable de la forme normale. Nous distinguons deux catégories de chemins ; ceux qui conduisent à des sous-termes libres engendrant des parties de début : « *chemins hauts* » et ceux qui conduisent à des sous-termes libres engendrant des parties de fin : « *chemins bas* ».

Intuitivement, pour construire les chemins maximums hauts $CH(t)$ et bas $CB(t)$ d'un terme t, il suffit de descendre récursivement dans l'arbre des symboles décrivant le terme. Pour chaque racine-fonction du terme, on choisit la position impliquant la construction d'une partie correspondante – de début pour haut et de fin pour bas (voir définition 4.6). *Tout sous-terme dans une position préfixe d'un chemin maximum haut, ou bas, est libre.*

Dans notre exemple, $CH(t) = CH(x * (x + x)) = 2CH(x + x) = 22CH(x) = 22$

et $CB(x*(x+x)) = 2CB(x+x) = 21CB(x) = 21$. Donc, $t/2$, $t/22$, et $t/21$ – soit respectivement $x+x$, x, et x – sont des sous-termes libres de t.

De même, $CH(s) = CH((x*x)+(x*x)) = 2CH(x*x) = 22CH(x) = 22$ et $CB((x*x)+(x*x)) = 1CB(x*x) = 12CB(x) = 12$. Donc, $s/2$, $s/22$, $s/1$, et $s/12$ – soit respectivement $x*x$, x, $x*x$, et x – sont des sous-termes libres de s.

3.3.2 Représentation des parties

Les termes représentant les parties engendrées par les sous-termes libres se calculent de manière similaire en parcourant l'arbre du terme. *Dans le cas des positions contextuelles et transversales, la construction de la représentation prend en compte le contexte du sous-terme dans cette position.* Nous définissons deux fonctions : **top**(**t**,**p**) calculant une représentation à partir d'un terme t et d'un chemin haut p et **bot**(**t**,**p**) calculant une représentation à partir d'un terme t et d'un chemin bas p (voir définition 4.8). Chacune de ces représentations se retrouve effectivement au début et à la fin de la forme normale du terme (voir théorème 4.6).

Dans notre exemple, $top(x*(x+x),22) = x*top(x+x,2) = x*top(x,\varepsilon) = x*x$ et $bot(x*(x+x),21) = x*bot(x+x,1) = x*bot(x,\varepsilon) = x*x$.

De même, $top((x*x)+(x*x),22) = top(x*x,2) = x*top(x,\varepsilon) = x*x$ et $bot((x*x)+(x*x),12) = bot(x*x,2) = x*bot(x,\varepsilon) = x*x$.

3.4 Généralisations

Une fois les sous-termes libres et leurs parties associées identifiées, une ou plusieurs généralisations correctes sont possibles. En effet, de par la notion même de sous-terme libre, remplacer ces sous-termes ne modifie dans la forme normale du terme les contenant que la partie qu'ils engendrent.

Prenons une équation dont les deux côtés possèdent un sous-terme libre syntaxiquement identique et qui engendre deux parties dont les représentation sont syntaxiquement égales et placées au même endroit dans la forme normale du terme (début ou fin). Si l'on remplace ces sous-termes par une nouvelle variable, cela ne modifie que les parties engendrées ; comme ces parties sont égales, la validité de l'équation n'en est pas affectée (voir théorème 5.2).

3.4.1 Exemple

Reprenons l'équation ci-dessus,

$$x * (x + x) = (x * x) + (x * x)$$

Si l'on prend les chemins hauts 22 à gauche et 22 à droite, on obtient les conditions nécessaires pour une généralisation. En effet, le sous-terme libre est $x = t/22 = s/22$ et les parties engendrées sont $top(t,22) = x * x$ à gauche et $top(s,22) = x * x$ à droite. Le sous-terme libre peut donc être substitué par une nouvelle variable y.

De même, si l'on prend les chemins bas 21 à gauche et 12 à droite ; le sous-terme libre est x et les parties engendrées sont $bot(t,21) = x * x$ et $bot(s,12) = x * x$. Ce sous-terme peut lui aussi être substitué par une nouvelle variable z.

La généralisation ainsi obtenue est donc :

$$x * (\mathbf{z} + \mathbf{y}) = (x * \mathbf{z}) + (x * \mathbf{y})$$

En utilisant la règle GEN de génération sur y (voir section 2.1.1), l'équation devient [3]

$$((x * z) + (x * y')) + x = (x * z) + ((x * y') + x)$$

3. Les cas de base $x * (z + 0) = (x * z) + (x * 0)$ et $(w + (x * y')) + 0 = w + ((x * y') + 0)$ étant triviaux, nous ne les traiterons pas ici.

et peut être généralisée suivant les chemins haut 2 à gauche et 22 à droite, ainsi que les chemins bas 11 à gauche est 1 à droite en :

$$(\mathbf{w} + (x * y')) + \mathbf{v} = \mathbf{w} + ((x * y') + \mathbf{v})$$

Cette nouvelle équation est « facile » à prouver car elle ne nécessite pas d'autre généralisation (ni lemme). Si on applique une deuxième fois la règle GEN sur v, cela donne[3] :

$$s(w + ((x * y') + v')) = s(w + ((x * y') + v'))$$

Les deux côtés de cette dernière équation étant syntaxiquement égaux, la conjecture initiale est bien un théorème inductif.

3.4.2 Algorithme

L'algorithme calculant des généralisations correctes d'une équation $t = s$ se déroule en trois étapes, une fois construite la table des arguments libres (voir section 4.1) :

Calculer les chemins maximums hauts et bas – CH et CB – des deux côtés de l'équation à l'aide de la table des arguments libres.

Rechercher les plus grands préfixes des deux chemins **hauts** maximums dans lesquels est placé un sous-terme identique. Si les représentations *top* des parties engendrées par ces sous-termes sont égales, alors la substitution de ces sous-termes à gauche et à droite de l'équation par une variable n'apparaissant ni dans t ni dans s est une généralisation correcte de l'équation.

Rechercher les plus grands préfixes des deux chemins **bas** maximums dans lesquels est placé un sous-terme identique. Si les représentations *bot* des parties engendrées par ces sous-termes sont égales, alors la substitution de ces sous-termes à gauche et à droite de l'équation par une variable n'apparaissant ni dans t ni dans s est une généralisation correcte de l'équation.

3.5 Partitions

Une fois identifiées deux parties de fin (ou de début) égales pour les deux côtés d'une conjecture, une possibilité autre que la généralisation nous est offerte : « la partition de termes ». En effet, comme ces deux parties sont inductivement égales nous pouvons *les éliminer des deux côtés sans que la validité de l'équation n'en soit affectée*.

De manière plus générale, la partition d'un terme est un couple de termes dont la « jonction » des formes normales est égale à la forme normale correspondante du terme (voir définition 6.1). La méthode de preuve est basée sur le fait que si (a, b) est une partition d'un terme t et (c, d) une partition d'un terme s alors a = c et b = d – syntaxiquement ou inductivement – implique que t = s soit un théorème inductif (voir théorème 6.1).

Pour obtenir un couple bien-fondé décrivant le langage des formes normales des instances closes d'un terme il faut pouvoir décrire une partie de début ou de fin de terme (comme ci-dessus) mais aussi son complément dans la forme normale du terme. Or, de manière similaire au calcul des représentations $top()$ et $bot()$ nous pouvons calculer leurs compléments respectifs $ntp()$ et $nbt()$ (voir définition 4.9). Les partitions utilisées dans notre méthode peuvent donc être indifféremment, pour un terme t donné, le couple formé des parties $top(t,p)$ et $ntp(t,p)$ pour un chemin haut p ou le couple formé des parties $nbt(t,q)$ et $bot(t,q)$ pour un chemin bas q.

3.5.1 Exemples

Illustrons notre méthode à travers deux exemples : celui déjà présent dans la section précédente ainsi qu'un autre faisant intervenir une nouvelle notion – celle de « *motif* ».

• **Première équation :**

$$x * (x + x) = (x * x) + (x * x) \quad (e1)$$

Les représentations complémentaires $ntp()$ et $nbt()$ sont calculées à partir du même chemin mais en faisant en sorte de garder ce qui a été écarté – et inversement – lors du calcul de $top()$ et $bot()$ (voir ci-dessus).

Nous avons une partition suivant le chemin haut 22 à gauche :

- $top(x * (x + x),22) = x * x$
- $ntp(x * (x + x),22) = x * ntp(x + x,2) = x * (x + ntp(x,\varepsilon)) = x * (x + 0)$

Nous avons aussi une partition suivant le chemin haut 2 à droite :

- $top((x * x) + (x * x),2) = x * x$
- $ntp((x * x) + (x * x),2) = (x * x) + ntp(x * x,\varepsilon) = (x * x) + 0$

Les deux débuts étant égaux, l'équation $e1$ est un théorème inductif si et seulement si les deux compléments sont égaux inductivement (voir théorème 6.1). On obtient ainsi l'équation :

$$x * (x + 0) = (x * x) + 0 \quad (e1')$$

Or, les deux côtés de $e1'$ se réécrivent pareillement en $x * x$. La preuve de $e1$ est donc directe et ne nécessite aucun schéma d'induction.

• **Deuxième équation :**

$$r(ap(l,r(l)) = ap(l,r(l)) \quad (e2)$$

Soit le système de réécriture $\mathcal{R}'$ monomorphique, terminant et confluent sur les termes clos suivant :

$$\{ \ (1)\ ap(c.l,L) \to c.ap(l,L);\ (2)\ ap(\varnothing,L) \to L;$$
$$(3)\ r(c.l) \to ap(r(l),c.\varnothing);\ (4)\ r(\varnothing) \to \varnothing \ \}$$

Dans le système $\mathcal{R}'$, la position 1 de la fonction $ap()$ est de tête, car la liste dans cette position va se recomposer – élément par élément – en tête de la forme normale et la position 2 est de queue car le sous-terme L dans cette position reste inchangé et immobile (par rapport à ap). Donc une partition de la partie droite est :

- $top(ap(l,r(l)),1) = top(l,\varepsilon) = l$
- $ntp(ap(l,r(l)),1) = ap(ntp(l,\varepsilon),r(l)) = ap(\varnothing,r(l))$

La position 1 de la fonction $r()$ est transversale. En effet, dans la partie droite de la règle (3) un élément $(c.\varnothing)$ est rajouté en position de queue. Donc, si un terme a pour racine r, sa forme normale est composée d'une liste construite à partir du sous-terme en position transversale mais en commençant par la fin du sous-terme (voir lemme 4.5). Le fait que l'argument unique de $r()$ soit transversal implique que la forme normale de $r(ap(l,r(l))$ finisse par la fonction $r()$ appliquée **au début** de la forme normale de $ap(l,r(l))$. Donc, nous avons la partition :

- $bot(r(ap(l,r(l))),11) = r(top(ap(l,r(l)),1)) = r(top(l,\varepsilon)) = r(l)$
- $nbt(r(ap(l,r(l))),11) = r(ntp(ap(l,r(l)),1)) = r(ap(ntp(l,\varepsilon),r(l))) = r(ap(\varnothing,r(l)))$

Les partitions suivant le chemin haut 1 à droite de $e2$ et le chemin bas 11 à gauche sont utilisables pour simplifier la conjecture. En effet, le complément à la partie haute $(ap(\varnothing,r(l)))$ et la partie basse ont la même forme normale : $r(l)$. Donc $e2$ est un théorème inductif si et seulement si la partie haute et le complément de la partie basse sont égaux inductivement ; ce qui donne l'équation :

$$r(ap(\varnothing,r(l))) = l$$

Ce qui se réécrit suivant $\mathcal{R}'$ en l'équation suivante :

$$r(r(l)) = l \quad (e3)$$

Comme les deux côtés de cette équation ne possèdent pas de sous-termes indépendants, elles ne peuvent donner de partition à l'aide des définitions vues précédemment. C'est pourquoi nous introduisons une nouvelle notion, celle de « motif ».

3.5.2 Motifs

Les formes normales closes des systèmes monomorphiques sont des « listes » et peuvent donc être composées d'une série régulière d'éléments « similaires ». Un motif est une représentation de ces éléments quand ils existent. Pour calculer le motif d'un terme nous prenons un sous-terme placé sur un chemin d'induction que l'on remplace par un élément non-constant de l'ensemble test du système[4] (voir définition 6.4).

Dans l'équation $e3$, si l'on remplace le sous-terme l commun aux deux côtés par l'élément non-constant de $\mathcal{R}'$ ($c.l'$) on obtient les termes $r(ap(r(l'),c.\varnothing))$ – forme normale de $r(r(c.l'))$ – pour $r(r(l))$ et $c.l'$ pour l.

Le motif haut – $\mathbf{motH}()$ – de chacun des termes est la partie qui se rajoute au début de la forme normale par rapport au terme original[5] (celui de l'équation $e3$). *Donc le motif et le terme original forment une partition du terme substitué.*

Or, pour le chemin bas 11 du côté gauche de l'équation $e3$, nous avons la partition :

- $bot(r(ap(r(l),c.\varnothing)),11) = r(top(ap(r(l),c.\varnothing),1)) = r(top(r(l),\varepsilon)) = r(r(l))$
- $nbt(r(ap(r(l),c.\varnothing),11) = r(ntp(ap(r(l),c.\varnothing),1)) = r(ap(ntp(r(l),\varepsilon),c.\varnothing)) = r(ap(\varnothing,c.\varnothing))$

Donc $r(ap(\varnothing,c.\varnothing))$ – dont la forme normale est $c.\varnothing$ – est le motif haut du terme $r(r(l))$.

Du côté droit, $bot(c.l,2) = l$ et $nbt(c.l,2) = c.\varnothing$ est le motif haut du terme l.

Les deux motifs étant égaux, les formes normales sont composées chacune d'une série de termes égaux. Les deux termes sont donc eux aussi égaux si et seulement

4. Rappelons que l'ensemble test décrit le modèle initial de l'ensemble d'axiome représenté par le système de réécriture et donc les formes normales closes générées par le système.

5. Nous définissons aussi un motif bas – $\mathbf{motB}()$ – qui se rajoute en fin de la forme normale.

si les cas de bases sont égaux inductivement (voir théorème 6.1). Les cas de bases représentent la partie du terme qui finit la série définie par le motif et sont formés par la substitution de la variable utilisée pour calculer le motif dans l'équation originale par le terme constant de l'ensemble test. Soit dans notre exemple, la substitution de l par la constante $\varnothing$ dans l'équation $e3$:

$$r(r(\varnothing)) = \varnothing$$

Équation qui est évidemment un théorème inductif pour $\mathcal{R}'$; donc $e3$ et $e2$ sont aussi des théorèmes inductifs.

3.5.3 Procédure de preuve

Récapitulons les étapes nécessaires à la preuve d'une conjecture donnée $u = v$ en utilisant la partition de termes :

Si les formes normales de u et v sont égales syntaxiquement alors $u = v$ est un théorème inductif.

Calculer toutes les partitions possibles de u et v en utilisant la définition 6.2.

S'il existe deux couples (a,b) et (c,d) partitions de u et v tels que $a \equiv c$ (respectivement $b \equiv d$) alors appliquer la procédure pour $b = d$ (respectivement $a = c$).

Sinon trouver un chemin d'induction contenant un sous-terme commun dans u et v et calculer les motifs associés en utilisant la définition 6.4.

Si les motifs des deux côtés de l'équation sont égaux syntaxiquement alors appliquer la procédure pour les remplacements dans u et v du sous-terme commun par la constante de l'ensemble test.

Sinon si les remplacements dans u et v du sous-terme commun par la constante de l'ensemble test sont égaux syntaxiquement alors appliquer la procédure pour les motifs respectifs de u et v.

Deuxième partie

Contributions Principales

Chapitre 4

Théorème de découpage

Dans ce chapitre, nous établirons une preuve constructive du théorème de découpage qui est au centre de nos techniques – aussi bien la généralisation que la partition de termes. Pour cela, nous allons définir formellement les notions et les fonctions introduites dans le chapitre précédent. Ensuite, nous combinerons ces définitions et les lemmes qu'elles impliquent pour montrer comment, et sous quelles conditions, les formes normales des instances closes d'un terme peuvent se découper en deux parties bien définies.

4.1 Table des arguments libres

Rappelons en premier lieu que notre réflexion porte sur le devenir des formes normales des instances closes des termes. Étudier ce devenir nous permet de tirer des conclusions sur les termes eux-mêmes dans le cadre du raisonnement inductif.

Rappelons aussi que le type de position d'un argument nous indique comment cet argument agit sur la forme normale close du terme.

Cette section introduit donc les définitions formelles de chaque type de position d'argument (voir section 3.2.1), définitions qui peuvent être vérifiées syntaxiquement à partir de l'analyse d'un système de réécriture donné. Ensuite, nous donnerons un

exemple de table associant chaque symbole de fonction d'un système de réécriture aux possibles types de ces positions d'argument. Enfin, nous montrerons formellement l'effet de chaque type de positions sur la forme normale close d'un terme.

4.1.1 Arguments libres dans la définition de fonctions

Dans les parties gauches des règles $\mathcal{R}_f$ définissant une fonction f, on peut trouver dans une position d'argument i, un terme quelconque x ou un terme constructeur $c[x]$[1]. Dans ce cas, nous disons que le terme x « représente » les arguments qui sont placés à la position i par rapport à f dans tous les termes appartenant à $T(\mathcal{F}_R,\mathcal{X})$.

• ***Exemple* 4.1.1.**

$$\mathcal{R}_+ = \{\ (1)\ z + 0 \to z;\ (2)\ x + s(y) \to s(x + y)\}$$

- Dans la règle (1), z représente les arguments de $+$ en position 1.
- Dans la règle (1), 0 représente les arguments de $+$ en position 2.
- Dans la règle (2), x représente les arguments de $+$ en position 1.
- Dans la règle (2), y, l'argument réflexif du terme constructeur $s(y)$, représente les arguments de $+$ en position 2.

La définition ci-dessous est essentielle pour notre approche car elle introduit formellement la notion d'argument libre. *En effet, dans un terme, un argument de fonction est libre s'il n'est pas « dupliqué » par cette fonction.* Il est en effet nécessaire que dans les règles définissant la fonction, le sous-terme représentant l'argument ne se retrouve qu'une seule fois dans chaque partie gauche.

Définition 4.1. Dans un système de réécriture $\mathcal{R}$, pour une fonction $f \in \mathcal{D}_R$, une position $p \in P_f$ est une *position d'argument libre* (respectivement une *position récursive d'argument libre*) pour f, si pour toute règle $l \to r \in \mathcal{R}_f$, il existe **au plus** une

1. c'est-à-dire $c[x]_p$ avec $c(\varepsilon) \in \mathcal{C}_R$ et $p = RA(\mathcal{R},c(\varepsilon))$

position q telle que :

$$l/p = r/q \text{ (respectivement } l/p = c[x] \text{ et } r/q = x)$$

• ***Exemple* 4.1.2.**

$$\mathcal{R}_{\Sigma I} = \{\ (1)\ \Sigma I(0,y) \to y;\ (2)\ \Sigma I(s(x),y) \to \Sigma I(x,s(y+x))\ \}$$

La fonction $\Sigma I(n,0)$ calcule la somme des entiers de 1 à n dans un « style itératif ».

- La position 1 pour ΣI n'est pas une position d'argument libre car x se retrouve deux fois à droite de la règle (2).
- La position 2 pour ΣI est une position récursive d'argument libre car y ne se retrouve qu'une seule fois à droite de la règle (2) ainsi qu'à droite de la règle (1).

Remarque : Pour la fonction +, la position 1 est une position d'argument libre et la position 2 est une position récursive d'argument libre.

Comme un argument libre n'est pas dupliqué, il n'influence qu'une partie des formes normales des instances closes du terme. On peut donc remplacer un argument en position libre par un autre sans que la « structure » du terme ne change lors d'une séquence de réécriture comme le montre la proposition ci-dessous.

Proposition 4.1. *Dans un système de réécriture $\mathcal{R}$ convergent sur les termes clos et linéaire à gauche, pour une fonction f, alors pour tout terme y, pour toute règle $l \to r \in \mathcal{R}_f$ et pour toute substitution close θ ;*

- *$fn(l[y]_p\theta) = fn(r[y]_q\theta)$ si $l/p = r/q$ et si p est une position d'argument libre de f,*
- *$fn(l[c[y]]_p\theta) = fn(r[y]_q\theta)$ si $l/p = c[x]$ et $r/q = x$ et si p est une position récursive d'argument libre de f.*

Démonstration. Posons $t = r/q = l/p$ si p est une position d'argument libre de f ou bien $t = r/q = l/pp'$ avec $p' = RA(\mathcal{R},l(p))$ si p est une position récursive d'argument libre de f.

Comme $\mathcal{R}$ est convergent sur les termes clos, $fn(l\theta) = fn(r\theta)$. Soit la substitution σ qui remplace les occurrences de t par y, de par les propriétés de la substitution, $fn(l\sigma\theta) = fn(r\sigma\theta)$.

Or $fn(l[y]_p\theta) = fn(l\sigma\theta)$ et $fn(r[y]_q\theta) = fn(r\sigma\theta)$ si et seulement si t est placée à une seule position p dans l et à une seule position q dans r. Comme $\mathcal{R}$ est linéaire à gauche et comme il n'existe pas de position $q' \neq q$ telle que $r(q) = r(q')$, nous avons bien $fn(l[x]_p\theta) = fn(r[x]_q\theta)$. □

• ***Exemple* 4.1.3.**

$$\mathcal{R}_* = \{\ (1)\ x * 0 \to 0;\ (2)\ x * s(y) \to (x * y) + x\ \}$$

Soit l'instance close $s(0) * s(0)$ qui se réécrit en $(s(0) * 0) + s(0)$.

– La position 2 est une position récursive d'argument libre de $*$, et donc si l'on ne remplace **que** l'argument en position 2, nous obtenons :

$$fn(s(0) * \mathbf{s(s(0))}) = fn((s(0) * \mathbf{s(0)}) + s(0))$$

– La position 1 n'est pas une position d'argument libre de $*$, et donc si l'on ne remplace **que** l'argument en position 1, nous obtenons :

$$fn(\mathbf{0} * s(0)) \neq fn((\mathbf{0} * 0) + \mathbf{s(0)})$$

4.1.2 Manipulation des arguments par une fonction définie

Rappelons tout d'abord que dans un système monomorphique $\mathcal{R}$, la position de l'argument réflexif d'un constructeur $f \in \mathcal{C}_R$ s'écrit $RA(\mathcal{R},f)$ (voir définition 2.14).

Les autres types de positions d'arguments libres se différencient par la manière dont est recopié l'argument de la partie droite à la partie gauche des règles de l'ensemble définition. Nous distinguons deux catégories principales d'arguments : ceux que la fonction ne modifie pas dans la partie gauche et ceux sur lesquels est définie

récursivement la fonction. Pour chaque type de position, nous apporterons un exemple de telle définition.

Remarque : Comme toutes ces définitions impliquent des arguments libres, elles suivent les conditions de la définition 4.1.

Arguments persistants

Pour qu'un argument soit réellement persistant tout au long d'une séquence de réécriture, il faut que dans toutes les parties droites de l'ensemble définition, il se retrouve une place où il ne pourra être modifié par aucune fonction. Un argument en position de queue se retrouve donc dans la partie droite soit à la racine soit dans un chemin qui est une suite de positions *de queue, réflexive, ou identique à celle qu'il occupait initialement par rapport à la fonction.*

Définition 4.2. Dans un système de réécriture $\mathcal{R}$, pour une fonction $f \in \mathcal{D}_R$, une position d'argument libre $p \in P_f$ est *de queue*, et s'écrit $DP(\mathcal{R},f)$, si pour tout $l \to r \in \mathcal{R}_f$ il existe une seule position q telle que $l/p = r/q$ et

- Soit $q = \varepsilon$
- Soit $q = q_1 \dots q_n$ avec pour tout $i \leq n$, $|q_i| = 1$ et
  - soit $q_i = RA(\mathcal{R},r(q_1 \dots q_{i-1}))$,
  - soit $q_i = DP(\mathcal{R},r(q_1 \dots q_{i-1}))$,
  - soit $q_i = p$ avec $r(q_1 \dots q_{i-1}) = f$.

• ***Exemple* 4.1.4.**

$$\mathcal{R}_{\Sigma I} = \{\ (1)\ \Sigma I(0,y) \to y;\ (2)\ \Sigma I(s(x),y) \to \Sigma I(x,s(y+x))\ \}$$

Prenons la position d'argument 2 pour ΣI.

- Dans la règle (1), $l/2 = y$ et $r/\varepsilon = y$.

- Dans la règle (2), $l/2 = y$ et $r/211 = y$, nous avons donc $q = 211$ et
  - $q_1 = 2$ avec[2] $r(\varepsilon) = \Sigma I$
  - $q_2 = 1 = RA(\mathcal{R},s) = RA(\mathcal{R},r(2))$
  - $q_3 = 1 = DP(\mathcal{R},+) = DP(\mathcal{R},r(21))$

Donc $DP(\mathcal{R},\Sigma I) = 2$.

Arguments de récurrence

Les positions d'arguments de récurrence sont des éléments de l'ensemble des positions d'induction définies dans la section 2.1.2. Or, pour le cas d'un système monomorphique $\mathcal{R}$, dans les parties gauches d'un ensemble définition, on trouve à la position d'un argument utilisé pour une récurrence (voir section 2.2),

- soit une constante $\perp_T$,
- soit un terme constructeur $c[x]$ – c-à-d $c[x]_p$ avec $c(\varepsilon) \in \mathcal{C}_R$ et $p = RA(\mathcal{R},c(\varepsilon))$.

Le type d'une position d'argument de récurrence dépend de la position dans laquelle se retrouve la fonction dans les parties droites. Le type de position dit « de tête » implique que le constructeur, à cette position dans la partie droite des règles de réécriture, se retrouve à la racine de la partie gauche correspondante.

Définition 4.3. Dans un système de réécriture monomorphique $\mathcal{R}$, pour une fonction $f \in \mathcal{D}_R$, une position récursive d'argument libre $p \in PI_{\mathcal{R}}(f)$ est *de tête*, et s'écrit $TP(\mathcal{R},f)$, si $\forall l \to r \in \mathcal{R}_f$

- Soit $l/p = \perp_{type(l)}$
- Soit $l/p = c[x]$ et $r = c[l[x]_p]$

• ***Exemple* 4.1.5.**

$$\mathcal{R}_+ = \{\ (1)\ x + 0 \to x;\ (2)\ x + s(y) \to s(x + y);\ (3)\ x + p(y) \to p(x + y)\ \}$$

2. Rappelons que pour $i = 1$ nous avons $q_1 \ldots q_{i-1} = \varepsilon$.

Prenons la position d'argument 2 pour $+$.

- Dans la règle (1), $l/2 = 0$.
- Dans la règle (2), $l/2 = s(y)$ et $r = s(x+y) = s(l[y]_2)$.
- Dans la règle (3), $l/2 = p(y)$ et $r = p(x+y) = p(l[y]_2)$.

Donc, $TP(\mathcal{R},+) = 2$.

Le type de position contextuelle – respectivement transversale – implique que la partie droite de la règle de réécriture se retrouve, réduite du constructeur, dans la partie gauche correspondante, en position de queue – respectivement de tête.

Définition 4.4. Dans un système de réécriture monomorphique $\mathcal{R}$, pour une fonction $f \in \mathcal{D}_R$, une position récursive d'argument libre $p \in PI_{\mathcal{R}}(f)$ telle que $\forall l \to r \in \mathcal{R}_f$

- Soit $l/p = \perp_{type(l)}$ et $r = \perp_{type(l)}$
- Soit $l/p = c[x]$ et il existe q tel que $r/q = l[x]_p$. Alors p est une position :
  - *contextuelle* et notée $CP(\mathcal{R},f)$ si $q = DP(\mathcal{R},r(\varepsilon))$
  - *transversale* et notée $SP(\mathcal{R},f)$ si $q = TP(\mathcal{R},r(\varepsilon))$

• ***Exemple* 4.1.6.**

$$\mathcal{R}_{mf} = \{\ (1)\ mf(x,0) \to 0;\ (2)\ mf(x,s(y)) \to \Sigma I(x,mf(x,y))\ \}$$

Prenons la position d'argument 2 pour mf.

- Dans la règle (1), $l/2 = 0$ et $r = 0$
- Dans la règle (2), $l/2 = s(y)$ et $r/2 = mf(x,y) = l[y]_2$.

Comme $2 = DP(\mathcal{R},\Sigma I)$, nous avons $CP(\mathcal{R},mf) = 2$

• ***Exemple* 4.1.7.**

$$\mathcal{R}_D = \{\ (1)\ D(0) \to 0;\ (2)\ D(s(x)) \to s(s(0)) + D(x)\ \}$$

Prenons la position d'argument 1 pour D.

- Dans la règle (1), $l/1 = 0$ et $r = 0$

– Dans la règle (2), $l/1 = s(x)$ et $r/2 = D(x) = l[x]_1$.

Comme $2 = TP(\mathcal{R},+)$, nous avons $SP(\mathcal{R},D) = 1$

Remarque : Des définitions de fonctions comme

$$\begin{aligned}
\mathcal{R}_{dbl} &= \{\ dbl(0) \to 0;\ dbl(s(x)) \to s(s(dbl(x)))\ \} \\
\mathcal{R}_{exp} &= \{\ exp(x,s(y)) \to exp(x,y) * x;\ exp(x,0) \to s(0)\ \} \\
\mathcal{R}_{demi} &= \{\ demi(0) \to 0;\ demi(s(0)) \to 0;\ demi(s(s(x))) \to s(demi(x))\ \}
\end{aligned} \tag{4.1.1}$$

impliquent à chaque fois un argument libre (par exemple, le premier argument de *dbl* est en essence contextuel). Mais pour plus de lisibilité, nous laissons le soin à de futures recherches de définir formellement ces types d'arguments libres ainsi que leurs effets sur les formes normales.

4.1.3 Définition d'une table

Définition 4.5. La *table des argument libres* d'un système de réécriture $\mathcal{R}$, notée $TAL(\mathcal{R})$, est un tableau à deux dimensions qui associe à chaque symbole de fonction f de $\mathcal{R}$, et à chaque type de position d'argument, l'indice de la position d'argument correspondante (s'il existe).

Par exemple, pour le système de réécriture $\mathcal{R}$ suivant :

$$\mathcal{R} = \left\{ \begin{array}{rlrl}
x + s(y) & \to s(x + y) & x + 0 & \to x \\
x * s(y) & \to (x * y) + x & x * 0 & \to 0 \\
exp(x,s(y)) & \to exp(x,y) * x & exp(x,0) & \to s(0) \\
m(x,s(y)) & \to x + m(x,y) & m(x,0) & \to 0 \\
\Sigma(s(x)) & \to s(x) + \Sigma(x) & \Sigma(0) & \to 0 \\
\Sigma I(s(x),y) & \to \Sigma I(x,s(y + x)) & \Sigma I(0,y) & \to y \\
r(c.l) & \to ap(r(l),c.\varnothing) & r(\varnothing) & \to \varnothing \\
ap(c.l,L) & \to c.ap(l,L) & ap(\varnothing,L) & \to L \\
R(c.l,L) & \to R(l,c.L) & R(\varnothing,L) & \to L
\end{array} \right. \tag{4.1.2}$$

La table $TAL(\mathcal{R})$, ainsi que les positions d'inductions (voir section 2.1.2), correspondantes sont :

f	$RA(\mathcal{R},f)$	$DP(\mathcal{R},f)$	$TP(\mathcal{R},f)$	$CP(\mathcal{R},f)$	$SP(\mathcal{R},f)$	$PI_{\mathcal{R}}(f)$
s	1					
$+$		1	2			$\{2\}$
$*$				2		$\{2\}$
e						$\{2\}$
m					2	$\{2\}$
Σ						$\{1\}$
ΣI		2				$\{1\}$
$.$	2					
ap		2	1			$\{1\}$
r					1	$\{1\}$
R		2				$\{1\}$

4.1.4 Effet des sous-termes libres sur la forme normale d'un terme

Remarque : Dorénavant, pour plus de lisibilité dans les preuves et comme nous ne prenons en compte qu'un seul système monomorphique $\mathcal{R}$ et un seul type T à la fois, nous noterons la constante $\perp_T$ de ce type T simplement $\perp$.

Les lemmes ci-dessous décrivent formellement le devenir dans la forme normale d'un terme, d'un argument clos placé dans un type de position précis par rapport à la racine du terme. Nous voyons ainsi que dans un système monomorphique :

• **La forme normale close d'un terme $f(\overrightarrow{A_{1\ldots n}})$, où le sous-terme clos normalisé A_p est en position de queue $p \leq n$ pour f, se finit par A_p.**

Lemme 4.2. *Soit $\mathcal{R}$ un système de réécriture monomorphique, pour un terme t, une position $p = DP(\mathcal{R},t(\varepsilon))$, une substitution close θ, et un terme A clos en forme*

normale pour $\mathcal{R}$;

$$fn(t[A]_p\theta) = fn(t[\perp_{type(t)}]_p\theta) \otimes A$$

Démonstration. Soit la séquence de réécriture $t[A]_p\theta \to t_1 \to t_2 \to \ldots \to t_n$ amenant à $fn(t[A]_p\theta)$. De par la définition 4.2 de DP, pour chaque étape $i \leq n$ de réécriture il existe une position q_i, une suite de positions RA ou DP, tel que $t_i/q_i = A$.

Soit $T = t_n$; comme T est un terme en forme normale close, il n'est composé que de constructeurs et $q = q_n$ est alors une suite de positions RA. Donc, d'après la définition 2.15 de $\otimes$,

$$fn(t[A]_p\theta) = T[A]_q = T[\perp]_q \otimes A$$

Or, d'après la proposition 4.1,

$$fn(t[\perp]_p\theta) = fn(t_1[\perp]_{q_1}) = \ldots = fn(t_n[\perp]_{q_n}) = T[\perp]_q$$

Donc, nous avons bien $fn(t[A]_p\theta) = fn(t[\perp]_p\theta) \otimes A$. □

• ***Exemple* 4.1.8.** Rappelons que $DP(\mathcal{R},\Sigma I) = 2$.

$$\Sigma I(s(s(0)),\mathbf{s}(\mathbf{s}(\mathbf{s}(\mathbf{0})))) \to \Sigma I(s(0),s(\mathbf{s}(\mathbf{s}(\mathbf{s}(\mathbf{0}))) + s(0))) \to \Sigma I(s(0),s(s(\mathbf{s}(\mathbf{s}(\mathbf{s}(\mathbf{0})))))) \to \Sigma I(0,s(s(s(\mathbf{s}(\mathbf{s}(\mathbf{s}(\mathbf{0})))))) + 0)) \to s(s(s(\mathbf{s}(\mathbf{s}(\mathbf{s}(\mathbf{0})))))))$$

• La forme normale close d'un terme $f(\overrightarrow{A_{1\ldots n}})$, où le sous-terme clos normalisé A_p est en position de tête $p \leq n$ pour f, se commence par A_p.

Lemme 4.3. *Soit $\mathcal{R}$ un système de réécriture monomorphique, pour un terme t, une position $p = TP(\mathcal{R},t(\varepsilon))$, une substitution close θ, et un terme A clos en forme normale pour $\mathcal{R}$;*

$$fn(t[A]_p\theta) = A \otimes fn(t[\perp_{type(t)}]_p\theta)$$

Démonstration. Par récurrence sur la structure du terme clos A ; comme A est un terme clos d'un système monomorphique, nous avons deux cas (voir section 2.2) :

– Soit $A = \perp$, et de par la définition de $\otimes$, $fn(t[\perp]_p\theta) = \perp \otimes fn(t[\perp]_p\theta)$.

– Soit $A = c[A']$, et de par la définition 4.4 de TP,

$$\begin{aligned} fn(t[c[A']]_p\theta) &= fn(c[t[A']]_p\theta) \\ &= c[fn(t[A']\theta)] \text{ (comme } c(\varepsilon) \text{ est un constructeur libre)} \\ &= c[A' \otimes fn(t[\bot]_p\theta)] \text{ (par hypothèse de récurrence)} \\ &= c[A'] \otimes fn(t[\bot]_p\theta) \text{ (d'après le lemme 2.3)} \end{aligned}$$

Dans les deux cas, nous avons bien $fn(t[A]_p\theta) = A \otimes fn(t[\bot]_p\theta)$. □

• ***Exemple* 4.1.9.** Rappelons que $TP(\mathcal{R},ap) = 1$.

$$\begin{aligned} ap(\mathbf{a}.\mathbf{b}.\mathbf{c}.\varnothing,d.\varnothing) &\to \mathbf{a}.ap(\mathbf{b}.\mathbf{c}.\varnothing,d.\varnothing) \to \mathbf{a}.\mathbf{b}.ap(\mathbf{c}.\varnothing,d.\varnothing) \\ &\to \mathbf{a}.\mathbf{b}.\mathbf{c}.ap(\varnothing,d.\varnothing) \to \mathbf{a}.\mathbf{b}.\mathbf{c}.d.\varnothing \end{aligned}$$

• La construction de la forme normale du terme $f(\overrightarrow{A_{1\ldots n}})$, où le sous-terme clos normalisé A_p est en position de contextuelle $p \leq n$ pour f, utilise le début de A_p pour engendrer de manière régulière le début de la forme normale close du terme.

Lemme 4.4. *Soit $\mathcal{R}$ un système de réécriture monomorphique, pour un terme t, une position $p = CP(\mathcal{R},t(\varepsilon))$, une substitution close θ, et deux termes A et B clos en forme normale pour $\mathcal{R}$;*

$$fn(t[A \otimes B]_p\theta) = fn(t[A]_p\theta) \otimes fn(t[B]_p\theta)$$

Démonstration. Par récurrence sur la structure du terme clos A :

– Soit $A = \bot$, et de par la définition de $\otimes$,

$$fn(t[\bot \otimes B]_p\theta) = fn(t[B]_p\theta) = \bot \otimes fn(t[B]_p\theta)$$

Or, de par la définition 4.4 de CP, $\bot$ est la forme normale de $t[\bot]_p$.

– Soit $A = c[A']$, et $fn(t[c[A'] \otimes B]_p\theta)$

$= fn(t[c[A' \otimes B]]_p\theta)$ (d'après le lemme 2.3)

$= fn(r[t[A' \otimes B]_p]_q\theta)$ avec $q = DP(\mathcal{R},r(\varepsilon))$ (de par la définition de CP)

$= fn(r[\bot]_q\theta) \otimes fn(t[A' \otimes B]_p\theta)$ (d'après le lemme 4.2)

$= fn(r[\bot]_q\theta) \otimes fn(t[A']_p\theta) \otimes fn(t[B]_p\theta)$ (par hypothèse de récurrence)

$= fn(r[t[A']_p]_q\theta) \otimes fn(t[B]_p\theta)$ (d'après le lemme 4.2)

$= fn(t[c[A']]_p\theta) \otimes fn(t[B]_p\theta)$ (d'après la proposition 4.1 et la définition 4.4)

Dans les deux cas, nous avons bien $fn(t[A \otimes B]_p\theta) = fn(t[A]_p\theta) \otimes fn(t[B]_p\theta)$. □

• ***Exemple* 4.1.10.** Rappelons que $CP(\mathcal{R},*) = 2$ et $DP(\mathcal{R},+) = 1$

$$fn(A * \mathbf{s}(\mathbf{s}(\mathbf{s}(B)))) = fn((A * s(s(B))) + A) = fn(((A * s(B)) + A) + A)$$
$$= fn((((A * B) + A) + A) + A) = fn(A * \mathbf{s}(\mathbf{s}(\mathbf{s}(\mathbf{0})))) \otimes fn(A * B)$$

• La construction de la forme normale du terme $f(\overrightarrow{A_{1\ldots n}})$, où le sous-terme clos normalisé A_p est en position de transversale $p \leq n$ pour f, utilise le début de A_p pour engendrer de manière régulière la fin de la forme normale close du terme.

Lemme 4.5. *Soit $\mathcal{R}$ un système de réécriture monomorphique, pour un terme t, une position $p = SP(\mathcal{R},t(\varepsilon))$, une substitution close θ, et deux termes A et B clos en forme normale pour $\mathcal{R}$;*

$$fn(t[A \otimes B]_p\theta) = fn(t[B]_p\theta) \otimes fn(t[A]_p\theta)$$

Démonstration. Par récurrence sur la structure du terme clos A :

– Soit $A = \bot$, et de par la définition de $\otimes$,

$$fn(t[\bot \otimes B]_p\theta) = fn(t[B]_p\theta) = fn(t[B]_p\theta) \otimes \bot$$

Or, de par la définition 4.4 de SP, $\bot$ est la forme normale de $t[\bot]_p$.

– Soit $A = c[A']$, et $fn(t[c[A'] \otimes B]_p\theta)$

$= fn(t[c[A' \otimes B]]_p\theta)$ (d'après le lemme 2.3)

$= fn(r[t[A' \otimes B]_p]_q\theta)$ avec $q = TP(\mathcal{R},r(\varepsilon))$ (de par la définition de SP)

$= fn(t[A' \otimes B]_p\theta) \otimes fn(r[\bot]_q\theta)$ (d'après le lemme 4.3)

$= fn(t[B]_p\theta) \otimes fn(t[A]_p\theta) \otimes fn(r[\bot]_q\theta)$ (par hypothèse de récurrence)

$= fn(t[B]_p\theta) \otimes fn(r[t[A']_p]_q\theta)$ (d'après le lemme 4.3)

$= fn(t[B]_p\theta) \otimes fn(t[c[A']]_p\theta)$ (d'après la proposition 4.1 et la définition 4.4)

Dans les deux cas, nous avons bien $fn(t[A \otimes B]_p\theta) = fn(t[B]_p\theta) \otimes fn(t[A]_p\theta)$. □

• ***Exemple* 4.1.11.** Rappelons que $SP(\mathcal{R},r) = 1$.

$r(\mathbf{a}.\mathbf{b}.c.d.\varnothing) \to ap(r(b.c.d.\varnothing),a.\varnothing) \to ap(ap(r(c.d.\varnothing),b.\varnothing),a.\varnothing)$

$\to ap(ap(ap(r(d.\varnothing),c.\varnothing),b.\varnothing),a.\varnothing) \to ap(ap(ap(ap(r(\varnothing),d.\varnothing),c.\varnothing),b.\varnothing),a.\varnothing)$

$\to ap(ap(ap(ap(\varnothing,d.\varnothing),c.\varnothing),b.\varnothing),a.\varnothing) \to^* d.c.b.a.\varnothing = fn(r(c.d.\varnothing)) \otimes fn(r(\mathbf{a}.\mathbf{b}.\varnothing))$

4.2 Parties engendrées par les sous-termes libres

Dans cette section, nous allons définir les chemins amenant dans un terme à tous les sous-termes libres engendrant des parties de début et de fin de la forme normale des instances closes du terme. En effet, grâce aux définitions de la section précédente nous sommes à même de formaliser le calcul des parties engendrées par les sous-termes libres déjà aperçu section 3.3. Nous donnerons quelques exemples supplémentaires, de chemins et de parties engendrées, basés sur le système monomorphique $\mathcal{R}$ de la section précédente (dont la table des arguments libres se trouve page 79).

Enfin, nous établirons formellement le découpage d'un terme suivant les parties engendrées et leurs compléments.

4.2.1 Chemins hauts et bas

Comme nous l'avons vu dans la section 3.3, une suite de positions d'arguments libres peut amener à un sous-terme libre. De plus, si ces positions sont correctement ordonnées, nous pouvons connaître la place de la partie engendrée par ce sous-terme (début ou fin de la forme normale des instances closes du terme).

Nous définissons en premier lieu les chemins hauts et bas maximums. Ce sont ceux qui descendent le plus loin possible dans le terme en utilisant les définitions 4.2, 4.3, et 4.4.

Définition 4.6. Soit $\mathcal{R}$ un système de réécriture monomorphique, pour un terme t, le *chemin haut maximum* noté $CH(t)$, et le *chemin bas maximum* noté $CB(t)$ sont calculés récursivement ainsi :

$$\exists p \in \{RA(\mathcal{R},t(\varepsilon)),TP(\mathcal{R},t(\varepsilon)),CP(\mathcal{R},t(\varepsilon))\} \Rightarrow CH(t) = pCH(t/p)$$
$$\exists p = SP(\mathcal{R},t(\varepsilon)) \Rightarrow CH(t) = pCB(t/p)$$
$$\nexists p \in \{RA(\mathcal{R},t(\varepsilon)),TP(\mathcal{R},t(\varepsilon)),CP(\mathcal{R},t(\varepsilon)),SP(\mathcal{R},t(\varepsilon))\} \Rightarrow CH(t) = \varepsilon$$

$$\exists p \in \{RA(\mathcal{R},t(\varepsilon)),DP(\mathcal{R},t(\varepsilon)),CP(\mathcal{R},t(\varepsilon))\} \Rightarrow CB(t) = pCB(t/p)$$
$$\exists p = SP(\mathcal{R},t(\varepsilon)) \Rightarrow CB(t) = pCH(t/p)$$
$$\nexists p \in \{RA(\mathcal{R},t(\varepsilon)),DP(\mathcal{R},t(\varepsilon)),CP(\mathcal{R},t(\varepsilon)),SP(\mathcal{R},t(\varepsilon))\} \Rightarrow CB(t) = \varepsilon$$

Remarque : Le caractère « spécial » du type de position SP est dû au fait qu'une fonction ayant un tel argument utilise la fin de la forme normale de l'argument pour construire le début de la forme normale du terme (voir lemme 4.5). Nous retrouverons cette particularité pour le calcul des parties.

- ***Exemple* 4.2.1. ap(r(ap(l,n)),R(n,l))**

  - $CH(ap(r(ap(l,n)),R(n,l))) = 1CH(r(ap(l,n))) = 11CB(ap(l,n)) = 112CB(n)$
    $= 112$

- $CB(ap(r(ap(l,n)),R(n,l))) = 2CB(R(n,l)) = 22CB(l) = 22$

- ***Exemple* 4.2.2.** $\mathbf{x} * \mathbf{m}(\mathbf{y},\mathbf{z} + \mathbf{x})$

  - $CH(x * m(y,z + x)) = 2CH(m(y,z + x)) = 22CB(z + x) = 221CB(z) = 221$
  - $CB(x * m(y,z + x)) = 2CB(m(y,z + x)) = 22CH(z + x) = 222CH(x) = 222$

Cependant, il n'est pas toujours adéquat d'utiliser un chemin maximum. Par exemple, dans une généralisation, plus le sous-terme choisi est important, plus la conjecture obtenue sera générale et plus la tentative de preuve possède de chance d'aboutir. C'est pourquoi nous utilisons une deuxième définition.

Définition 4.7. Soit $\mathcal{R}$ un système de réécriture monomorphique, pour un terme t, toute position $p \leq CH(t)$ préfixe d'un chemin haut maximum est un *chemin haut*; et toute position $p \leq CB(t)$ préfixe d'un chemin bas maximum est un *chemin bas*.

- ***Exemple* 4.2.3.** $\mathbf{t} = \mathbf{s}$: $\mathbf{x} * (\mathbf{y} + (\mathbf{x} + \mathbf{x})) = (\mathbf{y} * \mathbf{x}) + (\mathbf{y} * (\mathbf{x} + \mathbf{x}))$

  Nous avons les chemins hauts maximums suivants :

  - $CH(t) = CH(x * (y + (x + x))) = 222$ avec $t/222 = x$ et
  - $CH(s) = CH((y * x) + (y * (x + x))) = 222$ avec $s/222 = x$

Or, pour les chemins hauts 22 à gauche et 22 à droite, nous avons $t/22 = x + x$ et $s/22 = x + x$.

4.2.2 Calcul des parties engendrées

Le fait de savoir que les deux côtés d'une conjecture possèdent un même sous-terme libre qui engendre de chaque côté, une partie placée au même endroit, ne suffit pas à simplifier une conjecture. Il faut encore que les parties engendrées par ces sous-termes soit elles-même identiques.

On ne calcule pas directement ces parties car ce sont des parties de la forme normale, mais on calcule une représentation. Ce calcul suit récursivement le chemin correspondant – haut ou bas.

Définition 4.8. Soit $\mathcal{R}$ un système de réécriture monomorphique, pour un terme t, et un chemin haut p la *partie haute* de t notée $top(t,p)$ est calculée récursivement ainsi :

$$q = RA(\mathcal{R},t(\varepsilon)) \Rightarrow top(t,q.p) = t[top(t/q,p)]_q$$
$$q = TP(\mathcal{R},t(\varepsilon)) \Rightarrow top(t,q.p) = top(t/q,p)$$
$$q = CP(\mathcal{R},t(\varepsilon)) \Rightarrow top(t,q.p) = t[top(t/q,p)]_q$$
$$q = SP(\mathcal{R},t(\varepsilon)) \Rightarrow top(t,q.p) = t[bot(t/q,p)]_q$$
$$top(t,\varepsilon) = t$$

Si p est un chemin bas, la *partie basse* de t notée $bot(t,p)$ est calculée récursivement ainsi :

$$q = RA(\mathcal{R},t(\varepsilon)) \Rightarrow bot(t,q.p) = bot(t/q,p)$$
$$q = DP(\mathcal{R},t(\varepsilon)) \Rightarrow bot(t,q.p) = bot(t/q,p)$$
$$q = CP(\mathcal{R},t(\varepsilon)) \Rightarrow bot(t,q.p) = t[bot(t/q,p)]_q$$
$$q = SP(\mathcal{R},t(\varepsilon)) \Rightarrow bot(t,q.p) = t[top(t/q,p)]_q$$
$$bot(t,\varepsilon) = t$$

- ***Exemple* 4.2.4. ap(r(ap(l,n)),R(n,l))**

  Nous avons le chemin haut 112 et le chemin bas 22 :

  – $top(ap(r(ap(l,n)),R(n,l))),112) = top(r(ap(l,n)),12) = r(bot(ap(l,n),2)) = r(bot(n,\varepsilon)) = r(n)$

  – $bot(ap(r(ap(l,n)),R(n,l))),22) = bot(R(n,l),2) = bot(l,\varepsilon) = l$

  La définition ci-dessous des compléments aux parties hautes et basses n'est pas

indispensable pour l'algorithme de généralisation, mais elle sert pour la preuve de la correction de celui-ci ainsi que pour la partition de termes.

Définition 4.9. Soit $\mathcal{R}$ un système de réécriture monomorphique, pour un terme t, et un chemin haut p (resp. un chemin bas p) le *complément de la partie haute* de t notée $ntp(t,p)$ (resp. le *complément de la partie basse* de t notée $nbt(t,p)$) est calculé récursivement ainsi :

$$\begin{array}{rll|l}
q = RA(\mathcal{R},t(\varepsilon)) \Rightarrow & ntp(t,q.p) = ntp(t/q,p) & & nbt(t,q.p) = t[nbt(t/q,p)]_q \\
q = DP(\mathcal{R},t(\varepsilon)) \Rightarrow & & & nbt(t,q.p) = t[nbt(t/q,p)]_q \\
q = TP(\mathcal{R},t(\varepsilon)) \Rightarrow & ntp(t,q.p) = t[ntp(t/q,p)]_q & & \\
q = CP(\mathcal{R},t(\varepsilon)) \Rightarrow & ntp(t,q.p) = t[ntp(t/q,p)]_q & & nbt(t,q.p) = t[nbt(t/q,p)]_q \\
q = SP(\mathcal{R},t(\varepsilon)) \Rightarrow & ntp(t,q.p) = t[nbt(t/q,p)]_q & & nbt(t,q.p) = t[ntp(t/q,p)]_q \\
 & ntp(t,\varepsilon) = \perp_{type(t)} & & nbt(t,\varepsilon) = \perp_{type(t)}
\end{array}$$

- ***Exemple* 4.2.5.** $\mathbf{x} * \mathbf{m}(\mathbf{y},\mathbf{z} + \mathbf{x})$

  - $\mathbf{ntp}(\mathbf{x} * \mathbf{m}(\mathbf{y},\mathbf{z} + \mathbf{x}),\mathbf{222}) = x * \mathbf{ntp}(\mathbf{m}(\mathbf{y},\mathbf{z} + \mathbf{x}),\mathbf{22}) = x * m(y,\mathbf{nbt}(\mathbf{z} + \mathbf{x},\mathbf{2})) = x * m(y,z + \mathbf{nbt}(\mathbf{x},\varepsilon)) = x * m(y,z + \mathbf{0})$
  - $\mathbf{nbt}(\mathbf{x} * \mathbf{m}(\mathbf{y},\mathbf{z} + \mathbf{x}),\mathbf{221}) = x * \mathbf{nbt}(\mathbf{m}(\mathbf{y},\mathbf{z} + \mathbf{x}),\mathbf{21}) = x * m(y,\mathbf{ntp}(\mathbf{z} + \mathbf{x},\mathbf{1})) = x * m(y,\mathbf{nbt}(\mathbf{z},\varepsilon) + x) = x * m(y,\mathbf{0} + x)$

4.2.3 Théorème principal

Nous sommes maintenant près d'introduire le théorème principal de ce chapitre et sa preuve. Ce théorème nous permet d'établir la correction de nos méthodes de généralisation et de partition (voir chapitres suivants). *Le théorème de découpage ci-dessous montre qu'un terme muni d'un chemin haut – ou bas – possède deux parties bien définies.*

Théorème 4.6. *Soit $\mathcal{R}$ un système de réécriture monomorphique, pour un terme t, une position $p \in dom(p)$, et une substitution close θ ;*

- *Si p est un chemin haut alors $fn(t\theta) = fn(top(t,p)\theta) \otimes fn(ntp(t,p)\theta)$.*
- *Si p est un chemin bas alors $fn(t\theta) = fn(nbt(t,p)\theta) \otimes fn(bot(t,p)\theta)$.*

Démonstration. • Par induction sur le chemin haut p, d'après la définition 4.6, nous avons un cas de base (ε) et quatre cas de récurrence :

- Soit $p = \varepsilon$, d'après les définitions 4.8 et 4.9 de *top* et *ntp*,

$$fn(t\theta) = fn(t\theta) \otimes \bot = fn(top(t,\varepsilon)\theta) \otimes fn(ntp(t,\varepsilon)\theta)$$

- Soit $p = qp'$ avec a tel que $t = t[a]_q$, comme de par les propriétés de la substitution $fn(t[a]_q\theta) = fn(t[fn(a\theta)]_q\theta)$,
  - soit $q = RA(\mathcal{R},t(\varepsilon))$, et $fn(t[fn(a\theta)]_q\theta)$

    $= fn(t[fn(top(a,p')\theta) \otimes fn(ntp(a,p')\theta)]_q\theta)$ (par hypothèse de récurrence)

    $= fn(t[fn(top(a,p')\theta)]_q\theta) \otimes fn(ntp(a,p')\theta)$ (d'après le lemme 2.3)
  - soit $q = TP(\mathcal{R},t(\varepsilon))$, et $fn(t[fn(a\theta)]_q\theta)$

    $= fn(t[fn(top(a,p')\theta) \otimes fn(ntp(a,p')\theta)]_q\theta)$ (par hypothèse de récurrence)

    $= (fn(top(a,p')\theta) \otimes fn(ntp(a,p')\theta)) \otimes fn(t[\bot]_q\theta)$ (d'après le lemme 4.3)

    $= fn(top(a,p')\theta) \otimes fn(t[fn(ntp(a,p')\theta)]_q\theta)$ (d'après le lemme 4.3)
  - soit $q = CP(\mathcal{R},t(\varepsilon))$, et $fn(t[fn(a\theta)]_q\theta)$

    $= fn(t[fn(top(a,p')\theta) \otimes fn(ntp(a,p')\theta)]_q\theta)$ (par hypothèse de récurrence)

    $= fn(t[fn(top(a,p')\theta)]_q\theta) \otimes fn(t[fn(ntp(a,p')\theta)]_q\theta)$ (d'après le lemme 4.4)
  - soit $q = SP(\mathcal{R},t(\varepsilon))$ avec p' chemin bas, et $fn(t[fn(a\theta)]_q\theta)$

    $= fn(t[fn(ntb(a,p')\theta) \otimes fn(bot(a,p')\theta)]_q\theta)$ (par hypothèse de récurrence)

    $= fn(t[fn(bot(a,p')\theta)]_q\theta) \otimes fn(t[fn(ntp(a,p')\theta)]_q\theta)$ (d'après le lemme 4.5)

Dans chaque cas, de par les propriétés de la substitution et d'après la définition de *top* et *ntp*, $fn(t[a]_q\theta) = fn(top(t,qp')\theta) \otimes fn(ntp(t,qp')\theta)$.

• Par induction sur le chemin bas p, d'après la définition 4.6, nous avons un cas de base (ε) et quatre cas de récurrence :

– Soit $p = \varepsilon$, d'après les définitions 4.8 et 4.9 de *bot* et *ntp*,

$$fn(t\theta) = \bot \otimes fn(t\theta) = fn(nbt(t,\varepsilon)\theta) \otimes fn(bot(t,\varepsilon)\theta)$$

– Soit $p = qp'$ avec a tel que $t = t[a]_q$, comme de par les propriétés de la substitution $fn(t[a]_q\theta) = fn(t[fn(a\theta)]_q\theta)$,

– soit $q = RA(\mathcal{R},t(\varepsilon))$, et $fn(t[fn(a\theta)]_q\theta)$

$= fn(t[fn(nbt(a,p')\theta) \otimes fn(bot(a,p')\theta)]_q\theta)$ (par hypothèse de récurrence)

$= fn(t[fn(nbt(a,p')\theta)]_q\theta) \otimes fn(bot(a,p')\theta)$ (d'après le lemme 2.3)

– soit $q = DP(\mathcal{R},t(\varepsilon))$, et $fn(t[fn(a\theta)]_q\theta)$

$= fn(t[fn(nbt(a,p')\theta) \otimes fn(bot(a,p')\theta)]_q\theta)$ (par hypothèse de récurrence)

$= fn(t[\bot]_q\theta) \otimes (fn(nbt(a,p')\theta) \otimes fn(bot(a,p')\theta))$ (d'après le lemme 4.2)

$= fn(t[fn(nbt(a,p')\theta)]_q\theta) \otimes fn(bot(a,p')\theta)$ (d'après le lemme 4.2)

– soit $q = CP(\mathcal{R},t(\varepsilon))$, et $fn(t[fn(a\theta)]_q\theta)$

$= fn(t[fn(nbt(a,p')\theta) \otimes fn(bot(a,p')\theta)]_q\theta)$ (par hypothèse de récurrence)

$= fn(t[fn(nbt(a,p')\theta)]_q\theta) \otimes fn(t[fn(bot(a,p')\theta)]_q\theta)$ (d'après le lemme 4.4)

– soit $q = SP(\mathcal{R},t(\varepsilon))$ avec p' chemin haut, et $fn(t[fn(a\theta)]_q\theta)$

$= fn(t[fn(top(a,p')\theta) \otimes fn(ntp(a,p')\theta)]_q\theta)$ (par hypothèse de récurrence)

$= fn(t[fn(top(a,p')\theta)]_q\theta) \otimes fn(t[fn(ntp(a,p')\theta)]_q\theta)$ (d'après le lemme 4.5)

Dans chaque cas de par les propriétés de la substitution et d'après la définition de *nbt* et *bot*, $fn(t[a]_q\theta) = fn(nbt(t,qp')\theta) \otimes fn(bot(t,qp')\theta)$.

□

Chapitre 5

Généralisations correctes

Dans ce chapitre, nous montrerons formellement comment établir des généralisations correctes en nous basant sur les définitions du chapitre 4. Nous poserons notamment un algorithme généralisant des conjectures ainsi que le théorème établissant sa correction. Nous exposerons ensuite quelques exemples de telles généralisations pour finir par analyser les capacités et les limites de cette méthode.

5.1 Formalisation

Dans cette section nous formaliserons l'algorithme de la section 3.4, puis nous établirons sa correction à l'aide d'une proposition portant sur les sous-termes libres et d'un théorème.

5.1.1 Algorithme de généralisation

Comme nous l'avons vu dans le chapitre 3, généraliser une conjecture nécessite trois conditions :

1. Qu'il existe deux positions à droite et à gauche de l'équation qui contiennent le même sous-terme.

2. Que les sous-termes à ces positions engendrent des parties placées au même endroit – début ou fin – dans la forme normale de leur côté respectif.
3. Que les sous-termes à ces positions engendrent des parties identiques.

Ces conditions peuvent être vérifiées en utilisant la définition 4.7 des chemins hauts et bas ainsi que la définition 4.8 des parties hautes et basses.

Pour un système monomorphique $\mathcal{R}$, en supposant que la table des arguments libres $TAL(\mathcal{R})$ ait été calculée en utilisant la définition 4.5, notre algorithme de généralisation correcte est[1] :

Généralisation de $t = s$
Calculer $CH(t)$ et $CB(t)$, ainsi que $CH(s)$ et $CB(s)$;
Pour toutes les positions p préfixes de $CH(t)$ en ordre croissant de longueur
Pour toutes les positions q préfixes de $CH(s)$ en ordre croissant de longueur
Si $t/p \equiv s/q$ et $top(t,p) \equiv top(s,q)$ **Alors** $t[x]_p = s[x]_q$ – où x est une nouvelle variable – est une généralisation de $t = s$;
Pour toutes les positions p préfixes de $CB(t)$ en ordre croissant de longueur
Pour toutes les positions q préfixes de $CB(s)$ en ordre croissant de longueur
Si $t/p \equiv s/q$ et $bot(t,p) \equiv bot(s,q)$ **Alors** $t[x]_p = s[x]_q$ – où x est une nouvelle variable – est une généralisation de $t = s$;

Cette algorithme de généralisation peut être décrit sous la forme d'une règle d'inférence (voir section 2.1.1).

Définition 5.1. Pour un système de réécriture $\mathcal{R}$ monomorphique et pour un ensemble d'équations C, la *généralisation correcte* est la règle d'inférence

GC : $\quad C \cup \{t = s\} \vdash C \cup \{t' = s'\}$

Si $t' = s'$ *est un généralisation correcte de* $t = s$.

1. Rappelons que le symbole $\equiv$ représente l'égalité syntaxique et que $=_{ind(\mathcal{R})}$ indique que l'équation est un théorème inductif (voir chapitre 1)

Cette règle d'inférence doit être appliqué juste avant une induction. L'ordre de priorité des règle devient alors :

$$\mathbf{SUP} \gg \mathbf{INC} \gg \mathbf{GC} \gg \mathbf{GEN}$$

Ainsi toute généralisation possible d'une conjecture est effectuée avant même que la tentative de preuve par induction « classique » ne commence (et qu'une divergence ne s'engage). Elle peut aussi intervenir durant la tentative de preuve.

5.1.2 Théorème de correction

Pour démontrer que la validité de la conjecture ne change pas lors de la substitution du sous-terme libre, nous devons dans un premier lieu, montrer que le sous-terme désigné par nos définitions est « vraiment » libre. C'est-à-dire que les parties de la forme normale close du terme, autres que celle engendrée par le sous-terme, ne dépendent pas de ce sous-terme. C'est le but de la proposition ci-dessous.

Proposition 5.1. *Soit $\mathcal{R}$ un système de réécriture monomorphique, pour un terme t et une position $p \in dom(p)$,*

- *Si p est un chemin haut alors $ntp(t,p) = ntp(t[\bot_{type(t)}]_p,p)$.*
- *Si p est un chemin bas alors $nbt(t,p) = nbt(t[\bot_{type(t)}]_p,p)$.*

Démonstration. Par induction sur le chemin haut ou bas p, d'après la définition 4.9 de ntp et nbt, nous avons un cas de base (ε) et cinq cas possibles de récurrence :

- Soit $p = \varepsilon$, et

$$ntp(t,\varepsilon) = \bot = t[\bot]_\varepsilon = ntp(t[\bot]_\varepsilon,\varepsilon)$$
$$nbt(t,\varepsilon) = \bot = t[\bot]_\varepsilon = nbt(t[\bot]_\varepsilon,\varepsilon)$$

- Soit $p = qp'$ avec a tel que $t = t[a]_q$, nous pouvons avoir
  - soit $ntp(t[a]_q,qp') = ntp(a,p')$ avec p' chemin haut de a,

- soit $ntp(t[a]_q,qp') = t[ntp(a,p')]_q$ avec p' chemin haut de a,
- soit $ntp(t[a]_q,qp') = t[nbt(a,p')]_q$ avec p' chemin bas de a,
- soit $nbt(t[a]_q,qp') = t[nbt(a,p')]_q$ avec p' chemin bas de a,
- soit $nbt(t[a]_q,qp') = t[ntp(a,p')]_q$ avec p' chemin haut de a.

Or, par hypothèse de récurrence, $ntp(a,p') = ntp(a[\bot]_{p'},p')$ et $nbt(a,p') = nbt(a[\bot]_{p'},p')$ donc

$$ntp(t[a]_q,qp') = ntp(t[a[\bot]_{p'}]_q,qp') = ntp(t[a]_q[\bot]_{qp'},qp')$$
$$nbt(t[a]_q,qp') = nbt(t[a[\bot]_{p'}]_q,qp') = nbt(t[a]_q[\bot]_{qp'},qp')$$

Nous avons donc bien $ntp(t,p) = ntp(t[\bot]_p,p)$ et $nbt(t,p) = nbt(t[\bot]_p,p)$. □

La correction de l'algorithme de généralisation, c'est-à-dire *la conjecture est un théorème inductif si et seulement si la généralisation est un théorème inductif*, dépend du théorème ci-dessous.

Théorème 5.2. *Soit $\mathcal{R}$ un système de réécriture monomorphique, pour une équation $t = s$ et deux positions $p \leq CH(t)$ et $q \leq CH(s)$ (respectivement $p \leq CB(t)$ et $q \leq CB(s)$) telles que $t/p \equiv s/q$ et que $top(t,p) \equiv top(s,q)$ (respectivement $bot(t,p) \equiv bot(s,q)$)*

*$t =_{ind(\mathcal{R})} s$ **si et seulement si** $t[x]_p =_{ind(\mathcal{R})} s[x]_q$, avec x une nouvelle variable.*

Démonstration. • Posons $t[x]_p =_{ind(\mathcal{R})} s[x]_q$.

Pour toute substitution close θ, il existe une substitution σ telle que $x\sigma = t/p = s/q$. Or, quelque soit $\theta' = \sigma\theta$, $fn(t[x]_p\theta') = fn(s[x]_q\theta')$.

Donc nous avons bien

$$fn(t\theta) = fn(t[x]_p\theta') = fn(s[x]_q\theta') = fn(s\theta)$$

• Posons maintenant $t =_{ind(\mathcal{R})} s$

Soit une substitution close θ quelconque, d'après le théorème de découpage 4.6,

$$fn(t\theta) = fn(top(t,p)\theta) \otimes fn(ntp(t,p)\theta) \text{ et}$$
$$fn(t[x]_p\theta) = fn(top(t[x]_p,p)\theta) \otimes fn(ntp(t[x]_p,p)\theta)$$

Comme $top(t,p) \equiv top(s,q)$, et que $t/p \equiv s/q$, de par la définition 4.8 de *top*, $top(t[x]_p,p) = top(s[x]_q,q)$. De plus, nous avons $t =_{ind(\mathcal{R})} s$ et donc :

$$fn(top(t,p)\theta) \otimes fn(ntp(t,p)\theta) = fn(top(s,q)\theta) \otimes fn(ntp(s,q)\theta)$$

Comme $fn(top(t,p)\theta) = fn(top(s,q)\theta)$, d'après le lemme 2.1, on obtient $fn(ntp(t,p)\theta)$ $= fn(ntp(s,q)\theta)$, et, d'après la proposition 5.1, $fn(ntp(t[x]_p,p)\theta) = fn(ntp(s[x]_q,q)\theta)$. En conséquence, et puisque $top(t,p) \equiv top(s,q)$, nous avons $top(t[x]_p,p) \equiv top(s[x]_q,q)$ et

$$fn(top(t[x]_p,p)\theta) \otimes fn(ntp(t[x]_p,p)\theta) = fn(top(s[x]_q,q)\theta) \otimes fn(ntp(s[x]_q,q)\theta)$$
$$\text{donc } t[x]_p =_{ind(\mathcal{R})} s[x]_q$$

La démonstration avec $bot(t,p) \equiv bot(s,q)$ est totalement symétrique. □

5.2 Utilisation des généralisations

Dans cette section, nous illustrerons notre algorithme de généralisation sur quelques exemples portant sur les listes et l'arithmétique puis nous analyserons les possibilités offertes par cet algorithme.

5.2.1 Exemples

Le système de réécriture utilisé dans chacun des exemples est le système $\mathcal{R}$ suivant :

$$\mathcal{R} = \left\{ \begin{array}{rlrl} x + s(y) & \to s(x + y) & x + 0 & \to x \\ x * s(y) & \to (x * y) + x & x * 0 & \to 0 \\ exp(x,s(y)) & \to exp(x,y) * x & exp(x,0) & \to s(0) \\ m(x,s(y)) & \to x + m(x,y) & m(x,0) & \to 0 \\ \Sigma(s(x)) & \to s(x) + \Sigma(x) & \Sigma(0) & \to 0 \\ \Sigma I(s(x),y) & \to \Sigma I(x,s(y + x)) & \Sigma I(0,y) & \to y \\ r(c.l) & \to ap(r(l),c.\varnothing) & r(\varnothing) & \to \varnothing \\ ap(c.l,L) & \to c.ap(l,L) & ap(\varnothing,L) & \to L \\ R(c.l,L) & \to R(l,c.L) & R(\varnothing,L) & \to L \end{array} \right. \tag{5.2.1}$$

dont la table des arguments libres se trouve page 79.

- ***Exemple* 5.2.1.**

$$ap(l,ap(l,l)) = ap(ap(l,l),l)$$

Nous avons $CH(ap(l,ap(l,l))) = 1$ et $CH(ap(ap(l,l),l)) = 11$, et comme

- $ap(l,ap(l,l))/1 = ap(ap(l,l),l)/11 = l$ et
- $top(ap(l,ap(l,l)),1) = top(ap(ap(l,l),l),11) = l$

On obtient la généralisation : $\mathbf{ap(M,ap(l,l)) = ap(ap(M,l),l)}$

Nous avons aussi $CB(ap(l,ap(l,l))) = 22$ et $CB(ap(ap(l,l),l)) = 2$, et comme

- $ap(l,ap(l,l))/22 = ap(ap(l,l),l)/2 = l$ et
- $bot(ap(l,ap(l,l)),22) = bot(ap(ap(l,l),l),2) = l$

On obtient la généralisation : $\mathbf{ap(M,ap(l,N)) = ap(ap(M,l),N)}$

- ***Exemple* 5.2.2.**

$$ap(r(l),ap(l,c.\varnothing)) = ap(ap(r(l),l),c.\varnothing)$$

Nous avons $CH(ap(r(l),ap(l,c.\varnothing))) = 11$ et $CH(ap(ap(l,l),c.\varnothing)) = 111$, et comme

- $ap(r(l),ap(l,c.\varnothing))/1 = ap(ap(r(l),l),c.\varnothing)/11 = r(l)$ et
- $top(ap(r(l),ap(l,c.\varnothing)),1) = top(ap(ap(r(l),l),c.\varnothing),11) = r(l)$

On obtient la généralisation : $\mathbf{ap}(\mathbf{M},\mathbf{ap}(\mathbf{l},\mathbf{c}.\varnothing)) = \mathbf{ap}(\mathbf{ap}(\mathbf{M},\mathbf{l}),\mathbf{c}.\varnothing)$

Nous avons aussi $CB(ap(r(l),ap(l,c.\varnothing))) = 22$ et $CB(ap(ap(r(l),l),c.\varnothing)) = 2$, et comme

- $ap(r(l),ap(l,c.\varnothing))/22 = ap(ap(r(l),l),c.\varnothing)/2 = c.\varnothing$ et
- $bot(ap(r(l),ap(l,c.\varnothing)),22) = bot(ap(ap(r(l),l),c.\varnothing),2) = c.\varnothing$

On obtient la généralisation : $\mathbf{ap}(\mathbf{M},\mathbf{ap}(\mathbf{l},\mathbf{N})) = \mathbf{ap}(\mathbf{ap}(\mathbf{M},\mathbf{l}),\mathbf{N})$

- ***Exemple* 5.2.3.**

$$R(l,c.\varnothing) = ap(r(l),c.\varnothing)$$

Nous avons $CB(R(l,c.\varnothing)) = 22$ et $CB(ap(r(l),c.\varnothing)) = 22$, et comme

- $R(l,c.\varnothing)/2 = ap(r(l),c.\varnothing)/2 = c.\varnothing$ et
- $bot(R(l,c.\varnothing),2) = bot(ap(r(l),c.\varnothing),2) = c.\varnothing$

On obtient la généralisation : $\mathbf{R}(\mathbf{l},\mathbf{M}) = \mathbf{ap}(\mathbf{r}(\mathbf{l}),\mathbf{M})$

- ***Exemple* 5.2.4.**

$$c.r(r(l)) = r(ap(r(l),c.\varnothing))$$

Nous avons $CH(c.r(r(l))) = 211$ et $CH(r(ap(r(l),c.\varnothing))) = 111$, et comme

- $c.r(r(l))/21 = r(ap(r(l),c.\varnothing))/11 = r(l)$ et
- $top(c.r(r(l)),21) = top(r(ap(r(l),c.\varnothing)),11) = r(r(l))$

On obtient la généralisation : $\mathbf{c}.\mathbf{r}(\mathbf{M}) = \mathbf{r}(\mathbf{ap}(\mathbf{M},\mathbf{c}.\varnothing))$

- ***Exemple* 5.2.5.**

$$R(l,R(l,\varnothing)) = ap(R(l,\varnothing),R(l,\varnothing))$$

Nous avons $CB(R(l,R(l,\varnothing))) = 22$ et $CB(ap(R(l,\varnothing),R(l,\varnothing))) = 22$, et comme

- $R(l,R(l,\varnothing))/2 = ap(R(l,\varnothing),R(l,\varnothing))/2 = R(l,\varnothing)$ et
- $bot(R(l,R(l,\varnothing)),2) = bot(ap(R(l,\varnothing),R(l,\varnothing)),2) = R(l,\varnothing)$

On obtient la généralisation : $\mathbf{R(l,M) = ap(R(l,\varnothing),M)}$

- ***Exemple* 5.2.6.**

$$r(ap(R(l,\varnothing),c.l)) = ap(r(l),c.l)$$

Nous avons $CH(r(ap(R(l,\varnothing),c.l))) = 122$ et $CH(ap(r(l),c.l)) = 11$, et comme

- $r(ap(R(l,\varnothing),c.l))/122 = ap(r(l),c.l)/11 = l$ et
- $top(r(ap(R(l,\varnothing),c.l)),122) = top(ap(r(l),c.l),11) = r(l)$

On obtient la généralisation : $\mathbf{r(ap(R(l,\varnothing),c.M)) = ap(r(M),c.l)}$

- ***Exemple* 5.2.7.**

$$x * (y + (y * x)) = x * (y * (s(0) + x))$$

Nous avons $CH(x * (y + (y * x))) = 222$ et $CH(x * (y * (s(0) + x))) = 222$, et comme

- $x * (y + (y * x))/222 = x * (y * (s(0) + x))/222 = x$ et
- $top(x * (y + (y * x)),222) = top(x * (y * (s(0) + x)),222) = x * (y * x)$

On obtient la généralisation : $\mathbf{x * (y + (y * Z)) = x * (y * (s(0) + Z))}$

- ***Exemple* 5.2.8.**

$$exp(x,y) * (exp(x,z) + exp(y,x)) = exp(x,y + z) + (exp(x,y) * exp(y,x))$$

Nous avons $CH(exp(x,y)*(exp(x,z)+exp(y,x))) = 22$ et $CH(exp(x,y+z)+(exp(x,y)*exp(y,x))) = 22$, et comme

- $exp(x,y) * (exp(x,z) + exp(y,x))/22 = exp(x,y + z) + (exp(x,y) * exp(y,x))/22 = exp(y,x)$ et

– $top(exp(x,y)*(exp(x,z)+exp(y,x)),22) = top(exp(x,y+z)+(exp(x,y)*exp(y,x)),22) = exp(x,y) * exp(y,x)$

On obtient la généralisation :

$$\mathbf{exp(x,y) * (exp(x,z) + W) = exp(x,y + z) + (exp(x,y) * W)}$$

• ***Exemple* 5.2.9.**

$$m(x,(x * x) + m(x,x + x)) = m(x,m(x,x) + m(x,x)) + m(x,x * x)$$

Nous avons $CH(m(x,(x * x) + m(x,x + x))) = 212$ et $CH(m(x,m(x,x) + m(x,x)) + m(x,x * x)) = 222$, et comme

– $m(x,(x * x) + m(x,x + x))/21 = m(x,m(x,x) + m(x,x)) + m(x,x * x)/22 = x * x$,

– $top(m(x,(x * x) + m(x,x + x)),21) = top(m(x,m(x,x) + m(x,x)) + m(x,x * x),22) = m(x,x * x)$

On obtient la généralisation :

$$\mathbf{m(x,Y + m(x,x + x)) = m(x,m(x,x) + m(x,x)) + m(x,Y)}$$

Nous avons aussi $CB(m(x,(x * x) + m(x,x + x))) = 221$ et $CH(m(x,m(x,x) + m(x,x)) + m(x,x * x)) = 122$, et comme

– $m(x,(x * x) + m(x,x + x))/221 = m(x,m(x,x) + m(x,x)) + m(x,x * x)/122 = x$,

– $bot(m(x,(x * x) + m(x,x + x)),221) = bot(m(x,m(x,x) + m(x,x)) + m(x,x * x),122) = m(x,m(x,x))$

On obtient la généralisation :

$$\mathbf{m(x,Y + m(x,Z + x)) = m(x,m(x,x) + m(x,Z)) + m(x,Y)}$$

• ***Exemple* 5.2.10.**

$$s(x) + \Sigma(x) = \Sigma I(x,s(x))$$

Nous avons $CB(s(x) + \Sigma(x)) = 11$ et $CB(\Sigma I(x,s(x))) = 21$, et comme

– $s(x) + \Sigma(x)/1 = \Sigma I(x,s(x))/2 = s(x)$ et

– $bot(s(x) + \Sigma(x),1) = bot(\Sigma I(x,s(x)),2) = s(x)$

On obtient la généralisation : $\mathbf{Y} + \mathbf{\Sigma}(\mathbf{x}) = \mathbf{\Sigma I}(\mathbf{x},\mathbf{Y})$

5.2.2 Analyse des capacités de l'algorithme

Comme le montre les exemples ci-dessous, cette méthode de généralisation amène à des preuves élégantes. Contrairement à toutes les approches heuristiques, cet algorithme ne calcule que des généralisations correctes. De plus, il est capable de généraliser des variables qui ne sont pas des variables d'induction. Comme cet algorithme est relativement peu complexe, il peut être intégré directement dans n'importe quel système de preuve par induction.

Malheureusement, cette méthode possède les limites inhérentes aux systèmes monomorphiques, bien que ceux ci peuvent représenter l'arithmétique. D'autre part, elle ne peut fonctionner si les représentations des parties de début ou de fin ne sont pas syntaxiquement égales. Par exemple dans l'équation $x * (x * x) = (x * x) * x$, la généralisation correcte $x * (x * Y) = (x * x) * Y$ ne peut être trouvée directement.

Comme nous allons le voir dans le chapitre suivant, la partition de termes est une méthode plus « souple » car elle se base uniquement sur les parties en généralisant le concept de sous-terme libre.

Chapitre 6

Une nouvelle méthode pour l'induction mathématique : La partitions de termes

Dans ce chapitre, nous décrirons la méthode basée sur les partitions de termes seulement apperçus dans la section 3.5.

Notre approche de l'induction mathématique est basée directement sur une spécification finie du langage des formes normales des instances closes de termes. En effet, dans beaucoup de cas, les formes normales closes des termes peuvent être identifiées à l'avance. Les notions clés « couple - partition » et « motif » nous permettent de décrire ces formes normales.

Nous allons donc d'abord définir formellement la notion de partition qui décompose un terme et celle de motif qui décrit sa structure. Ensuite, nous poserons un algorithme basé sur ces concepts pour prouver des théorèmes inductifs ; algorithme dont nous démontrerons la correction. Et enfin, nous appliquerons cette méthodes sur diverses exemples de conjecture et nous analyserons ses capacités.

6.1 Définitions

Dans cette section, nous commencerons par donner une définition formelle aux couples formant une partition ainsi que le moyen de les calculer. Nous définirons ensuite les motifs, mais comme le moyen de les calculer diffère légèrement de ce que nous avons déjà vu, nous illustrerons ces définitions par quelques exemples.

6.1.1 Couples - partitions

La définition ci-dessous montre qu'une partition d'un terme est un couple de termes dont la réunion des formes normales est égale à la forme normale correspondante du terme.

Définition 6.1. Soit $\mathcal{R}$ un système de réécriture monomorphique, pour t un terme, une *partition* de t est un couple (a, b) tel que, pour toute substitution close θ,

$$fn(t\theta) = fn(a\theta) \otimes fn(b\theta)$$

Pour calculer les partitions – et donc les motifs (voir plus bas) – d'un terme nous utilisons les définitions du chapitre 4. Mais pour plus de lisibilité et comme nous utilisons indifféremment les découpages issus des chemins hauts ou des chemins bas, nous ajoutons une nouvelle définition.

Définition 6.2. Soit $\mathcal{R}$ un système de réécriture monomorphique, pour t un terme, l'*ensemble des découpages* de t, noté $DEC(t)$, est l'ensemble formé par les couples :

$$\{(top(t,p), fn(ntp(t,p))) \mid p \leq CH(t)\} \cup \{(fn(nbt(t,p)), bot(t,p)) \mid p \leq CB(t)\}$$

Remarque : Nous utilisons les formes normales des compléments car ceux-ci sont calculés avec l'apport d'une constante et ne sont donc pas forcément irréductibles.

L'ensemble $DEC(t)$ est calculable simplement en prenant un à un les préfixes de $CH(t)$ et $CB(t)$. De plus, le théorème 4.6 nous assure que tous les éléments de $DEC(t)$ sont bien des partitions de t.

6.1.2 Motifs

Les formes normales des instances closes d'un terme peuvent parfois être composées d'une série d'éléments similaires. Le motif d'un terme est la représentation de ces éléments. Donc la répétition de la forme normale close du motif d'un terme compose la forme normale de l'instance close correspondante du terme.

Pour le construire nous utilisons l'élément non-constant de l'ensemble test (voir définition 1.21) que nous substituons à un sous-terme placé sur un chemin d'induction du terme. Un chemin est dit d'induction s'il est constitué d'une suite de positions d'induction. Un sous-terme placé sur un chemin d'induction d'un terme est donc un argument de récurrence pour toutes les fonctions qui le précèdent dans le terme (voir définition 2.8).

Définition 6.3. Pour un système de réécriture $\mathcal{R}$ et un terme t, les *chemins d'induction* de t, sont les positions de l'ensemble

$$CI(t) = \{pp' \in dom(t) \mid p' \in PI_{\mathcal{R}}(t(p))\}$$

Un argument de récurrence étant utilisé pour définir inductivement la fonction, c'est celui qui va le plus naturellement former un motif. En effet, ce chemin de par sa définition, engendre un élément qui sera porté à se « détacher » du terme substitué (voir section 2.1.2). Or, selon la définition ci-après, le motif d'un terme et le terme lui-même forme une partition du terme substitué.

Nous distinguons deux types de motifs, les motifs « hauts » qui se rajoutent en tête de la forme normale du terme substitué et les motifs « bas » qui se rajoutent en queue. En fait, il peut exister un motif par constructeur.

Définition 6.4. Soit $\mathcal{R}$ un système de réécriture monomorphique, pour t un terme, $p \in CI(t)$ une position d'induction, et $c[x]$ un élément non-constant de $TS(\mathcal{R})$, le *motif haut* de t pour p et $c[x]$ noté $motH(t,p,c[x])$ et le *motif bas* de t pour q et $c[x]$

noté $motB(t,p,c[x])$ sont les termes, s'ils existent, tel que

$$(motH(t,p,c[x]),t[x]_p) \in DEC(fn(t[c[x]]_p))$$
$$(t[x]_p,motB(t,p,c[x])) \in DEC(fn(t[c[x]]_p))$$

Exemples

Soit le système monomorphique $\mathcal{R}'$ suivant :

$$\left\{\begin{array}{lll} x+0 \to x & x+s(y) \to s(x+y) & x+p(y) \to p(x+y) \\ neg(0) \to 0 & neg(s(x)) \to p(0)+neg(x) & neg(p(x)) \to s(0)+neg(x) \\ x*0 \to 0 & x*s(y) \to (x*y)+x & x*p(y) \to (x*y)+neg(x) \end{array}\right\} \quad (6.1.1)$$

• La table des argument $TAL(\mathcal{R}')$ et les positions d'induction sont :

f	$RA(\mathcal{R},f)$	$DP(\mathcal{R},f)$	$TP(\mathcal{R},f)$	$CP(\mathcal{R},f)$	$SP(\mathcal{R},f)$	$PI_{\mathcal{R}}(f)$
s	1					
$+$		1	2			$\{2\}$
neg					1	$\{1\}$
$*$				2		$\{2\}$

• L'ensemble test $TS(\mathcal{R}')$ est $\{0,s(x),p(x)\}$.

Les éléments non-constants de $\mathcal{R}'$ sont donc $s(x)$ et $p(x)$. Voici quelques exemples de motifs :

• ***Exemple* 6.1.1. $\mathbf{t = neg(x * x)}$**

$CI(t) = \{\varepsilon,1,12\}$ et pour le chemin d'induction $p = 12$:

Nous avons $fn(t[s(y)]_{12}) = fn(neg(x*s(y))) = neg((x*y)+x)$, et comme

– $fn(nbt(t[s(y)]_{12},12)) = fn(neg((x*y)+0)) = neg(x*y) = t[y]_{12}$ et

– $bot(t[s(y)]_{12},12) = neg(x)$

Le motif bas $motB(neg(x*x),12,s(y))$ est donc $\mathbf{neg(x)}$.

Nous avons $fn(t[p(y)]_{12}) = fn(neg(x * p(y))) = neg((x * y) + neg(x))$, et comme

- $fn(nbt(t[p(y)]_{12},12)) = fn(neg((x * y) + 0)) = neg(x * y) = t[y]_{12}$ et
- $bot(t[p(y)]_{12},12) = neg(neg(x))$

Le motif bas $motB(neg(x * x),12,p(y))$ est donc $\mathbf{neg(neg(x))}$.

- ***Exemple* 6.1.2.** $\mathbf{t = neg(neg(neg(x)))}$

  $CI(t) = \{\varepsilon,1,11,111\}$ et pour le chemin d'induction $p = 11$:

  Nous avons $fn(t[s(y)]_{11}) = fn(neg(neg(s(y)))) = neg(p(0) + neg(y))$, et comme

- $fn(nbt(t[s(y)]_{11},12)) = fn(neg(p(0) + 0)) = fn(neg(p(0))) = s(0)$ et
- $bot(t[s(y)]_{11},12) = neg(neg(y)) = t[y]_{11}$

Le motif haut $motH(neg(neg(neg(x))),11,s(y))$ est donc $\mathbf{s(0)}$.

Nous avons $fn(t[p(y)]_{11}) = fn(neg(neg(p(y)))) = neg(s(0) + neg(y))$, et comme

- $fn(nbt(t[s(y)]_{11},12)) = fn(neg(s(0) + 0)) = fn(neg(s(0))) = p(0)$ et
- $bot(t[s(y)]_{11},12) = neg(neg(y)) = t[y]_{11}$

Le motif haut $motH(neg(neg(neg(x))),11,p(y))$ est donc $\mathbf{p(0)}$.

6.2 Formalisation

Dans cette section, nous exposerons notre algorithme pour prouver des théorèmes inductifs. Puis, nous établirons sa correction grâce à deux théorèmes portant sur les partitions et sur les motifs.

6.2.1 Algorithme de preuve

Comme nous l'avons vu dans la section 3.5, simplifier une conjecture en utilisant la partition de termes nécessite seulement de trouver un membre d'une partition ou

des motifs égaux pour les deux côtés de la conjecture. S'il n'y a ni partition ni motif exploitable, on utilise la règle GEN de génération (voir section 2.1.1).

Pour un système monomorphique $\mathcal{R}$, en supposant que la table des arguments libres $TAL(\mathcal{R})$ ait été calculée en utilisant la définition 4.5, notre algorithme de preuve de théorème inductif est présenté sous la forme d'une fonction – **inductif()** – indiquant si l'équation fournie en argument est un théorème inductif.

Fonction inductif($t = s$) : booléen

Si $fn(t) \equiv fn(s)$ **Alors Retourner** VRAI;
Si $vars(t) = \varnothing$ ou $vars(s) = \varnothing$ **Alors Retourner** FAUX;
Pour tous les éléments (a,b) de $DEC(t)$ **et** tous les éléments (c,d) de $DEC(s)$
    Si $a \equiv c$ **Alors Retourner** inductif($b = d$);
    Si $b \equiv d$ **Alors Retourner** inductif($a = c$);
Soit T = type(t);
Pour tous les éléments p de $CI(t)$ **et** tous les éléments q de $CI(s)$
    Si $t/p \equiv s/q$ **Alors**
        Pour P dans $\{BAS,HAUT\}$
            Si $\forall c[x] \in TS(\mathcal{R})$. motif($P$,$t$,$p$,$c[x]$) $\neq$ nul **Alors**
                Si $\forall c[x] \in TS(\mathcal{R})$. motif($P$,$t$,$p$,$c[x]$) $\equiv$ motif(P,s,q,$c[x]$) **Alors**
                    Retourner inductif($t[\perp_T]_p = s[\perp_T]_q$);
                Sinon Si $fn(t[\perp_T]_p) \equiv fn(s[\perp_T]_q)$ **Alors**
                    Retourner $\forall c[x] \in TS(\mathcal{R})$. inductif(motif($P$,$t$,$p$,$c[x]$) = motif($P$,$s$,$q$,$c[x]$));
Retourner inductif(GEN($t = s$));

La fonction ci-dessous calcule le motif haut ou bas d'un terme pour une position et un élément non-constant de l'ensemble test donné.

Fonction motif(place, t, p, c[x]) : terme
Pour tous les éléments (a,b) de $DEC(fn(t[c[x]]_p))$
Si $place = BAS$ et $a \equiv t[x]_p$ **Alors Retourner** b;
Sinon Si $place = HAUT$ et $b \equiv t[x]_p$ **Alors Retourner** a;
Retourner nul;

Remarque : Notre algorithme utilise plusieurs fois le même motif d'un même terme ainsi que le découpage dont il est issu. Une programmation effective peut bien sûr pré-calculer tous les motifs d'un terme une seule fois, par exemple sous la forme d'une table associant chaque chemin d'induction et chaque constructeur au motif correspondant. Nous le laissons sous cette forme pour plus de clarté.

6.2.2 Correction de la méthode

Pour démontrer la validité de notre approche, nous devons nous assurer que la validité de l'équation ne change pas si

- l'on « supprime » deux parties égales des deux côtés de l'équation.
- l'on substitue à la variable d'induction un « cas de base » s'il existe des motifs égaux pour les deux côtés de l'équation ou inversement.

Le théorème ci-dessous énonce la propriété clé de la partition de termes :

Théorème 6.1. *Soit $\mathcal{R}$ un système de réécriture monomorphique, pour une équation $t = s$, (a,b) une partition de t, et (c,d) une partition de s,*

Si $a =_{ind(\mathcal{R})} c$ alors, $t =_{ind(\mathcal{R})} s$ ***si et seulement si*** *$b =_{ind(\mathcal{R})} d$*

Si $b =_{ind(\mathcal{R})} d$ alors, $t =_{ind(\mathcal{R})} s$ ***si et seulement si*** *$a =_{ind(\mathcal{R})} c$*

Démonstration. Soit θ une substitution close ; comme (a,b) est une partition de t et

(c,d) est une partition de s, de par la définition 6.1,

$$fn(t\theta) = fn(a\theta) \otimes fn(b\theta)$$

$$fn(s\theta) = fn(c\theta) \otimes fn(d\theta)$$

• Assumons $t =_{ind(\mathcal{R})} s$, comme $\mathcal{R}$ est convergent sur les termes clos, $fn(t\theta) = fn(s\theta)$ et

$$fn(a\theta) \otimes fn(b\theta) = fn(c\theta) \otimes fn(d\theta)$$

– Si $a =_{ind(\mathcal{R})} c$, alors d'après le lemme 2.1, $b =_{ind(\mathcal{R})} d$

– Similairement, si $b =_{ind(\mathcal{R})} d$, toujours d'après le lemme 2.1, $a =_{ind(\mathcal{R})} c$

• Assumons $a =_{ind(\mathcal{R})} c$ et $b =_{ind(\mathcal{R})} d$, comme $\mathcal{R}$ est monomorphique, nous avons une unique position p telle que $fn(a\theta)/p = fn(c\theta)/p = \perp$; donc, d'après la définition 2.15,

$$fn(a\theta) \otimes fn(b\theta) = fn(c\theta) \otimes fn(d\theta)$$

Donc nous avons bien $t =_{ind(\mathcal{R})} s$ □

Le théorème ci-dessous énonce la propriété sur les motifs :

Théorème 6.2. *Soit $\mathcal{R}$ un système de réécriture monomorphique, pour une équation $t = s$ tel que $\exists p \in CI(t)$, $\exists q \in CI(s)$, $t/p =_{ind(\mathcal{R})} s/q$.*

S'il existe des motifs hauts de t et s pour p et q et pour tous les éléments non-constants de l'ensemble test $TS(\mathcal{R})$, alors

– *Si $\forall c[x] \in TS(\mathcal{R})$. $motH(t,p,c[x]) =_{ind(\mathcal{R})} motH(s,q,c[x])$, alors $t =_{ind(\mathcal{R})} s$*

si et seulement si *$t[\perp_{type(t)}]_p =_{ind(\mathcal{R})} s[\perp_{type(t)}]_q$*

– *Si $t[\perp_{type(t)}]_p =_{ind(\mathcal{R})} s[\perp_{type(t)}]_q$, alors $t =_{ind(\mathcal{R})} s$*

si et seulement si *$\forall c[x] \in TS(\mathcal{R})$, $motH(t,p,c[x]) =_{ind(\mathcal{R})} motH(s,q,c[x])$*

Respectivement, s'il existe des motifs bas de t et s pour p et q et pour tous les éléments non-constants de l'ensemble test $TS(\mathcal{R})$, alors

– *Si $\forall c[x] \in TS(\mathcal{R})$. $motB(t,p,c[x]) =_{ind(\mathcal{R})} motB(s,q,c[x])$, alors $t =_{ind(\mathcal{R})} s$*

si et seulement si *$t[\perp_{type(t)}]_p =_{ind(\mathcal{R})} s[\perp_{type(t)}]_q$*

– *Si* $t[\bot_{type(t)}]_p =_{ind(\mathcal{R})} s[\bot_{type(t)}]_q$, *alors* $t =_{ind(\mathcal{R})} s$

si et seulement si $\forall c[x] \in TS(\mathcal{R})$, $motB(t,p,c[x]) =_{ind(\mathcal{R})} motB(s,q,c[x])$

Démonstration. Soit θ une substitution close, comme $t/p =_{ind(\mathcal{R})} s/q$, il existe un terme clos en forme normal A tel que

$$A = fn((t/p)\theta) = fn((s/q)\theta)$$

De par les propriétés de la substitution,

$$fn(t\theta) = fn(t[fn((t/p)\theta)]_p\theta) = fn(t[A]_p\theta)$$
$$fn(s\theta) = fn(s[fn((s/q)\theta)]_q\theta) = fn(s[A]_q\theta)$$

• Tentons d'établir la preuve de $t =_{ind(\mathcal{R})} s$.

Par récurrence sur la structure du terme clos A :

– Soit $A = \bot$, alors si $t[\bot]_p =_{ind(\mathcal{R})} s[\bot]_q$,

$$fn(t[\bot]_p\theta) = fn(s[\bot]_q\theta)$$

– Soit $A = c[A']$. De par la définition 6.4 des motifs, s'il existe des motifs haut de t et s pour p et q et pour tous les éléments non-constants de l'ensemble test, alors pour chacun

$$fn(t\theta) = fn(t[c[A']]_p\theta) = fn(motH(t,p,c[x])\theta) \otimes fn(t[A']_p\theta)$$
$$fn(s\theta) = fn(s[c[A']]_q\theta) = fn(motH(s,q,c[x])\theta) \otimes fn(s[A']_q\theta)$$

Car x est une nouvelle variable et peut donc être substituée par A'.

Or, si $motH(t,p,c[x]) =_{ind(\mathcal{R})} motH(s,p,c[x])$,

$$fn(motH(t,p,c[x])\theta) = fn(motH(s,p,c[x])\theta)$$

De plus, par hypothèse de récurrence $fn(t[A']_p\theta) = fn(s[A']_q\theta)$, et donc d'après le lemme 2.1,

$$fn(t[c[A']]_p\theta) = fn(s[c[A']]_q\theta)$$

• De ce raisonnement, on peut tirer les déductions suivantes :

– Si $t[\bot]_p =_{ind(\mathcal{R})} s[\bot]_q$ et si pour chaque élément non-constant de l'ensemble test, il existe des motifs hauts tels que $motH(t,p,c[x]) =_{ind(\mathcal{R})} motH(s,q,c[x])$, alors $t =_{ind(\mathcal{R})} s$, mais aussi,

– Si $\forall c[x] \in TS(\mathcal{R}).\ motH(t,p,c[x]) =_{ind(\mathcal{R})} motH(s,q,c[x])$, alors $t =_{ind(\mathcal{R})} s$ si $t[\bot]_p =_{ind(\mathcal{R})} s[\bot]_q$

– Si $t[\bot]_p =_{ind(\mathcal{R})} s[\bot]_q$, alors $t =_{ind(\mathcal{R})} s$ si $\forall c[x] \in TS(\mathcal{R}).\ motH(t,p,c[x]) =_{ind(\mathcal{R})} motH(s,q,c[x])$

La démonstration avec des motifs bas étant totalement symétrique, on recouvre bien tous les cas possibles du théorème 6.2. □

Remarque : L'algorithme de preuve n'utilise pas toute la puissance des théorèmes car l'algorithme ne prend en compte, dans ses tests, que l'égalité syntaxique, alors que le théorème se « contente » de l'égalité inductive. L'égalité syntaxique n'est donc pas indispensable – contrairement au théorème 5.2 portant sur les généralisations – pour établir la correction de la méthode.

Cependant, nous avons choisit de restreindre l'algorithme pour limiter l'espace de recherche. En effet, tester une validité inductive est beaucoup plus coûteux (en terme d'efficacité) et peut même faire diverger la tentative preuve.

6.3 Utilisation des partitions

Dans cette section, nous illustrerons notre algorithme de preuve sur quelques exemples considérés comme « difficiles ». puis nous analyserons les possibilités offertes par cette méthode.

Remarque : Tous les exemples du chapitre précédent peuvent bien sûr être traité par cette méthode. L'inverse n'est pas toujours possible.

6.3.1 Exemples

Le système de réécriture utilisé dans chacun des exemples est le système

$$\mathcal{R} = \left\{ \begin{array}{rlrl} x + s(y) & \to s(x + y) & x + 0 & \to x \\ x * s(y) & \to (x * y) + x & x * 0 & \to 0 \\ exp(x,s(y)) & \to exp(x,y) * x & exp(x,0) & \to s(0) \\ m(x,s(y)) & \to x + m(x,y) & m(x,0) & \to 0 \\ \Sigma(s(x)) & \to s(x) + \Sigma(x) & \Sigma(0) & \to 0 \\ \Sigma I(s(x),y) & \to \Sigma I(x,s(y + x)) & \Sigma I(0,y) & \to y \\ r(c.l) & \to ap(r(l),c.\varnothing) & r(\varnothing) & \to \varnothing \\ ap(c.l,L) & \to c.ap(l,L) & ap(\varnothing,L) & \to L \\ R(c.l,L) & \to R(l,c.L) & R(\varnothing,L) & \to L \end{array} \right. \tag{6.3.1}$$

dont la table des arguments libres se trouve page 79.

- ***Exemple* 6.3.1.** $\mathbf{t = s}$: $\mathbf{r(ap(l,l)) = ap(r(l),r(l))}$

Nous avons $CB(t) = 11$ et $CH(s) = 11$, ainsi que les partitions :

– $(fn(nbt(t,11)),bot(t,11)) = (fn(r(ap(\varnothing,l))),r(l)) = (r(l),r(l))$

– $(top(s,1),fn(ntp(s,1))) = (r(l),fn(r(ap(\varnothing,l)))) = (r(l),r(l))$

Donc, $\mathbf{inductif(}r(ap(l,l)) = ap(r(l),r(l))\mathbf{)} = \mathbf{inductif(}r(l) = r(l)\mathbf{)} =$ VRAI.

- ***Exemple* 6.3.2.** $\mathbf{r(r(ap(l,l)))) = ap(l,l)}$

Nous avons $CI(t) = \{\varepsilon,1,11,111\}$ et $CI(s) = \{\varepsilon,1\}$, ainsi que les termes :

– $fn(t[c'.l']_{11}) = fn(r(r(c'.l'))) = r(ap(r(l'),c'.\varnothing))$ et

– $fn(s[c'.l']_{\varepsilon}) = fn(c'.l') = c'.l'$

dont les partitions sont :

– $(fn(nbt(fn(t[c'.l']_{11}),11)),bot(fn(t[c'.l']_{11}),11)) = (c'.\varnothing,r(r(l')))$ et

– $(fn(nbt(fn(s[c'.l']_{\varepsilon}),2)),bot(fn(s[c'.l']_{\varepsilon}),2)) = (c'.\varnothing,l')$

Alors, $motH(t,11,c'.l') = motH(s,\varepsilon,c'.l') = c'.\varnothing$ et donc
inductif$(r(r(ap(l,l))) = ap(l,l))$ = **inductif**$(r(r(\varnothing)) = \varnothing)$ = VRAI (on a alors $\varnothing \equiv \varnothing$).

• ***Exemple* 6.3.3.** $\mathbf{R(l,\varnothing) = r(l)}$

Nous avons $CI(t) = \{\varepsilon,1\}$ et $CI(s) = \{\varepsilon,1\}$, ainsi que les termes :

– $fn(t[c.l']_1) = fn(R(c.l',\varnothing)) = R(l',c.\varnothing)$ et

– $fn(s[c.l']_1) = fn(r(c.l')) = ap(r(l'),c.\varnothing)$

dont les partitions sont :

– $(fn(nbt(fn(t[c.l']_1),2)),bot(fn(t[c.l']_1),2)) = (R(l',\varnothing),c.\varnothing)$ et

– $(top(fn(s[c.l']_1),1),fn(ntp(fn(s[c.l']_1),1))) = (r(l'),ap(\varnothing,c.\varnothing)) = (r(l'),c.\varnothing)$

Alors, $motB(t,c.l') = motB(s,c.l') = c.\varnothing$ et donc **inductif**$(R(l,\varnothing) = r(l))$ = **inductif**$(R(\varnothing,\varnothing) = r(\varnothing))$ = VRAI (on a alors $\varnothing \equiv \varnothing$).

• ***Exemple* 6.3.4.** $\mathbf{x + (x + x) = (x + x) + x}$

Nous avons $CH(t) = 22$ et $CH(s) = 2$, ainsi que les partitions :

– $(top(t,22),fn(ntp(t,22))) = (x,fn(x + (x + 0))) = (x,x + x)$

– $(top(s,2),fn(ntp(s,2))) = (x,fn((x + x) + 0)) = (x,x + x)$

Donc, **inductif**$(x + (x + x) = (x + x) + x)$ = **inductif**$(x + x = x + x)$ = VRAI.

• ***Exemple* 6.3.5.**

$$\mathbf{m(x,m(x,x + x) + x) = m(x,x) + (m(x,m(x,x)) + m(x,m(x,x)))}$$

Nous avons $CH(t) = 2122$ et $CH(s) = 2222$, ainsi que les partitions :

– $(top(t,2122),fn(ntp(t,2122))) = (m(x,m(x,x)),m(x,m(x,x) + x))$

– $(top(s,22),fn(ntp(s,22))) = (m(x,m(x,x)),m(x,x) + m(x,m(x,x)))$

Donc **inductif**$(m(x,m(x,x + x) + x) = m(x,x) + (m(x,m(x,x)) + m(x,m(x,x))))$ = **inductif**$(m(x,m(x,x) + x)) = m(x,x) + m(x,m(x,x)))$.

• Nous avons maintenant $CH(t) = 212$ et $CH(s) = 222$, ainsi que les partitions :

– $(top(t,212), fn(ntp(t,212))) = (m(x,m(x,x)), m(x,0+x))$

– $(top(s,2), fn(ntp(s,2))) = (m(x,m(x,x)), m(x,x))$

Donc **inductif(**$m(x,m(x,x)+x)) = m(x,x) + m(x,m(x,x))$**)** = **inductif(**$m(x,0+x) = m(x,x)$**)**.

• Nous avons maintenant $CB(t) = 22$ et $CB(s) = 2$, ainsi que les partitions :

– $(fn(nbt(t,22)), bot(t,22)) = (fn(m(x,0+0)), m(x,x)) = (0, m(x,x))$

– $(fn(nbt(s,2)), bot(s,2)) = (fn(m(x,0)), m(x,x)) = (0, m(x,x))$

Donc **inductif(**$m(x,0+x) = m(x,x)$**)** = **inductif(**$m(x,x) = m(x,x)$**)** = VRAI.

• ***Exemple* 6.3.6.** $(\mathbf{x} * \mathbf{x}) * (\mathbf{x} * \mathbf{x}) = \mathbf{x} * ((\mathbf{x} * \mathbf{x}) * \mathbf{x})$

Nous avons $CI(t) = \{\varepsilon, 2, 22\}$ et $CI(s) = \{\varepsilon, 2, 22, 222\}$, ainsi que les termes :

– $fn(t[s(y)]_{22}) = fn((x*x)*(x*s(y))) = (x*x)*((x*y)+x)$ et

– $fn(s[s(y)]_{222}) = fn(x*((x*x)*s(y))) = x*((x*x)*y+(x*x))$

dont les partitions sont :

– $(top(fn(t[s(y)]_{22}),22), fn(ntp(fn(t[s(y)]_{22}),22))) = ((x*x)*x, (x*x)*(x*y))$,

– $(top(fn(s[s(y)]_{222}),22), fn(ntp(fn(s[s(y)]_{222}),22))) = (x*(x*x), x*((x*x)*y))$

Alors, $motH(t,22,s(y)) = (x*x)*x$ et $motH(s,222,s(y)) = x*(x*x)$. Or, pour les cas de base :

– $fn(t[0]_{22}) = fn((x*x)*(x*0)) = 0$ et

– $fn(s[0]_{222}) = fn(x*((x*x)*0)) = 0$

Donc **inductif(**$(x*x)*(x*x) = x*((x*x)*x)$**)** = **inductif(**$(x*x)*x = x*(x*x)$**)**.

• ***Exemple* 6.3.7.** $(\mathbf{x} * \mathbf{x}) * \mathbf{x} = \mathbf{x} * (\mathbf{x} * \mathbf{x})$

Nous avons $CI(t) = \{2\}$ et $CI(s) = \{22\}$, ainsi que les termes :

– $fn(t[s(y)]_2) = fn((x*x)*s(y)) = ((x*x)*y)+(x*x)$ et

– $fn(s[s(y)]_{22}) = fn(x*(x*s(y))) = x*((x*y)+x)$

dont les partitions sont :

– $(top(fn(t[s(y)]_2),2),fn(ntp(fn(t[s(y)]_2),2))) = (x * x,(x * x) * y)$ et

– $(top(fn(s[s(y)]_{22}),22),fn(ntp(fn(s[s(y)]_{22}),22))) = (x * x,x * (x * y))$

Alors, $motH(t,2,s(y)) = motH(s,22,s(y)) = x * x$ et donc
inductif$((x * x) * x = x * (x * x))$ = **inductif**$((x * x) * 0 = x * (x * 0))$ = VRAI (on a alors $0 \equiv 0$).

• ***Exemple* 6.3.8. $\mathbf{\Sigma(x) = \Sigma I(x,0)}$**

Nous avons $CI(t) = \{\varepsilon,1\}$ et $CI(s) = \{\varepsilon,1\}$, ainsi que les termes :

– $fn(t[s(y)]_1) = fn(\Sigma(s(y))) = s(y) + \Sigma(y)$ et

– $fn(s[s(y)]_1) = fn(\Sigma I(s(y),0)) = \Sigma I(y,s(0 + y))$

dont les partitions sont :

– $(top(fn(t[s(y)]_1),2),fn(ntp(fn(t[s(y)]_1),2)) = (\Sigma(y),s(y))$ et

– $(fn(nbt(fn(s[s(y)]_1),211)),bot(fn(s[s(y)]_1),211)) = (\Sigma I(y,0),s(0 + y))$

Alors, $motB(t,s(y)) = s(y)$ et $motB(s,s(y)) = s(0 + y)$. Or, pour les cas de base :

– $fn(t[0]_1) = fn(\Sigma(0)) = 0$ et

– $fn(s[0]_1) = fn(\Sigma I(0)) = 0$

Donc **inductif**$(\Sigma(x) = \Sigma I(x,0))$ = **inductif**$(s(y) = s(0 + y))$.

• Nous avons maintenant $CH(t) = 1$ et $CH(t) = 12$, ainsi que les partitions :

– $(top(t,1),fn(ntp(t,1))) = (s(y),0)$

– $(top(s,12),fn(ntp(s,12))) = (s(y),fn(0 + 0)) = (s(y),0)$

Donc **inductif**$(s(y) = s(0 + y))$ = **inductif**$(0 = 0)$ = VRAI.

• ***Exemple* 6.3.9. $\mathbf{exp(x,y + z) = exp(x,y) * exp(x,z)}$**

Comme il n'y a ni partition ni motif utilisable, on a donc
inductif$(exp(x,y + z) = exp(x,y) * exp(x,z))$ =

inductif(GEN($exp(x,y+z) = exp(x,y) * exp(x,z)$**))** [1] $=$

inductif($(exp(x,y) * exp(x,z')) * x = exp(x,y) * (exp(x,z') * x)$**)**

• Nous avons alors $CI(t) = \{\varepsilon,2,22\}$ et $CI(s) = \{\varepsilon,2,22\}$, ainsi que les termes :

– $fn(t[s(y)]_{22}) = ((exp(x,y) * exp(x,z')) * y) + (exp(x,y) * exp(x,z'))$ et

– $fn(s[s(y)]_{22}) = exp(x,y) * ((exp(x,z') * y) + exp(x,z'))$

dont les partitions sont :

– $(top(fn(t[s(y)]_{22}),2),fn(ntp(fn(t[s(y)]_{22}),2))) =$
$(exp(x,y) * exp(x,z'),(exp(x,y) * exp(x,z')) * y)$ et

– $(top(fn(s[s(y)]_{22}),22),fn(ntp(fn(s[s(y)]_{22}),22))) =$
$(exp(x,y) * exp(x,z'),exp(x,y) * (exp(x,z') * y))$

Alors, $motH(t,s(y)) = motH(s,s(y)) = exp(x,y) * exp(x,z')$ et donc

inductif($(exp(x,y) * exp(x,z')) * x = exp(x,y) * (exp(x,z') * x)$**)** $=$

inductif($(exp(x,y) * exp(x,z')) * 0 = exp(x,y) * (exp(x,z') * 0)$**)** $=$ VRAI (on a alors $0 \equiv 0$).

• ***Exemple* 6.3.10.** $(\mathbf{x} * \mathbf{y}) * \mathbf{exp}(\mathbf{x},\mathbf{y}) = \mathbf{y} * (\mathbf{x} * \mathbf{exp}(\mathbf{x},\mathbf{y}))$

Nous avons $CI(t) = \{\varepsilon,2,22\}$ et $CI(s) = \{\varepsilon,2,22,222\}$, ainsi que les termes :

– $fn(t[s(z)]_2) = fn((x * y) * s(z)) = ((x * y) * z) + (x * y)$ et

– $fn(s[s(z)]_{22}) = fn(y * (x * s(z))) = y * ((x * z) + x)$

dont les partitions sont :

– $(top(fn(t[s(z)]_2),2),fn(ntp(fn(t[s(z)]_2),2))) = (x * y,(x * y) * z)$ et

– $(top(fn(s[s(z)]_{22}),22),fn(ntp(fn(s[s(z)]_{22}),22))) = (y * x,y * (x * z))$

Alors, $motH(t,2,s(y)) = x * y$ et $motH(s,22,s(y)) = y * x$. Or, pour les cas de base :

– $fn(t[0]_{22}) = fn((x * y) * 0) = 0$ et

– $fn(s[0]_{222}) = fn(y * (x * 0)) = 0$

1. Le cas de base $exp(x,y+0) = exp(x,y) * exp(x,0)$ se réécrit en $exp(x,y) = exp(x,y)$.

Donc **inductif**$((x * y) * exp(x,y) = y * (x * exp(x,y)))$ = **inductif**$(x * y = y * x)$.

• ***Exemple* 6.3.11. x * y = y * x**

Comme il n'y a ni partition ni motif utilisable, on a **inductif**$(x * y = y * x)$ = **inductif(GEN**$(x * y = y * x)$**)** = **inductif**$((y' * x) + x = s(y') * x)$

De plus, on a **inductif**$((y' * x) + x = s(y') * x)$ = **inductif(GEN**$((y' * x) + x = s(y') * x)$**)** = **inductif**$(s(((y' * x') + y') + x') = s(((y' * x') + x') + y'))$

• Nous avons maintenant $CH(t) = 12$ et $CH(s) = 12$, ainsi que les partitions :

– $(top(t,1), fn(ntp(t,1))) = (((y' * x') + y') + x', s(0))$

– $(top(s,1), fn(ntp(s,1))) = (((y' * x') + x') + y', s(0))$

Donc **inductif**$(s(((y' * x') + y') + x') = s(((y' * x') + x') + y'))$ = **inductif**$(((y' * x') + y') + x' = ((y' * x') + x') + y')$.

• Nous avons maintenant $CB(t) = 112$ et $CB(s) = 112$, ainsi que les partitions :

– $(fn(nbt(t,11)), bot(t,11)) = (y' * x', (0 + y') + x')$

– $(fn(nbt(s,11)), bot(s,11)) = (y' * x', (0 + x') + y')$

Donc, **inductif**$(((y' * x') + y') + x' = ((y' * x') + x') + y')$ = **inductif**$((0 + y') + x' = (0 + x') + y')$.

• Comme il n'y a ni partition ni motif utilisable, on a **inductif**$((0 + y') + x' = (0+x')+y')$ = **inductif(GEN**$((0+y')+x' = (0+x')+y')$**)** = **inductif**$(s((0+x'')+y') = s(0 + x'') + y')$.

• Nous avons alors $CI(t) = \{\varepsilon, 1, 12\}$ et $CI(s) = \{\varepsilon, 1\}$, ainsi que les termes :

– $fn(t[s(y'')]_{12} = fn(s((0 + x'') + s(y''))) = s(s((0 + x'') + y''))$ et

– $fn(s[s(y'')]_1 = fn(s(0 + x'') + s(y'')) = s(s(0 + x'') + y'')$

dont les partitions sont :

– $(fn(nbt(fn(t[s(y'')]_{12}),1)), bot(fn(t[s(y'')]_{12}),1)) = (s(0), s((0 + x'') + y''))$

– $(fn(nbt(fn(s[s(y'')]_1),1)), bot(fn(s[s(y'')]_1),1)) = (s(0), s(0 + x'') + y'')$

Alors $motH(t,12,s(y'')) = motH(s,1,s(y'')) = s(0)$ et donc **inductif(**$s((0+x'')+y') = s(0+x'')+y'$**)** = **inductif(**$s((0+x'')+0) = s(0+x'')+0$**)** = VRAI (on a alors $s(0+x'') \equiv s(0+x'')$).

6.3.2 Analyse des capacités de la méthode

Cette méthode de preuve par induction est générale (dans le cadre des systèmes monomorphiques). En effet nous avons exposé un nombre important de fonction possédant un type de position permettant un découpage : DP, TP, CP, ou SP (voir sections 6.1.2, 4.1.2 et ci-dessus).

De plus, cette méthode permet de prouver des théorèmes inductifs même quand les fonctions présentes n'ont aucun type de position exploitable (voir exemples 6.3.8 et 6.3.9). Elle est donc capable, en combinaison ou non avec une méthode de preuve par induction « classique », d'achever totalement automatiquement la preuve de théorèmes réellement non-triviaux.

Cependant, on peut lui reprocher son caractère « peu naturel » car elle ne ressemble à aucune autre méthode de preuve. Il n'est en effet pas *directement* fait appel aux axiomes mais à une interprétation de l'effet de ceux-ci sur les formes normales des instances closes.

Troisième partie

Travaux Annexes

Chapitre 7

Heuristique de généralisation pour les théories conditionnelles et ses applications

Dans ce chapitre, nous allons proposer un heuristique de généralisation dans le cas des théories conditionnelles. Pour cela, nous allons dans un premier temps introduire les notions relatives à la logique conditionnelle. Après avoir décrit l'algorithme de généralisation, nous l'appliquerons à une série d'exemples. Nous exposerons notamment un exemple de preuve de contrainte d'intégrité de base de données formalisée par le calcul de situation.

Comme nous l'avons vu dans la section 1.3, tous les heuristiques existant de généralisation et de découverte de lemme [BW93, BSvH+93, Wal94a, IB96b, IB96a, KS96] travaillent sur des conjectures non-conditionnelles. De plus, notre approche, comme nos méthodes correctes, propose une généralisation *juste avant une induction* et ne nécessite donc pas de commencer une divergence pour l'analyser et la corriger.

7.1 Introduction

Dans cette section, nous apportons les notions nécessaires à la preuve par induction dans les théories conditionnelles. Après avoir défini la notion de théorie conditionnelle nous exposerons des règles de transitions formalisant l'induction dans de telles théories.

7.1.1 Définitions

Les théories conditionnelles sont formées par un ensemble d'axiomes sous la forme d'équations conditionnelles et traitent sur des clauses.

Définition 7.1. Une *équation conditionnelle* noté $e_1 \wedge \ldots \wedge e_n \implies e$ est formée par

– la conjonction des équations $e_1, \ldots, e_n$ appelées *prémisses* et

– une équation e appelée *conclusion*

Définition 7.2. Une *clause*, notée $\neg e_1 \vee \ldots \vee \neg e_n \vee e'_1 \vee \ldots \vee e'_m$, est formée par la disjonction des équations *négatives* $e_1, \ldots, e_n$ et *positives* $e'_1, \ldots, e'_m$.

Remarque : Une clause peut aussi se noter sous la forme d'une *clause de Horn*: $e_1 \wedge \ldots \wedge e_n \implies e'_1 \vee \ldots \vee e'_m$.

Une clause ϕ est une *conséquence d'inductive* d'un ensemble $\mathcal{A}$ d'axiomes si elle est valide dans le modèle initial ([Pad88]); ceci est dénoté $\mathcal{A} \models_{ind} \phi$. Les conséquences inductives sont reliées aux *conséquences déductive* de la manière suivante :

$$\mathcal{A} \models_{ind} \neg e_1 \vee \ldots \vee \neg e_n \vee e'_1 \vee \ldots \vee e'_m$$

si et seulement si pour toute substitution clause θ

$$\forall i \leq n \text{ t.q. } \mathcal{A} \models e_i\theta \text{ implique } \exists i \leq m \text{ t.q. } \mathcal{A} \models e'_j\theta$$

7.1.2 Règles de transition pour les théories conditionnelles

L'ensemble initial $\mathcal{A}$ d'équations conditionnelles doit d'abord être orienté, à l'aide un ordre bien-fondé $\succ$, en un système de réécriture $\mathcal{R}$.

Par exemple, prenons la théorie $\mathcal{A}$ suivante :

$$\mathcal{A} = \left\{ \begin{array}{lcl} in(c,L) = true & \Longrightarrow & sub(c.l,L) = sub(l,L) \\ in(c,L) = false & \Longrightarrow & sub(c.l,L) = false \\ & & sub(\varnothing,L) = true \\ eq(x,c) = true & \Longrightarrow & in(x,c.l) = true \\ eq(x,c) = false & \Longrightarrow & in(x,c.l) = in(x,l) \\ & & in(x,\varnothing) = false \\ in(c,l) = true & \Longrightarrow & dif(c.l) = false \\ in(c,l) = false & \Longrightarrow & dif(c.l) = dif(l) \\ & & dif(\varnothing) = true \\ & & eq(x,x) = true \\ true = false & \Longrightarrow & \end{array} \right. \tag{7.1.1}$$

En comparant les termes dans l'ordre lexicographique généré par la précédence

$$sub >_p dif >_p in >_p eq >_p . >_p \varnothing$$

On obtient un ordre lexicographique monotonique et bien fondé $\succ$ sur les termes. Par exemple, $sub(c.l,L) \succ sub(l,L)$ et $sub(c.l,L) \succ in(c,L)$ et ceci même quand les termes ont probablement la même sémantique ([Der87]). Ainsi tous les axiomes de $\mathcal{A}$ peuvent être orientés de gauche à droite.

La procédure d'induction, dans le sens de [BKR95, BR95, KR95], est formalisée par un ensemble de règles d'inférence appliquées à des paires de la forme (E,H). E est un ensemble de clauses qui sont les conjectures à prouver, et H un ensemble d'hypothèses d'induction.

Nous étendons simplement la définition des variables d'induction de la section 2.1.2 pour l'adapter aux axiomatisations conditionnelles.

Définition 7.3. Pour une clause C, les *atomes* de C, sont les termes de l'ensemble

$$atomes(C) = \{t; s \mid (t = s) \in C \text{ ou } \neg(t = s) \in C\}$$

Définition 7.4. Pour une clause C, $x \in \mathcal{V}_R(C)$ est une *variable d'induction* de C si x est une variable généralisée[1] ou si pour toute variable $y \in \mathcal{V}_R(C)$

$$\sum_{t \in atomes(C)} pi(x,t) \geq \sum_{t \in atomes(C)} pi(y,t)$$

• ***Exemple* 7.1.1.** Prenons le système de réécriture $\mathcal{R}$ basé sur l'axiomatisation $\mathcal{A}$ précédente (7.1.1) ansi que la clause C

$$in(x,l) = false \wedge in(y,L) = true \implies sub(l,x.L) = false$$

Les positions d'inductions des différents symboles de fonction de $\mathcal{R}$ sont :

f	eq	in	sub	dif
$PI_{\mathcal{R}}(f)$	{}	{2}	{1}	{1}

Le total des poids d'induction, pour chaque variable de C, est

- $x : 0 + 0 + 0 = 0$
- $y : 0 + 0 + 0 = 0$
- $l : 2 + 0 + 2 = 4$
- $L : 0 + 2 + 0 = 2$

Donc la seule variable d'induction de C est donc l.

1. Nous appelons « généralisée » une variable introduite par la règle de généralisation GT (voir section 7.2.1).

7.2 Règle de généralisation

Dans cette section nous introduisons une nouvelle règle de transition : la règle « *GT* ». L'idée essentielle derrière cette règle est de proposer une forme généralisé d'une conclusion juste avant qu'une autre application de la règle de génération soit tentée et que le système ne diverge.

Comme dans une clause les occurrences du sous-terme à généraliser peuvent être multiples, nous définissons, pour plus de clarté le « remplacement » des occurrences d'un sous-terme par un autre dans une clause.

Définition 7.5. Le *remplacement* de toutes les occurrences d'un sous-terme a par un terme b dans une clause C se note $C[a \mapsto b]$.

7.2.1 Transformation Généralisée

Cette règle généralise des conjectures en abstrayant un sous-terme commun non-trivial dans une position suffixe d'une position d'induction. Un sous-terme est *trivial* dans une clause si c'est une variable linéaire (c'est-à-dire dont au plus une seule occurrence se trouve dans chacun des côtés d'une équation).

La règle GT fonctionne en deux étapes :

1. Elle « Transforme » un des littéraux positifs d'une clause B en une équation dont les deux côtés partagent un sous-terme non-trivial commun.
2. Puis elle « Généralise » l'équation obtenue en remplaçant le sous-terme commun par une nouvelle variable.

Pour créer le sous-terme commun la règle GT utilise l'hypothèse d'induction A de la conjecture B. Il existe deux raisons principales d'utiliser la clause A. Premièrement, A est « similaire » à B dans le sens où certains sous-termes de A peuvent rester inchangés lors du processus de simplification. Deuxièmement, plusieurs applications

de la règle de génération créent souvent le même motif de divergence. La définition ci-dessous formalise cette idée :

Définition 7.6. Soit $A = P \vee (l = r)$ une clause où x est une variable d'induction pour A. Soit $B = Q \vee (a[s]_p = b)$ une clause dérivée de A. Si

1. s est le plus grand sous-terme non-trivial de B à une position p suffixe d'une position d'induction de B.
2. il existe une substitution θ t.q. $l\theta \equiv d[s]_q \neq a[s]_p$, $r\theta \equiv b$ et $P\theta$ subsume Q.
3. $A \models_{ind} Q[s \mapsto \bot] \vee (a[\bot]_p = d[\bot]_q)$ pour tous les termes clos $\bot$ de $TS(\mathcal{R})$ de même type que s.

Alors $C = \neg Q[s \mapsto x] \vee (a[x]_p/u = d[x]_q/u)$ où

– x est une nouvelle variable, dite *variable généralisée*,

– $u < p$ la plus longue position t.q. $a[]_u \equiv d[]_u$ et $\mathcal{V}_R(a[]_u) \cap \mathcal{V}_R(C) = \varnothing$,

est une *transformation généralisée* de B.

Définition 7.7.

$$\mathbf{GT}: (E \cup \{C\}, H) \vdash_\iota (E \cup \{T\}, H)$$

Si T est une transformation généralisée de C.

Comme nous l'avons précisé plus haut, la règle GT fonctionne en deux étapes. La première étape construit une équation $(a[s]_p = d[s]_q)$ dont les deux côtés partagent un sous-terme non-trivial commun s. Ceci grâce aux trois conditions de la définition.

– La condition 1. identifie un sous-terme non-trivial dans B.

– La condition 2. « rapproche » l'hypothèse A de la conjecture B à l'aide de la substitution θ.

– La condition 3. filtre les clauses obtenues à travers un ensemble représentatif – les termes clos – du l'ensemble test pour prémunir contre une sur-généralisation non valide.

La deuxième étape généralise le plus largement possible l'équation obtenue. En effet prenons une clause $C' = P \vee (c[t]_u = c[s]_u)$ telle que $vars(c[]_u) \cap vars(P \vee (t = s)) = \varnothing$, alors $\mathcal{A} \models_{ind} C'$ si $\mathcal{A} \models_{ind} P \vee (t = s)$[2].

Remarque : Nous ne précisons pas comment retrouver l'hypothèse d'induction d'une conjecture donnée. Cependant, dans l'implantation logiciel d'un système de preuve il est aisé de garder un lien vers l'hypothèse d'induction depuis une conjecture.

Exemple détaillé

Prenons la théorie conditionnelle $\mathcal{A}$ suivante définissant le tri par insertion ($is(l)$) ainsi que la fonction $perm(l,L)$ indiquant si une liste l est la permutation d'une liste L.

$$\mathcal{A} = \left\{ \begin{array}{lcl} & & is(c.l) = ins(c,is(l)) \\ & & is(\varnothing) = \varnothing \\ in(c,L) = true & \Longrightarrow & perm(c.l,L) = perm(l,del(c,L)) \\ in(c,L) = false & \Longrightarrow & perm(c.l,L) = false \\ & & perm(\varnothing,c.l) = false \\ & & perm(\varnothing,\varnothing) = true \\ true = false & \Longrightarrow & \end{array} \right. \tag{7.2.1}$$

L'ensemble test de $\mathcal{A}$ est $\{c.l, \varnothing\}$. Soit la conjecture $A = P \vee (l = r)$:

$$perm(l,is(l)) = true$$

Une induction sur A donne $perm(c.l',is(c.l')) = true$, et après simplification la conjecture $B = Q \vee a[s]_p = b$

$$in(c,ins(c,is(l'))) = false \vee perm(l',del(c,ins(c,is(l')))) = true$$

Nous avons :

1. La position $p = 222$, avec $a[s]_{222} = perm(l',del(c,ins(c,is(l'))))$ et $s = is(l')$.

2. Notons que s'il y a pas de contexte $c[]_u$ commun, la position u peut être la position ε.

2. La substitution θ de l par l' avec $d[is(l')]_2 = perm(l',is(l')) = l\theta$, $r\theta = b = true$, et $P\theta$ (qui est vide) subsumant Q.
3. $\mathcal{A} \models_{ind} Q[is(l') \mapsto \varnothing] \vee (a[\varnothing]_{222} = d[\varnothing]_2)$ [3].

Si l'on prend la position $u = 2$, nous obtenons la transformation généralisée C

$$in(c,ins(c,X)) = false \vee del(c,ins(c,X)) = X$$

en effet, $vars(perm(l',)) \cap vars(C) = \varnothing$.

Remarque : Cette généralisation permet de réécrire le terme $perm(l',del(c,ins(c,is(l'))))$ en $perm(l',is(l'))$, et ainsi d'utiliser l'hypothèse d'induction pour finir la preuve avec la tautologie $true = true$. Avec celle de l'exemple 7.2.3, cette preuve forme une vérification de la correction du tri par insertion.

7.2.2 Exemples

Prenons la théorie conditionnelle $\mathcal{A}$ (7.1.1) de la section 7.1.2.

- ***Exemple* 7.2.1. dif(l) = true $\Longrightarrow$ sub(l,l) = true**

Après la substitution test de l par $x.l'$, la seule conjecture obtenue après simplification est : $in(x,l') = true \vee dif(l') = false \vee sub(l',x.l') = true$

La règle GT propose alors la clause

$$in(x,Z) = true \vee dif(Z) = false \vee sub(Z,x.l') = sub(Z,l')$$

3. Car $in(c,ins(c,\varnothing)) = false \vee perm(l',del(c,ins(c,\varnothing))) = perm(l',\varnothing)$ se simplifie par $\mathcal{R}$ en $true = false \vee perm(l',\varnothing) = perm(l',\varnothing)$

Ajoutons les axiomes suivants à $\mathcal{A}$:

$$\left\{\begin{array}{lcl} x+s(y)=s(x+y) & | & x+0=x \\ x*s(y)=x*y+x & | & x*0=0 \\ eq(s(x),s(y))=eq(x,y) & | & eq(0,0)=true \\ eq(0,s(y))=false & | & eq(s(x),0)=false \\ x<y=true & \Longrightarrow & min(x.y.l)=min(x.l) \\ x<y=false & \Longrightarrow & min(x.y.l)=min(y.l) \\ & & min(x.\varnothing)=x \end{array}\right. \tag{7.2.2}$$

- ***Exemple* 7.2.2. in(min(l),l) = true**

Après la substitution test de l par $x.y.l'$, la seule conjecture obtenue après simplification est : $min(y.l') < x = false \vee in(min(y.l'),x.y.l') = true$

La règle GT propose alors la clause

$$Z < x = false \vee in(Z,x.y.l') = in(Z,y.l')$$

Ajoutons maintenant les axiomes de l'exemple détaillé ainsi que les axiomes suivants :

$$\left\{\begin{array}{lcl} c<x=true & \Longrightarrow & ins(x,c.l)=c.ins(x,l) \\ c<x=false & \Longrightarrow & ins(x,c.l)=x.c.l \\ & & ins(x,\varnothing)=x.\varnothing \\ eq(x,c)=true & \Longrightarrow & del(x,c.l)=l \\ eq(x,c)=false & \Longrightarrow & del(x,c.l)=c.del(x,l) \\ c<x=true & \Longrightarrow & sorted(c.x.l)=sorted(x.l) \\ c<x=false & \Longrightarrow & sorted(c.x.l)=false \\ & & sorted(x.\varnothing)=true \\ & & sorted(\varnothing)=true \end{array}\right. \tag{7.2.3}$$

- ***Exemple* 7.2.3. sorted(is(l)) = true**

Après la substitution test de l par $c.l'$, la seule conjecture obtenue après simplification est : $sorted(ins(c,is(l))) = true$

La règle GT propose alors la clause

$$sorted(ins(c,Z)) = sorted(Z)$$

- ***Exemple* 7.2.4. in(x,ins(x,is(l))) = true**

Après la substitution test de l par $c.l'$, la seule conjecture obtenue après simplification est : $in(x,ins(x,ins(c,is(l)))) = true$

La règle GT propose alors la clause

$$in(x,ins(x,ins(c,Z))) = in(x,ins(x,Z))$$

7.3 Preuve de contrainte d'intégrité

Dans cette section, nous montrons comment notre méthode s'applique à la preuve d'une contrainte d'intégrité d'une base de données relationnelles. La spécification logique de la base de données est formalisée par le calcul de situation. Nous commençons par décrire comment une telle spécification peut être traduite dans le contexte de la logique équationnelle. Ensuite nous montrerons comment une telle preuve peut être totalement automatisée par nos techniques. Une telle approche ressemble à celle de [BPS+96], mais ici notre but est de montrer que l'on peut prouver une contrainte d'intégrité non triviale sans aucune interaction humaine.

7.3.1 Spécification logique d'une base de données relationnelle

Le calcul de situation proposé par McCarthy en 1969 ([MH69]) se donne pour objectif de donner une représentation logique d'un monde réel ou informatique. Intuitivement, le monde étudié n'est pas décrit dans sa globalité mais il représente chaque état de ce monde par une « *situation* ». A partir d'une situation initiale, on évolue vers d'autres grâce à des « *actions* »[4]. Le but de cette représentation est d'inférer

4. Plus précisément, une situation est la résultante d'une action et d'une situation précédente.

des formules logiques valides dans toutes les situations, ces formules représentent des notions de possibilité, causalité ou connaissance du monde étudié.

Le représentation que nous utilisons pour décrire une base de données est celle proposé par Reiter dans [Rei91, Rei93a, Rei93b]. Cette représentation est issue des travaux pour les systèmes dynamiques pour l'intelligence artificielle. Chaque état successif d'une base de données est une situation, chaque transaction sur la base est une action et les relations sont représentée par des prédicats.

L'axiomatisation spécifiant une base de données dynamique contient alors les informations suivantes :

- Des connaissances indépendantes des états de la base sur les objets que manipule cette base,
- Des connaissances sur l'état de la base dans sa situation initiale,
- Des pré-conditions sur les différentes transactions affectant la base de données, et
- Les effets de ces transactions sous la formes de prédicats dont les valeurs de vérité changent suivant les transactions qui ont amené à l'état courant.

Les *contraintes d'intégrité* qui sont propriétés vraies quelles que soient les transactions affectant la base sont alors naturellement représentées par des formules logiques universellement quantifiées sur les situations. Le besoin de raisonner formellement sur les contraintes d'intégrité à l'aide de l'induction mathématique est donc directement issu de la définition récursive des situations.

Types et constructeurs

La base de données que nous nous proposons d'axiomatiser est une version simplifié de la base de donnée dynamique éducationnelle de [Rei93a]. Les types « primitifs » présents dans cette axiomatisation sont *situation* (état), *action* (transaction), *student* (étudiant), *course* (cours) et *boolean* (booléen).

Les constructeurs du type *situation* sont :

1. $S0$: l'état initial de la base,
2. $do(act,s)$: la résultante d'une transaction act sur un état s de la base.

Les constructeurs du type *action* représentent les transactions qui affectent la base.

1. $register(etu,c)$: Inscription de l'étudiant etu au cours c.
2. $drop(etu,c)$: Désinscription de l'étudiant etu au cours c.

Les constructeurs du type *boolean* sont $true$ et $false$.

$$true = false \implies \tag{7.3.1}$$

Remarque : Les types *student* et *course* n'ont pas de constructeurs propres, car ces constructeurs sont inutiles dans notre étude mais peuvent très bien être définis si-besoin. Nous regrouperons donc ces deux types sous un seul pour plus de lisibilité des domaines (voir plus bas).

Fonctions définies

Cette base implique deux relations, une dépendante de l'état de la base, et une indépendante :

1. $enrolled(etu,c,s)$: L'étudiant etu est inscrit au cours c quand la base est dans l'état s.
2. $prerequ(pre,c)$: Le cours pre est un pré-requis du cours c.

Les prédicats dépendant de l'état de la base sont définis par des « *axiomes d'état suivant* ». Par exemple, le prédicat *enrolled* est défini par l'axiome implicitement universellement quantifié signifiant « *Un étudiant est inscrit à un cours si et seulement s'il y a été inscrit par la dernière transaction ou s'il y était déjà inscrit dans l'état*

précédent et s'il n'y a pas été désinscrit par la dernière transaction ».

$$enrolled(etu,co,do(a,s)) \equiv (a = register(etu,co)) \vee (enrolled(etu,do,s) \wedge (a \neq drop(etu,co)))$$

Axiome qui peut être traduit par :

$$\begin{aligned}
& enrolled(etu,co,S0) = false \\
eqAct(a,register(etu,co)) = true \implies & enrolled(etu,co,do(a,s)) = true \\
eqAct(a,register(etu,co)) = false \wedge & \; eqAct(a,drop(etu,co)) = true \\
\implies & enrolled(etu,co,do(a,s)) = true \\
eqAct(a,register(etu,co)) = false \wedge & \; eqAct(a,drop(etu,co)) = false \\
\implies & enrolled(etu,co,do(a,s)) = enrolled(etu,co,s)
\end{aligned} \tag{7.3.2}$$

Ici, *eqAct* exprime la relation d'égalité sur les transaction (voir [Llo87] p. 79).

$$\begin{aligned}
& eqAct(register(x,c),drop(y,d)) = false \\
& eqAct(drop(x,c),register(y,d)) = false \\
x \neq y \implies & eqAct(register(x,c),register(y,d)) = false \\
x = y \wedge c \neq d \implies & eqAct(register(x,c),register(y,d)) = false \\
x = y \wedge c = d \implies & eqAct(register(x,c),register(y,d)) = true \\
x \neq y \implies & eqAct(drop(x,c),drop(y,d)) = false \\
x = y \wedge c \neq d \implies & eqAct(drop(x,c),drop(y,d)) = false \\
x = y \wedge c = d \implies & eqAct(drop(x,c),drop(y,d)) = true
\end{aligned} \tag{7.3.3}$$

Comme nous l'avons vu ci-dessus, les transactions ont des pré-conditions qui doivent être satisfaites avant que la transaction affecte la base. Ces pré-conditions sont représentées par deux fonctions *poss* et *reach* à valeur booléenne. $poss(act,s)$

indique si une transaction *act* est possible dans l'état s et $reach(s)$ indique si l'état s est atteignable (c'est-à-dire s'il est la résultante uniquement d'une série d'actions possibles). Ceci donne les axiomes suivants :

$$\begin{aligned} reach(S0) &= true \\ poss(act,s) = true &\implies reach(do(act,s)) = reach(s) \\ poss(act,s) = false &\implies reach(do(act,s)) = false \end{aligned} \tag{7.3.4}$$

Pour plus de lisibilité et de simplicité nous assumons qu'il est impossible de désinscrire un étudiant. La pré-condition de l'action *drop* s'exprime donc ainsi :

$$poss(drop(etu,co),s) = false \tag{7.3.5}$$

Traduction des quantificateurs

Les formules logiques utilisées par Reiter aussi bien les pré-conditions et les effets des transactions que les contraintes d'intégrité peuvent être quantifiés. Par exemple, voici la pré-condition « *On peut inscrire un étudiant à un cours si et seulement s'il est inscrit à tout les pré-requis du cours* » :

$$poss(register(etu,co),s) \equiv \forall co'(prerequ(co',co) \implies enrolled(etu,co'))$$

Pour traduire cette formule logique, nous introduisons la notion de *quantificateur borné* développé par Boyer et Moore dans [BM79]. Ceci est possible car il existe un nombre fini d'objets présents dans la base dans chaque état de la base de données. On introduit donc une fonction $dom(s)$ représentant le *domaine* des objets présents dans la base à l'état s. Cette fonction est du type *list* dont les constructeurs sont . et $\varnothing$. Le premier argument du constructeur . est du type unifié $course/student$[5].

5. Si l'on désire séparer les deux types on peut introduire un type abstrait T dont les constructeurs sont alors $st(etu)$ et $ct(co)$ prenant un étudiant ou un cours pour former un objet de type T.

$$
\begin{aligned}
dom(S0) &= \varnothing \\
dom(do(register(etu,co),s)) &= etu.(co.dom(s)) \\
dom(do(drop(etu,co),s)) &= etu.(co.dom(s))
\end{aligned}
\tag{7.3.6}
$$

Remarque : Le domaine de l'état initial est ici vide mais il pourrait contenir tous les cours définis.

Nous pouvons alors définir la pré-condition de l'action *register* à l'aide du quantificateur borné *ball* :

$$
poss(register(etu,co),s) = ball(etu,co,s,dom(s)) \tag{7.3.7}
$$

dont la définition est :

$$
\begin{aligned}
& ball(etu,co,s,\varnothing) = true \\
prerequ(x,co) = false \implies & ball(etu,co,s,x.d) = ball(etu,co,s,d) \\
prerequ(x,co) = true \wedge enrolled(etu,x,s) = false \implies & \\
& ball(etu,co,s,x.d) = false \\
prerequ(x,co) = true \wedge enrolled(etu,x,s) = true \implies & \\
& ball(etu,co,s,x.d) = ball(etu,co,s,d)
\end{aligned}
\tag{7.3.8}
$$

7.3.2 Preuve de la contrainte d'intégrité

Toutes les définitions ci-dessus (de 7.3.1 à 7.3.8) peuvent être regroupées en un système de réécriture $\mathcal{R}$ orienté par l'ordre lexicographique de simplification basé sur la précédence

$$
\begin{aligned}
reach \; > poss > ball > enrolled > dom > eqAct > prerequ & \\
> do > S0 > register > drop > . > \varnothing > true > false &
\end{aligned}
$$

L'ensemble test de ce système est

$$TS(\mathcal{R}) = \{true,\ false,\ do(register(x,y),s),\ do(drop(x,y),s),\ \varnothing,\ c.l\}$$

Les positions d'inductions de fonctions définies du système de réécriture $\mathcal{R}$ sont :

f	$reach$	$poss$	$enrolled$	$prerequ$	$eqAct$	dom	$ball$
$PI_{\mathcal{R}}(f)$	{1}	{1}	{3}	{}	{1,2}	{1}	{4}

La contrainte d'intégrité dont nous allons dérouler la preuve signifie : « *Pour tout état atteignable de la base, si un étudiant est inscrit à un cours alors il est inscrit à tous les pré-requis de ce cours* ». Cette contrainte peut se formuler ainsi :

$$\begin{aligned} reach(s) = true \wedge enrolled(etu,co) = true \wedge prerequ(co',co) = true \\ \Longrightarrow\ enrolled(etu,co',s) = true \end{aligned} \tag{7.3.9}$$

En utilisant la règle d'inférence « *complément* »[6] on obtient :

$$\begin{aligned} reach(s) = false \vee enrolled(etu,co,s) = false \vee prerequ(co',co) = false \\ \vee\ enrolled(etu,co',s) = true \end{aligned}$$

La seule variable se trouvant en position d'induction est s. On effectue donc une induction sur s, ce qui donne :

$$\begin{aligned} &reach(S0) = false \vee enrolled(etu,co,S0) = false \vee prerequ(co',co) = false \\ &\quad \vee enrolled(etu,co',S0) = true \\ &reach(do(register(x,y),s')) = false \vee enrolled(etu,co,do(register(x,y),s')) = false \\ &\quad \vee prerequ(co',co) = false \vee enrolled(etu,co',do(register(x,y),s')) = true \\ &reach(do(drop(x,y),s')) = false \vee enrolled(etu,co,do(drop(x,y),s')) = false \\ &\quad \vee prerequ(co',co) = false \vee enrolled(etu,co',do(drop(x,y),s')) = true \end{aligned}$$

La première et la troisième conjecture sont simplifiées en tautologies respectivement à l'aide des définitions de $enrolled$ (7.3.2) et de $reach$ et $poss$ (7.3.4 et 7.3.5).

6. Voir définition formelle des règles d'inférence dans [BKR95].

La deuxième conjecture est simplifiée par cas en[7] :

$$reach(s') = false \vee enrolled(etu,co',s') = false \vee prerequ(co',co) = false$$
$$\vee ball(etu,co,s',dom(s')) = false$$

Ici, la seule variable d'induction est s'. Les conjectures générées par les éléments de l'ensemble test $S0$ et $do(drop(x',y'),s'')$ sont encore simplifiées en tautologies. La conclusion d'induction générée à partir de $do(resister(x',y'),s'')$ est simplifiée en :

$$reach(s'') = false \vee enrolled(etu,co',s'') = false \vee prerequ(co',co) = false \vee$$
$$ball(etu,co,do(register(x',y'),s''),dom(s'')) = false \vee prerequ(y',co) = true$$

Nous appliquons maintenant la règle GT qui construit l'équation

$$ball(etu,co,do(register(x',y'),s''),dom(s'')) = ball(etu,co,s'',dom(s''))$$

Le sous-terme commun est $dom(s'')$ et la clause proposée est donc :

$$reach(s'') = false \vee enrolled(etu,co',s'') = false \vee prerequ(co',co) = false \vee$$
$$ball(etu,co,do(register(x',y'),s''),Z) = ball(etu,co,s'',Z) \vee prerequ(y',co) = true$$

Cette clause peut être orientée pour former la règle de réécriture

$$reach(s'') = false \wedge enrolled(etu,co',s'') = false \wedge prerequ(co',co) = false \wedge$$
$$prerequ(y',co) = true \implies ball(etu,co,do(register(x',y'),s''),Z) \rightarrow ball(etu,co,s'',Z) \quad (7.3.10)$$

La variable d'induction de cette clause est Z car c'est une variable généralisée (voir définition 7.4). Les conclusions d'induction sont alors

$$reach(s'') = false \vee enrolled(etu,co',s'') = false \vee prerequ(co',co) = false \vee$$
$$ball(etu,co,do(register(x',y'),s''),\varnothing) = ball(etu,co,s'',\varnothing) \vee prerequ(y',co) = true$$
$$reach(s'') = false \vee enrolled(etu,co',s'') = false \vee prerequ(co',co) = false \vee$$
$$ball(etu,co,do(register(x',y'),s''),c.l) = ball(etu,co,s'',c.l) \vee prerequ(y',co) = true$$

7. Les tautologies résultant des ces simplifications par cas ayant été supprimées.

Ces équations sont simplifiées en tautologies grâce à la définition de *ball* (7.3.8) et à l'hypothèse d'induction (7.3.10).

Chapitre 8

NICE : Un système de preuve

Dans ce chapitre, nous allons décrire le logiciel NICE (“*Nice Induction for Conditional Equations*”). Ce logiciel implante les méthodes et les heuristiques présentés dans ce manuscrit. La méthode de preuve par induction utilisée est l'induction implicite dans le sens de [BKR95, BR95]. Ce système présente donc des similitudes avec le logiciel SPIKE ([BKR92]).

Notre but en développant ce logiciel, n'était pas de rajouter un système de preuve parmi les relativement nombreux existants, mais de tester nos méthodes. Néanmoins, ce logiciel c'est révélé au fur et à mesure de son évolution, satisfaisant du point de vue de son interface et de son efficacité. Nous avons donc choisi de le présenter en fin de ce manuscrit.

Dans un premier temps, nous retracerons les raisons qui nous ont amené à concevoir ce logiciel. Nous exposerons ensuite l'ensemble de ces fonctionnalités. Enfin, nous montrerons quelques « captures d'écran » retraçant le déroulement de preuves.

En annexe de ce manuscrit, le lecteur trouvera quelques spécifications (c'est à dire des axiomatisations) écrites pour NICE et quelques « traces » de preuves réalisées.

8.1 Historique

En 1999, une première version purement textuelle de ce logiciel a été implanté en langage `C`. Ce langage de programmation fut choisi pour des raisons d'efficacité. Nous y avons intégré les heuristiques de généralisation et de choix des variables d'induction (voir sections 2.1.2 et 7.2.1). Nous n'avons pas choisit de modifier un logiciel de preuve existant car nous trouvions ces logiciels trop lents. De plus, nous avons trouvé leurs « codes »[1] trop obscures à appréhender car relativement peu commentées.

Pour permettre une meilleur interaction avec l'utilisateur nous avons décidé d'ajouter une interface graphique. Comme la construction d'une telle interface est une chose peu aisée et peu portable en `C`, nous somme passé au langage `JAVA`. En effet, ce langage possède comme atouts une plus grande facilité de développement des interfaces graphiques et une meilleure portabilité entre les différents systèmes d'exploitation. De plus, une documentation précise et agréable de la bibliothèque sur laquelle est basé l'implémentation peut être générée grâce à l'outil `javadoc`.

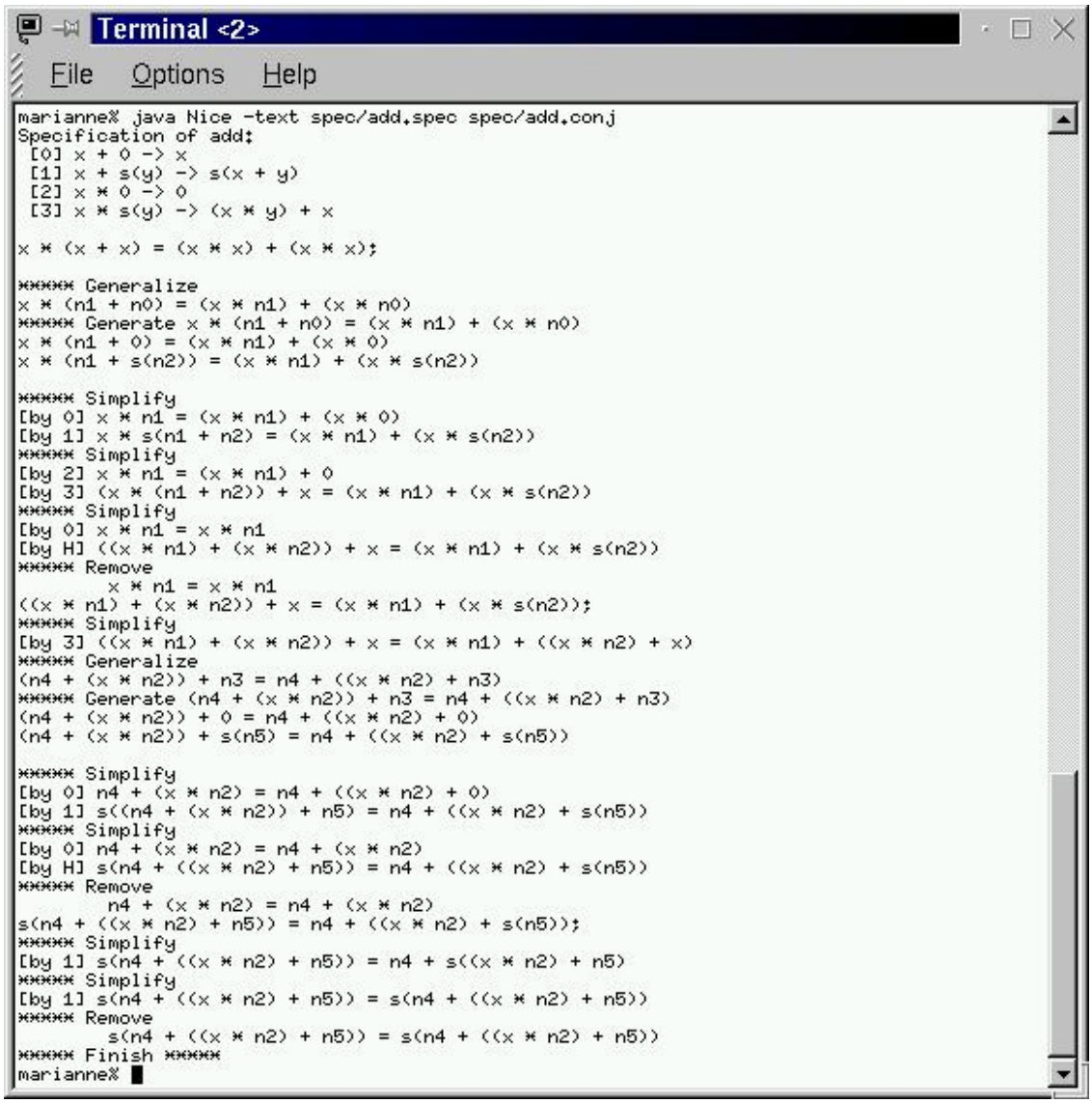
Dans cette version, nous avons implanté toutes les définitions et notions présentes dans ce manuscrit. `NICE` est en autres, capable de vérifier l'orientation d'un système de réécriture ainsi que reconnaître automatiquement si un système donné est monomorphique. Il calcule les positions d'induction des fonctions apparaissant dans un système quelconque et applique l'heuristique de sélection des variables d'induction. Il utilise soit l'algorithme calculant des généralisations correctes soit l'heuristique du chapitre 7 selon le type de système de réécriture. Il implante aussi l'algorithme basé sur la partition de termes tel qu'il se trouve à la section 6.2.1. Il contient à l'heure actuelle, 46 classes et près de 8000 lignes de code (commentaires inclus) et est disponible à l'addresse web :

`http://www-sop.inria.fr/coprin/urso`

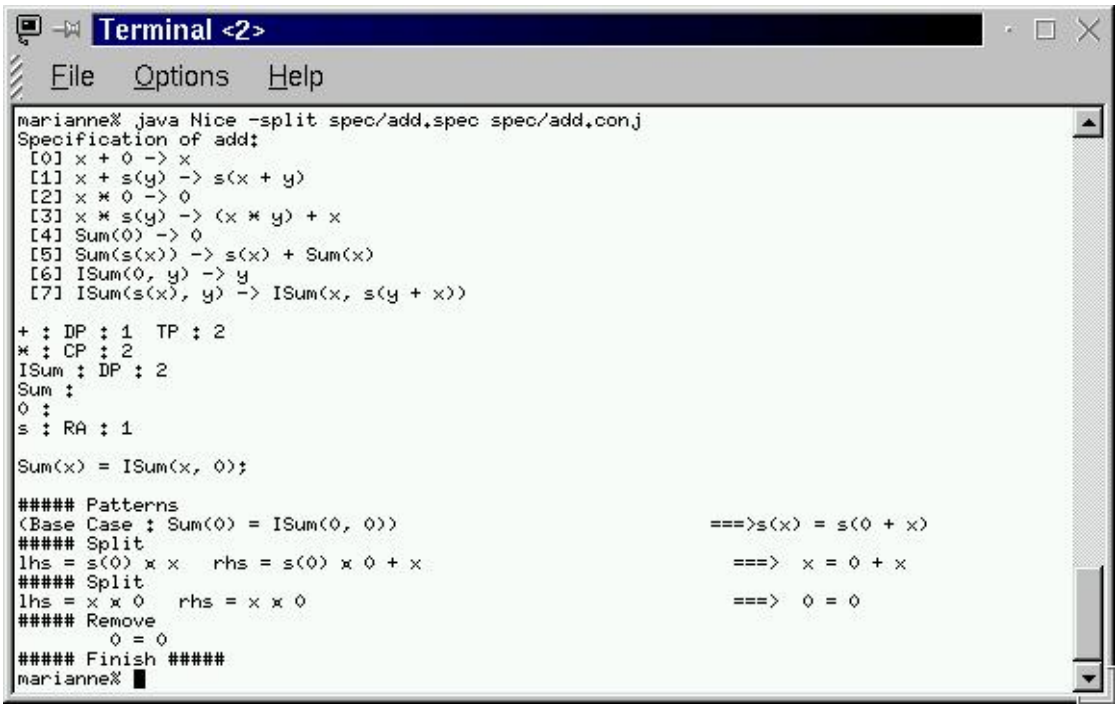
1. Particulièrement celui en `lisp` de `RRL` et dans une moindre mesure, celui en `caml` de `SPIKE`

8.2 Fonctionnalités

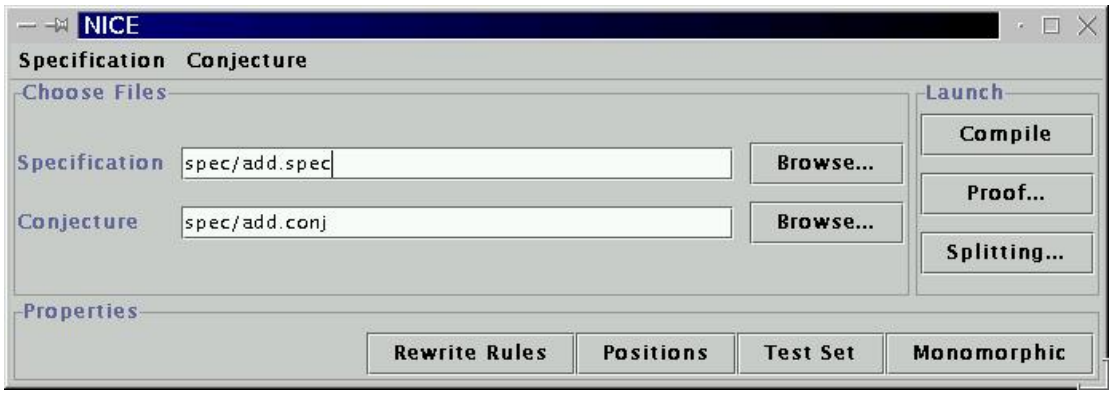
Le logiciel NICE se présente sous deux formes. La première, purement textuelle, est appelée depuis la ligne de commande et lance directement la preuve d'une (ou de plusieurs) conjecture(s) donnée(s).



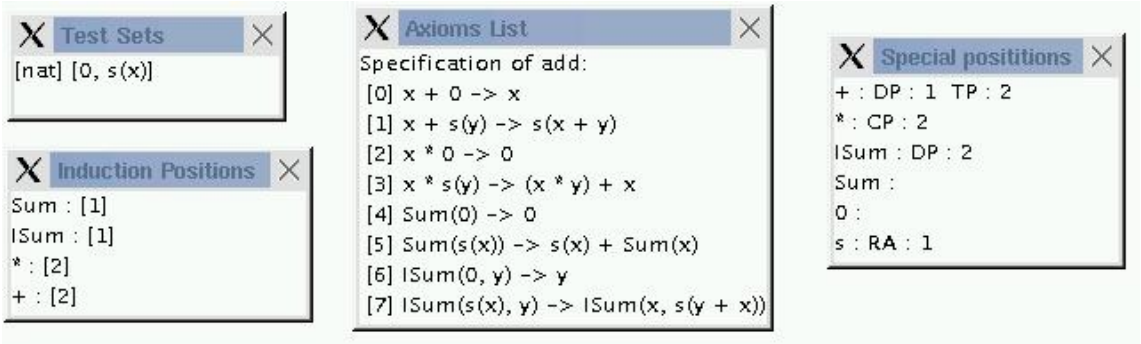
Cette forme permet aussi l'utilisation de l'algorithme de partition de termes. Les types de positions d'arguments des différentes fonctions sont affichées puis, si le système est monomorphique, la preuve est lancée.



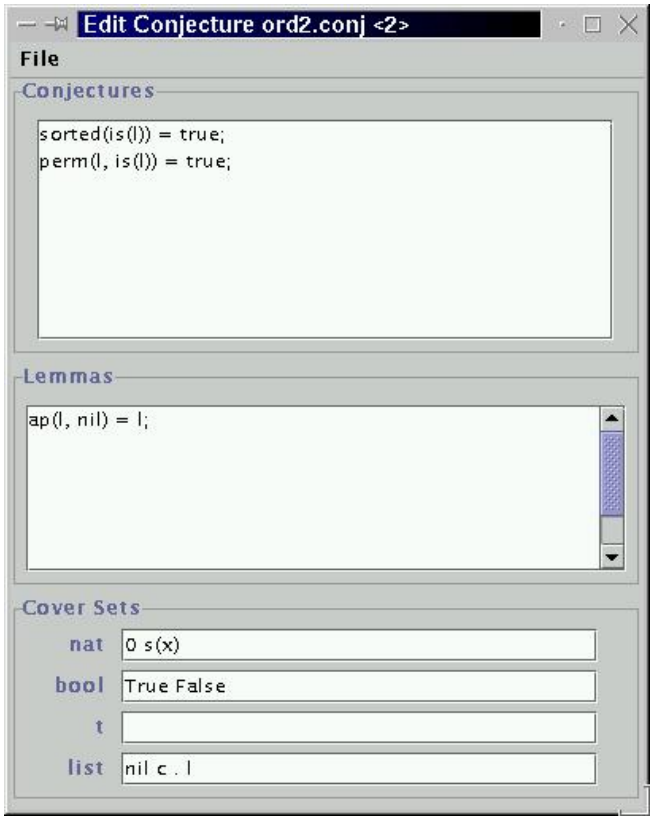
Sous sa forme graphique, le logiciel `NICE` présente d'abord une fenêtre permettant le choix des fichiers contenant la spécification et les conjectures ainsi que l'appel des différentes composantes du logiciel.



Chaque bouton situé en bas de la fenêtre décrit une propriété de la spécification calculé automatiquement par `NICE`.



Le logiciel propose une édition interactive des spécifications et surtout des conjectures sous forme d'un éditeur spécifique.

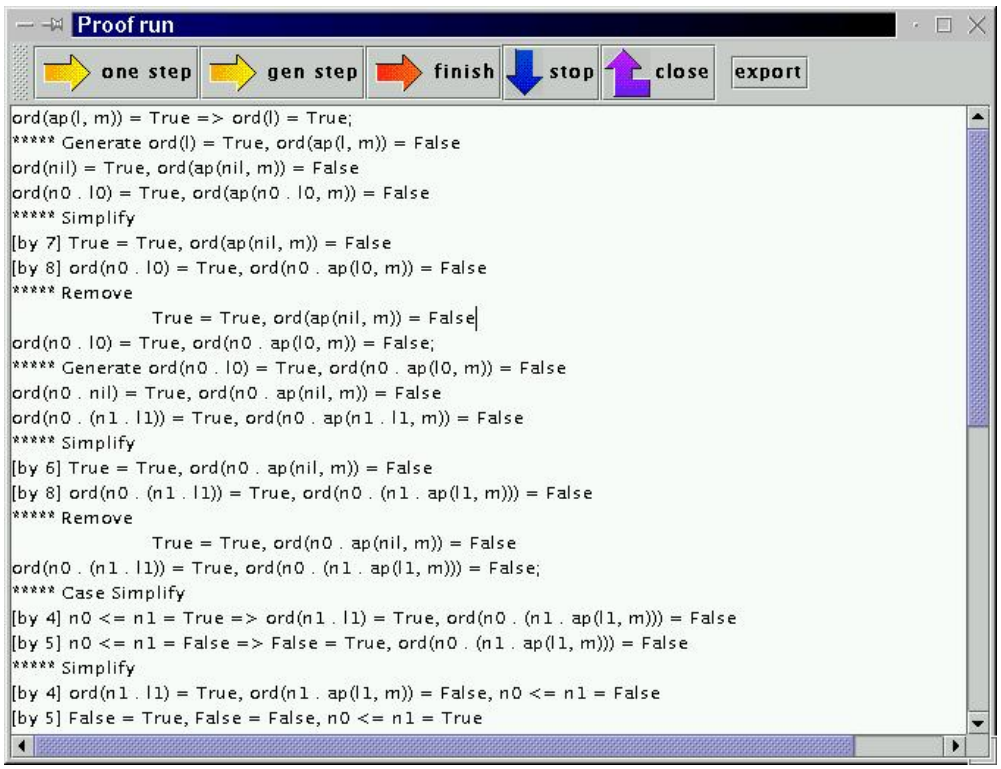


Cet éditeur permet aussi de sélectionner les lemmes qui seront pris en compte lors de la tentative de preuve. L'utilisateur peut choisir des ensembles couvrants pour chaque type présent dans la spécification, mais si il ne le fait pas, le logiciel calculera automatiquement des ensembles tests.

Remarque : Les fichiers contenant les spécifications ou les conjectures sont des fichiers de textes manipulables aussi par les éditeurs de fichiers classiques.

8.3 Déroulement des preuves

Les tentatives de preuves effectuées grâce à l'induction implicite avec généralisations ou grâce à l'algorithme de partition de termes se déroule dans une fenêtre particulière.



L'utilisateur peut dérouler la tentative de preuve étape par étape. Une étape est constituée d'une simplification, d'une suppression de tautologie, d'une réécriture par cas, ou d'une généralisation dans le cas de l'induction « classique ». Pour la partition de termes, une étape est soit une simplification par découpage, soit une simplification par motifs, soit une suppression de tautologie. La preuve peut aussi être déroulée jusqu'à la prochaine induction (règle de « génération ») ou directement jusqu'à la fin.

Grâce à la présence des `threads` dans la bibliothèque du langage `JAVA`, plusieurs tentatives de preuves peuvent être lancées en même temps sur plusieurs conjectures différentes.

8.4 Limitations et perspectives

Le logiciel NICE, bien que implantant complètement deux méthodes de preuve par induction (implicite et partition de termes), ne contient pas certains outils qui sont présents habituellement dans ce genre de logiciel. Notamment, aucune technique de complétion automatique (ou semi-automatique) n'est fournie pour transformer une spécification en un système convergent sur les termes clos. Il n'est pas non plus capable de vérifier si un système de réécriture est suffisamment complet et de proposer si-besoin, les règles manquantes[2]. Cependant il peut vérifier la terminaison d'un système si l'utilisateur lui fourni une précédence sur les symboles de fonctions présents dans la spécification.

La bibliothèque du logiciel NICE possède une documentation importante sous forme de pages HTML. De plus, l'ensemble de ses composants est destiné à être disponible sur internet. Nous encouragerons donc toutes suggestions d'améliorations ou contributions. Toutefois, son but n'est pas de devenir un « grand » logiciel de preuve mais juste une plate-forme de tests ouverte.

2. Contrairement aux logiciels RRL et SPIKE.

Conclusion

Tout d'abord, nous tenons à préciser que nous avons présenté les premiers travaux proposant des généralisations correctes. De plus, pour la première fois, ces généralisations portent sur des équations contenant de chaque côté, trois ou plus occurrences d'une même variable ainsi que des équations basées essentiellement sur la multiplication.

Comme nous l'avons expliqué à la fin des chapitres 5 et 6, les limitations de nos contributions principales sont dues à celles inhérentes aux systèmes monomorphiques.

Rappelons que la première caractéristique de ces systèmes est la convergence sur les termes clos et la complétude suffisante. Ces propriétés sont pourtant requises par de nombreuses méthodes de preuve (particulièrement pour assurer la complétude réfutationnelle) et sont naturelles si l'on définit une axiomatisation de manière structurelle.

La deuxième caractéristique est la limitation à un seul argument du même type pour chaque constructeur. Même si l'on peut ainsi représenter des structures telles que des listes d'entiers, on ne peut pas appréhender les structures en graphes. Cependant, pour autant que nous le sachions, aucun heuristique n'a proposé des généralisations portant sur des graphes non-triviaux.

La troisième caractéristique est la limitation à une seule constante par type. Nous croyons que cette limitation est la plus « gênante », car elle nous empêche de

manipuler des types booléens et des prédicats. En effet, il est difficile d'étudier la structure des formes normales des instances closes quand celles-ci ne peuvent être que $vrai$ ou $faux$. Dans ce cas, seuls les heuristiques présentés dans la section 1.3 et celui proposé dans le chapitre 7, se révèlent efficaces.

Le première proposition visant à étendre nos travaux est donc de dépasser ces limitations, peut-être en s'inspirant des notions de co-réductibilité (voir, par exemple, [Kou92]). Nous pensons que l'unicité de la position contenant la constante dans un terme clos est moins important que l'unicité de la forme normale des termes non-clos.

Nous proposons aussi d'étendre le nombre de types d'arguments libres et la définition de ceux existants. En effet, de nombreuses définitions de fonctions (exp, $double$, $length$, ...) ont un argument qui est libre mais qui n'est pas formellement identifié par nos définitions. Nous avons intentionnellement limité ces définitions pour assurer une meilleure lisibilité des définitions et des preuves.

La dernière proposition est plus théorique. Il s'agit d'étudier formellement les limites des généralisations correctes. Il conviendrait d'essayer de comprendre ce qui peut être appris, en avance, sur les formes normales des instances closes des termes à partir des seuls axiomes.

Bibliographie

[Aub79] R. Aubin, *Mechanizing structural induction*, Theorical Computer Science **9** (1979), 329–362.

[Bac88] L. Bachmair, *Proof by consistency in equational theories*, Proceedings of Third IEEE LICS, 1988.

[Bac91] ———, *Canonical equational proof*, Birkhäuser, Boston, 1991.

[BJ97] A. Bouhoula and J.P. Jouannaud, *Automata-driven automated induction*, Proceedings of Twelfth IEEE LICS, 1997.

[BKR92] A. Bouhoula, E. Kounalis, and M. Rusinowitch, *Spike: an automatic theorem prover*, Proceedings of LPAR'92, 1992.

[BKR95] ———, *Automated mathematical induction*, Journal of Logic and Computation **5(5)** (1995), 631–668.

[BM79] R.S. Boyer and J.S. Moore, *A computational logic*, Academic Press, NY, 1979.

[BM88] ———, *A computational logic handbook*, Perspectives in Computing, vol. 23, Academic Press, NY, 1988.

[BPS+96] L. Bertossi, J. Pinto, P. Saez, D. Kapur, and M. Subramaniam, *Automated proofs of integrity constraints in situation calculus*, Proceedings of ISMIS'96, 1996.

[BR95] A. Bouhoula and M. Rusinowitch, *Implicit induction in conditional theories*, Journal of Automated Reasoning **14(2)** (1995), 189–235.

[BRH96] F. Bronsard, U. Reddy, and R. Hasker, *Induction using term orders*, Journal of Automated Reasoning **16** (1996), 3–37.

[BSvH+93] A. Bundy, A. Stevens, F. van Hermelen, A. Ireland, and A. Smail, *Rippling: A heuristic for guiding inductive proofs*, Artificial Intelligence **62** (1993), 185–253.

[Bur69] R. Burstall, *Prooving properties of programs by structural induction*, Computer Journal **12(1)** (1969), 41–48.

[BvHHS90] A. Bundy, F. van Hermelin, C. Horn, and A. Smail, *The oyster-clam system*, Proceedings of Tenth Int. CADE, 1990.

[BW92] D. Basin and T. Walsh, *Difference matching*, Proceedings of Eleventh Int. CADE, 1992.

[BW93] ———, *Difference unification*, Proceedings of Thirteenth International Joint Conference on Artificial Intelligence, 1993.

[Com00] H. Comon, *Inductionless induction*, Handbook of Theorical Computer Science, 2000.

[Der87] N. Dershowitz, *Termination of rewriting*, Journal of Symbolic Computation **3** (1987), 69–116.

[Der97] ———, *Completion and its applications*, Actes du Séminaire d'Informatique Théorique (Paris), 1997.

[DJ90] N. Dershowitz and J.P. Jouannaud, *Rewriting systems*, Handbook of Theorical Computer Science (J. van Leeuwen, ed.), vol. B, North-Holland, 1990, pp. 243–309.

[Göd31] K. Gödel, *Über formal unentscheidbare sätze der principa mathematica und verwandter system, I*, Monatsh. Math. Phys. **38** (1931), 173–198, traduction en anglais dans [vH67].

[GS92] H. Ganzinger and J. Stuber, *Inductive theorem proving by consistency for first order clause*, Proceedings of CTRS'92, 1992.

[Gut78] J. Guttag, *Abstract data types and software specification*, Communications of the ACM **21** (1978), 1048–1064.

[HH82] G. Huet and J.M. Hullot, *Proofs by induction in equational theories without constructor*, JCSS **25 (2)** (1982),??

[IB96a] A. Ireland and A. Bundy, *Extensions to a generalization critic for inductive proof*, Proceedings of Thirteenth Int. CADE, 1996.

[IB96b] ———, *Using failure to guide inductive proof*, Journal of Automated Reasoning **16** (1996), 38–85.

[JK86] J.P. Jouannaud and E. Kounalis, *Automatic proofs by induction in equational theories without constructor*, Proceedings of First IEEE LICS, 1986.

[JK89] ———, *Automatic proofs by induction in theories without constructor*, Information and Control **82(1)** (1989).

[Kap87] S. Kaplan, *Simplifying conditional term rewriting systems: Unification, termination and confluence*, Journal of Symbolic Computation **4** (1987), 295–334.

[KB73] D.E. Knuth and P.B. Bendix, *Computational problems in abstract algebra*, ed. J. Leech, 1973.

[KM87] D. Kapur and D. Musser, *Proof by consistency*, Artificial Intelligence **31(2)** (1987), 125–157.

[Kou92] E. Kounalis, *How to check for the ground-reducibility property in term rewriting systems*, TCS **106 (1)** (1992), 87–117.

[KR90] E. Kounalis and M. Rusinowitch, *Mechanizing inductive reasoning*, Proceedings of Eighth AAAI, 1990, pp. 240–245.

[KR95] ———, *Reasoning with conditional axioms*, Annals of Mathematics and Artificial Intelligence **15** (1995), 125–149.

[Kre65] G. Kreisel, *Mathematical logic*, Lectures on Modern Mathematics (T. Saady, ed.), vol. 3, J. Wiley & sons, 1965, pp. 95–195.

[KS96] D. Kapur and M. Subramaniam, *Lemma discovery in automating induction*, Proceedings of Thirteenth Int. CADE, 1996.

[KU99a] E. Kounalis and P. Urso, *Generalization discovery for proofs by induction in conditional theories*, Proceedings of FLAIRS-99 (Orlando, FL), AAAI Press, 1999.

[KU99b] ———, *Mechanizing proofs of integrity constraints in the situation calculus*, Proceedings of IEA/AIE-99 (Cairo, Egypt), Springer-Verlag, 1999.

[KZ95] D. Kapur and H. Zang, *An overview of rewrite rule laboratory (rrl)*, Journal of Computer Mathematics with Applications **29(2)** (1995), 91–114.

[Lan81] D. Lankford, *A simple explanation of inductionless induction*, Tech. Report MTP-14, Mathematics Departement, 1981.

[Llo87] J.W. Lloyd, *Foundation of logic programming*, Springer Verlag, 1987.

[MH69] J. McCarthy and P. Hayes, *Some philosophical problems from the standpoint of the artificial intelligence*, Machine Intelligence **4** (1969), 463–502.

[Mus80] D.R. Muser, *On proving inductive properties of abstract data types*, Proceedings of Seventh ACM Symp. POPL, 1980.

[Pad88] P. Padawitz, *Computing in horn clause theories*, Springer-Verlag, 1988.

[Pla85] D. Plaisted, *Semantic confluence tests and completion methods*, Information and Control **65** (1985), 182–215.

[Red90] U. Reddy, *Term rewriting induction*, Proceedings of Tenth Int. CADE, 1990.

[Rei91] R. Reiter, *The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression*, A.I. and .T.C.: Papers in Honor of J. McCarthy (1991), 359–380.

[Rei93a] ———, *Formalizing database evolution in the situation calculus*, Proceedings of Fifth Generation Computer Systems, 1993.

[Rei93b] ———, *Proving properties of states in the situation calculus*, Artificial Intelligence **64** (1993), 337–351.

[vH67] J. van Heijenoort, *From frege to goedel: a source book in mathematical logic, 1879–1931*, Harvard University Press, 1967.

[Wal94a] T. Walsh, *A divergence critic*, Proceedings of Twelfth Int. CADE, 1994.

[Wal94b] C. Walther, *Mathematical induction*, Handbook of Logic in Artificial Intelligence and Logic Programming (Dov M. Gadday, ed.), vol. 2, Oxford University Press, 1994, pp. 127–228.

[Wal96] T. Walsh, *A divergence critic for inductive proof*, Journal of Artificial Intelligence Research **4** (1996), 209–235.

[Zha88] H. Zhang, *Reduction, superposition and induction: Automated reasoning in equational logic*, Ph.D. thesis, Rensselaer Polytechnic Institute, Troy, N.Y., 1988.

[ZKK88] H. Zhang, D. Kapur, and M.S. Krishnamoorthy, *A mechanizable induction principle for equational specification*, Proceedings of Ninth Int. CADE, 1988.

Résumé

L'un des plus importants défis pour l'induction est de mécaniser le plus possible la règle-ω. En d'autres termes, un système de preuve ne peut pas être réellement automatique si plusieurs interactions humaines – telles que l'apport de lemmes, de généralisations, ou de schémas d'induction – sont nécessaires pour prouver des théorèmes (particulièrement des théorèmes *non-linéaires*). En fait, les systèmes de preuve et les contributions théoriques les plus avancées, soit ne proposent pas de règles permettant une spéculation *correcte* de lemme ou de généralisation, soit évitent l'appellation « automatique » pour préférer celle de « vérificateur de preuve ».

Dans cette thèse, nous proposons des apports aux méthodes de preuve par induction dans le sens d'un plus grande automatisation. Ces apports sont constitués de deux heuristiques efficaces et surtout de deux algorithmes corrects. Le premier heuristique introduit une nouvelle définition des *variables d'induction*. Le second est une règle de généralisation pour les théories conditionnelles . Le premier algorithme calcule des généralisations correctes pour des théories non-conditionnelles. Le second est une méthode d'induction originale – la « partition de termes » – permettant la preuve automatique de théorèmes inductifs.

Mots clés : Induction mathématique, Preuve automatique, Généralisations, Logique équationnelle.

Abstract

A major challenge for induction is to mechanize the ω-rule as much as possible. In other words, a theorem prover cannot be truly automatic if many human interactions – such as lemmas, generalizations, or induction schemes – are needed to prove theorems (particularly *non-linear* theorems) that seem trivial for any human. Actually, the most advanced induction theorem provers and theoretical contributions either do not provide *sound* rules speculating lemma or generalization or simply prefer appellation "proof-checker" instead of "automatic prover".

This thesis essentially proposes enhancements to mechanize mathematical inductive proofs methods. These enhancements are composed of two efficient heuristics and, mainly, two sound algorithms. The first heuristic provides a new definition of induction variables. The second is a generalization rule for conditional theories. The first sound algorithm computes sound generalization for unconditional theories. The second is a completely new method – "term partition" – to prove inductive theorem.

Keywords: Mathematical Induction, Automated Reasonning, Generalizations, Equational Logic.