

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/213879499>

Resolution d'équations dans les langages d'ordre 1, 2, ..., omega.

Thesis · January 1976

CITATIONS

284

READS

509

1 author:



Gérard P. Huet

National Institute for Research in Computer Science and Control

144 PUBLICATIONS 9,755 CITATIONS

SEE PROFILE

N° d'ordre

THÈSE DE DOCTORAT D'ÉTAT

présentée

A L'UNIVERSITÉ PARIS VII

par

Gérard HUET

pour obtenir

**LE GRADE DE DOCTEUR ES-SCIENCES
SPÉCIALITÉ : MATHÉMATIQUES**

RÉSOLUTION D'ÉQUATIONS DANS DES LANGAGES D'ORDRE 1,2,..., ω .

Soutenue le 28 Septembre 1976 devant la Commission d'examen composée de :

MM. L. NOLIN

Président

M. DEMAZURE

J. GIRAUD

S. GRIGORIEFF

M. NIVAT

} Examinateurs

G. PLOTKIN

Invité

N° d'ordre

THÈSE DE DOCTORAT D'ÉTAT

présentée

A L'UNIVERSITÉ PARIS VII

par

Gérard HUET

pour obtenir

**LE GRADE DE DOCTEUR ES-SCIENCES
SPÉCIALITÉ : MATHÉMATIQUES**

RÉSOLUTION D'ÉQUATIONS DANS DES LANGAGES D'ORDRE $1, 2, \dots, \omega$.

Soutenue le 28 Septembre 1976 devant la Commission d'examen composée de :

MM. L. NOLIN	Président
M. DEMAZURE	Examinateurs
J. GIRAUD	
S. GRIGORIEFF	
M. NIVAT	
G. PLOTKIN	Invité



Je tiens à exprimer toute ma reconnaissance à Monsieur le Professeur L. Nolin, qui me fait l'honneur de présider ce jury, et dont les encouragements chaleureux m'ont été précieux. Je remercie Messieurs les Professeurs M. Demazure, S. Grigorieff et G. Plotkin d'avoir bien voulu s'intéresser à ce travail. Merci également à Monsieur le Professeur J. Giraud pour m'avoir proposé un intéressant sujet de deuxième thèse.

Que Maurice Nivat trouve ici l'expression de ma profonde gratitude. Comme Directeur Scientifique, ses conseils et ses critiques m'ont toujours été précieux. Comme ami, son soutien constant est inestimable.

Ma dette est vive à l'égard de mon ami Gilles Kahn, qui a contribué directement à la recherche de certains résultats présentés dans ce travail.

Je profite de l'occasion qui m'est offerte pour remercier tous mes camarades de travail de l'IRIA pour l'ambiance de travail amicale qu'ils savent créer.

Merci enfin à Chantal Le Moal, qui a frappé cette thèse avec beaucoup de gentillesse.

TABLE DES MATIERES

INTRODUCTION	I.1	
<u>CHAPITRE 1 : Le λ-calcul typé</u>		
1.1	Types	1.1
1.2	Termes	1.2
1.3	Contextes, liberté	1.3
1.4	Egalité	1.7
1.5	β -réduction	1.9
1.6	Normalisation forte	1.11
1.7	Cohérence	1.15
1.8	Forme canonique	1.18
<u>CHAPITRE 2 : Substitutions solutions d'équations</u>		
2.1	Substitutions	2.1
2.2	Filtrage	2.8
2.3	Unification	2.11
2.4	Le préordre \leq_V	2.14
2.5	Ensembles complets d'unificateurs	2.18
2.6	Finitude des ensembles d'unificateurs	2.22
2.7	Redondance des ensembles d'unificateurs	2.23
2.8	Indécidabilité de l'unification	2.32
2.9	Ensembles complets de filtres	2.36
2.10	Unification multiple	2.40
<u>CHAPITRE 3 : Semi-algorithmes d'unification à l'ordre ω.</u>		
3.1	Définitions et notations	3.1
3.2	Algorithme de simplification	3.7
3.3	Algorithme de choix	3.11
3.4	Arbres de pré-unification	3.19
3.5	Arbres de pré-unification indépendants	3.29
3.6	Arbres d'unifiabilité	3.38
3.7	Cas particuliers	3.49

CHAPITRE 4 : Extension au λ - η -calcul

4.1	Forme extensionnelle	4.1
4.2	Substitutions extensionnelles	4.8
4.3	Unification extensionnelle	4.8
4.4	Arbres de pré- η -unification	4.9
4.5	L'algorithme de Jensen-Pietrzykowski	4.11

CHAPITRE 5 : Unification dans les langages de premier ordre

5.1	Termes de premier ordre	5.1
5.2	Substitutions, filtrage	5.3
5.3	Inf-demi-treillis bien fondés	5.10
5.4	\hat{T} est un ITF	5.12
5.5	Unification dans \hat{T}	5.24
5.6	Congruences rationnelles	5.35
5.7	Un algorithme d'unification rapide	5.43
5.8	Unification sur les arbres	5.52
5.9	Unification linéaire	5.67

CHAPITRE 6 : Unification dans les langages de second ordre

6.1	Définitions générales	6.2
6.2	Termes linéaires	6.8
6.3	Termes monadiques	6.22
6.4	Unification dans \bar{T}	6.30
6.5	Conclusion	6.37

CONCLUSION

1	Résultats obtenus, problèmes ouverts	C.1
2	Applications	C.4

BIBLIOGRAPHIE

INTRODUCTION

Nous étudions dans ce travail le problème de la résolution de systèmes d'équations entre termes, dans des langages typés d'ordre un et plus.

Résoudre une équation $E = E'$, où E et E' sont des termes, c'est trouver une substitution σ de termes pour les variables libres de E et E' , qui rende identiques les termes substitués σE et $\sigma E'$. Une telle solution σ s'appelle un unificateur de E et E' .

Nous nous intéresserons également à l'ensemble des substitutions σ qui identifient un terme E à un terme E' , c'est-à-dire telles que $\sigma E = E'$. On dira alors que σ est un filtre de E vers E' .

Pour résoudre de telles équations, nous aurons besoin d'étudier la structure de l'ensemble T des termes, muni du préordre de filtrage défini par :

$$E \leq E' \Leftrightarrow \exists \sigma \quad \sigma E = E'.$$

Le préordre \leq s'étend par extensionnalité à l'ensemble des substitutions. De manière équivalente :

$$\sigma \leq \rho \Leftrightarrow \exists \eta \quad \rho = \eta \sigma$$

Autrement dit, si σ et ρ sont des solutions à une équation aux termes et $\sigma \leq \rho$, alors ρ découle de σ par composition avec η : la solution ρ est moins générale que la solution σ .

Nous allons nous attacher ici à l'étude des ensembles complets de solutions d'équations dans T , définis comme des ensembles de substitutions :

- 1) qui sont solution de l'équation (cohérence)
- et 2) dont toutes les solutions découlent (complétude).

Dans le cas où l'on peut imposer à l'ensemble des solutions d'être une base, c'est-à-dire si les solutions considérées sont minimales, nous parlerons d'ensemble complet de solutions minimales.

Si de plus ces solutions sont indépendantes, en ce sens qu'elles n'ont pas de majorant commun, nous dirons qu'on a un ensemble complet de solutions indépendantes. Enfin, s'il est possible de réduire un ensemble complet de solutions à un élément unique, nous dirons qu'on a une solution principale.

Nous allons étudier trois types d'équations :

- l'unification : $U = \{\sigma \mid \sigma E = \sigma E'\}$.
- la demi-unification ou filtrage : $\mathcal{D} = \{\sigma \mid \sigma E = E'\}$.
- la pré-unification : $P = \{\sigma \mid \sigma E \sim \sigma E'\}$, où \sim est une relation syntaxique entre termes.

Donnons maintenant brièvement le cadre dans lequel s'inscrit notre étude, motivée initialement par des problèmes issus de la théorie de la démonstration.

L'unification, ou identification de formules par substitution, a été étudiée pour la première fois, pour la logique de premier ordre, dans les Recherches sur la théorie de la démonstration, thèse de J. Herbrand datant de 1930. Dans ce travail, Herbrand a donné trois caractérisations équivalentes de la notion de proposition valide, qu'il a appelées les propriétés A, B et C. Les propriétés voisines B et C donnèrent naissance à ce qui a été appelé par la suite le théorème d'Herbrand, tandis que la propriété A tombait provisoirement dans l'oubli. Pour reconnaître si une proposition a la propriété A, Herbrand définit un algorithme de résolution d'égalités, qui est la première version publiée de l'algorithme d'unification de premier ordre [HJ1, page 124].

Ce n'est qu'après les années 50 que l'apparition des ordinateurs suscita tout un courant de recherches visant à la génération automatique de preuves mathématiques formelles. Bien sûr, les travaux de Church et de Gödel limitent au départ l'ambition d'une telle entreprise : on ne peut espérer définir de procédure générale décidant en un temps fini de la validité d'une proposition de premier ordre. On appelle donc système de preuve complet tout algorithme construisant une démonstration formelle de tout théorème. Les propositions non valides peuvent donner lieu soit à l'arrêt du système (une réfutation est obtenue dans ce cas), soit à des calculs infinis.

Les premiers programmes réalisés visaient à l'implémentation du théorème d'Herbrand, par génération de tronçons initiaux de l'univers d'Herbrand d'une proposition. Une telle recherche exhaustive était vouée à l'échec, et cette approche fut abandonnée.

Au début des années 60, J. Guard conçut et réalisa un système de démonstration semi-automatique, implémentant des règles d'inférences dérivées de celles de Gentzen. Divers langages étaient abordés : calcul des propositions, des prédictats, et même une théorie des types d'ordre ω [GJ1]. La principale règle d'inférence, pour le 1er ordre, faisait appel à la notion de "general matching formula", qui n'est autre que le sup (plus petit majorant) des deux formules dans l'ordre \leq . Un algorithme d'unification de premier ordre (dit "matching algorithm") était décrit correctement, et des suggestions étaient faites pour l'extension d'un tel algorithme aux ordres supérieurs. Ce travail fut largement ignoré, en dépit des succès enregistrés (le programme, avec l'aide d'un mathématicien, prouva une conjecture de théorie des treillis).

A la même époque, une équipe d'Argonne National Laboratory construisait des programmes de démonstration automatique inspirés de la propriété A de Herbrand. Leur travail culmina avec la découverte de la règle de résolution, exposée dans le papier maintenant classique

de J.A. Robinson [RJA1]. Cette règle d'inférence, à elle seule, définit un système de preuve complet pour la logique de premier ordre. L'application de la règle met en oeuvre une procédure d'unification, présentée pour la première fois sous ce nom. Cette approche suscita rapidement un grand nombre de travaux : définition d'heuristiques tendant à limiter le nombre de formules engendrées par la résolution, extension au calcul des prédictats avec égalité [RW1]. La plupart des programmes de démonstration automatique en usage à ce jour sont basés sur la règle de résolution.

L'utilisation de langages logiques d'ordre supérieur fut peu suivie, en particulier à cause de la difficulté d'étendre l'unification à de tels langages. Gould [GW1], suivant les suggestions de Guard, étudia le problème de la recherche d'instances communes à des formules de théorie des types d'ordre ω . Cette approche ne peut mener à une règle d'inférence complète, plusieurs unificateurs distincts pouvant correspondre à la même instance. De plus les algorithmes présentés sont incomplets, voire incorrect [HG2].

Le problème fut repris ensuite par T. Pietrzykowski qui définit un algorithme de recherche d'unificateurs complet pour le second-ordre. Ceci lui permit d'étendre la règle de résolution au calcul des prédictats du second ordre [PT1]. Cette méthode fut ensuite étendue aux ordres supérieurs dans [JP1]. Malheureusement la règle d'inférence utilisée fait appel à l'énumération d'un ensemble complet d'unificateurs, ce qui est non effectif, un tel ensemble étant généralement infini, et redondant, la minimalité ne pouvant être garantie, comme nous le verrons.

Dans notre thèse de Ph.D. [HG1], nous avons montré comment il était possible d'étendre la règle de résolution aux ordres supérieurs, sans pour cela générer explicitement un ensemble complet d'unificateurs. On demande simplement de pouvoir vérifier l'existence

ou l'absence de solutions. Bien que ce problème soit indécidable, comme nous le verrons, il est possible d'énumérer, sans redondance, un ensemble complet de minorants d'unificateurs, que nous appellerons pré-unificateurs: Nous décrirons de tels algorithmes au Chapitre 3, et prouverons leur correction.

Ce travail a donc été originellement motivé par des problèmes de théorie de la démonstration. Mais les méthodes utilisées sont essentiellement de nature algébrique. Nous pensons que les résultats présentés dans ce travail sont des résultats de base d'une théorie des manipulations formelles d'expressions. Nous présenterons dans la conclusion un certain nombre d'applications, en particulier à la théorie de la programmation.

Résumé

Le Chapitre 1 introduit le langage d'ordre ω , cadre général de notre travail. C'est un λ -calcul typé, avec α - et β -réductions. Nous donnons les principales propriétés de la relation de conversion, en particulier l'existence d'une forme canonique des termes. Ce chapitre ne contient pas de résultat original et peut être omis par le lecteur familier avec le λ -calcul.

Le Chapitre 2 définit les ensembles complets d'unificateurs dans ce langage, et donne les principaux résultats concernant leur structure. En particulier nous prouvons qu'il n'est pas possible de leur imposer la condition de minimalité, et que l'unifiabilité est indécidable dès l'ordre 3.

Le Chapitre 3 présente divers algorithmes permettant d'engendrer un ensemble complet de pré-unificateurs. En particulier, un algorithme énumérant un ensemble complet de solutions indépendantes permet de tester sans redondances l'unifiabilité d'un ensemble de termes. Dans le cas particulier de la recherche de demi-unificateurs, cet algorithme permet d'engendrer un ensemble complet de solutions minimales.

Le Chapitre 4 donne les modifications à apporter à ces résultats en présence de la règle η , ou extensionnalité restreinte du λ -calcul.

Le Chapitre 5 reprend l'ensemble du problème, mais dans le cas des langages de premier ordre de cette fois. Il est montré que l'ensemble des termes muni de l'ordre de filtrage, possède une structure de treillis complet sous condition. L'existence d'un unificateur principal est montrée par une nouvelle méthode, basée sur l'étude des congruences rationnelles entre termes. Cette méthode fournit un algorithme d'unification plus efficace que les algorithmes traditionnels. De plus nous montrons comment on peut étendre l'existence d'une solution principale à une équation entre termes, les solutions étant prises maintenant dans l'ensemble des termes finis ou infinis.

Cette solution si elle existe, est rationnelle, c'est-à-dire reconnaissable par un automate fini. Ce chapitre peut être lu de manière indépendante.

Le Chapitre 6 traite du cas des langages de second ordre. Deux ordres de filtrage sont étudiés, et il est montré que pour ces deux ordres la structure des termes de seond ordre est beaucoup plus complexe qu'au premier ordre, même en se limitant au cas monadique. Dans ce dernier cas, l'unifiabilité est reliée à un problème ouvert de la théorie des monoïdes. Enfin nous donnons un algorithme permettant d'engendrer un ensemble complet fini de demi-unificateurs de deux termes quelconques de second ordre.

Dans la conclusion nous présentons un certain nombre d'applications de nos résultats, principalement en démonstration automatique et en théorie de la programmation. Chaque application est illustrée par un exemple.

CHAPITRE I : Le λ -calcul typé

Ce chapitre présente succinctement les principales propriétés du λ -calcul typé, muni de la β -réduction.

1.1. Types

Notre langage est dérivé de la théorie des types de Church [CA1]. Chaque expression du langage possède un *type* unique, qui définit sa position dans une hiérarchie fonctionnelle. Pour cela, on se donne un ensemble fini T_0 de *types élémentaires*, et l'ensemble T des types est le plus petit ensemble contenant T_0 et fermé par :

$$\alpha, \beta \in T \implies (\alpha \rightarrow \beta) \in T.$$

(T est donc un langage algébrique sur l'alphabet $T_0 \cup \{(\ ,), +\}$).

Le type composé $(\alpha \rightarrow \beta)$ est le type des applications d'un ensemble d'éléments de type α dans un ensemble d'éléments de type β . Nous désignerons les types par les lettres de l'alphabet grec $\alpha, \beta, \gamma, \delta$.

Abréviation : Nous utiliserons la notation $(\alpha_1 \times \alpha_2 \times \dots \times \alpha_n \rightarrow \beta)$ avec $n \geq 1$, comme abréviation pour le type

$$(\alpha_1 \rightarrow (\alpha_2 \rightarrow \dots (\alpha_n \rightarrow \beta) \dots)).$$

1.2. Termes

Les *termes*, ou *expressions bien formées*, comprennent les *atomes*, les *applications* et les *abstractions*. Les termes sont désignés par $e, e', e_1, \dots, E, E', \dots$, et quelquefois par $M, M', \dots, N, N', \dots$. Chaque terme e est affecté d'un type unique $\tau(e)$.

1.2.1. Atomes

On se donne un ensemble dénombrable V_α de variables de type α pour tout α dans T , et un ensemble fini ou dénombrable C de constantes de types quelconques. Les ensembles V_α et C sont disjoints deux à deux. L'ensemble des *atomes* est :

$$A = V \cup C, \text{ avec } V = \bigsqcup_{\alpha \in T} V_\alpha.$$

Les variables sont dénotées par des minuscules $x, y, z, \dots, f, g, h, \dots$ et les constantes par des majuscules $A, B, C, \dots, F, G, H, \dots$. Un atome quelconque est dénoté par $\emptyset, \emptyset', \dots$.

1.2.2. Applications

Si e_1 est un terme de type $(\alpha \rightarrow \beta)$ et e_2 est un terme de type α , alors $(e_1 e_2)$ est un terme de type β .

Le terme $(e_1 e_2)$ dénote la valeur de la fonction dénotée par e_1 pour l'argument dénoté par e_2 .

1.2.3. Abstractions

Si e est un terme de type β et $x \in V_\alpha$, alors $\lambda x e$ est un terme de type $(\alpha \rightarrow \beta)$.

Le terme $\lambda x e$ sert à dénoter la fonction qui, à l'argument x , fait correspondre la valeur dénotée par e .

1.2.4. Termes, ordre

L'ensemble T des termes est donc défini comme le plus petit ensemble contenant A et fermé par les opérations d'*application* et d'*abstraction*. On désigne par T_α l'ensemble des termes de type α .

Exemple : Si $x \in V_\alpha$ et $y \in V_{(\alpha \rightarrow \alpha) \rightarrow \beta}$ alors $(\lambda y(y\lambda xx)y) \in T_\beta$.

Définissons la fonction $0 : T \rightarrow N$ par :

- si $\alpha \in T_0$ alors $0(\alpha) = 1$
- si $\alpha = (\beta \rightarrow \gamma)$ alors $0(\alpha) = \max\{0(\beta) + 1, 0(\gamma)\}$.

On appelle *ordre* d'un terme e l'entier $0(\tau(e))$. Ceci correspond à la notion usuelle d'ordre : à l'ordre 1 les termes représentent des individus, à l'ordre 2 des fonctions, à l'ordre 3 des fonctionnelles, etc...

On appelle *langage d'ordre n* un langage où l'on n'autorise que des variables d'ordre au plus n . Notre langage complet est une théorie des types d'ordre ω , c'est à dire où l'on peut définir des fonctionnelles de tout ordre fini.

1.3. Contextes, liberté

Un *contexte* est un terme dans lequel on a supprimé une occurrence de sous-terme. Les contextes vont nous servir à nommer les occurrences de sous-termes.

On dénotera par $E^{<\alpha>} , E'^{<\alpha>} , \dots$ un contexte où le sous-terme manquant est de type α . On utilise le symbole $<\alpha>$ pour marquer la place de ce sous-terme. On dénote par X_β l'ensemble des contextes de type β .

Plus précisément, on a la définition récursive :

- (i) $<\alpha> \in X_\alpha$
- (ii) $e \in T_{(\alpha \rightarrow \beta)}$ et $F^{<\gamma>} \in X_\alpha \Rightarrow (e F^{<\gamma>}) \in X_\beta$
- (iii) $E^{<\gamma>} \in X_{(\alpha \rightarrow \beta)}$ et $e \in T_\alpha \Rightarrow (E^{<\gamma>} e) \in X_\beta$
- (iv) $x \in V_\alpha$ et $E^{<\gamma>} \in X_\beta \Rightarrow \lambda x E^{<\gamma>} \in X_{(\alpha \rightarrow \beta)}$

Exemple : En reprenant l'exemple ci-dessus,

$$E^{<\alpha>} \equiv (\lambda y(y\lambda x^{<\alpha>})y) \in X_\beta.$$

(On utilise \equiv pour dénoter l'identité syntaxique des termes et des contextes).

Remarquons que dans $E^{<\alpha>}$, α dénote le type du sous-terme manquant, et non le type du contexte.

Si $E^{<\alpha>} \in X_\beta$ et $e \in T_\alpha$, alors $E^{<e>}$ dénote le terme de type β obtenu en remplaçant $<\alpha>$ par e dans $E^{<\alpha>}$. La définition récursive de la construction $E^{<e>}$ suivant (i) à (iv) de la définition de $E^{<\alpha>}$ est évidente et laissée au lecteur.

Par exemple, pour (i) on a $<e> \equiv e$.

De même si $E'^{<\gamma>} \in X_\alpha$, alors $E^{<E'<\gamma>>}$ dénote le contexte de type β obtenu en remplaçant $<\alpha>$ par $E'<\gamma>$ dans $E^{<\alpha>}$.

Exemple :

Avec $E^{<\alpha>}$ défini comme ci-dessus et $z \in V_\alpha$, alors $E^{<z>} \equiv (\lambda y(y \lambda x z) y)$.

Définitions :

$E^{<\gamma>}$ lie x si et seulement si $E^{<\gamma>} \equiv E_1^{<\lambda x E_2 <\gamma>>}$. (Cette définition s'applique même si $\tau(x) \neq \gamma$).

Soit $E \equiv E^{<x>}$. L'occurrence de x distinguée par le contexte $E^{<\gamma>}$, avec $\gamma = \tau(x)$, est dite liée dans E si $E^{<\gamma>}$ lie x , libre dans E sinon. On appelle variable libre de E toute variable qui possède au moins une occurrence libre dans E .

L'ensemble $V[E]$ des variables libres de E peut être défini récursivement par :

- (i) $V[x] = \{x\}$ pour $x \in V$, $V[A] = \emptyset$ pour $A \in C$.
- (ii) $V[e_1 e_2] = V[e_1] \cup V[e_2]$
- (iii) $V[\lambda x e] = V[e] - \{x\}$.

Soit $e \in T_\alpha$, $x \in V_\alpha$ et $E \in T$. On dénote par $S_e^x[E]$ le terme obtenu en remplaçant chaque occurrence libre de x par e dans E . C'est bien sûr un terme de type $\tau(E)$.

Plus précisément, on a la définition récursive :

- (i) $S_e^x[x] \equiv e$
- (ii) $S_e^x[@] \equiv @$ $\quad @ \in A - \{x\}$

- (iii) $S_e^x[(e_1 e_2)] \equiv (S_e^x[e_1] S_e^x[e_2])$
- (iv) $S_e^x[\lambda x e_1] \equiv \lambda x e_1$
- (v) $S_e^x[\lambda y e_1] \equiv \lambda y S_e^x[e_1] \quad \text{si } y \neq x$

Exemple : Si $E \equiv (\lambda y(y\lambda xx)y)$ et $e \equiv \lambda uv$, avec $\tau(u) = (\alpha \rightarrow \alpha)$ et $\tau(v) = \beta$, alors

$$S_e^y[E] \equiv (\lambda y(y\lambda xx)\lambda uv).$$

Lemme 1.1

$$x \notin V[e] \Rightarrow S_e^x[e] \equiv e$$

Démonstration :

Par récurrence structurelle sur e :

(i) si $e \equiv x$ alors $x \in V[e]$

(ii) si $e \equiv \emptyset \neq x$ alors $S_e^x[e] \equiv \emptyset \equiv e$

(iii) si $e \equiv (e_1 e_2)$ alors

$$x \notin V[e] \Rightarrow x \notin V[e_1] \text{ et } x \notin V[e_2]$$

$$S_e^x[e] = (S_e^x[e_1] S_e^x[e_2]) \equiv (e_1 e_2) \text{ par hypothèse de récurrence.}$$

$$\equiv e$$

(iv) si $e \equiv \lambda x e_1$ alors $S_e^x[e] \equiv e$

(v) si $e \equiv \lambda y e_1$, $y \neq x$, alors

$$x \notin V[e] \Rightarrow x \notin V[e_1]$$

$$S_e^x[e] \equiv \lambda y S_e^x[e_1] \equiv \lambda y e_1 \text{ par hypothèse de récurrence.}$$

$$\equiv e. \quad \square$$

Lemme 1.2

$$S_x^x[e] \equiv e$$

Démonstration :

Facile par récurrence structurelle sur e . \square

Remarque :

De la même manière, on peut définir l'opération S sur les contextes :

$E'<\gamma> = S_e^x[E<\gamma>]$ est défini récursivement suivant la structure de $E<\gamma>$. On peut alors prouver facilement que, pour tout terme E de type γ , on a :

$$S_e^x[E<\gamma>] = \begin{cases} E'<e> & \text{si } E<\gamma> \text{ lie } x \\ E'<S_e^x[E]> & \text{sinon.} \end{cases}$$

Remarquons que la définition de $S_e^x[e']$ permet des "captures de variables libres".

Par exemple, $S_x^y[\lambda xy] \equiv \lambda xx$.

Pour remédier à cette situation, nous définissons une relation $L(x,y,E)$:

" x est libre pour y dans E " comme suit. Si $x,y \in V_\alpha$ et $E \in T$, alors $L(x,y,E)$ si et seulement si pour tout contexte $E<>$ de E tel que $E \equiv E<\lambda ye>$, toutes les occurrences de x dans cette occurrence de e sont liées dans E .

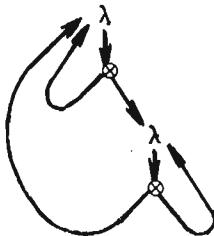
Plus précisément, si $y = x$, alors pour tout e on a $L(x,x,e)$; sinon :

- (i) $L(x,y,@) \quad \forall @ \in A$
- (ii) $L(x,y,(e_1 e_2)) \Leftrightarrow L(x,y,e_1) \text{ et } L(x,y,e_2)$
- (iii) $L(x,y,\lambda xe)$
- (iv) $L(x,y,\lambda ye) \Leftrightarrow x \notin V[e]$
- (v) $L(x,y,\lambda ze) \Leftrightarrow L(x,y,e) \quad \forall z \notin \{x,y\}$.

Si $L(x,y,E)$, alors les occurrences libres de x dans E peuvent être remplacées par y , sans que ces occurrences de y soient "capturées" par un symbole λ . Autrement dit, $S_y^x[e]$ est une opération valide lorsque $L(x,y,e)$.

1.4. Égalité

Deux termes sont dits égaux s'ils ne diffèrent que par renommage de leurs variables liées, sans capture de variable libre. Intuitivement, les variables liées sont muettes, elles ne servent qu'à marquer des positions. Notre égalité est l'identité des représentations "par pointeurs", où par exemple $\lambda x(x\lambda y(xy))$ est représenté par



Une notation similaire est suggérée par Bourbaki, avec l'assemblage $\boxed{\boxed{(\boxed{\boxed{}})}} (\boxed{\boxed{(\boxed{\boxed{}})}})$. Voir également la notation de de Bruijn [DB1] et les n-wffs de Andrews [AP1]. Nous allons maintenant définir formellement cette égalité.

Définition : α -conversion.

Soit $E \equiv E<\lambda x e>$, avec $x \in V_\gamma$. Si $y \in V_\gamma - V[e]$ est tel que $L(x,y,e)$, alors

$$E \xrightarrow{\alpha} E<\lambda y S_y^x[e]>.$$

Lemme 1.3 La relation $\xrightarrow{\alpha}$ est symétrique.

Démonstration :

Soit $E \equiv E<\lambda x e>$, avec $x \in V_\gamma$, et $y \in V_\gamma - V[e]$, tel que $L(x,y,e)$.

Soit $E' \equiv E<\lambda y S_y^x[e]>$. Si $y = x$ alors $S_y^x[e] \equiv e$ par le lemme 1.2 et donc dans ce cas $E' \equiv E$, d'où $E' \xrightarrow{\alpha} E$ trivialement.

Sinon ($y \neq x$), on prouve par récurrence structurelle sur e que :

$$y \notin V[e] \text{ et } L(x,y,e) \text{ et } e' \equiv S_y^x[e] \implies x \notin V[e'] \text{ et } L(y,x,e') \text{ et } e \equiv S_x^y[e'].$$

La preuve se fait en considérant les différents cas, et suit directement les définitions de V , L et S ; nous ne donnons pas les détails ici. Ceci prouve donc que toutes les conditions sont remplies pour que

$$E' \xrightarrow{\alpha} E . \square$$

L'égalité = est maintenant définie comme la fermeture réflexive et transitive de $\xrightarrow{\alpha}$. C'est donc une relation d'équivalence qui préserve les types. Cette relation est bien sûr trivialement décidable en temps linéaire.

Par exemple : $\lambda x((A\lambda yx)\lambda yz) = \lambda y((A\lambda zy)\lambda uz)$

mais : $\lambda x(Ay) \neq \lambda y(Ay)$

et : $f \neq \lambda x(fx)$.

Lemme 1.4 $E = E' \implies V[E] = V[E']$.

Démonstration : On prouve, par récurrence structurelle sur e , que

$$y \notin V[e] \implies V[e] - \{y\} = V[S_y^x[e]] - \{y\}.$$

Donc $E \xrightarrow{\alpha} E'$ implique $V[E] = V[E']$,

car $V[e] = V[e'] \implies V[E<e>] = V[E<e'>]$.

La proposition suit immédiatement. \square

Il est facile de prouver que l'égalité respecte les règles de substitutivité :

$$\begin{cases} e_1 = e_2 \implies E<e_1> = E<e_2> \\ e_1 = e_2 \implies S_{e_1}^x[E] = S_{e_2}^x[E]. \end{cases}$$

Par contre, nous n'avons pas :

$e_1 = e_2 \implies S_E^x[e_1] = S_E^x[e_2]$, toujours à cause des captures. Pour remédier à cela, nous supposons dorénavant que l'écriture de $S_E^x[e]$ suppose qu'on a pris e dans sa classe tel que $\forall y \in V[E] \ L(x, y, e)$. Nous emploierons donc librement les équations structurelles :

$$\begin{cases} S_E^x(e_1 e_2) = (S_E^x[e_1] S_E^x[e_2]) \\ S_E^x[\lambda ue'] = \lambda u S_E^x[e']. \end{cases}$$

Dans ce qui suit, nous utiliserons M, N, P, Q pour dénoter des termes, afin de ne pas avoir trop d'indices compliqués.

Lemme 1.5 de commutation des S

$\forall M, N, P \in T \quad \forall x, y \in V, \quad x \neq y$ tels que $y \notin V[M]$,
on a $S_M^x[S_N^y[P]] = S_Q^y[S_M^x[P]],$ avec $Q = S_M^x[N].$

Démonstration : La preuve est aisée, par récurrence structurelle sur P , en utilisant les équations ci-dessus. \square

1.5. β -réduction :

Soit $e = E<(\lambda x M N)>; e' = E<S_N^x[M]>$ est dit dériver de e par β -réduction,
et on note :

$$e \xrightarrow{\beta} e'.$$

Le sous-terme $(\lambda x M N)$ s'appelle *redex* de e , le sous-terme de e' correspondant $S_N^x[M]$ s'appelle le *rédion*. On dit qu'un terme e est en *forme β -normale* s'il n'est pas de la forme $E<(\lambda x M N)>.$

La règle de β -réduction est la règle de calcul de notre langage. Elle exprime la règle de substitution de l'argument N pour l'argument formel x dans le calcul de la valeur en N de la fonction définie par $f(x) \equiv M$. La β -réduction conserve bien sûr les types. Elle n'est ni symétrique, ni réflexive. On dénote par $\xrightarrow{\beta}^*$ sa fermeture réflexive et transitive.

Lemme 1.6

$$\forall e, e' \in T \quad e \xrightarrow{\beta} e' \implies V[e'] \subset V[e].$$

Démonstration :

Soit $e = E<(\lambda x M N)>$.

$$V[(\lambda x M N)] = (V[M] - \{x\}) \cup V[N]$$

$$V[S_N^x[M]] = \begin{cases} (V[M] - \{x\}) \cup V[N] & \text{si } x \in V[M] \\ V[M] & \text{sinon} \end{cases}$$

On a donc bien $V[S_N^x[M]] \subset V[(\lambda x M N)]$, et le résultat suit immédiatement. \square

Lemme 1.7

$$e \xrightarrow{\beta} e' \implies S_E^x[e] \xrightarrow{\beta} S_E^x[e'].$$

Démonstration :

Soit $e = E<(\lambda u M N)>$, $e' = E<S_N^u[M]>$.

On peut supposer que les variables liées dans e et e' sont distinctes des variables de $V[E] \cup \{x\}$. Alors, par application des règles structurelles, en notant $E'<> = S_E^x[E<>]$:

$$S_E^x[e] = E'<(\lambda u S_E^x[M] S_E^x[N])>$$

$$\text{et } S_E^x[e'] = E'<S_E^x[S_N^u[M]]>. \text{ Soit } P = S_E^x[N].$$

On a $S_E^x[e] \xrightarrow{\beta} E'<S_P^u[S_E^x[M]]>$, qui est précisément égal à $S_E^x[e']$ par le lemme 1.5. \square

Dans la suite nous utiliserons, sans l'énoncer, la propriété suivante, facile à montrer par récurrence sur la longueur de la réduction :

$$e \xrightarrow{\beta} e' \implies F<e> \xrightarrow{\beta} F<e'>.$$

Lemme 1.8

$$e \xrightarrow{\beta} e' \implies S_e^x [E] \xrightarrow{\beta} S_{e'}^x [E]$$

Démonstration : par récurrence structurelle sur E :

i) $E = x \quad S_e^x [E] = e \xrightarrow{\beta} e' = S_{e'}^x [E].$

ii) $E = @ \# x \quad S_e^x [E] = @ = S_{e'}^x [E]$

iii) $E = (e_1 e_2) \quad S_e^x [E] = (S_e^x [e_1] S_e^x [e_2])$

$$\xrightarrow{\beta} (S_{e'}^x [e_1] S_{e'}^x [e_2]) \text{ par hypothèse de récurrence}$$

$$\xrightarrow{\beta} (S_{e'}^x [e_1] S_{e'}^x [e_2]) \text{ de même}$$

$$= S_{e'}^x [E]$$

iv) $E = \lambda x e_1 \quad S_e^x [E] = S_{e'}^x [E] = E$

v) $E = \lambda y e_1 \quad S_e^x [E] = \lambda y S_e^x [e_1] \xrightarrow{\beta} \lambda y S_{e'}^x [e_1] \text{ par hypothèse de récurrence}$

$$= S_{e'}^x [E] . \square$$

1.6. Normalisation forte

Nous nous proposons maintenant de montrer que la β -réduction est progressivement bornée, c'est à dire que pour tout terme e_1 , il existe une borne supérieure à la longueur des chaînes :

$$e_1 \xrightarrow{\beta} e_2 \xrightarrow{\beta} e_3 \xrightarrow{\beta} \dots$$

La preuve de ce théorème (qui s'appelle généralement propriété de normalisation forte dans la terminologie de logique combinatoire), nécessite plusieurs lemmes préliminaires. La méthode utilisée est due à Tait [TW1].

Définitions :

Pour tout $e \in T$, on définit $\Lambda(e)$ comme la longueur de la plus longue β -chaîne issue de e . Soit $N = \{e \mid \Lambda(e) < \infty\}$. On appelle N l'ensemble des termes fortement normalisables.

L'ensemble des termes dérivables est $\mathcal{D} = \bigsqcup_{\gamma \in T} \mathcal{D}_\gamma$, avec \mathcal{D}_γ défini par :

$$\text{i)} \quad \text{si } \gamma \in T_0 \quad \text{alors } \mathcal{D}_\gamma = T_\gamma \cap N.$$

$$\text{ii)} \quad \text{si } \gamma = (\alpha \rightarrow \beta) \text{ alors } \mathcal{D}_\gamma = \{e \in T_\gamma \mid \forall e' \in \mathcal{D}_\alpha (ee') \in \mathcal{D}_\beta\}.$$

Remarques :

1) Si $e = F<e'$, alors $\Lambda(e) \geq \Lambda(e')$; donc $e \in N \implies e' \in N$.

2) Si $e \not\in \mathcal{D}_\beta$, alors $\Lambda(e) > \Lambda(e')$; donc $e \in N \implies e' \in N$.

De même $e \in \mathcal{D}_\gamma \implies e' \in \mathcal{D}_\gamma$, par une récurrence facile sur γ .

Nous allons maintenant montrer que $T = N$, en prouvant $\mathcal{D} \subset N$ puis $T \subset \mathcal{D}$.

Lemme 1.9 Soit $e_1, e_2, \dots, e_n \in N$, $n \geq 0$.

$$\text{Soit } @ \in A, \text{ avec } \tau(@) = \begin{cases} (\tau(e_1) \times \dots \times \tau(e_n)) \rightarrow \gamma & \text{si } n > 0 \\ \gamma & \text{si } n = 0, \text{ où } \gamma \in T \text{ quelconque.} \end{cases}$$

$$\text{On définit } e = \begin{cases} (\dots ((@e_1)e_2) \dots e_n) & \text{si } n > 0 \\ @ & \text{si } n = 0. \end{cases}$$

Alors $e \in \mathcal{D}_\gamma$.

Lemme 1.10 $\mathcal{D}_\gamma \subset N$.

Démonstrations :

On prouve les lemmes 1.9 et 1.10 par récurrence simultanée sur γ .

i) $\gamma \in T_0$. On a $\mathcal{D}_\gamma \subset N$ par définition. Pour le lemme 1.9, on remarque que

$$\Lambda(e) = \sum_{i=1}^n \Lambda(e_i), \text{ ce qui entraîne } e \in N, \text{ et donc } e \in \mathcal{D}_\gamma \text{ par définition de } \mathcal{D}_\gamma.$$

ii) $\gamma = (\alpha \rightarrow \beta)$. Désignons par A_γ l'énoncé du lemme 1.9 et par B_γ celui du lemme 1.10.

Par hypothèse de récurrence, on suppose $A_\alpha, A_\beta, B_\alpha, B_\beta$. Soit maintenant

e_1, e_2, \dots, e_n, e comme dans A_γ . Soit $e_{n+1} \in D_\alpha$, $e' = (\dots ((@e_1)e_2) \dots e_{n+1})$.

Par B_α , on a $e_{n+1} \in N$. $\tau(e') = \beta$, et donc $e' \in D_\beta$, par A_β . Comme cela est vrai pour tout $e_{n+1} \in D_\alpha$, on a $e \in D_\gamma$, par définition de D_γ , ce qui prouve A_γ .

Soit maintenant $e \in D_\gamma$ quelconque, et $\tau(@)=\alpha$. On a $@ \in D_\alpha$, par A_α (cas $n=0$).

Par définition de D_γ , on doit avoir $(e@) \in D_\beta$, et donc $(e@) \in N$, par B_β .

Par la remarque 1 ci-dessus, ceci entraîne $e \in N$, c'est à dire B_γ . \square

Lemme 1.11

Soit $N \in D_\gamma$. Si $S_N^x[M] \in D_\delta$, alors $(\lambda x M N) \in D_\delta$.

Démonstration :

Soit $N \in D_\gamma$, $e = S_N^x[M] \in D_\delta$. Si $\delta \notin T_0$, alors soit

$\delta = (\gamma_1 \rightarrow (\gamma_2 \rightarrow \dots (\gamma_n \rightarrow \delta_0) \dots))$ avec $\delta_0 \in T_0$.

Choisissons, pour tout $i \leq n$, e_i quelconque dans D_{γ_i}

(on sait que $D_{\gamma_i} \neq \emptyset$, car $V_{\gamma_i} \subset D_{\gamma_i}$ par le lemme 1.9).

Soit $e' = (\dots ((\lambda x M N) e_1) \dots e_n)$. (si $\delta \in T_0$, $n=0$).

Nous allons montrer que $e' \in N$. Pour cela, considérons toutes les β -réductions à partir de e' . Il y a deux sortes : celles qui réduisent le rédex $(\lambda x M N)$ (plus précisément son descendant dans la réduction considérée), et celles qui ne le réduisent pas. Pour ces dernières, leur longueur est majorée par $\Lambda(M) + \Lambda(N) + \sum_{i=1}^n \Lambda(e_i)$. Mais par hypothèse $S_N^x[M] \in D_\gamma$, et donc $S_N^x[M] \in N$ par le lemme 1.10, ce qui entraîne $M \in N$ par le lemme 1.7. On a également $N \in N$ et $e_i \in N$ par le lemme 1.10, et donc toutes ces réductions se terminent.

Les réductions qui réduisent $(\lambda x M N)$ se décomposent en :

(i) $M \xrightarrow[\beta]{*} M'$ toujours finies, car on vient de voir $M \in N$.

(ii) $N \xrightarrow[\beta]{*} N'$ toujours finies, puisque $N \in N$.

(iii) $e_i \xrightarrow{\beta} e'_i$ toujours finies de même

(iv) une réduction à partir de $e'' = (\dots (S_N^x[M']e'_1)\dots e'_n)$.

Mais par le lemme 1.7 $S_N^x[M] \xrightarrow{\beta} S_N^x[M']$, donc par le lemme 1.8

$S_N^x[M] \xrightarrow{\beta} S_N^x[M']$, et $(\dots (S_N^x[M]e_1)\dots e_n) \xrightarrow{\beta} e''$.

Mais par l'hypothèse $S_N^x[M] \in \mathcal{D}_\delta$, on a $(\dots (S_N^x[M]e_1)\dots e_n) \in \mathcal{D}_{\delta_0} \subset N$, donc

$e'' \in N$, et ces réductions là se terminent aussi. Donc $e' \in N$, et aussi $e' \in \mathcal{D}_{\delta_0}$, puisque $\tau(e') \in T_0$. Ceci entraîne $(\lambda xMN) \in \mathcal{D}_\delta$, car les e_i sont quelconques. \square

Lemme 1.12 Pour tout $n \geq 0$ soit $u_i \in V_{\gamma_i}$, $e_i \in \mathcal{D}_{\gamma_i}$, $1 \leq i \leq n$, tels que

$u_i \notin (V[e_j] \cup \{u_j\}) \forall j < i$. Alors $\forall e \in T$ $S_{e_n}^{u_n}[\dots S_{e_2}^{u_2}[S_{e_1}^{u_1}[e]]\dots] \in \mathcal{D}$.

Démonstration

Pour tout $e \in T$, notons $\phi(e) = S_{e_n}^{u_n}[\dots S_{e_1}^{u_1}[e]\dots]$.

La preuve se fait par récurrence structurelle sur e .

i) $e = u_i$; alors $\phi(e) = e_i \in \mathcal{D}$ par hypothèse.

ii) $e = @ \neq u_j$ $1 \leq j \leq n$; alors $\phi(e) = @ \in \mathcal{D}$ par le lemme 1.9.

iii) $e = (MN)$; alors $\phi(e) = (\phi(M)\phi(N))$ avec par hypothèse de récurrence $\phi(M) \in \mathcal{D}$, $\phi(N) \in \mathcal{D}$, ce qui entraîne $\phi(e) \in \mathcal{D}$ par définition de \mathcal{D} .

iv) $e = \lambda xM$, avec $\tau(x) = \alpha$ et $\tau(M) = \beta$; on peut supposer x tel que

$x \notin (\{u_i\} \cup V[e_i]) \forall i \leq n$. Alors, pour tout $N \in \mathcal{D}_\alpha$, on a

$S_N^x[\phi(M)] = S_N^x[S_{e_n}^{u_n}[\dots [S_{e_1}^{u_1}[M]\dots]]] \in \mathcal{D}_\beta$ par hypothèse de récurrence, et

donc $(\lambda x\phi(M)N) \in \mathcal{D}_\beta$ par le lemme 1.11. Comme $\phi(e) = \lambda x\phi(M)$, on a $\phi(e) \in \mathcal{D}$

par définition de $\mathcal{D}_{(\alpha \rightarrow \beta)}$. \square

Théorème 1 de normalisation $T = N$.

Démonstration

- (i) $T \subset D$ par le lemme 1.12, avec $n=0$.
 - (ii) $D \subset N$ par le lemme 1.10
- $N \subset T \subset N$, d'où le théorème. \square

Remarque : ce théorème n'est pas vrai du λ -calcul non typé ; par exemple, en prenant $e = (\lambda u(uu)\lambda u(uu))$, on aurait $e \not\rightarrow_{\beta} e$.

1.7. Cohérence :

Nous allons maintenant prouver une propriété de cohérence de la relation \rightarrow_{β} , qui va nous permettre d'effectuer les β -réductions dans un ordre arbitraire. Dans la terminologie de logique combinatoire, cette propriété est connue comme la propriété de Church-Rosser :

$$\forall e_1, e_2, e_3 \in T \\ e_1 \xrightarrow{\beta}^* e_2 \text{ et } e_1 \xrightarrow{\beta}^* e_3 \Rightarrow \exists e_4 \in T \quad e_2 \xrightarrow{\beta}^* e_4 \text{ et } e_3 \xrightarrow{\beta}^* e_4.$$

Cette propriété est vraie du λ -calcul complet, c'est à dire sans les restrictions de type, au contraire du théorème de normalisation. Mais lorsque la propriété de normalisation est vraie, la propriété de cohérence est plus facile à montrer par son intermédiaire, et c'est ce que nous avons choisi de faire ici.

Lemme 1.13 (propriété du losange).

$$\forall e_1, e_2, e_3 \in T \quad e_1 \not\rightarrow_{\beta} e_2 \text{ et } e_1 \not\rightarrow_{\beta} e_3 \Rightarrow \exists e_4 \in T \quad e_2 \xrightarrow{\beta}^* e_4 \text{ et } e_3 \xrightarrow{\beta}^* e_4$$

Démonstration

Si $e_2 = e_3$, c'est trivial ; sinon il y a deux cas, suivant les positions respectives des deux rédex réduites dans e_1 .

(i) Les rédex sont des sous-termes n'ayant pas de partie commune, c'est à dire :

$$e_1 = E_1 \ll(E_2 \langle (\lambda u_2 M_2 N_2) \rangle E_3 \langle (\lambda u_3 M_3 N_3) \rangle) \gg$$

avec $e_2 = E_1 \langle E_2 S_{N_2}^{u_2} [M_2] \rangle E_3 \langle (\lambda u_3 M_3 N_3) \rangle \rangle$

et $e_3 = E_1 \langle (E_2 \langle \lambda u_2 M_2 N_2 \rangle) E_3 \langle S_{N_3}^{u_3} [M_3] \rangle \rangle$.

Il suffit de prendre :

$$e_4 = E_1 \langle (E_2 \langle S_{N_2}^{u_2} [M_2] \rangle E_3 \langle S_{N_3}^{u_3} [M_3] \rangle) \rangle \text{ et } e_2 \not\rightarrow e_4 \text{ et } e_3 \not\rightarrow e_4 \text{ sans difficulté.}$$

(ii) le rédex réduit pour donner e_3 est interne à celui réduit pour donner e_2 .

Il y a de nouveau deux cas :

a) $e_1 = E_1 \langle (\lambda u_2 E_2 \langle (\lambda u_3 M_3 N_3) \rangle N_2) \rangle$

avec $e_2 = E_1 \langle S_{N_2}^{u_2} [E_2 \langle (\lambda u_3 M_3 N_3) \rangle] \rangle$

et $e_3 = E_1 \langle (\lambda u_2 E_2 \langle S_{N_3}^{u_3} [M_3] \rangle N_2) \rangle$.

Considérons $e_4 = E_1 \langle S_{N_2}^{u_2} [E_2 \langle S_{N_3}^{u_3} [M_3] \rangle] \rangle$.

On a $e_3 \not\rightarrow e_4$ sans difficulté.

Aussi : $e = E_2 \langle (\lambda u_3 M_3 N_3) \rangle \not\rightarrow E_2 \langle S_{N_3}^{u_3} [M_3] \rangle = e'$ d'où

$S_{N_2}^{u_2} [e] \not\rightarrow S_{N_2}^{u_2} [e']$ par le lemme 1.7 et donc $e_2 \not\rightarrow e_4$.

b) $e_1 = E_1 \langle (\lambda u_2 M_2 E_2 \langle (\lambda u_3 M_3 N_3) \rangle) \rangle$

avec $e_2 = E_1 \langle S_{E_2}^{u_2} [M_2] \rangle$ où $E_2 = E_1 \langle (\lambda u_3 M_3 N_3) \rangle$

et $e_3 = E_1 <(\lambda u_2 M_2 E_3)>$ où $E_3 = E_2 <S_{N_3}^{u_3}[M_3]>.$

Soit $e_4 = E_1 <S_{E_3}^{u_2}[M_2]>; e_3 \xrightarrow{*} e_4$ sans difficulté.

Comme $E_2 \xrightarrow{*} E_3$, on a $e_2 \xrightarrow{*} e_4$ par le lemme 1.8. \square

Théorème 2 de cohérence

$$\forall e_1, e_2, e_3 \in T \quad e_1 \xrightarrow{*} e_2 \text{ et } e_1 \xrightarrow{*} e_3 \implies \exists e_4 \in T \quad e_2 \xrightarrow{*} e_4 \text{ et } e_3 \xrightarrow{*} e_4.$$

Démonstration

Par récurrence sur $\Lambda(e_1)$ (qui est fini par le théorème 1). Soit $e_1 \xrightarrow{n} e_2$ et $e_1 \xrightarrow{m} e_3$. Si $n=0$, on peut prendre $e_4 = e_3$, si $m=0$ on peut prendre $e_4 = e_2$.

Sinon soit :

$$e_1 \xrightarrow{*} e'_1 \xrightarrow{n-1} e_2 \text{ et } e_1 \xrightarrow{*} e''_1 \xrightarrow{m-1} e_3.$$

Par le lemme 1.13, $\exists \bar{e} \quad e'_1 \xrightarrow{*} \bar{e}$ et $e''_1 \xrightarrow{*} \bar{e}$.

Par hypothèse de récurrence, comme $\Lambda(e'_1) < \Lambda(e_1)$, on sait que $\exists E_2 \quad e'_1 \xrightarrow{*} E_2$ et $\bar{e} \xrightarrow{*} E_2$. De même, $\exists E_3 \quad e''_1 \xrightarrow{*} E_3$ et $\bar{e} \xrightarrow{*} E_3$. Mais aussi $\Lambda(\bar{e}) < \Lambda(e)$, et donc $\exists e_4 \quad E_2 \xrightarrow{*} e_4$ et $E_3 \xrightarrow{*} e_4$. \square

Corollaire

Soit $\xrightarrow{\beta}$ l'équivalence engendrée par $\xrightarrow{*}$. Alors $e \xrightarrow{\beta} e'$ ssi $\exists e''$

$$e \xrightarrow{\beta} e'' \text{ et } e' \xrightarrow{\beta} e''.$$

Démonstration :

$$\text{Soit } e = e_1 \xrightarrow{*} e_2 \xrightarrow{*} e_3 \xrightarrow{*} \cdots \xrightarrow{*} e_{2n} \xrightarrow{*} e_{2n+1} = e'.$$

La preuve se fait par récurrence sur n . Si $n=0$, alors prendre $e'' = e = e'$.

Sinon, on a :

$\exists \bar{e}_1 \quad e_1 \xrightarrow{\beta} \bar{e}_1 \quad \text{et} \quad e_3 \xrightarrow{\beta} \bar{e}_1 \quad \text{par le théorème 2}$

$\exists \bar{e}_2 \quad e_3 \xrightarrow{\beta} \bar{e}_2 \quad \text{et} \quad e' \xrightarrow{\beta} \bar{e}_2 \quad \text{par hypothèse de récurrence}$

et donc $\exists e' \quad \bar{e}_1 \xrightarrow{\beta} e' \quad \text{et} \quad \bar{e}_2 \xrightarrow{\beta} e' \quad \text{par le théorème 2}$

appliqué à e_3, \bar{e}_1 et \bar{e}_2 . \square

Ce corollaire montre que la β -réduction est suffisante pour exprimer l'inter-convertibilité.

1.8. Forme canonique

Théorème 3 de la forme canonique

Tout terme possède une forme normale unique, que l'on obtient au bout d'un temps fini par une séquence arbitraire de β -réductions.

Démonstration :

Par le théorème 1, toutes les réductions à partir d'un terme e se terminent en un temps fini sur une forme normale ; considérons deux telles séquences de réductions :

$$e \xrightarrow{\beta} e_1 \quad e \xrightarrow{\beta} e_2.$$

Par le théorème 2, il existe e_3 tel que :

$$e_1 \xrightarrow{\beta} e_3 \quad e_2 \xrightarrow{\beta} e_3.$$

e_1 et e_2 étant des formes normales, on doit avoir $n=m=0$, et donc $e_3 = e_1 = e_2$. \square

Pour insister sur le caractère d'unicité de cette forme normale, nous l'appellerons *forme canonique de e* , dénotée par $N[e]$.

Corollaire :

$e \equiv e'$ ssi $N[e] = N[e']$, et cela fournit une méthode de décision pour l'interconvertibilité.

Abréviations :

A partir de maintenant, nous allons utiliser la notation fonctionnelle classique, en utilisant les abréviations suivantes :

$$\left\{ \begin{array}{l} (\dots((e_1 e_2) e_3) \dots e_n) \longrightarrow e_1(e_2, e_3, \dots, e_n) \\ \lambda x_1 x_2 \dots \lambda x_n e \longrightarrow \lambda x_1 x_2 \dots x_n e \quad \text{si les } x_i \text{ sont distincts.} \end{array} \right.$$

Tout terme e en forme normale peut ainsi être abrégé en une expression de la forme :

$$\lambda x_1 x_2 \dots x_n @ (e_1, e_2, \dots, e_p) \quad \text{où :}$$

- $@ \in A$ $@$ est appelé tête de e .
- $n \geq 0$; si $n=0$ λ n'apparaît pas.
- $p \geq 0$; si $p=0$ () n'apparaît pas.
- $\{x_1, \dots, x_n\}$ est un ensemble de n variables distinctes.
- $\forall i \in [1, p]$ e_i est une expression de la même forme; on dit que e_i est un argument de e .
- $\forall i \in [1, n]$, $\forall j \in [1, p]$ x_i n'apparaît pas lié dans e_j .

La preuve en est facile, et n'utilise que les propriétés de l'égalité.

Remarquons que notre notation n'est pas ambiguë pour les termes en forme normale.

Par exemple, $\lambda u \cdot u(v)$ est une abréviation pour $\lambda u(uv)$ et non (λuuv) , qui a pour forme canonique v .

Définitions :

Soit $e = \lambda x_1 x_2 \dots x_n \cdot @ (e_1, e_2, \dots, e_p)$ un terme en forme normale.

On définit :

- $H[e] = \lambda x_1 x_2 \dots x_n \cdot @$; ce terme est appelé *en tête* de e .
- $R[e] = \{x_1, \dots, x_n\}$; cet ensemble est appelé *lieur* de e .
- $H[e] = n$; ce nombre ≥ 0 est appelé *arité* de e .
- $\pi(e)$ est défini récursivement par : $\pi(e) = p + \sum_{i=1}^p \pi(e_i)$

C'est le nombre d'applications utilisées dans la construction de e ; on appelle $\pi(e)$ la *profondeur* de e .

Si $@ \in C \cup R[e]$, on dit que e est *rigide*.

Sinon (i.e. $@ \in V[e]$) e est dit *flexible*.

On dénote par T_r l'ensemble des termes rigides, et par T_f l'ensemble des termes flexibles.

A partir du chapitre 2, nous ne considérons plus que des termes en forme normale. Donc T désignera l'ensemble des termes en forme normale, $=$ dénotera l'identité des formes β -normales, modulo α -conversion, $V[e]$ dénotera l'ensemble des variables libres de la forme canonique de e . Nous ferons les preuves directement par récurrence sur la structure des termes mis sous forme abrégée.

CHAPITRE 2 : Substitutions solutions d'équations

Ce chapitre a pour objet l'étude des équations $e_1 = e_2$ dans T .

Résoudre une telle équation, c'est trouver une substitution pour les variables libres de e_1 et e_2 qui rende ces termes identiques, après conversion.

Nous allons tout d'abord définir et étudier les substitutions.

2.1. Substitutions

On appelle *composante de substitution* une paire $\langle x, e \rangle$, où $x \in V_Y$, et $e \in T_Y$ est en forme normale.

On appelle *substitution* un ensemble fini de composantes de substitution se rapportant à des variables distinctes :

$$\sigma = \{ \langle x_i, e_i \rangle \mid 1 \leq i \leq n \} \quad \forall i, j \leq n \quad x_i = x_j \Rightarrow i = j.$$

Nous dénotons les substitutions par $\sigma, \rho, \eta, \xi, \zeta$, et désignons par S l'ensemble des substitutions.

Nous ignorons dans les substitutions les paires $\langle x, x \rangle$.

Nous définissons donc l'égalité dans S par :

$$\sigma = \sigma' \Leftrightarrow (\sigma - \sigma') \cup (\sigma' - \sigma) \subset \{ \langle x, x \rangle \mid x \in V \}$$

Soit $\sigma \in S$. On définit, pour tout x dans V , $\sigma x \in T$ par :

$$\sigma x = \begin{cases} e & \text{si } \langle x, e \rangle \in \sigma \\ x & \text{sinon.} \end{cases}$$

On appelle *domaine* de la substitution σ l'ensemble fini

$D(\sigma) = \{x \in V \mid \sigma x \neq x\}$ et ensemble des variables *introduites* par σ l'ensemble fini

$$I(\sigma) = \bigsqcup_{x \in D(\sigma)} V[\sigma x].$$

Autrement dit, une substitution est une application de V dans T , pré-servant les types, et égale à l'identité presque partout. Cette application est étendue à T comme suit.

L'*application* de la substitution $\sigma = \{ \langle x_i, e_i \rangle \mid 1 \leq i \leq n \}$ à un terme e est définie par :

$$\sigma e = N\Gamma(\lambda x_1 \dots x_n. e)(e_1, e_2, \dots, e_n).$$

On vérifie facilement que cette définition est cohérente avec la notation précédente σx . De plus, les conflits de variables étant résolus automatiquement lors de la mise sous forme normale, σe ne dépend pas de l'ordre dans lequel on prend les paires $\langle x_i, e_i \rangle$ dans σ , comme le montre le lemme suivant.

Lemme 2.1 de permutation des composantes.

Soit X un ensemble de n variables distinctes de V :

$$X = \{x_1, \dots, x_n\}.$$

Soit $\{i_1, i_2, \dots, i_n\}$ une permutation quelconque de $\{1, 2, \dots, n\}$, et n termes $e_i \in T_{\tau}(x_i)$ $1 \leq i \leq n$. Alors, pour tout $e \in T$:

$$(\lambda x_1 \dots x_n . e)(e_1, \dots, e_n) = (\lambda x_{i_1} \dots x_{i_n} . e)(e_{i_1}, \dots, e_{i_n}).$$

Démonstration

Il suffit de prouver le lemme pour une transposition de deux variables.

On veut montrer :

$(\lambda x_1 x_2 . e)(e_1, e_2) = (\lambda x_2 x_1 . e)(e_2, e_1)$. Alors, en supposant qu'il n'y a pas de conflits, et en particulier que $x_2 \notin V[e_1]$:

$$\begin{aligned} & (\lambda x_1 x_2 . e)(e_1, e_2) \stackrel{x_1}{\not\in} (S_{e_1}^{x_1}[\lambda x_2 e] e_2) \\ &= (\lambda x_2 S_{e_1}^{x_1}[e] e_2) \stackrel{x_2}{\not\in} S_{e_2}^{x_2} [S_{e_1}^{x_1}[e]] \\ &= S_{e_2}^{x_2} [S_{e_2}^{x_2}[e]] \quad \text{par le lemme de commutation 1.5} \\ &= (\lambda x_2 x_1 . e)(e_2, e_1) \text{ par symétrie. } \square \end{aligned}$$

Ce lemme nous conduit à une autre caractérisation de l'opération σe , comme substitution parallèle.

Soit $\sigma = \{<x_i, e_i> \mid 1 \leq i \leq n\}$. On définit une opération $S_{e_i}^{x_i}$, par la définition récursive de $S_{e_i}^{x_i}[e]$ suivant la structure de $e \in T$:

$$(i) \quad S_{e_i}^{x_i}[x_k] = e_k \quad 1 \leq k \leq n$$

$$(ii) \quad S_{e_i}^{x_i}[y] = y \quad y \notin \{x_i \mid 1 \leq i \leq n\}$$

$$(iii) \quad S_{e_i}^{x_i}[(ee')] = N(S_{e_i}^{x_i}[e] S_{e_i}^{x_i}[e'])$$

$$(iv) \quad S_{\{e_i\}}^{x_i}[\lambda ye] = \lambda z \quad S_{\{e_i\}}^{x_i}[S_z^y[e]] \text{ où } z \notin \bigcup_{i=1}^n (\{x_i\} \cup V[e_i]) \text{ et } R(y, z, e).$$

Pour (iv), si on choisit y tel que $y \neq x_i$ et $y \notin V[e_i]$ $\forall i \leq n$, alors (par le lemme 1.2) on peut écrire :

$S_{\{e_i\}}^{x_i}[\lambda ye] = \lambda y \quad S_{\{e_i\}}^{x_i}[e]$ ce qui montre que $S_{\{e_i\}}^{x_i}$ vérifie toutes les équations structurelles.

Lemme 2.2 Soit $\sigma = \{<x_i, e_i> \mid 1 \leq i \leq n\}$.

$$\forall e \in T \quad \sigma e = S_{\{e_i\}}^{x_i} e.$$

Démonstration

Soit $E \equiv (\lambda x_1 \dots x_n . e)(e_1, \dots, e_n)$. Soit y_1, \dots, y_n des variables distinctes de même types respectivement que x_1, \dots, x_n , et n'apparaissant pas dans E . On a

$$\lambda x_1 \dots x_n . e = \lambda y_1 \dots y_n . e', \text{ avec } e' \equiv S_{y_n}^{x_n} [\dots S_{y_1}^{x_1} [e] \dots] \text{ et}$$

$$\sigma e = N[F] = N[S_{e_n}^{y_n} [\dots S_{e_1}^{y_1} [e'] \dots]] \text{ (par } \beta\text{-conversion).}$$

Il est maintenant facile de vérifier que ce dernier terme est égal à $S_{\{e_i\}}^{x_i} e$, par récurrence sur e .

Faisons par exemple le cas atomique :

(i) $e = y_i$ est impossible car par hypothèse y_i n'a pas d'occurrence dans E .

(ii) $e = x_i$; alors $e' = y_i$ et $S_{e_n}^{y_n} [\dots S_{e_1}^{y_1} [y_i] \dots] = e_i$

par applications successives de la définition de S et du lemme 1.1.

(iii) $e = z \notin \{x_i, y_i\}$; alors $\sigma e = z$ par définition de S . \square

Ce lemme nous montre que les substitutions se comportent comme des morphismes, modulo la mise sous forme normale.

Dans la suite, nous emploierons librement cette propriété.

A partir de maintenant, nous utiliserons de préférence la rotation fonctionnelle pour les termes. La relation = dénotera l'égalité des formes canoniques. Dans cette notation, le lemme 2.2 nous donne :

Lemme 2.3

Soit $e = \lambda u_1 \dots u_n @ (e_1, \dots, e_p)$, $\sigma \in S$.

Si $\forall i \quad 1 \leq i \leq n \quad u_i \notin D(\sigma) \cup I(\sigma)$, alors $\sigma e = \lambda u_1 \dots u_n . \sigma @ (\sigma e_1, \dots, \sigma e_p)$.

Démonstration

Par $n+p$ applications de la définition de $S\{x_i^e\}_e$. \square

Définition :

Etendons à S la définition de π donnée en 1.8.

Soit $\sigma = \{<x_i, e_i> \mid 1 \leq i \leq n\}$; on définit :

$$\pi(\sigma) = n + \sum_{i=1}^n \pi(e_i).$$

Soit V un ensemble fini de variables, $\sigma \in S$. On appelle *restriction de σ à V* la substitution :

$$\sigma \upharpoonright V = \{<x_i, \sigma x_i> \mid x_i \in V\}.$$

Nous finissons ce paragraphe par quelques propriétés élémentaires des substitutions.

Lemme 2.4

$$\forall e, \sigma, v \quad V[e] \subset v \implies \sigma e = (\sigma \upharpoonright v)e.$$

Démonstration

par récurrence sur e . \square

Lemme 2.5

$$\forall e, \sigma \quad D(\sigma) \cap V[e] = \emptyset \implies \sigma e = e.$$

Démonstration

Immédiat par le lemme précédent, puisque en prenant $V = V[e]$ on a $\sigma \upharpoonright V = \emptyset$. \square

Remarque :

La réciproque de ce lemme n'est pas vraie en général. Par exemple, considérer $e = f(A)$ et $\sigma = \{<f, \lambda u. f(u)>\}$ ou $\sigma = \{<f, \lambda u. f(A)>\}$.

Lemme 2.6

L'en-tête d'un terme rigide est invariant par substitution. L'arité d'un terme flexible ne diminue pas par substitution.

$$\text{i.e. } \forall \sigma \in S \quad \begin{cases} \forall e \in T_r \quad H[\sigma e] = H[e] \\ \forall e \in T_f \quad \bar{H}[\sigma e] \geq \bar{H}[e]. \end{cases}$$

Démonstration

Par application directe du lemme 2.3. \square

Lemme 2.7

$$\forall \sigma, \sigma' \quad (\forall x \in V \quad \sigma x = \sigma' x) \Leftrightarrow (\forall e \in T \quad \sigma e = \sigma' e).$$

Démonstration

\Leftarrow est trivial

\Rightarrow est obtenue simplement par récurrence structurelle sur e ,

en utilisant le lemme 2.3. \square

Définition : La composition des substitutions σ et ρ , notée $\rho\sigma$, est définie comme la substitution :

$$\rho\sigma = \{<x, \rho(\sigma x)> \mid x \in V\}.$$

Remarquons que, pour toutes substitutions σ et ρ :

$$\mathcal{D}(\rho\sigma) \subset \mathcal{D}(\sigma) \cup \mathcal{D}(\rho)$$

$$\text{et : } \mathcal{I}(\rho\sigma) \subset \mathcal{I}(\sigma) \cup \mathcal{I}(\rho).$$

La composition des substitutions est exactement la composition au sens des applications de T dans T , comme le montre le lemme suivant.

Lemme 2.8 $\forall \sigma, \rho \in S$

$$\text{a)} \quad \forall e \in T \quad (\rho\sigma)e = \rho(\sigma e)$$

$$\text{b)} \quad \forall n \in S \quad (\rho\sigma)n = \rho(\sigma n)$$

Démonstration

On prouve a) par récurrence sur e :

$$\text{i)} \quad e = x \quad (\rho\sigma)x = \rho(\sigma x) \quad \text{par définition}$$

$$\text{ii)} \quad e = (e_1 e_2) \quad (\rho\sigma)e = ((\rho\sigma)e_1 (\rho\sigma)e_2) \\ = (\rho(\sigma e_1) \rho(\sigma e_2)) \quad \text{par hypothèse de récurrence}$$

$$\begin{aligned}
 &= \rho(\sigma e_1 \sigma e_2) \\
 &= \rho(\sigma(e_1 e_2)) = \rho(\sigma e).
 \end{aligned}$$

iii) $e = \lambda x e'$ on suppose x choisi en dehors de $D(\rho) \cup D(\sigma) \cup I(\rho) \cup I(\sigma)$.

$$\begin{aligned}
 (\rho\sigma)e &= \lambda x(\rho\sigma)e' \text{ car } x \notin D(\rho\sigma) \cup I(\rho\sigma) \text{ par la remarque ci-dessus.} \\
 &= \lambda x\rho(\sigma e') \text{ par hypothèse de récurrence} \\
 &= \rho(\lambda x\sigma e') \text{ car } x \notin D(\rho) \cup I(\rho) \\
 &= \rho(\sigma\lambda x e') \text{ car } x \notin D(\sigma) \cup I(\sigma) \\
 &= \rho(\sigma e).
 \end{aligned}$$

b) soit $x \in V$ quelconque.

$$\begin{aligned}
 [\rho(\sigma\eta)]x &= \rho[(\sigma\eta)x] \text{ par a)} \\
 &= \rho[\sigma(\eta x)] \text{ de même} \\
 &= (\rho\sigma)(\eta x) \text{ de même} \\
 &= [\rho(\sigma)\eta]x \text{ de même. } \square
 \end{aligned}$$

Ce lemme nous permet de nous dispenser des parenthèses, et d'écrire $\rho\sigma e$ et $\rho\sigma\eta$.

2.2. Filtrage

Définition

On définit un préordre \leq dans T par :

$$E \leq E' \iff \exists \sigma \in S \quad \sigma F = E'.$$

On dit que E schématis⁽⁺⁾ E' , ou que E est un schéma pour E' . De manière équivalente, on dira que E généralise E' , ou que E' est une instance de E .

(+) en anglais : E subsumes E' ou E matches E' ; le filtrage est appelé subsumption ou pattern-matching, suivant les domaines d'application.

Les substitutions σ solutions de $\sigma E = E'$ sont appelées *filtres*⁽⁺⁾ de E vers E' . On dénote par $F(E, E')$ l'ensemble de ces substitutions. On appelle problème du *filtrage* la recherche de tels filtres.

Remarquons que s'il existe un filtre de E vers E' , il n'est pas unique en général, même si on le restreint à prendre son domaine dans $V[E]$. Ceci est vrai même si on se limite à des termes de second ordre.

Par exemple, prenons

$$\begin{cases} E = f(A) \\ E' = A \end{cases} \quad \text{avec} \quad \begin{cases} \tau(A) = \alpha \\ \tau(f) = (\alpha \rightarrow \alpha). \end{cases}$$

Alors $\sigma_1 = \{\langle f, \lambda u. A \rangle\}$
 et $\sigma_2 = \{\langle f, \lambda u. u \rangle\}$ sont des filtres de E vers E' .

Par contre, nous montrerons au chapitre 5 que la solution d'un problème de filtrage de premier ordre est unique, si elle existe.

Au chapitre 6, nous verrons qu'à l'ordre 2 il est possible de décrire toutes les solutions par un nombre fini de filtres, qui sont "solutions principales".

Afin de définir plus précisément cette notion, il nous faut exprimer en quoi une solution peut découler d'une autre. Pour cela, nous étendons le préordre \leq à S comme suit.

Définition :

$$\sigma \leq \rho \iff \exists \eta \rho = \eta \sigma.$$

Nous dirons que σ schématisse ρ , on que σ "est plus générale" que ρ , ou encore que ρ est une *instance* de σ .

Par exemple, si $\sigma = \{\langle x, y \rangle\}$ et $\rho = \{\langle x, A \rangle, \langle y, A \rangle\}$, avec $\tau(x) = \tau(y) = \tau(A) = \alpha$, alors $\sigma \leq \rho$ car $\rho = \eta \sigma$ avec $\eta = \{\langle y, A \rangle\}$.

Si σ et ρ sont des filtres de E vers E' tels que $\sigma \leq \rho$, on peut dire que la solution ρ découle de la solution σ .

(+) Cette notion n'a rien à voir avec la notion topologique de filtre.

On appelle *filtre principal* (FP) de E vers E' un filtre σ d'où découlent tous les autres :

$$(i) \quad \sigma \in F(E, E')$$

$$(ii) \quad \forall \rho \in F(E, E') \quad \sigma \leq \rho$$

Autrement dit, un filtre principal est un élément **minimum** de $F(E, E')$.

Lemme 2.9

Il existe des termes de second ordre E et E' , avec $F(E, E') \neq \emptyset$, qui n'admettent pas de filtre principal.

Démonstration

Il suffit de prendre E et E' comme dans l'exemple ci-dessus :

$$\left\{ \begin{array}{l} E = f(A) \\ E' = A \end{array} \right. \quad \text{avec} \quad \left\{ \begin{array}{l} \tau(A) = \alpha \\ \tau(f) = (\alpha \rightarrow \alpha). \end{array} \right.$$

Considérons $\sigma_1 = \{\langle f, \lambda u, A \rangle\}$ avec $\tau(u) = \alpha$.
et $\sigma_2 = \{\langle f, \lambda u, u \rangle\}$

On a $\sigma_1, \sigma_2 \in F(E, E')$. Mais $\sigma_1 \neq \sigma_2$, car pour toute substitution ρ :
 $\rho \sigma_1 f = \lambda u. A \neq \lambda u. u = \sigma_2 f$.

De même $\sigma_2 \neq \sigma_1$. Montrons plus généralement que σ_1 et σ_2 n'admettent pas de minorant commun dans $F(E, E')$.

Soit $\sigma \in F(E, E')$ quelconque.

Si $\bar{H}[\sigma f] = 0$ alors $\sigma f \equiv (\sigma f A)$ ne peut donc être égal à $E' = A$. On doit donc avoir, à cause du type de f , $\bar{H}[\sigma f] = 1$, i.e. $\sigma f = \lambda u. @e_1, \dots, e_n$.

(i) Si $@ \equiv u$, alors $n = 0$ à cause des types. Mais alors, pour tout $\rho \in S$,

$$\rho \sigma f = \lambda u. u \neq \lambda u. A = \sigma_1 f.$$

(ii) Si $\emptyset \neq u$, alors $\sigma f(A) = \emptyset(\dots, S_A^u[e_i], \dots)$ ne peut être égal à $\sigma A = A$ que si $\emptyset \equiv A$ et $n=0$.

Mais alors, pour tout $\rho \in S$, $\rho\sigma f = \lambda u. A \neq \lambda u. u = \sigma_2 f$. Ceci conclut la preuve que $\exists (\sigma \leq \sigma_1 \text{ et } \sigma \leq \sigma_2)$. \square

Nous allons donc devoir définir des ensembles complets de filtres, comme ensembles de solutions d'où découlent toutes les autres. Avant de préciser ces définitions, nous allons tout d'abord étudier les ensembles de solutions d'équations entre termes plus générales.

2.3. Unification

On appelle *unificateur* de deux termes E et E' de même type toute substitution σ telle que $\sigma E = \sigma E'$.

Nous nous intéressons à l'ensemble des unificateurs de E et E' :

$$\mathcal{U}(E, E') = \{\sigma \in S \mid \sigma E = \sigma E'\}$$

E et E' sont dits *unifiables* si et seulement si $\mathcal{U}(E, E') \neq \emptyset$.

Les unificateurs sont les solutions de l'équation $E = E'$, dans la structure T .

Lemme 2.10

Pour tous $E, E' \in T$, pour tout unificateur σ de E et E' :

- 1) $\forall V \subset V \quad (V[E] \cup V[E']) \subset V \implies \sigma \upharpoonright V \in \mathcal{U}(E, E')$
- 2) $\forall \rho \in S \quad \sigma \leq \rho \implies \rho \in \mathcal{U}(E, E')$.

Démonstration

Ces deux propriétés de fermeture de $\mathcal{U}(E, E')$ sont aisées à montrer

- 1) découle du lemme 2.4 , 2) de la définition de $\sigma \leq \rho$. \square

Définition :

On appelle *unificateur principal* (UP) de E et E' tout élément minimum de $U(E, E')$, c'est à dire toute substitution σ telle que :

- (i) $\sigma \in U(E, E')$
- (ii) $\forall \rho \in U(E, E') \quad \sigma \leq \rho$.

De même que pour le filtrage, il n'existe pas toujours d'unificateur principal :

Lemme 2.11

Il existe des termes de second ordre unifiables, qui n'admettent pas d'unificateur principal.

Démonstration

Il suffit de reprendre les termes considérés au lemme 2.9. E' étant un terme fermé : $V[E'] = \emptyset$, tout unificateur de E et E' est un filtre de E vers E' , et réciproquement.

Le résultat s'ensuit. \square

Nous verrons au chapitre 5 que, dans les langages de premier ordre, toute paire de termes unifiables possède un unificateur principal. Cette propriété est une propriété fondamentale de la logique de premier ordre, dont la première trace remonte à la thèse de Herbrand. Robinson [RJA1] a donné un algorithme d'unification qui, pour une paire quelconque de termes de premier ordre, détermine s'ils sont unifiables, et en cas de succès retourne un unificateur principal ("most general unifier"). Nous montrerons au chapitre 5 comment retrouver cette propriété à partir de considérations purement algébriques.

Remarque :

Relations entre le problème de l'unification et le problème de la recherche d'instances communes dans T .

Tout unificateur σ de E et E' détermine une instance commune $\sigma E = \sigma E'$ de E et de E' . Réciproquement, si E et E' sont séparés, i.e. si $V[E] \cap V[E'] = \emptyset$, alors le majorant commun $\sigma_1 E = \sigma_2 E'$ détermine l'unificateur $\sigma = (\sigma_1 \upharpoonright V[E] \cup \sigma_2 \upharpoonright V[E'])$.

Si les termes E et E' ne sont pas séparés, on peut rechercher leurs instances communes en unifiant E et E'' , où E'' est obtenu à partir de E' par une bijection renommant les variables communes en de nouvelles variables du même type : $E'' = \xi E'$. Tout unificateur de E et E'' est bien instance de E et E' . Réciproquement, soit l'instance commune $\sigma_1 E = \sigma_2 E'$. Comme $E' = \xi^{-1} E''$, la substitution $(\sigma_1 \upharpoonright V[E] \cup (\sigma_2 \xi^{-1}) \upharpoonright V[E''])$ est un unificateur de E et E'' . On peut donc trouver les instances communes par unification. Inversement, peut-on unifier E et E' si l'on sait déterminer les instances communes de deux termes quelconques ?

Soit $\{x_1, \dots, x_n\} = V[E] \cap V[E']$. Considérons les termes :

$$\begin{cases} E_1 = F_1(x_1, F_2(x_2, \dots, F_n(x_n, E) \dots)) \\ E_2 = F_1(x_1, F_2(x_2, \dots, F_n(x_n, E') \dots)) \end{cases}$$

où F_1, \dots, F_n sont des constantes de types convenables. Toute instance commune de E_1 et E_2 est de la forme :

$E_3 = F_1(x_1, F_2(x_2, \dots, F_n(x_n, e) \dots))$, avec $e = \sigma_1 E$, $e = \sigma_2 E'$, et $\forall i \leq n \quad \sigma_1 x_i = \sigma_2 x_i = x_i$.

On peut donc former $\sigma = (\sigma_1 \upharpoonright V[E] \cup \sigma_2 \upharpoonright V[E'])$, et par construction $\sigma \in U(E, E')$.

Dans tous les cas, à un problème d'unifiabilité correspond bien un problème de recherche d'instances communes, et réciproquement. Mais à plusieurs unificateurs distincts peut correspondre le même majorant, puisque le résultat d'un filtrage n'est pas nécessairement unique (et donc σ_1 et σ_2 ci-dessus ne sont pas uniquement déterminés). C'est ainsi qu'à un unificateur principal correspond un plus petit commun majorant des deux termes, mais la réciproque n'est pas vraie.

Par exemple, avec F et E' choisis comme dans la preuve du lemme 2.9, E et E' admettent A comme plus petit majorant, mais pourtant n'ont pas d'unificateur principal. La recherche d'unificateurs est donc en général un problème plus difficile que la recherche d'instances communes.

Gould [GW1], qui est le premier à avoir étudié le problème de l'unification en langages d'ordre supérieur, a malheureusement manqué de faire cette distinction importante.

Nous allons maintenant définir la notion d'ensemble complet d'unificateurs, comme ensemble de solutions d'où découlent toutes les autres. Mais auparavant nous avons besoin, pour des raisons essentiellement techniques, de raffiner l'ordre \leq entre substitutions. C'est l'objet du prochain paragraphe.

2.4. Le préordre \leq_V

Le préordre \leq dans S n'est pas assez fin pour nos besoins, car, contrairement au cas des termes de premier ordre, nous allons avoir besoin d'introduire de "nouvelles" variables (i.e. hors de $V[F] \cup V[E']$) afin de décrire les unificateurs de F et E' . Par exemple, considérons :

$$\begin{cases} F = f(A) \\ E' = f(B) \end{cases} \quad \text{avec} \quad \tau(f) = (\alpha \rightarrow \alpha), \quad \tau(A) = \tau(B) = \alpha.$$

Toute substitution d'une fonction constante à f , par exemple

$\rho = \{\langle f, \lambda u. C \rangle\}$, est un unificateur de E et E' .

Inversement, c'est la seule manière d'unifier E et E' . On aimeraît donc pouvoir dire que

$\sigma = \{\langle f, \lambda u. x \rangle\}$ est un UP de E et E' .

Pourtant, c'est faux, car par exemple avec $\eta = \{\langle x, C \rangle\}$, on a

$\eta\sigma = \rho \cup \eta \neq \rho$. Aucun autre η ne peut d'ailleurs convenir, car si $x \in D(\eta)$ alors $x \in D(\eta\sigma)$, et si $x \notin D(\eta)$ alors $\eta\sigma f = \sigma f \neq \rho f$. On a donc $\sigma \not\models \rho$.

Par contre, en prenant $\eta = \{\langle x, C \rangle\}$, il est vrai que $\rho e = \eta e$ pour tout terme e tel que $x \notin V[e]$, et c'est tout ce qui nous importe. C'est cette considération qui motive les définitions suivantes.

Définitions :

Soit V un ensemble fini de variables. On définit une équivalence \equiv_V dans S par :

$$\sigma \equiv_V \rho \iff \sigma \upharpoonright V = \rho \upharpoonright V.$$

On définit maintenant une relation \leq_V dans S par :

$$\sigma \leq_V \rho \iff \exists \eta \in S \quad \rho \equiv_V \eta\sigma.$$

On dit que σ schématisse ρ sur V .

Nous utiliserons V comme un ensemble de variables protégées, dans lequel on s'interdit de puiser les "nouvelles" variables. Par exemple, avec σ et ρ définis comme dans l'exemple ci-dessus, on a $\sigma \leq_V \rho$ pour tout V ne contenant pas x .

Remarquons que pour tout $e \in T$, pour tout $V \subset U$ tel que $U[e] \subset V$, on a :

$$\sigma \equiv_V \rho \implies \sigma e = \rho e.$$

Par contre, la réciproque n'est pas vraie. Par exemple, considérons $e = f(x)$ et $V = \{f, x\}$. Avec $\sigma = \{\langle f, \lambda u. u \rangle\}$ et $\rho = \{\langle f, \lambda u. x \rangle\}$, on a $\sigma e = \rho e = x$, mais pourtant on a $\sigma f \neq \rho f$.

Mais nous avons conservé la propriété qu'avait d'être compatible avec l'unification, sous la forme suivante :

Lemme 2.12

$$\forall E, E' \in T, \quad \forall V \supseteq V[E] \cup V[F'], \quad \forall \sigma \in U(E, E'), \quad \forall \rho \in S$$

$$\sigma \underset{V}{\leq} \rho \implies \rho \in U(E, E').$$

Démonstration

Directement à partir du lemme 2.4. \square

Lemme 2.13

$$\forall \sigma, \rho, V \quad \sigma \underset{V}{\equiv} \rho \implies \forall \eta \in S \quad \eta \sigma \underset{V}{\equiv} \eta \rho$$

Démonstration

$$\sigma \underset{V}{\equiv} \rho \quad \text{implique} \quad \forall x \in V \quad \sigma x = \rho x.$$

$$\eta \sigma \upharpoonright V = \{ \langle x, \eta e \rangle \mid \sigma x = e \text{ et } x \in V \} = \eta \rho \upharpoonright V. \quad \square$$

Remarquons, par contre qu'il est faux que

$$\sigma \underset{V}{\equiv} \rho \implies \sigma \eta \underset{V}{\equiv} \rho \eta$$

Par exemple, considérer $V = \{x\}$, $\sigma = \emptyset$, $\rho = \{\langle z, A \rangle\}$, $\eta = \{\langle x, z \rangle\}$.

La propriété correcte doit s'énoncer comme suit.

Lemme 2.14 $\forall v, \eta$ tels que $I(\eta) \subset v$:

$$\forall \sigma, \rho \quad \frac{\sigma = \rho}{v} \Rightarrow \frac{\sigma\eta = \rho\eta}{v}.$$

Démonstration

$$(i) \quad \forall x \in I(\eta) \quad \begin{aligned} \sigma\eta x &= (\sigma|_V)\eta x && \text{par le lemme 2.4} \\ &= (\rho|_V)\eta x && \text{par } \sigma \equiv_V \rho \\ &= \rho\eta x && \text{par le lemme 2.4} \end{aligned}$$

$$(ii) \quad \forall x \in V - I(\eta) \quad \begin{aligned} \sigma\eta x &= \sigma x && \text{par le lemme 2.5} \\ &= \rho x && \text{par } \sigma \equiv_V \rho \\ &= \rho\eta x && \text{par le lemme 2.5. } \square \end{aligned}$$

Lemme 2.15

La relation \leq_V est transitive.

Démonstration

Soit $\sigma_1 \leq_V \sigma_2 \leq_V \sigma_3$; $\sigma_3 \equiv_V \eta_2\sigma_2$ et $\sigma_2 \equiv_V \eta_1\sigma_1$.

On a $\sigma_3 \equiv_V \eta_2(\eta_1\sigma_1)$ par le lemme 2.13.

$\eta_2(\eta_1\sigma_1) = (\eta_2\eta_1)\sigma_1$ par le lemme 2.8.

d'où $\sigma_1 \leq_V \sigma_3$. \square

Remarque :

La relation \leq_V n'est pas antisymétrique, même sur le quotient par \equiv_V .

Par exemple, avec

$$\sigma = \{(x, y)\} \quad \rho = \{(x, y)\} \quad V = \{x\},$$

on a $\frac{\sigma \leq \rho}{V} \leq_V \sigma$ mais $\sigma \not\equiv_V \rho$. \leq_V ne définit donc qu'une structure de préordre sur S .

Définitions :

On dénote \equiv_V l'équivalence associée au préordre \leq_V :

$$\sigma \equiv_V \sigma' \Leftrightarrow \sigma \leq_V \sigma' \text{ et } \sigma' \leq_V \sigma.$$

On dit que σ et σ' sont *congrus modulo V*.

Enfin, on définit une relation \perp_V sur S par :

$$\begin{aligned} \sigma \perp_V \sigma' &\Leftrightarrow \exists \rho \quad \sigma \leq_V \rho \text{ et } \sigma' \leq_V \rho \\ &\Leftrightarrow \exists n, n' \quad n\sigma = n'\sigma' \end{aligned}$$

On dit que σ et σ' sont *indépendants sur V*.

2.5. Ensembles complets d'unificateursDéfinition :

Soit E, E' deux termes de même type, $L = V[E] \cup V[E']$, V un ensemble fini de variables tel que $L \subset V$. On appelle *ensemble complet d'unificateurs (ECU) de F et F'* sur V tout ensemble de substitutions Σ tel que :

- (1) $\Sigma \subset U(E, E')$ cohérence
- (2) $\forall \rho \in U(F, F') \quad \exists \sigma \in \Sigma \quad \sigma \leq_V \rho$ complétude
- (3) $\forall \sigma, \sigma' \in \Sigma \quad \begin{matrix} \sigma \neq \sigma' \\ \sigma \neq \sigma' \end{matrix} \quad L$ non-congruence

Remarquons que pour tous E, E' et V , il existe toujours un ECU :

il suffit de choisir un représentant dans chaque classe modulo \equiv_L de $U(E, E')$.

Si E et E' ne sont pas unifiables, alors \emptyset est le seul ECU, par 1).

Par contre, si $U(E, E') \neq \emptyset$, alors \emptyset n'est pas un ECU, par 2).

La condition (3) est une condition technique non essentielle. On pourrait exiger de Σ les conditions supplémentaires :

$$(4) \quad \forall \sigma \in \Sigma \quad D(\sigma) \subset L$$

$$(5) \quad \forall \sigma \in \Sigma \quad I(\sigma) \cap V = \emptyset.$$

Les ECU sont la généralisation à notre langage des UP du langage de logique de premier ordre.

On peut qualifier un ECU d'ensemble de solutions principales à l'équation $E = E'$, d'où découlent toutes les autres par composition. En effet, la condition de complétude et le lemme 2.12 entraînent que pour tout ECU Σ on a :

$$\mathcal{U}(E, E') = \{\rho \mid \forall \sigma \in \Sigma \quad \sigma \leq \rho\}.$$

Autrement dit, tout ECU est un système de générateur de l'idéal $\mathcal{U}(E, E')$.

Remarques sur l'ensemble V .

Pour tous V et V' tels que $L \subset V \subset V'$, on a

$$\sigma \leq_{V'} \sigma' \implies \sigma \leq_V \sigma'.$$

Tout ECU Σ de E et E' sur V' est donc aussi un ECU de E et E' sur V .

Réciproquement, soit Σ un ECU de E et E' sur L . On peut lui faire correspondre un ECU Σ' de E et E' sur V comme suit.

Soit $\bar{V} = V - L$, ξ une bijection de \bar{V} dans $V - V$, préservant les types.

Soit

$$\Sigma' = \{(\xi\sigma) \upharpoonright L \mid \sigma \in \Sigma\}. \text{ On vérifie :}$$

i) cohérence

$$[(\xi\sigma) \upharpoonright L]E = \xi\sigma E = \xi\sigma E' = [(\xi\sigma) \upharpoonright L]E'$$

ii) complétude

Soit $\rho \in \mathcal{U}(E, E')$, $\sigma \in \Sigma$ tel que $\exists n \quad \rho = n\sigma$.

Nous laissons au lecteur le soin de montrer que

$$\rho \equiv_{\bar{V}} n'\sigma, \text{ avec } n' = (n\xi^{-1}) \upharpoonright (V - \bar{V}) \cup \rho \upharpoonright \bar{V}.$$

iii) non congruence

$$\xi\sigma = n\xi\sigma' \implies \sigma = \xi^{-1}n\xi\sigma', \text{ d'où}$$

$$\sigma \not\equiv_L \sigma' \implies (\xi\sigma) \upharpoonright L \not\equiv_L (\xi\sigma') \upharpoonright L.$$

Σ' est donc bien un ECU de E et E' sur V .

Remarquons que si Σ vérifie les conditions (4) et (5) ci-dessus, alors $\Sigma' = \Sigma$. Les conditions (4) et (5) sont donc suffisantes pour qu'un ECU sur L soit aussi FCU sur V.

Nous utiliserons en général $V = L$. Mais pour certaines applications de nos algorithmes, nous voulons pouvoir spécifier des variables supplémentaires protégées, par exemple des variables libres dans un contexte contenant E ou E' . Si l'on veut par exemple définir une règle de résolution [RJA1] entre deux clauses C et C' , coupées suivant les littéraux F et E' , alors il faudra prendre $V = V[C] \cup V[C']$.

Gould [GW1] avait défini la notion d'ensemble complet d'instances. Cette notion n'est malheureusement pas suffisante pour définir une règle de résolution. Par exemple, avec

$$\begin{aligned} C &= P(f(A)) \vee Q(f(B)) \\ \text{et} \quad C' &= \neg P(A) \end{aligned}$$

on veut pouvoir engendrer les deux résolvantes $Q(B)$ et $Q(A)$, bien que $f(A)$ et A aient une instance commune unique A .

Plotkin [PG1] utilise, pour des langages de premier ordre munis d'une relation d'équivalence sur les termes définie par une théorie équationnelle, une notion d'ensemble d'unificateurs les plus généraux. Cette notion est très voisine de celle d'ensemble complet d'unificateurs minimaux, que nous introduirons en 2.7.

Jensen et Pietrzykowski [JP1] ont défini, pour un λ -calcul typé avec η -conversion, la notion d'ensemble général d'unificateurs. Un tel ensemble Σ vérifie deux propriétés :

- 1) cohérence
- 2) $\forall \rho \in \mathcal{U}(F, E') \quad \exists \sigma \in \Sigma \quad \exists \eta \in S \quad \rho \subset \eta\sigma.$

Mais la condition 2) n'est pas une condition de complétude suffisante, comme le montre le contre exemple suivant :

$$\left\{ \begin{array}{l} E = f(x) \\ E' = A \end{array} \right. \quad \text{avec} \quad \left\{ \begin{array}{l} \tau(x) = \tau(A) = \alpha \\ \tau(f) = (\alpha \rightarrow \alpha) \end{array} \right.$$

Soit $\sigma_1 = \{\langle f, \lambda u \cdot A \rangle, \langle x, B \rangle\}$ et $\sigma_2 = \{\langle f, \lambda u \cdot u \rangle, \langle x, A \rangle\}$ avec $\tau(u) = \tau(B) = \alpha$.

Alors $\{\sigma_1, \sigma_2\}$ est un ensemble général d'unificateurs de E et E' . Pourtant $\sigma_3 = \{\langle f, \lambda u \cdot A \rangle\}$ est un unificateur qu'on ne peut engendrer par composition avec σ_1 ou σ_2 .

Définition :

On appelle *unificateur V-principal* de E et E' tout unificateur σ tel que $\{\sigma\}$ est un ECU de E et E' sur V . C'est donc un élément minimum de $U(E, E')$, relativement à \leq_V .

Exemple :

Soit $\left\{ \begin{array}{l} E = x \\ E' = y \end{array} \right.$ et $V = \{x, y, z\}$ avec $\tau(x) = \tau(y) = \tau(z) = \alpha$.

Alors $\{\langle x, y \rangle\}, \{\langle y, x \rangle\}, \{\langle x, u \rangle, \langle y, u \rangle\}$ et $\{\langle x, y \rangle, \langle z, x \rangle\}$ sont des unificateurs *V-principaux*. Par contre $\{\langle x, z \rangle, \langle y, z \rangle\}$ n'est pas *V-principal* non plus que $\{\langle x, y \rangle, \langle z, y \rangle\}$.

Remarquons que le lemme 2.11 est toujours vrai, en remplaçant principal par *V-principal*. Les ECU devront donc en général contenir plus d'un élément. La première question qu'on peut alors se poser concerne leur finitude : si deux termes E et E' sont unifiables, possèdent-ils toujours un ECU fini ? Nous répondront négativement à cette question dans le prochain paragraphe.

2.6. Finitude des ensembles d'unificateurs

Théorème 4 : Il existe des termes de second ordre E et E' , unifiables, et tels que, pour tout V tel que $L = V[F] \cup V[F'] \subset V$, tout ECU de F et E' sur V est infini.

Démonstration

$$\text{Soit } \begin{cases} E = f(F(A)) \\ E' = F(f(A)) \end{cases} \quad \text{avec} \quad \begin{cases} \tau(A) = \alpha \\ \tau(f) = \tau(F) = (\alpha \rightarrow \alpha) \end{cases}$$

Il suffit de prendre $V = L = \{f\}$.

Considérons $\Sigma = \{\{<f, F>\}\} \cup \{\{<f, \lambda u.F^n(u)>\} \mid n \geq 0\}$

$$\text{où } F^0(u) = u \text{ et } F^n(u) = F(F^{n-1}(u)).$$

Montrons tout d'abord que Σ est un ECU de E et E' sur V .

La condition de cohérence est facile, on vérifie que $\sigma E = \sigma E'$ pour tout σ de Σ . Pour la condition de complétude, soit $\rho \in U(E, E')$, $e = \rho f$, $\bar{\rho} = \rho - \{<f, e>\}$.

Cas 1 : $\bar{\rho}[e] = 0$. Alors $\rho E = e(F(A))$, $\rho E' = F(e(A))$ ne peuvent être égaux que si $e = F$; dans ce cas on a bien $\rho = \bar{\rho} \{<f, F>\}$

Cas 2 : $\bar{\rho}[e] > 0$. A cause du type de f , l'arité de e ne peut dépasser 1 : $e = \lambda u.e'$, avec $\tau(u) = \tau(e') = \alpha$.

Soit p le plus grand entier tel que $e' = F^p(e'')$, avec $\tau(e'') = \alpha$.

$$\text{Alors } \rho E = F^p(S_{F(A)}^u[e''])$$

et $\rho E' = F^{p+1}(S_A^u[e''])$. Par cas sur e'' , il est facile de montrer que $\rho E = \rho E'$ impose $e'' = u$, et alors on a bien $\rho = \bar{\rho} \{<f, \lambda u.F^p(u)>\}$.

Enfin, pour tous σ et σ' distincts dans Σ , il n'existe pas de n tel que $\sigma' \models_V^n \sigma$ puisque $I(\sigma) = \emptyset$. Donc

$$\forall \sigma, \sigma' \in \Sigma \quad \sigma \not\models_V^{\sigma'} \quad (1)$$

Ceci permet donc de conclure que Σ est un ECU de E et E' sur V .

Soit maintenant Σ' un ECU fini de F et F' sur V . Σ étant infini :

$\exists \sigma, \sigma' \in \Sigma, \sigma \neq \sigma'$ tels que $\exists \rho \in \Sigma' \rho \underset{V}{\leq} \sigma$ et $\rho \underset{V}{\leq} \sigma'$.

Mais, Σ étant un ECU : $\exists \sigma'' \in \Sigma \sigma'' \underset{V}{\leq} \rho$.

Par transitivité de $\underset{V}{\leq}$:

$$\sigma'' \underset{V}{\leq} \sigma \quad (2)$$

$$\text{et : } \sigma'' \underset{V}{\leq} \sigma' \quad (3):$$

Si $\sigma'' \neq \sigma$ (2) contredit (1), sinon (3) contredit (1). Ceci montre donc qu'il ne peut exister d'ECU fini de E et E' . \square

Remarquons que le théorème 4 reste vrai, même si on se limite à des symboles de fonction monadiques. Ce cas correspond à la résolution d'équations dans un monoïde libre, comme nous le verrons au chapitre 6 (remarquer qu'on peut décrire Σ ci-dessus par $\{\langle f, F^* \rangle\}$).

2.7. Redondance des ensembles d'unificateurs

Bien que l'on doive parfois considérer des ECU infinis, on peut se demander si on peut au moins imposer aux ECU une condition de non-redondance plus forte que la condition (3) de non-congruence. Nous aimerions remplacer dans la définition d'ECU la condition (3) par la condition de minimalité :

$$\forall \sigma, \sigma' \in \Sigma \quad \underset{\sigma \neq \sigma'}{\sigma \notin L} \quad \underset{L}{\sigma \neq \sigma'}$$

Définition :

Soit E, E' deux termes de même type, V un ensemble fini de variables tel que $L = V[E] \cup V[E'] \subset V$.

On appelle ensemble complet d'unificateurs minimaux (ECUM) de E et E' sur V tout ensemble de substitutions Σ tel que :

- (1) $\Sigma \subset U(E, E')$ cohérence
- (2) $\forall \rho \in U(E, E') \quad \exists \sigma \in \Sigma \quad \sigma \leq_V \rho$ complétude
- (3) $\forall \sigma, \sigma' \in \Sigma \quad \begin{matrix} \sigma \not\leq_L \sigma' \\ \sigma \neq \sigma' \end{matrix}$ minimalité.

Un ECUM peut être qualifié d'ensemble de solutions principales indépendantes de l'équation $F = F'$. En considérant ses éléments comme des générateurs minimaux, on peut dire qu'un ECUM est une base de $U(E, E')$.

Lemme 2.16.

S'il existe un FCUM de E et E' sur V il est unique, à un isomorphisme près. Plus exactement, entre deux tels ECUM Σ_1 et Σ_2 il existe une bijection

$$\phi : \Sigma_1 \rightarrow \Sigma_2 \quad \text{telle que} \quad \forall \sigma \in \Sigma_1 \quad \sigma \equiv_V \phi(\sigma).$$

Démonstration :

Considérons deux ECUM Σ_1 et Σ_2 . Pour tout ρ dans Σ_1 , on choisit $\phi(\rho)$ dans Σ_2 comme un σ tel que $\sigma \leq_V \rho$.

Pour tout ρ dans Σ_2 , on définit $\psi(\rho)$ dans Σ_1 , comme un σ tel que $\sigma \leq_V \rho$. Par transitivité on a

$$\forall \sigma \in \Sigma_1 \quad \psi(\phi(\sigma)) \leq_V \sigma \quad \text{qui impose par (3) :}$$

$$\psi(\phi(\sigma)) = \sigma.$$

Donc $\phi(\sigma) \leq_V \sigma \leq_V \psi(\phi(\sigma))$, ce qui entraîne $\sigma \equiv_V \phi(\sigma)$. \square

Remarques :

1) Si $L \subset V \subset V'$, alors tout FCUM sur V' est aussi un ECUM sur V . C'est pour conserver cette propriété que nous imposons \leq_L dans (3) plutôt que \leq_V .

Comme pour les FCU, si on impose de plus :

$$(4) \quad \forall \sigma \in \Sigma \quad D(\sigma) \subset L$$

$$(5) \quad \forall \sigma \in \Sigma \quad I(\sigma) \cap V = \emptyset,$$

alors on peut remplacer dans (2) \leq_V par \leq_L .

La définition d'ECUM est alors très proche de celle d'ensemble d'unificateurs les plus généraux de Plotkin [PG1]. Mais dans sa définition, la condition (5) est remplacée par une condition plus faible :

$$(6) \quad V' \cap V = \emptyset$$

avec $V' = V[\sigma E] \cup V[\sigma E']$. Cette condition n'est pas suffisante pour assurer la complétude, lorsque l'inclusion $V' \subset (L \cup I(\sigma))$ est stricte.

Par exemple,

Soit $\left\{ \begin{array}{l} E = f(x) \\ E' = A \\ L = \{x, f\} \\ V = \{x, f, y\} \end{array} \right.$ avec $\left\{ \begin{array}{l} \tau(x) = \tau(y) = \tau(A) = \alpha \\ \tau(f) = (\alpha \rightarrow \alpha). \end{array} \right.$

Considérons $\sigma_1 = \{\langle f, \lambda u \cdot A \rangle, \langle x, y \rangle\}$

et $\sigma_2 = \{\langle f, \lambda u \cdot u \rangle, \langle x, A \rangle\}$

$\{\sigma_1, \sigma_2\}$ est un ECUM de E et E' sur L qui satisfait (4) et (6).

Pourtant ce n'est pas un ECUM de E et E' sur V comme on peut le constater en considérant l'unificateur

$$\rho = \{\langle f, \lambda u \cdot A \rangle, \langle x, B \rangle, \langle y, C \rangle\}.$$

2) Soit $U = \{\sigma \in \mathcal{U}(E, E') \mid D(\sigma) \subset L \text{ et } I(\sigma) \cap V = \emptyset\}$, M l'ensemble des éléments de U minimaux par rapport à l'ordre induit par \leq_L . Si U est bien fondé, alors M est l'ECUM de E et E' . Plus généralement, disons que U est *inductif* si toute chaîne décroissante dans U , relativement à \leq_L , admet un minorant.

Si U est inductif, alors M est l'ECUM de E et E' , en vertu du lemme de Zorn.

Par contre, si U n'est pas inductif, alors E et E' peuvent ne pas admettre d'ECUM. Nous allons voir que cette situation peut en fait se produire ici.

Théorème 5 Certaines paires de termes d'ordre 3 unifiables ne possèdent pas d'ECUM.

Démonstration

$$\text{Soit } \begin{cases} F = f(x, A) \\ F' = f(x, B) \\ L = \{x, f\} \end{cases} \quad \text{avec} \quad \begin{cases} \tau(A) = \tau(B) = \alpha \\ \tau(x) = (\alpha \rightarrow \alpha) \\ \tau(f) = ((\alpha \rightarrow \alpha) \times \alpha \rightarrow \alpha) \end{cases}$$

Il suffit de prendre $V = L$.

Considérons :

$$\begin{aligned} \rho &= \{<f, \lambda uv. h(u)>\} \\ \sigma_0 &= \{<f, \lambda u. u>, <x, \lambda v. z>\} \\ \text{et } \sigma_n &= \{<f, \lambda uv. g_n(u, u(h_1^n(u, v)), \dots, u(h_n^n(u, v)))>, <x, \lambda v. z>\} \quad (n > 0), \\ \text{avec } & \begin{cases} \tau(v) = \tau(z) = \alpha, \quad \tau(u) = (\alpha \rightarrow \alpha), \\ \tau(h) = ((\alpha \rightarrow \alpha) \rightarrow \alpha), \quad \tau(h_i^j) = ((\alpha \rightarrow \alpha) \times \alpha \rightarrow \alpha), \\ \tau(g_i) = ((\alpha \rightarrow \alpha) \times \underbrace{\alpha \times \dots \times \alpha}_{i} \rightarrow \alpha) \end{cases} \\ \text{et soit } \Sigma &= \{\sigma_i \mid i \geq 0\} \cup \{\rho\}. \end{aligned}$$

La proposition suit d'un certain nombre de lemmes, comme le théorème 4.

La première étape consiste à montrer que Σ est un ECU de E et E' sur V .

Tout d'abord, on vérifie que pour tout $\sigma \in \Sigma$, $\sigma E = \sigma E'$, d'où la condition de cohérence. La preuve se poursuit en trois parties A, B et C.

A) Complétude de Σ

Soit maintenant σ un unificateur quelconque de E et E' , $e_f = \sigma f$ et $e_x = \sigma x$. On montre par cas sur e_f qu'il existe dans Σ une substitution plus générale que σ . D'après le type de f , $\bar{H}[e_f] \leq 2$, d'où trois cas.

(1) $\bar{H}[e_f] = 0$. Alors $\sigma E = e_f(e_x, A) \neq e_f(e_x, B) = \sigma E'$, et ce cas ne peut donc se produire.

(2) $\bar{H}[e_f] = 1$, i.e., $e_f = \lambda u.e$ avec $\bar{H}[e] = 0$.

Soit $e' = N[S_{e_x}^u[e]]$. On a :

$$\begin{cases} \sigma E = e'(A) \\ \sigma E' = e'(B) \end{cases}$$

$\sigma E = \sigma E'$ implique (par cas sur e') que

$$e' = \lambda v.e'' \text{ avec } v \notin V[e''].$$

On doit donc résoudre $S_{e_x}^u[e] = \lambda v.e''$. Par cas sur $e = y(e_1, \dots, e_n)$: (rappelons que $\bar{H}[e] = 0$)

(i) $y = u$; alors à cause des types $n = 0$,

$$e' = e_x = \lambda v.e'' \text{ , et } \sigma = \{\langle z, e'' \rangle\} \sigma_0.$$

(ii) $y \neq u$; alors

$$e' = y(\dots, S_{e_x}^u[e_i], \dots) \neq \lambda v.e'' \text{ et ce cas est donc impossible.}$$

(3) $\bar{H}[e_f] = 2$, i.e. $e_f = \lambda u v . e$. La preuve se poursuit sur la structure de e .

(i) $v \notin V[e]$; alors $\sigma \equiv_V \{\langle h, \lambda u . e \rangle, \langle x, e_x \rangle\} \rho$.

(ii) Si e possède une occurrence libre de v non contenue dans un argument de u , alors on prouve que, pour tout terme e_x , $v \in V[e']$, avec $e' = N[S_{e_x}^u[e]]$.

Soit $e = \lambda w_1 \dots w_n . @ (e_1, \dots, e_p)$.

Si $@ = v$ (avec $p=0$) alors c'est trivial. Sinon, $@ \neq u$ par hypothèse, et on a donc :

$S_{e_x}^u[e] = \lambda w_1 \dots w_n . @ (\dots, S_{e_x}^u[e_i], \dots)$. Comme au moins un des e_i doit satisfaire à l'hypothèse de récurrence, on obtient aisément le résultat.

On a alors :

$\sigma(f(x, A)) = S_A^V[e'] \neq S_B^V[e'] = \sigma(f(x, B))$, ce qui est contraire à l'hypothèse. Ce cas ne peut donc pas se produire.

(iii) Sinon on peut écrire e sous la forme :

$e = E < u, u, \dots, u, u(e_1), u(e_2), \dots, u(e_n) >$

où toutes les occurrences de u et v apparaissent entre

les guillemets. Comme $v \in V[e]$, cela impose $\exists k \leq n \quad v \in V[e_k]$.

Dans cette notation, nous supposons que les occurrences isolées de u n'apparaissent pas dans la partie gauche d'une application (auquel cas on aurait explicité le terme $u(e_i)$ correspondant).

Maintenant, pas cas sur e_x :

a) $e_x = \lambda v . e'$, où $v \notin V[e']$.

Alors $\sigma \equiv_V \eta \sigma_n$ avec

$\eta = \{ \langle g_n, \lambda u w_1 \dots w_n . E < u, u, \dots, u, w_1, w_2, \dots, w_n > >, \langle z, e' \rangle \} \cup \{ \langle h_i^n, \lambda u v . e_i \rangle \mid 1 \leq i \leq n \}$.

b) Dans tous les autres cas, il est facile de montrer que $v \in V[N[S_{e_x}^u [u(e_k)]]]$, et donc aussi que $v \in V[N[S_{e_x}^u [e]]]$. On en conclut comme en (ii) que ce cas ne peut pas se produire.

Ceci termine la preuve de la complétude de Σ . Il nous reste à montrer la non-congruence. En fait nous allons étudier en détail la structure de Σ vis à vis de la relation \leq_V .

B) \leq_V n'est pas un ordre bien fondé sur Σ

$$(1) \quad \forall i > 0 \quad \sigma_{i+1} \leq_V \sigma_i$$

En effet, soit

$$\eta_i = \{<g_{i+1}, \lambda u v_1 v_2 \dots v_{i+1} \cdot g_i(u, v_1, \dots, v_i)>\} \cup \{<h_j^{i+1}, h_j^i> \mid 1 \leq j \leq i\}.$$

On a bien $\sigma_i \leq_V \eta_i \leq_V \sigma_{i+1}$.

$$(2) \quad \forall i \geq 0 \quad \sigma_i \not\leq_V \rho.$$

En effet, supposons qu'il existe n tel que $\rho \leq_V^n \sigma_i$. On devrait avoir $x = n(\lambda v.z) = \lambda v'.h(z)$, ce qui est impossible.

$$(3) \quad \rho \not\leq_V \sigma_1$$

Supposons qu'il existe n tel que $\sigma_1 \leq_V^n \rho$. On devrait avoir $\sigma_1 f = \eta(\lambda u v \cdot h(u)) = \lambda u v \cdot g_1(u(h_1^1(u, v)))$, ce qui est impossible, car $\eta(\lambda u v \cdot h(u)) = \lambda u' v' \cdot e$, où $v' \notin V[e]$.

$$(4) \quad \forall i > 0 \quad \sigma_i \not\leq_V \sigma_0.$$

De la même manière, si $\sigma_0 \leq_V^n \sigma_i$, on devrait avoir

$$\sigma_0 f = \lambda u \cdot u = \eta(\lambda u v \cdot g_i(\dots)) = \lambda u' v' \cdot e, \text{ ce qui est impossible.}$$

(5) $\sigma_0 \nleq_V \sigma_1$. De même, si $\sigma_1 = n\sigma_0$, on devrait avoir

$\sigma_1 f = n(\lambda u \cdot u) = \lambda u \cdot u$, ce qui est impossible.

(6) $\forall i > 0 \quad \sigma_0 \nleq_V \sigma_i \text{ et } \rho \nleq_V \sigma_i$.

Conséquence immédiate de (1), (3), (5) et de la transitivité de \leq_V .

(7) $\rho \nleq_V \sigma_0$.

Preuve comme pour (4).

(8) $\forall i > 0 \quad \sigma_i \nleq_V \sigma_{i+1}$

Cette preuve est assez longue, la difficulté étant essentiellement notationnelle. Nous allons montrer que $\sigma_1 \nleq_V \sigma_2$; la preuve générale est similaire.

Supposons que $\sigma_2 \equiv_V n\sigma_1$. Par le lemme 2.13 on doit avoir, pour toute substitution ξ : $\xi \sigma_2 \equiv_V \xi n \sigma_1$. Choisissons

$$\xi = \{\langle g_2, G \rangle, \langle h_1^2, \lambda u v \cdot C(v) \rangle, \langle h_2^2, \lambda u v \cdot D(v) \rangle\}$$

avec $\tau(G) = \tau(g_2)$, $\tau(C) = \tau(D) = (\alpha \rightarrow \alpha)$

et soit $\xi' = \xi n$. On doit avoir, en particulier pour f :

$$\xi'(\lambda u v \cdot g_1(u, u(h_1^1(u, v)))) = \lambda u v \cdot G(u, u(C(v)), u(D(v))).$$

Soit $e = \xi' g_1, e' = \xi' (h_1^1(u, v))$. On peut supposer que ni u ni v n'apparaissent libres dans $\xi' g_1$ et $\xi' h_1^1$. On doit donc avoir $C(u, u(C(v)), u(D(v)))$ comme forme normale de $e(u, u(e'))$ avec u et v non libres dans e . Une étude par cas sur la structure de e montre qu'on doit avoir $e = \lambda w_1 w_2 \cdot G(e_1, e_2, e_3)$.

Maintenant, par cas sur e_2 , on réduit à deux possibilités :

i) $e_2 = w_1(C(\bar{e}_2))$. Dans ce cas, on doit avoir $v = S_u^{w_1}[S_{u(e')}^{w_2}[\bar{e}_2]]$, avec $v \notin V[\bar{e}_2]$ ce qui est clairement impossible.

ii) $e_2 = w_2$. On doit alors avoir $e' = C(v)$, et par le même raisonnement appliqué à $e_3, e' = D(v)$, ce qui est impossible. Ceci complète la preuve de $\sigma_1 \not\leq_V \sigma_2$.

Nous venons de prouver que Σ a la structure suivante :

$$\rho, \sigma_0, \sigma_1 > \sigma_2 > \dots > \sigma_n > \dots$$

où $\sigma > \sigma'$ dénote $\sigma \leq_V \sigma'$ et $\sigma \not\leq_V \sigma'$. En particulier, l'ordre induit par \leq_V n'est pas bien fondé sur Σ .

En corollaire de (4), (6), (7) et (8) on obtient la condition de non-congruence dans Σ , ce qui achève de prouver que Σ est un ECU de E et E' sur V .

C) Soit maintenant Σ' un ECUM de E et E' sur V .

Σ' étant un F CU, $\exists \mu \in \Sigma' : \mu \leq_V \sigma_1$. Mais, Σ étant un ECU, $\exists \sigma \in \Sigma \quad \sigma \leq_V \mu$.

Par transitivité $\sigma \leq_V \sigma_1$, et donc $\sigma \neq \rho, \sigma \neq \sigma_0$ par (3) et (5).

Donc $\exists k \geq 1 : \sigma = \sigma_k$. Maintenant soit $\mu' \in \Sigma'$ tel que $\mu' \leq_V \sigma_{k+1}$ (μ' existe car Σ' est un F CU).

$\sigma_{k+1} \leq_V \sigma_k$ par (1) et donc $\mu' \leq_V \mu$ par transitivité. Σ' étant un ECUM, cela implique $\mu = \mu'$ et maintenant par transitivité on obtient $\sigma_k \leq_V \sigma_{k+1}$. Ceci n'est pas possible, par (8), ce qui contredit l'existence de Σ' .

Ceci achève la preuve du théorème 5. \square

Remarques :

1) Cas du second ordre.

Nous conjecturons que $U(E, E')$ muni de \leq est inductif au 2nd ordre, ce qui entraînerait l'existence de l'ECUM de deux termes unifiables.

2) L'examen de la preuve précédente nous montre que l'ensemble des termes

$e_n = \sigma_n f$ forme une suite infinie décroissante, donc que la structure $\langle T, \leq \rangle$ n'est pas bien fondée, même si on se restreint à l'ordre 3.

Nous verrons au chapitre 5 qu'on a un ordre bien fondé au 1er ordre et au chapitre 6 qu'il y a par contre des chaînes infinies décroissantes de termes de second ordre.

2.8. Indécidabilité de l'unification

Nous montrons finalement que l'existence d'unificateur pour deux termes de notre langage est un problème indécidable, dès l'ordre 3.

Théorème 6. Il est indécidable si deux termes quelconques d'ordre 3 sont unifiables.

Démonstration

Soit l'alphabet de deux variables $\Delta = \{u, v\}$, avec $\tau(u) = \tau(v) = (\alpha \rightarrow \alpha)$.
A tout mot $\xi \in \Delta^*$ on associe le terme $\tilde{\xi} \in T_{(\alpha \rightarrow \alpha)}$ comme suit :

$$\begin{cases} \tilde{\Lambda} = \lambda z \cdot z \\ \tilde{u\xi} = \lambda z \cdot u(\tilde{\xi}(z)) & \text{avec } \tau(z) = \alpha, \Lambda \text{ est le mot vide de } \Delta^*. \\ \tilde{v\xi} = \lambda z \cdot v(\tilde{\xi}(z)) \end{cases}$$

A tout problème de Post $\begin{pmatrix} x_1 x_2 \cdots x_n \\ y_1 y_2 \cdots y_n \end{pmatrix}$ sur Δ on associe le

problème de l'existence d'un unificateur pour le couple :

$$\begin{cases} E = \lambda u v w \cdot w(f(\tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_n), f(u, u, \dots, u)) \\ E' = \lambda u v w \cdot w(f(\tilde{y}_1, \tilde{y}_2, \dots, \tilde{y}_n), u(g(u))) \end{cases}$$

où

$$\left\{ \begin{array}{l} \tau(f) = ((\alpha \rightarrow \alpha) \times (\alpha \rightarrow \alpha) \times \cdots \times (\alpha \rightarrow \alpha)) \rightarrow \alpha \\ \tau(g) = ((\alpha \rightarrow \alpha) \rightarrow \alpha) \\ \tau(w) = (\alpha \times \alpha \rightarrow \alpha) \end{array} \right.$$

Nous allons montrer que ces deux problèmes se réduisent l'un à l'autre.

2.8.1.

Supposons qu'il existe une solution

$$i_1, i_2, \dots, i_p \quad p \geq 1$$

au problème de Post. Considérons

$$\sigma = \{\langle f, \lambda w_1 w_2 \cdots w_n \cdot w_{i_1} (w_{i_2} (\cdots (w_{i_p} (z)) \cdots)) \rangle, \\ \langle g, \lambda u \cdot u \underbrace{(\cdots (u(z)) \cdots)}_{p-1} \rangle\}.$$

Si l'on dénote $S = X_{i_1} X_{i_2} \cdots X_{i_p} = Y_{i_1} Y_{i_2} \cdots Y_{i_p}$,
on vérifie que :

$\sigma E = \sigma E' = \lambda u v w \cdot w(S(z), u^P(z))$, et σ est donc un unificateur de E et E' .

2.8.2.

Réciproquement, soit σ un unificateur quelconque de E et E' . Soit $e_f = \sigma f$, $e_g = \sigma g$. On suppose, sans perte de généralité, que ni e_f ni e_g ne contiennent u ou v .

On a : $\sigma E = \sigma E' = \lambda u v w \cdot w(e_1, e_2)$

avec $\begin{cases} e_1 = e_f(\tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_n) = e_f(\tilde{y}_1, \tilde{y}_2, \dots, \tilde{y}_n) \\ e_2 = e_f(u, u, \dots, u) = u(e_g(u)). \end{cases}$

Ignorons e_2 pour l'instant (qui est là seulement pour éliminer la solution triviale $p=0$ du problème de Post) et considérons la structure de e_1 , par cas suivant la structure de e_f .

Soit $e_f = \lambda w_1 \cdots w_k \cdot @(\cdots)$, sous forme normale.

Comme $\tau(e_f) = \tau(f)$, on doit avoir $k \leq n$, et

$\tau(@) = (\gamma_1, \gamma_2, \dots, \gamma_m, \underbrace{(\alpha \rightarrow \alpha), \dots, (\alpha \rightarrow \alpha)}_{n-k} \rightarrow \alpha)$ avec $m \geq 0$, $\gamma_1, \dots, \gamma_m \in T$.

Deux cas sont possibles :

2.8.1.1. $0 \leq k < n$

A cause de son type, $@$ ne peut être de la forme w_j . e_1 admet donc une forme normale :

$$@ (A_1, A_2, \dots, A_m, \tilde{x}_{k+1}, \dots, \tilde{x}_n)$$

et par symétrie on doit donc avoir

$$\tilde{x}_j = \tilde{y}_j \quad k < j \leq n,$$

et donc $x_j = y_j$, puisque l'opération \sim est une bijection. Dans ce cas le problème de Post admet comme solution n'importe quelle séquence d'entiers compris entre $k+1$ et n .

2.8.2.2. k=n

Soit p le plus grand entier ≥ 0 tel que

$$e_f = \lambda w_1 \cdots w_n \cdot w_{i_1} (w_{i_2} (\cdots (w_{i_p} (e')) \cdots)), \text{ avec } 1 \leq i_1, \dots, i_p \leq n.$$

Maintenant e_1 admet comme forme normale :

$$\tilde{x}_{i_1} (\tilde{x}_{i_2} (\cdots (\tilde{x}_{i_p} (e'')) \cdots)) = \tilde{y}_{i_1} (\tilde{y}_{i_2} (\cdots (\tilde{y}_{i_p} (e'')) \cdots)).$$

Il est clair que e'' ne commence pas par un u ou un v , par définition de p . Le premier symbole e'' est donc le premier symbole de e_1 qui ne soit ni un u ni un v . Par symétrie c'est aussi le premier symbole de e''' . En supprimant les parenthèses de la partie initiale de e_1 , on obtient donc

$$x_{i_1} x_{i_2} \cdots x_{i_p} = y_{i_1} y_{i_2} \cdots y_{i_p}.$$

Il nous reste à vérifier que $p > 0$, et pour cela nous considérons e_2 .

Si $p=0$, alors par le même raisonnement la forme normale de $e_f(u, \dots, u)$ ne peut commencer par un u , ce qui est contradictoire avec le fait qu'elle est aussi la forme normale de $u(e_g(u))$. Nous avons donc $p > 0$, et i_1, i_2, \dots, i_p est solution du problème de Post correspondant au couple E, E' .

2.8.1 et 2.8.2 prouvent l'interréductibilité du problème de Post et d'un sous problème de l'unification à l'ordre trois, ce qui prouve bien l'indécidabilité de ce dernier problème. \square

Corollaire : Il est indécidable si deux termes quelconques d'ordre 3 ont une instance commune.

Démonstration

Découle de la remarque à la fin du paragraphe 2.3 : il suffit de considérer l'exemple ci-dessus, où l'on ajoute à w un troisième argument égal à f pour E et F' . \square

Remarque : Pour le second ordre, le problème est encore ouvert.

Nous y reviendrons au chapitre 6.

Le théorème 6 a été utilisé par P. Andrews [AP2] pour montrer l'indécidabilité de la provabilité de certaines classes de formules en théorie des types.

2.9. Ensembles complets de filtres , demi-unification

Nous revenons maintenant au problème du filtrage.

Rappelons que nous cherchons les solutions σ de $\sigma E = E'$.

Tout d'abord, remarquons que nous avons besoin de variables auxiliaires pour décrire des filtres, de même que pour l'unification.

Par exemple, soit

$$\begin{cases} E = f(\lambda u \cdot A) & \text{avec} \\ E' = A & \end{cases} \quad \begin{cases} \tau(u) = \tau(A) = \alpha \\ \tau(f) = ((\alpha \rightarrow \alpha) + \alpha). \end{cases}$$

On veut pouvoir exprimer que l'ensemble constitué des deux substitutions :

$$\sigma_1 = \{<f, \lambda w \cdot w(h(w))>\}$$

et $\sigma_2 = \{<f, \lambda w \cdot A>\}$ constitue un ensemble complet de filtres de E vers E' . Nous allons donc utiliser ici aussi la relation \leq_V , où V doit contenir toutes les variables libres de E et de E' .

Définition :

Soit E, E' deux termes de même type, $L = V[E] \cup V[E']$, V un ensemble fini de variables tel que $L \subset V$.

On appelle *ensemble complet de filtres de E vers E'* sur V (ECF) tout ensemble de substitutions Σ tel que :

- (1) $\Sigma \subset F(E, E')$ cohérence
- (2) $\forall \rho \in F(F, E') \quad \exists \sigma \in \Sigma \quad \sigma \leq_V \rho$ complétude
- (3) $\forall \sigma, \sigma' \in \Sigma \quad \begin{matrix} \sigma \neq \sigma' \\ L \end{matrix} \quad \text{non-congruence}$

Nous n'allons pas étudier les ECF davantage, car cette notion soulève un problème difficile, lorsque $V[E'] \neq \emptyset$.

Considérons par exemple les termes de premier ordre :

$$\left\{ \begin{array}{l} E = y \\ E' = A(x) \end{array} \right. \quad V = L = \{x, y\}$$

La substitution $\sigma = \{<y, A(x)>\}$ est un filtre de E vers E' .

Pourtant ce n'est pas un filtre principal, car par exemple

$\sigma' = \{<y, A(x)>, <x, B>\}$ est un filtre tel que $\sigma \not\leq_V \sigma'$. Les filtres principaux de E vers E' sont les substitutions $\{<y, A(x)>, <x, z>\}$ avec $z \notin V$, ce qui semble peu naturel. De plus ces filtres principaux n'ont pas la propriété, que possédait σ , de laisser F' invariant.

Ce problème peut être résolu simplement au 1er ordre, en imposant aux filtres σ de E vers E' de posséder la propriété supplémentaire : $\mathcal{D}(\sigma) \subset V[E]$. Malheureusement cela ne suffit pas au second ordre. Considérons par exemple :

$$\left\{ \begin{array}{l} E = f(x) \\ E' = A(x) \end{array} \right. \quad \text{avec} \quad \left\{ \begin{array}{l} \tau(x) = \alpha \\ \tau(f) = \tau(A) = (\alpha \rightarrow \alpha) \end{array} \right.$$

et soit :

$$\left\{ \begin{array}{l} \sigma_1 = \{<f, A>\} \\ \sigma_2 = \{<f, \lambda u \cdot A(u)>\} \\ \sigma_3 = \{<f, \lambda u \cdot A(x)>, <x, y>\} \end{array} \right.$$

$\{\sigma_1, \sigma_2, \sigma_3\}$ est un ECF de E vers E' sur $\{f, x\}$, et $\sigma_4 = \{<f, \lambda u.A(x)>\}$ est un filtre strictement moins général que σ_3 .

Dorénavant nous allons nous limiter aux filtres tels que σ_1, σ_2 et σ_4 , qui laissent E' invariant. Nous les appelerons demi-unificateurs, pour souligner le fait qu'ils sont aussi unificateurs.

Dans les applications pratiques du filtrage ceci n'est pas vraiment une restriction, car en général E et E' n'ont pas de variables en commun.

Définitions : Soit E et E' deux termes de même type, $L = V[E] \cup V[E']$.

On appelle *demi-unificateur de E vers E'* toute substitution σ telle que $\mathcal{D}(\sigma) \subset V[E] - V[E']$ et $\sigma E = E'$.

On dénote $\mathcal{D}(E, E')$ l'ensemble de ces substitutions.

Remarquons qu'avec $S = \{\sigma | \mathcal{D}(\sigma) \subset V[E] - V[E']\}$, on a

$$\mathcal{D}(E, E') = F(E, E') \cap S = U(E, E') \cap S.$$

Soit V un ensemble fini de variables contenant L .

On appelle *ensemble complet de demi-unificateurs de E vers E' sur V* (ECD) tout ensemble de substitutions Σ tel que :

- (1) $\Sigma \subset \mathcal{D}(E, E')$ cohérence
- (2) $\forall \rho \in \mathcal{D}(E, E') \quad \exists \sigma \in \Sigma \quad \sigma \leq_V \rho$ complétude
- (3) $\forall_{\substack{\sigma, \sigma' \in \Sigma \\ \sigma \neq \sigma'}} \quad \sigma \not\leq_L \sigma'$ non-congruence.

Un ensemble complet de demi-unificateurs minimaux (ECDM) est un ECD qui possède en plus la condition de minimalité :

$$\forall_{\substack{\sigma, \sigma' \in \Sigma \\ \sigma \neq \sigma'}} \quad \sigma \not\leq_L \sigma'.$$

Par exemple, dans l'exemple ci-dessus, $\{\sigma_1, \sigma_2, \sigma_4\}$ est un ECDM de E vers E' sur $\{f, x\}$.

A l'ordre 1, x est demi-unifiable vers e ssi $x \notin V[e]$, et dans ce cas $\{x, e\}$ est demi-unificateur principal. D'une manière générale, nous verrons au chapitre 5 que deux termes E et E' admettent au plus un demi-unificateur, qui est donc principal.

Nous verrons au chapitre 3 que deux termes E et E' admettent toujours un ECDM, que l'on peut énumérer récursivement. L'algorithme d'énumération peut ne pas terminer, même lorsque $D(E, E') = \emptyset$.

Nous verrons au chapitre 6 que cet ECDM est fini à l'ordre 2. L'algorithme d'énumération peut alors être utilisé comme algorithme de décision.

A partir de l'ordre 3 toutefois, nous obtenons un analogue du théorème 4 :

Théorème 7 : Il existe des termes d'ordre 3 E et E' , avec $D(E, E') \neq \emptyset$, et tels que, pour tout V tel que $L = V[E] \cup V[E'] \subset V$, les ECD de E vers E' sur V sont infinis.

Démonstration :

Considérons :

$$\begin{cases} E = f(\lambda u \cdot u) \\ E' = A \end{cases} \quad \text{avec} \quad \begin{cases} \tau(u) = \tau(A) = \alpha \\ \tau(f) = ((\alpha \rightarrow \alpha) \rightarrow \alpha). \end{cases}$$

Il suffit de prendre $V = L = \{f\}$.

Soit $\Sigma = \{\sigma_n \mid n \geq 0\}$,
avec $\sigma_n = \{<f, \lambda w \cdot w^n(A)>\}$, où $\tau(w) = (\alpha \rightarrow \alpha)$.

Il est ais  de v rifier que $\Sigma \subset \mathcal{D}(F, F')$. En fait, on peut montrer, en utilisant les m mes techniques que dans la preuve du th or me 4, que Σ est un ECDM de E vers E' sur V . Comme pour le th or me 4, Σ  tant infini, il ne peut exister d'FCD fini de E vers E' sur V . \square

La d cidabilit  du probl me de la demi-unification en λ -calcul typ  d'ordre 3 ou plus est encore un probl me ouvert :

Conjecture : Il existe un algorithme qui, pour tous E, E' , d cide si $E \leq E'$.

2.10 Unification multiple

D finition :

On appelle *unificateur simultan * de

$\{<E_i, E'_i> \mid 1 \leq i \leq n\}$, avec $\tau(E_i) = \tau(E'_i) \quad 1 \leq i \leq n$, toute substitution σ telle que $\sigma E_i = \sigma E'_i \quad 1 \leq i \leq n$, i.e. $\sigma \in \bigcap_{i \leq n} U(E_i, F'_i)$.

Soit F une constante arbitraire, de type $(\tau(E_1) \times \dots \times \tau(E_n)) \nrightarrow \gamma$, avec $\gamma \in T$ quelconque. Il est imm diat que σ est un unificateur simultan  des E_i, E'_i ssi σ est un unificateur de E et E' , avec

$$E = F(E_1, \dots, E_n), \quad E' = F(E'_1, \dots, E'_n).$$

Nous pouvons donc traiter l'unification simultan e avec le formalisme précédent.

D finition :

On appelle *unificateur multiple* d'un ensemble fini $E = \{E_1, \dots, E_n\}$ de termes de m me type toute substitution σ telle que

$$\sigma E_1 = \sigma E_2 = \dots = \sigma E_n.$$

Les unificateurs multiples de E sont exactement les unificateurs simultanés de (par exemple) :

$$\{<E_1, E_2>, \dots, <E_1, E_n>\}.$$

Nous pouvons donc également traiter de l'unification multiple, et de même de l'unification multiple simultanée d'ensembles finis E_1, \dots, E_p .

Il n'y a donc pas de perte de généralité à se limiter à l'étude de l'unification de deux termes, puisqu'il est facile de ramener les systèmes d'équations entre termes à une seule équation.

Le prochain chapitre est consacré à la recherche d'unificateurs, par énumération d'ECUs. Nous montrerons en particulier qu'il est toujours possible de générer une solution, si elle existe, sans faire de recherches redondantes, même dans les cas où le théorème 5 s'applique.

CHAPITRE 3 : Semi-algorithmes d'unification à l'ordre w.

Dans ce chapitre, nous allons développer une méthode permettant de générer des ensembles complets d'unificateurs. Plus exactement, nous générerons des ensembles complets de minorants d'unificateurs. Cette recherche est dirigée de l'extérieur vers l'intérieur, et nous montrerons que les "préunificateurs" obtenus sont non redondants. Cette méthode fournit en particulier un semi-algorithme de décision pour le problème de l'unification en λ -calcul typé, qui est pratiquement implémentable.

3.1. Définitions et notations

3.1.1. Unificandes

On appelle *unificande* tout ensemble fini N de paires de termes de même type :

$$N = \{ \langle e_i^1, e_i^2 \rangle \mid 1 \leq i \leq n \} , \quad \tau(e_i^1) = \tau(e_i^2)$$

On définit :

$$V[N] = \bigcup_{j=1}^2 \bigcup_{i=1}^n V[e_i^j]$$

$$\begin{aligned} \pi_1(N) &= n + \sum_{i=1}^n \pi(e_i^1) \\ \pi_2(N) &= n + \sum_{i=1}^n \pi(e_i^2) \end{aligned} \quad \left. \begin{array}{l} \\ \end{array} \right\} \text{avec } \pi(e) \text{ défini comme en 1.8.}$$

Soit σ une substitution quelconque. On note :

$$\sigma N = \{\langle \sigma e_i^1, \sigma e_i^2 \rangle \mid 1 \leq i \leq n\}.$$

On dit que σ unifie N ssi $\forall i \leq n \quad \sigma e_i^1 = \sigma e_i^2$. On dénote l'ensemble de telles substitutions par $U(N)$. On a bien sûr $U(N) = \bigcap_{i \leq n} U(e_i^1, e_i^2)$.

L'unificande $N = \{\langle e_i^1, e_i^2 \rangle \mid 1 \leq i \leq n\}$ est dit réduit si :

$$\left\{ \begin{array}{l} n > 0 \\ \forall i \leq n \quad e_i^1 \in T_f \\ \exists i \leq n \quad e_i^2 \in T_r \end{array} \right.$$

On définit l'ensemble des noeuds N comme l'ensemble des unificandes réduits, auquel on adjoint deux éléments particuliers :

(S) noeud "succès"

et (F) noeud "échec".

Nous présenterons en 3.2 un algorithme de simplification, SIMPL, qui réduit un unificande en un élément de N , en préservant l'unifiabilité.

3.1.2. Congruence de simplification

Pour chaque type élémentaire $\alpha \in T_0$, choisissons une variable $h_\alpha \in V_\alpha$.

Pour chaque type $\alpha \in T$, on définit un terme $\hat{E}_\alpha \in T_\alpha$ par :

- si $\alpha \in T_0$ $\hat{F}_\alpha = h_\alpha$

- sinon, soit $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_n \rightarrow \beta)$ avec $\beta \in T_0$; alors

$\hat{F}_\alpha = \lambda w_1 \dots w_n \cdot h_\beta$, où les w_i sont des variables distinctes

et différentes de h_β , telles que $\tau(w_i) = \alpha_i$.

On définit maintenant :

$$\zeta = \bigsqcup_{\alpha \in T} \{ \langle x, \hat{E}_\alpha \rangle \mid x \in V_\alpha \}.$$

ζ est une substitution généralisée (parce que de domaine infini), qui représente une élimination universelle (tous les arguments des termes flexibles sont éliminés).

Pour tout terme e , on définit ζe comme étant $(\zeta \setminus V[e])e$. On définit de même ζN pour un unificande N , et pour toute substitution σ :

$\zeta \sigma = \{ \langle x, \zeta \sigma x \rangle \mid x \in V \}$ est également une substitution de domaine infini.

On définit la *congruence de simplification* \sim , entre termes de même type, par :

$$e_1 \sim e_2 \iff \zeta e_1 = \zeta e_2.$$

\sim est une congruence sur T , qui préserve les types et la propriété d'être flexible ou rigide. Parmi les congruences de ce type, c'est la congruence la plus grossière qui vérifie : $H[e_1] \neq H[e_2] \implies e_1 \not\sim e_2$ pour $e_1, e_2 \in T_r$.

Lemme 3.1 $\forall e_1, e_2 \in T_f \quad \tau(e_1) = \tau(e_2) \implies e_1 \sim e_2.$

Démonstration :

Soit e_1 et e_2 deux termes flexibles de même type. Notons :

$$\left\{ \begin{array}{l} e_1 = \lambda u_1 \cdots u_{m_1} \cdot f(e_1^1, e_2^1, \dots, e_r^1) \\ e_2 = \lambda v_1 \cdots v_{m_2} \cdot g(e_1^2, e_2^2, \dots, e_s^2). \end{array} \right.$$

Puisque $\tau(e_1) = \tau(e_2)$, il doit exister $\alpha_1, \alpha_2, \dots, \alpha_n \in T$ et $\beta \in T_0$ tels que :

$$\left\{ \begin{array}{ll} \tau(u_i) = \alpha_i & 1 \leq i \leq m_1 \\ \tau(v_i) = \alpha_i & 1 \leq i \leq m_2 \\ \tau(f) = (\tau(e_1^1), \dots, \tau(e_r^1), \alpha_{m_1+1}, \dots, \alpha_n \rightarrow \beta) \\ \tau(g) = (\tau(e_1^2), \dots, \tau(e_s^2), \alpha_{m_2+1}, \dots, \alpha_n \rightarrow \beta). \end{array} \right.$$

Soit w_i , $1 \leq i \leq n$ des variables distinctes et différentes de h_β , avec

$\tau(w_i) = \alpha_i$. Alors

$$\zeta e_1 = \zeta e_2 = \lambda w_1 \cdots w_n \cdot h_\beta. \quad \square$$

Corollaire :

Tout unificande N ne contenant que des paires de termes flexibles est unifiable par $\zeta|V[N]$. On dit qu'un tel noeud est "pré-unifié". Nous allons maintenant définir ce terme précisément.

3.1.3. Préunification

Soit E et E' deux termes de même type, $L = V[E] \cup V[E']$. On appelle pré-unificateur de E et E' toute substitution σ telle que

$$\sigma F \sim \sigma E'.$$

Autrement dit, un préunificateur σ est une substitution qui peut être étendue en un unificateur par composition avec ζ : $\zeta \sigma|L \in U(E, E')$.

On dénote par $P(F, E')$ l'ensemble des préunificateurs de F et E' .

Il est immédiat que $U(F, E') \subset P(F, E')$, et que $U(F, E') = \emptyset \Leftrightarrow P(F, E') = \emptyset$.

Soit V un ensemble fini de variables, avec $L \subset V$.

On appelle *ensemble complet de pré-unificateurs (ECP) de F et E' sur V* tout ensemble de substitutions Σ tel que :

- (1) $\Sigma \subset P(F, E')$ (cohérence)
- (2) $\forall \rho \in P(F, E') \quad \exists \sigma \in \Sigma : \sigma \leq_V \rho$ (complétude).

De même que pour les ECU, il existe toujours un ECP de F et E' sur V , puisque $P(F, E')$ est solution. Si $U(F, E') = \emptyset$ alors \emptyset est le seul ECP, sinon \emptyset n'est pas un ECP.

Nous allons présenter dans ce chapitre un algorithme qui, pour tous F , E' et V , énumère un ECP Σ de F et E' sur V . Comme nous venons de le remarquer, cet algorithme peut être utilisé pour décider si F et E' sont unifiables : si l'algorithme génère un élément ($\Sigma \neq \emptyset$) alors F et E' sont unifiables, sinon ($\Sigma = \emptyset$) F et E' sont incompatibles. Naturellement, d'après le théorème 6, il ne peut s'agir que d'un algorithme de semi-décision, c'est à dire qu'il peut boucler sans jamais ne rien générer.

On pourrait bien sûr énumérer $U(F, E')$ directement (par exemple en générant exhaustivement toutes les substitutions, et en les testant sur F et E'). Il est donc bien évident qu'un tel algorithme existe. L'intérêt d'énumérer un ECP vient du fait que le théorème 5 ne s'applique pas aux ECP, et qu'au contraire des ECU on peut imposer aux ECP une condition supplémentaire de non-redondance très forte. Ceci assure à notre algorithme une grande directivité sans faire de calculs redondants, ce qui le rend utilisable en pratique.

On appelle ensemble complet de pré-unificateurs indépendants (ECPI) de E et E' sur V tout ECP Σ de E et E' sur V qui vérifie la condition supplémentaire :

$$(3) \quad \forall \sigma_1, \sigma_2 \in \Sigma \quad \begin{matrix} \sigma_1 \mid \sigma_2 \\ \sigma_1 \neq \sigma_2 \end{matrix} \quad (\text{indépendance}), \text{ où } L = V[E] \cup V[E'].$$

Nous montrerons que de tels ECPI existent, et qu'on peut réaliser un algorithme qui énumère un FCPI, ce qui nous permet de tester l'unifiabilité de deux termes avec un minimum d'exploration.

Pour les ECU, même dans le cas où il existe un ECUM Σ , on ne peut pas en général imposer à Σ la condition d'indépendance, même au second ordre.

Par exemple, pour $F = f(x)$, $F' = f(A)$ et $V = L = \{f, x\}$, avec $\tau(x) = \tau(A) = \gamma$, $\tau(f) = (\gamma \rightarrow \gamma)$, on peut montrer que $\{\sigma_1, \sigma_2\}$ est un FCUM de E et E' sur V , avec $\sigma_1 = \{<x, A>\}$ et $\sigma_2 = \{<f, \lambda y \cdot z>\}$, $\tau(y) = \tau(z) = \gamma$, mais $\rho = \sigma_1 \cup \sigma_2$ est tel que $\sigma_1 \leq_V \rho$ et $\sigma_2 \leq_V \rho$. Ici $\{\emptyset\}$ est bien sûr un ECPI.

Remarquons que pour un FCPI, la condition (2) peut s'écrire :

$\forall \rho \in P(E, E') \quad \exists! \sigma \in \Sigma \quad \sigma \leq_V \rho$, où $\exists! \sigma$ dénote "il existe un σ unique". En effet, si on avait $\sigma_1 \leq_V \rho$ et $\sigma_2 \leq_V \rho$ avec $\sigma_1 \neq \sigma_2$, cela contredirait (3).

Lemme 3.2

S'il existe un ECPI de E et E' sur V il est unique, à un isomorphisme près. Plus exactement, entre deux tels ECPI Σ_1 et Σ_2 il existe une bijection

$$\phi : \Sigma_1 \rightarrow \Sigma_2 \text{ telle que } \forall \sigma \in \Sigma_1 \quad \sigma \leq_V \phi(\sigma).$$

La démonstration de ce lemme est identique à celle du lemme 2.16.

Nous allons maintenant décrire des algorithmes qui vont permettre d'énumérer un ECP. Le paragraphe 3.2 présente l'algorithme SIMPL, qui simplifie les unificandes. Le paragraphe 3.3 présente l'algorithme CHOIX, qui énumère des composantes de substitution, segments initiaux d'un ECP. L'algorithme de génération utilise alternativement ces deux algorithmes dans la construction d'un arbre, dont les noeuds terminaux définissent un ECP. Ces arbres sont définis en 3.4, et la construction prouvée correcte. En 3.5 on montre qu'on peut en fait engendrer un ECPI. En 3.6 on montre comment on peut encore restreindre le nombre de solutions, si on s'intéresse simplement à l'unifiabilité. Enfin le paragraphe 3.7 présente quelques heuristiques pour des cas particuliers, avec application à la demi-unification.

3.2. Algorithme de simplification

SIMPL prend comme argument un unificande, et retourne comme résultat un élément de N , c'est à dire soit un unificande réduit, soit \mathbb{S} ou \mathbb{E} .

3.2.1. Description algorithmique de SIMPL(N).

Etape 1 : S'il n'existe pas dans N de paire dans $T_r \times T_r$ alors aller à l'étape 2, sinon soit $\langle e_1, e_2 \rangle$ une telle paire :

$$N = \{ \langle e_1, e_2 \rangle \} \sqcup N', \text{ avec}$$

$$\begin{cases} e_1 = \lambda u_1 u_2 \cdots u_{n_1} \cdot @_1 (e_1^1, \dots, e_{p_1}^1) & n_1 \geq 0 \quad p_1 \geq 0 \\ e_2 = \lambda v_1 v_2 \cdots v_{n_2} \cdot @_2 (e_2^2, \dots, e_{p_2}^2) & n_2 \geq 0 \quad p_2 \geq 0 ; \end{cases}$$

Si $H[e_1] \neq H[e_2]$ alors résultat = "E".

Sinon, soit $p = p_1 = p_2$, $n = n_1 = n_2$.

$$\text{Soit } \begin{cases} \tilde{e}_i^1 = \lambda u_1 \cdots u_n \cdot e_i^1 \\ \tilde{e}_i^2 = \lambda v_1 \cdots v_n \cdot e_i^2, \end{cases}$$

répéter l'étape 1, avec $N = N' \cup \{ \langle \tilde{e}_i^1, \tilde{e}_i^2 \rangle \mid 1 \leq i \leq p \}$.

Etape 2 : Remplacer dans N toute paire $\langle e_1, e_2 \rangle$ de $T_r \times T_f$ par la paire $\langle e_2, e_1 \rangle$, puis faire l'étape 3.

Etape 3 : Si $N \subset T_f \times T_f$ alors résultat = "S", sinon résultat = N. \square

Remarques :

1. Dans l'étape 1, on a $H[e_1] \neq H[e_2]$

ssi $\begin{cases} - \text{ ou bien } n_1 \neq n_2 \\ - \text{ ou bien } @_1 \in C, @_2 \neq @_1 \\ - \text{ ou bien } @_1 = u_i, @_2 \neq v_i \quad 1 \leq i \leq n_1 = n_2 \end{cases}$

2. Dans la définition de \tilde{e}_i^1 , si $e_i^1 = \lambda w_1 \cdots w_k \cdot @_1 (\dots)$, alors

$e_i^1 = \lambda u_1 \cdots u_n w_1 \cdots w_k \cdot @_1 (\dots)$. L'utilisation de la forme abrégée définie en 1.8 nous permet de ne pas avoir de confusion de variables liées.

3.2.2. Exemples de calculs de SIMPL (N) :

$$1^\circ \quad N = \{A(\lambda u \cdot B(x, u), C), A(\lambda v \cdot B(y, v), f(C))\}$$

Les valeurs successives de N sont :

étape 1 $N = \{\langle \lambda u \cdot B(x, u), \lambda v \cdot B(y, v) \rangle, \langle C, f(C) \rangle\}$

" 1 $N = \{\langle \lambda u \cdot x, \lambda v \cdot y \rangle, \langle \lambda u \cdot u, \lambda v \cdot v \rangle, \langle C, f(C) \rangle\}$

" 1 $N = \{\langle \lambda u \cdot x, \lambda v \cdot y \rangle, \langle C, f(C) \rangle\}$

" 2 $N = \{\langle \lambda u \cdot x, \lambda v \cdot y \rangle, \langle f(C), C \rangle\}$ résultat retourné à l'étape 3.

$$2^\circ \quad N = \{A(\lambda u \cdot B(x, u)), A(\lambda v \cdot B(y, v))\}$$

Après 3 pas de calcul comme en 1°, on obtient :

$$N = \{\langle \lambda u \cdot x, \lambda v \cdot y \rangle\},$$

et à l'étape 3 on retourne comme résultat "S".

$$3^\circ \quad N = \{\langle \lambda u v \cdot A(u, \lambda w \cdot v), \lambda v w \cdot A(v, \lambda u \cdot v) \rangle\}.$$

étape 1 $N = \{\langle \lambda u v \cdot u, \lambda v w \cdot v \rangle, \langle \lambda u v w \cdot v, \lambda v w u \cdot v \rangle\}$

" 1 $N = \{\langle \lambda u v w \cdot v, \lambda v w u \cdot v \rangle\}$

" 1 on retourne comme résultat "E".

3.2.3. Preuve de correction de SIMPL

Lemme 3.3 Soit N un unificande quelconque. SIMPL(N) s'arrête au bout d'un nombre fini de pas, et retourne un résultat N' tel que :

(1) - si $N' = "E"$ alors $U(N) = \emptyset$

(2) - si $N' = "S"$ alors $U(N) \neq \emptyset$

(3) - sinon $U(N') = U(N)$.

Démonstration

Soit $N_0 = N$ initialement, $N_k = N$ après k itérations de l'étape 1.

On prouve par récurrence sur k que $U(N_k) = U(N_0)$. Pour $k=0$ c'est trivialement vérifié. Supposons la propriété vraie pour k .

Si $N_k \cap T_r \times T_r \neq \emptyset$, soit $\langle e_1, e_2 \rangle$ la paire sélectionnée à l'étape 1.

Si $H[e_1] \neq H[e_2]$, alors on sait par le lemme 2.6 que $U(N_k) = \emptyset$, et donc $U(N_0) = \emptyset$, ce qui prouve (1). Sinon, considérons une substitution σ quelconque.

Soit, avec les notations de 3.2.1. $V = \{u_i \mid 1 \leq i \leq n\} \cup \{v_i \mid 1 \leq i \leq n\}$.

Sans perte de généralité, on peut supposer les u_i et v_i choisis tels que $V \cap I(\sigma) = \emptyset$.

Dans ces conditions :

$$\begin{cases} \sigma e_1 = \lambda u_1 \dots u_n \cdot @_1(\sigma e_1^1, \dots, \sigma e_p^1) & \text{puisque } @_1 \in C \cup \{u_i \mid 1 \leq i \leq n\} \\ \sigma e_2 = \lambda v_1 \dots v_n \cdot @_2(\sigma e_1^2, \dots, \sigma e_p^2) & \text{de même.} \end{cases}$$

$\sigma e_1 = \sigma e_2$ si et seulement si $\forall i \leq p$:

$$\lambda u_1 \dots u_n \cdot [\sigma e_i^1] = \lambda v_1 \dots v_n \cdot [\sigma e_i^2]$$

c'est à dire, en allant maintenant vers l'extérieur, ssi $\tilde{\sigma e}_i^1 = \tilde{\sigma e}_i^2$.

N_{k+1} étant identique à N_k sur les autres paires, on a bien

$$U(N_{k+1}) = U(N_k) = U(N_0).$$

De plus, on vérifie que $\pi_1(N_{k+1}) = \pi_1(N_k) - 1$, ce qui prouve que l'itération de l'étape 1 doit s'arrêter. S'il n'y a pas eu échec, on passe donc à l'étape 2 avec N tel que $U(N) = U(N_0)$. Puisque l'unification est symétrique cette propriété est encore vraie après l'étape 2. Si alors $N \subset T_f \times T_f$, on a $U(N) \neq \emptyset$ par le corollaire au lemme 3.1, ce qui prouve (2). Sinon le résultat est N , avec $U(N) = U(N_0)$, ce qui prouve (3). \square

3.3. Algorithme de choix

Nous allons maintenant donner un algorithme CHOIX qui accepte trois arguments e_1, e_2, V .

e_1 et e_2 sont des termes de même type, $e_1 \in T_f$ et $e_2 \in T_r$. V est un ensemble fini de variables. CHOIX retourne comme résultat un ensemble fini Σ de composantes de substitution de la forme $\langle x, e \rangle$, où x est la variable de tête de e_1 .

Notre intention est que Σ puisse être complété par composition en un ECPI de e_1 et e_2 sur V .

Pour faciliter les preuves ultérieures, nous présentons CHOIX en justifiant à chaque pas les décisions prises par de nombreux commentaires.

3.3.1. Description algorithmique de CHOIX (e_1, e_2, V)

Soit

$$\begin{cases} e_1 = \lambda u_1 \dots u_{n_1} \cdot f(e_1^1, e_2^1, \dots, e_{p_1}^1) & n_1 \geq 0 \quad p_1 \geq 0 \\ e_2 = \lambda v_1 \dots v_{n_2} \cdot @ (e_1^2, e_2^2, \dots, e_{p_2}^2) & n_2 \geq 0 \quad p_2 \geq 0 \end{cases}$$

$$\text{avec } \tau(f) = (\alpha_1 \times \alpha_2 \times \dots \times \alpha_{p_1} \times \alpha_{p_1+1} \times \dots \times \alpha_{q_1} \rightarrow \beta). \quad q_1 \geq p_1.$$

e_2 étant rigide, on sait par le lemme 2.6 que $\forall \sigma \quad H[\sigma e_2] = H[e_2]$.

Donc pour tout préunificateur σ de e_1 et e_2 on doit avoir σf tel que

$$H[\sigma e_1] = \lambda v_1 \dots v_{n_2} \cdot @. C'est sur cette simple idée que CHOIX va construire un$$

ensemble Σ de composantes de substitutions relatives à f : $\langle f, \lambda z_1 \dots z_r \cdot @'(\dots) \rangle$.

L'atome $@'$ va être :

- soit \emptyset , et la manière la plus générale de construire une telle composante est donnée par la règle d'*imitation* ci-dessous.
- soit $z_k \quad 1 \leq k \leq r$, cas de la règle de *projection* ci-dessous.

Dans les deux cas, r aura toutes les valeurs compatibles avec un bon typage.

Tout d'abord nous remarquons que l'en tête d'un terme flexible ne peut que s'allonger par substitution, par le lemme 2.6. Donc si $n_1 > n_2$, CHOIX s'arrête avec le résultat $\Sigma = \emptyset$; sinon, soit $n = n_2 - n_1 \geq 0$. On introduit p_1 variables $w_1 \dots w_{p_1}$, avec $\forall i \leq p_1 \tau(w_i) = a_i$. Pour éviter les conflits, on impose $w_i \notin \{v_{n_1+1}, \dots, v_{n_2}\}$.

CHOIX retourne comme résultat l'union des solutions données par les règles d'imitation et de projection ci-dessous.

3.3.1.1. Imitation

Nous cherchons des préunificateurs de e_1 et e_2 qui "imitent" e_2 par e_1 , en substituant pour f un terme de tête θ . Si $\theta = v_i$ avec $1 \leq i \leq n_1$, ce n'est pas possible, puisque le u_i correspondant est protégé, étant lié par l'en tête de e_1 . On ne peut donc l'introduire qu'indirectement par la règle de projection ci-après. Nous nous limitons donc ici aux cas $\theta \in \mathcal{C}$ ou $\theta = v_i$ avec $n_1 < i \leq n_2$.

Il y a deux sous-cas, suivant la valeur de n .

3.3.1.1.1. $n > 0$

Dans ce cas l'en tête du terme à substituer est déterminé de manière unique par les types. Plus précisément, on prend comme solution unique la composante :

$$\langle f, \lambda w_1 \dots w_{p_1} v_{n_1+1} \dots v_{n_2} \cdot \theta(F_1, F_2, \dots, F_{p_2}) \rangle$$

avec : $F_i = h_i(w_1, \dots, w_{p_1}, v_{n_1+1}, \dots, v_{n_2}) \quad 1 \leq i \leq p_2,$

où les h_i sont des variables distinctes du bon type n'appartenant pas à V.

Par exemple, $\tau(h_1) = (\alpha_1 \times \alpha_2 \times \dots \times \alpha_{p_1} \times \alpha_{p_1+1} \times \dots \times \alpha_{p_1+n} \rightarrow \tau(e_1^2))$. Il ne peut y avoir de conflit de h_i avec w_j ou v_k , puisque leurs types diffèrent.

3.3.1.1.2. n=0

On doit donc avoir $\theta \in C$. Nous allons substituer à f un terme d'en-tête $\lambda w_1 \dots w_k \cdot \theta$, avec $0 \leq k \leq p_1$. Tout k entre 0 et p_1 ne convient pas, car il y a des conditions de typage à respecter. Tout d'abord, dans ce terme θ doit être appliquée à $p_2 - (p_1 - k)$ arguments. Ce nombre doit être ≥ 0 , ce qui nous donne la première condition :

$$(1) \quad \max(0, p_1 - p_2) \leq k \leq p_1.$$

Enfin les arguments intouchés de e_1 doivent être de types égaux aux arguments correspondants de e_2 :

$$(2) \quad \alpha_j = \tau(e_{p_2-p_1+j}^2) \quad \forall j (k < j \leq p_1).$$

Sous ces deux conditions, les composantes données ci-dessous sont bien typées. Donc, pour tout k qui satisfait à (1) et (2), on prend comme solution la composante :

$$\langle f, \lambda w_1 \dots w_k \cdot \theta(E_1, E_2, \dots, E_{p_2-p_1+k}) \rangle$$

avec $E_i = h_i (w_1, \dots, w_k) \quad 1 \leq i \leq p_2 - p_1 + k$, où les h_i sont des variables distinctes du bon type non dans V.

Remarquons que (1) et (2) sont satisfaites par $k = p_1$, ce qui garantit au moins une solution. A cause de (1), on a au plus $\min(p_1, p_2) + 1$ solutions.

Il n'est pas possible, dans le $\lambda\beta$ -calcul, de se limiter au seul cas $k = p_1$. Par exemple, considérons

$$e_1 = f(B(A)) \quad e_2 = A(B(f)), \text{ avec}$$

$$\tau(f) = \tau(A) = (\alpha \rightarrow \alpha), \quad \tau(B) = ((\alpha \rightarrow \alpha) \rightarrow \alpha). \text{ Les termes } e_1 \text{ et } e_2$$

admettent un unificateur unique $\sigma = \{\langle f, A \rangle\}$. CHOIX ne peut donc se limiter à la composante $\rho = \langle f, \lambda u \cdot A(h(u)) \rangle$, qui ne minore aucun préunificateur de e_1 et e_2 sur $\{f\}$. Par contre, nous verrons au chapitre 4 que ce cas suffit lorsque l'on se donne la règle de conversion supplémentaire :

$$(\eta\text{-conversion}) \quad E\langle \lambda u \cdot e(u) \rangle = E\langle e \rangle \quad \text{si } u \notin V[e].$$

3.3.1.2. Projection

Nous cherchons maintenant à "projeter" f sur l'un de ses arguments. Le terme substitué à f peut avoir un en-tête :

$$\text{soit } (a) \quad \lambda w_1 \cdots w_k \cdot w_i \quad 1 \leq i \leq k \leq p_1$$

$$\text{soit } (b) \quad \lambda w_1 \cdots w_{p_1} v_{n_1+1} \cdots v_{n_1+k} \cdot w_i \quad 1 \leq k \leq n, \quad 1 \leq i \leq p_1$$

$$\text{soit } (c) \quad \lambda w_1 \cdots w_{p_1} v_{n_1+1} \cdots v_{n_1+k} \cdot v_{n_1+i} \quad 1 \leq i \leq k \leq n$$

Toutefois, dans le cas (c), la composante σ ainsi construite serait telle que :

$$\begin{cases} \sigma e_1 = \lambda u_1 \cdots u_{n_1} v_{n_1+1} \cdots v_{n_1+k} \cdot v_{n_1+i} (\dots) \\ \sigma e_2 = \lambda v_1 \cdots v_{n_1} v_{n_1+1} \cdots v_{n_2} \cdot @(\dots) \end{cases}$$

Mais alors σ ne peut mener à un préunificateur de e_1 et e_2 que si $k=n$ et $@ = v_{n_1+i}$, et nous avons déjà considéré ce cas avec la règle d'imitation.

3.3.1.1.1. Nous nous limiterons donc aux cas (a) et (b).

3.3.1.2.1. en-tête $\lambda w_1 \cdots w_k \cdot w_i$

Comme pour l'imitation, nous avons des conditions de type.

$$(1) \quad 1 \leq i \leq k \leq p_1.$$

Le type de w_i doit être tel qu'on puisse le compléter par des arguments en un terme du bon type. Plus précisément, il doit exister $m \geq 0$ et $\gamma_1, \dots, \gamma_m \in T$ tels que :

$$(2) \alpha_i = (\gamma_1 \times \gamma_2 \times \cdots \times \gamma_m \times \alpha_{k+1} \times \cdots \times \alpha_{q_1} \rightarrow \beta)$$

c'est à dire que les types de f et α_i doivent avoir en commun le facteur droit $\alpha_{k+1} \times \cdots \times \alpha_{q_1} \rightarrow \beta$. Remarquons que, donnés i et k , m est déterminé de façon unique. Par exemple, si $k = q_1$ et i est tel que $\alpha_i = \beta$, alors on a la solution $m=0$.

Pour tous i et k satisfaisant (1) et (2), nous prenons comme résultat la composante

$$\langle f, \lambda w_1 \cdots w_k \cdot w_i (E_1, E_2, \dots, E_m) \rangle$$

avec $E_j = h_j(w_1, \dots, w_k) \quad , \quad 1 \leq j \leq m,$

où les h_j sont des variables distinctes non dans V . On a bien sûr $\tau(h_j) = (\alpha_1 \times \cdots \times \alpha_k \rightarrow \gamma_j)$.

Ce sous-cas nous donne un maximum de

$$p_1(p_1+1)/2 \text{ solutions.}$$

3.3.1.2.2. en-tête $\lambda w_1 \cdots w_{p_1} v_{n_1+1} \cdots v_{n_1+k} \cdot w_i$

Ce cas est similaire au précédent, la notation seule change. Nous avons de même les deux conditions :

$$(1) \quad 1 \leq k \leq n \quad 1 \leq i \leq p_1$$

$$(2) \quad \alpha_i = (\gamma_1 \times \gamma_2 \times \cdots \times \gamma_m \times \alpha_{p_1+k+1} \times \cdots \times \alpha_{q_1} \rightarrow \beta) \quad \text{et nous}$$

prenons comme résultat toutes les composantes :

$$\langle f, \lambda w_1 \cdots w_{p_1} v_{n_1+1} \cdots v_{n_1+k} \cdot w_i (E_1, E_2, \dots, F_m) \rangle$$

avec $E_j = h_j(w_1, \dots, w_{p_1}, v_{n_1+1}, \dots, v_{n_1+k})$ où les h_i sont des variables distinctes de bon type non dans V .

Ici nous avons au plus $n \times p_1$ solutions.

fin de CHOIX. □

3.3.2. Exemple de calcul de CHOIX

Soit $e_1 = f(x, B)$, $e_2 = A(B)$, $V = \{f, x\}$

avec $\tau(f) = ((\gamma \times \gamma \rightarrow \gamma) \times \gamma \rightarrow \gamma)$, $\tau(x) = (\gamma \times \gamma \rightarrow \gamma)$, $\tau(B) = \gamma$, $\tau(A) = (\gamma \rightarrow \gamma)$.

Avec les notations de CHOIX, on a :

$\alpha_1 = (\gamma \times \gamma \rightarrow \gamma)$, $\alpha_2 = \gamma$, $\beta = \gamma$, $n_1 = n_2 = 0$, $p_1 = q_1 = 2$, $p_2 = 1$.

3.3.2.1. Imitation

$n = n_2 - n_1 = 0$ et on prend donc le cas 3.2.1.1.2. $p_1 - p_2 = 1 \leq k \leq p_1 = 2$

nous donne deux sous-cas :

(i) $k=1$ alors (2) : $\alpha_2 = \tau(B) = \gamma$ est vérifiée, et on prend $\sigma_1 = \langle f, \lambda w_1 \cdot A \rangle$.

Remarquons que σ_1 unifie e_1 et e_2 , x étant éliminé.

(ii) $k=2$ alors (2) est trivialement vérifiée, et on prend

$\sigma_2 = \langle f, \lambda w_1 w_2 \cdot A(h(w_1, w_2)) \rangle$

avec $\tau(h) = ((\gamma \times \gamma \rightarrow \gamma) \times \gamma \rightarrow \gamma)$.

3.3.2.2. Projection

Puisque $n=0$ on ne considère que le cas 3.2.1.2.1. On a deux sous-cas :

(i) $k=1$; la condition (1) oblige $i=1$; la condition (2) est alors satisfaite

avec $m=1$, $\gamma_1 = \gamma$, ce qui donne la solution :

$\sigma_3 = \langle f, \lambda w_1 \cdot w_1(h'(w_1)) \rangle$

avec $\tau(h') = ((\gamma \times \gamma \rightarrow \gamma) \rightarrow \gamma)$.

(ii) $k=2$; la condition (1) nous donne deux solutions :

$i=1$ avec $m=2$, $\gamma_1 = \gamma_2 = \gamma$

$\sigma_4 = \langle f, \lambda w_1 w_2 \cdot w_1(h_1(w_1, w_2), h_2(w_1, w_2)) \rangle$

avec $\tau(h_1) = \tau(h_2) = \tau(f)$.

$i=2$ avec $m=0$

$\sigma_5 = \langle f, \lambda w_1 w_2 \cdot w_2 \rangle$

Sur cet exemple, on voit donc que CHOIX (e_1, e_2, V) retournera l'ensemble des 5 composantes $\sigma_1 \dots \sigma_5$, qui sont indépendantes sur V .

De plus on peut vérifier que tout unificateur de e_1 et e_2 peut être obtenu par composition à partir d'un σ_i , $1 \leq i \leq 4$. Nous allons maintenant montrer que c'est toujours le cas.

3.3.3. Preuve de correction de CHOIX

Lemme 3.4 de factorisation unique

Soit $e_1 \in T_f$, $e_2 \in T_r$ avec $\tau(e_1) = \tau(e_2)$, V un ensemble de variables fini. CHOIX (e_1, e_2, V) retourne comme résultat un ensemble fini Σ de composantes de substitutions indépendantes sur V tel que :

$$\forall \rho \in P(e_1, e_2) \quad \exists! \sigma \in \Sigma \quad \exists \eta(\rho \models \eta \sigma \text{ et } \pi(\eta) < \pi(\rho)).$$

Démonstration :

Nous utilisons les notations de 3.3.1.

Si $U(e_1, e_2) = \emptyset$ alors $\Sigma' = \emptyset$ satisfait le théorème. Supposons donc $U(e_1, e_2) \neq \emptyset$, ce qui impose en particulier $n = n_2 - n_1 \geq 0$.

Soit ρ un préunificateur quelconque de e_1 et e_2 , avec

$$\rho f = \lambda z_1 \dots z_k \cdot @'(E'_1, \dots, E'_k),$$

et soit $\bar{\rho} = \rho \upharpoonright (V - \{f\})$.

1er cas : $\left| \begin{array}{l} \textcircled{a} @' \in C \quad \text{et} \quad @ = @' \\ \text{ou} \quad \textcircled{b} @' = z_i, \quad p_1 < i \leq p_1 + n = k, \quad @ = v_{n_1 + i - p_1} \\ \text{ou} \quad \textcircled{c} @' = z_i, \quad 1 \leq i \leq p_1, \quad i \leq k \leq p_1 + n \end{array} \right.$

Par l'analyse de 3.3.1, nous avons considéré tous les cas de tels en-tête compatibles avec un bon typage : **a** et **b** par la règle d'imitation, **c** par la règle de projection. Il existe donc dans le résultat Σ une composante de substitution :

$$\sigma = \langle f, \lambda w_1 \cdots w_k \cdot @''(E_1, \dots, E_\ell) \rangle$$

avec $\begin{cases} \tau(w_i) = \tau(z_i) & \forall i \quad 1 \leq i \leq k \\ @'' = \begin{cases} @ & \text{dans le cas (a)} \\ w_i & \text{si } @' = z_i \text{ dans les cas (b) et (c)} \end{cases} \\ E_j = h_j(w_1, \dots, w_k) \quad \forall j \quad 1 \leq j \leq \ell, \quad h_j \notin V. \end{cases}$

De plus, σ est unique, car tous les en-tête construits par CHOIX sont distincts. Maintenant, considérons :

$$\eta = \{ \langle h_j, \lambda z_1 \cdots z_k \cdot E'_j \rangle \mid 1 \leq j \leq \ell \} \cup \bar{\rho}.$$

η est une substitution, car les h_j sont distincts, hors de V , et $\mathcal{D}(\bar{\rho}) \subset V$.

On vérifie bien que

$$(\eta\sigma) \upharpoonright V = \bar{\rho} \cup \{ \langle f, \eta\sigma f \rangle \} = \rho.$$

Enfin : $\pi(\eta) = \pi(\bar{\rho}) + \sum_{j=1}^{\ell} \pi(\lambda z_1 \cdots z_k \cdot E'_j) + \ell = \pi(\rho \upharpoonright V) - 1 < \pi(\rho).$

2ème cas : Dans tous les autres cas, c'est à dire :

$$\begin{cases} \text{(d)} \quad @' \in C \quad @ \neq @' \\ \text{(e)} \quad \rho f \text{ flexible} \\ \text{(f)} \quad @' = z_i, \quad i \leq k \text{ et } \begin{cases} k > p_1 + n \\ \text{ou } k = p_1 + n, \quad i > p_1, \quad @ \neq v_{n_1 + i - p_1} \\ \text{ou } k < p_1 + n, \quad i > p_1 \end{cases} \end{cases}$$

il est facile de montrer que $H[\rho e_1] \neq H[e_2]$, ce qui est incompatible avec $\rho \in P(e_1, e_2)$. Ce cas ne peut donc se produire, ce qui finit la preuve que les règles d'imitation et de projection sont suffisantes pour produire un ensemble complet de minorants des préunificateurs de e_1 et e_2 . \square

3.4. Arbres de pré-unification

Soit E et E' deux termes de même type, V un ensemble fini de variables contenant $L = V[E] \cup V[E']$. Nous allons montrer comment énumérer un ECP de E et E' sur V , en construisant un *arbre de préunification* (AP).

Un arbre de préunification est un arbre orienté, dont les noeuds sont étiquetés par des éléments de N et les arcs sont étiquetés par des composantes de substitutions.

Dans tout ce qui suit, nous désignerons les noeuds d'arbres de préunification par l'élément de N qui leur est associé, pour simplifier la notation.

3.4.1. Construction d'un AP de E et E' sur V .

Nous allons montrer comment construire progressivement un AP A de E et E' sur V , en définissant, pour tout noeud N construit, un ensemble fini de variables V_N . On définit également, pour tout noeud N non terminal, une variable v_N de $V[N]$.

Le noeud initial de A est

$$N_0 = SIMPL(\{<E, E'>\}). \text{ On prend } v_{N_0} = V.$$

Soit N le noeud courant dans A . Si $N = \textcircled{S}$ ou $N = \textcircled{F}$, N est dit terminal et n'a pas de successeur. Sinon, on choisit dans N une paire $\langle e_1, e_2 \rangle \in T_f \times T_r$ (il en existe au moins une par définition de N). On construit $\Sigma = CHOIX(e_1, e_2, v_N)$. Si $\Sigma = \emptyset$, N est remplacé par \textcircled{F} , le noeud terminal échec, et n'a pas de successeur.

Sinon, pour tout σ dans Σ , on construit l'arc :

$$N \xrightarrow{\sigma} N', \text{ avec}$$

$$N' = SIMPL(\sigma N).$$

On définit v_N comme étant la variable de tête de e_1 (i.e., $\sigma = \langle v_N, \dots \rangle$), et $V_{N'} = V_N \cup V[\sigma v_N]$.

Autrement dit, pour tout noeud N de A , V_N contient V et toutes les variables h_i introduites par CHOIX sur la branche reliant N_0 à N . On a donc également $V[N] \subset V_N$.

Remarquons qu'il peut y avoir plusieurs manières de choisir la paire $\langle e_1, e_2 \rangle$ dans N , et donc plusieurs arbres possibles. Nous verrons que ce choix n'importe pas, et qu'il peut donc être effectué par une heuristique quelconque.

Donnons un exemple d'un pas de construction, avec $N = \{e_1, e_2\}$ et $V_N = V$ comme dans l'exemple 3.3.2. Alors $\sigma_N = f$, et les 5 solutions $\sigma_1, \dots, \sigma_5$ de CHOIX (e_1, e_2, V) donnent les 5 successeurs de N :

$$\left| \begin{array}{l} N_1 = \textcircled{S} \\ N_2 = \{h(x, B), B\} \\ N_3 = \{x(h'(x), B), A(B)\} \\ N_4 = \{x(h_1(x, B), h_2(x, B)), A(B)\} \\ N_5 = \textcircled{E} \end{array} \right.$$

et, par exemple, $V_{N_4} = \{f, x, h_1, h_2\}$.

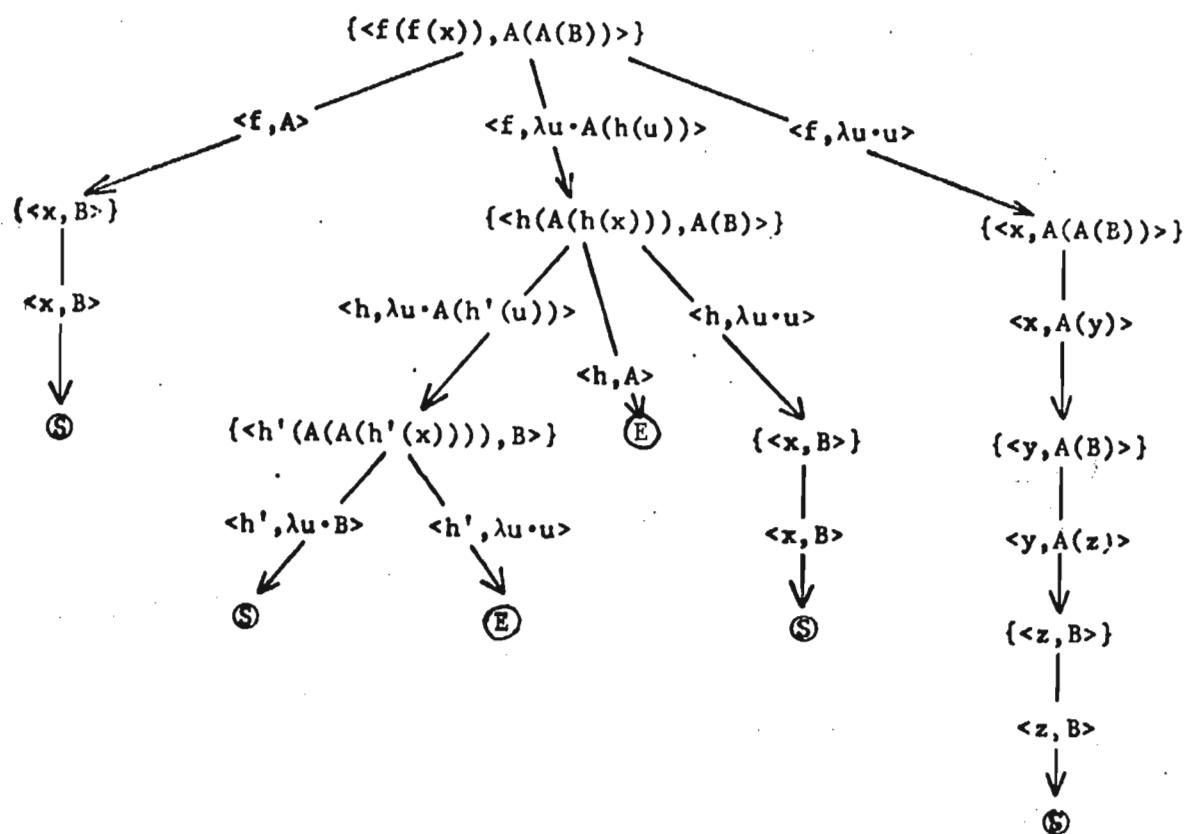
d

3.4.2. Exemples d'AP.3.4.2.1. Exemple 1 :

$E = f(f(x))$, $E' = A(A(B))$, $V = \{f, x\}$, avec :

$$\tau(x) = \tau(B) = \gamma, \tau(f) = \tau(A) = (\gamma \rightarrow \gamma).$$

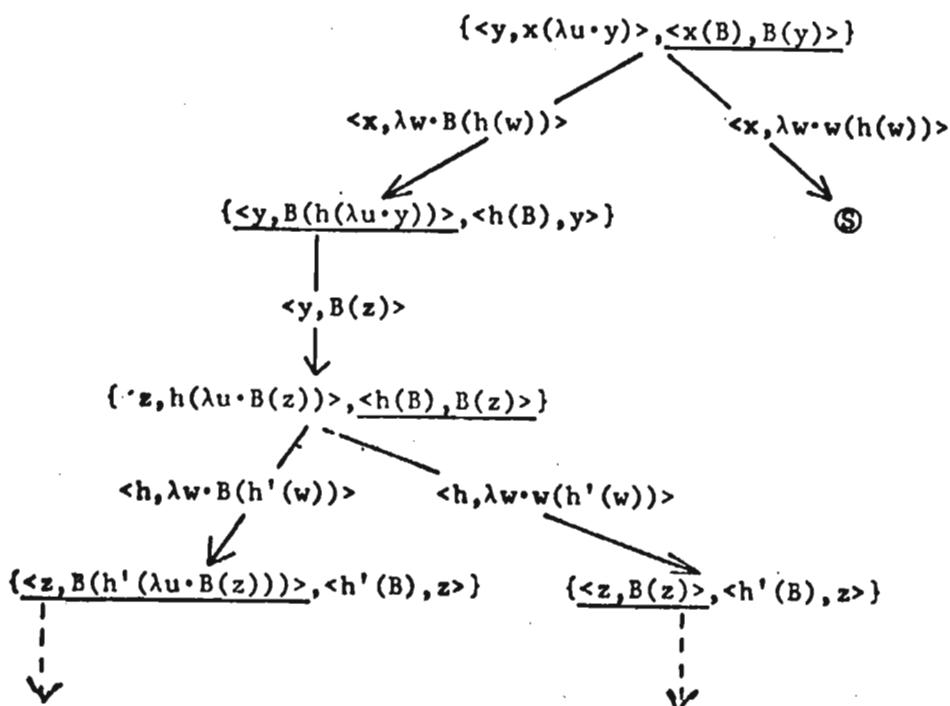
Il existe un AP unique, qui est fini :



3.4.2.1. Exemple 2 :

$E = A(y, x(B))$, $E' = A(x(\lambda u \cdot y), B(y))$, $V = \{x, y\}$,
 avec $\tau(y) = \tau(u) = \gamma$, $\tau(B) = (\gamma \rightarrow \gamma)$, $\tau(x) = ((\gamma \rightarrow \gamma) \rightarrow \gamma)$, $\tau(A) = (\gamma \times \gamma \rightarrow \gamma)$.

Ici on a un arbre unique infini, avec un noeud succès unique au niveau 1. Dans chaque noeud, on a souligné la paire $\langle e_1, e_2 \rangle$ sélectionnée.



Remarquons que le noeud succès provient directement de
 SIMPL ($\{\langle y, y \rangle, \langle B(h(B)), B(y) \rangle\}$).

3.4.2.3. Exemple 3 :

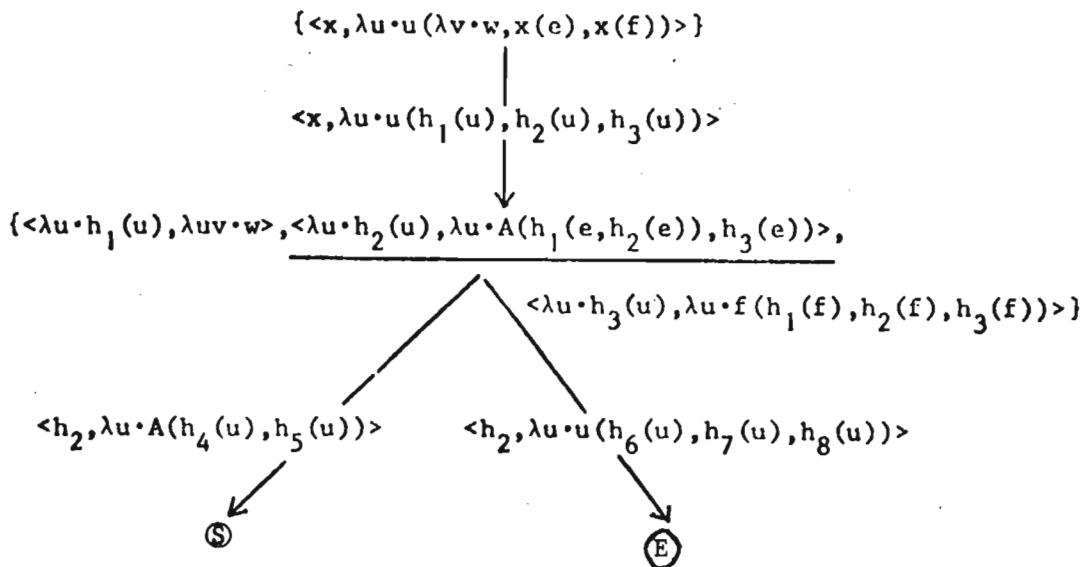
$$E = x, E' = \lambda u \cdot u(\lambda v \cdot w, x(e), x(f)), V = \{x, w, f\},$$

$$\text{où } e = \lambda u' v' w' \cdot A(u'(v'), w'),$$

$$\text{avec : } \tau(u) = \tau(f) = ((\gamma \rightarrow \gamma) \times \gamma \times \gamma \rightarrow \gamma), \tau(x) = (((\gamma \rightarrow \gamma) \times \gamma \times \gamma \rightarrow \gamma) \rightarrow \gamma),$$

$$\tau(u') = (\gamma \rightarrow \gamma), \tau(v) = \tau(w) = \tau(v') = \gamma, \tau(A) = (\gamma \times \gamma \rightarrow \gamma).$$

De nouveau un arbre unique fini :



Soit σ la composition des composantes de substitution le long du chemin qui mène au noeud succès.

Avec ζ défini en 3.1.2., en choisissant $h_\gamma = z$, on obtient l'unificateur :

$$\zeta \sigma | V = \{<x, \lambda u \cdot u(\lambda v \cdot z, A(z, z), z)>, <f, \lambda u' v' w' \cdot z>, <w, z>\}.$$

Inversement, tout unificateur de E et E' peut être obtenu par composition avec $\sigma | V = \{<x, \lambda u \cdot u(h_1(u), A(h_4(u), h_5(u)), h_3(u))>\}$.

Nous allons voir dans le paragraphe suivant qu'il en est toujours ainsi.

Remarquons sur cet exemple que la convergence est très rapide.

3.4.3. Preuve de correction de la méthode

Définitions : Soit A un AP de E et E' sur V , de noeud initial N_0 .

Pour tout noeud N de A , on définit σ_N comme la composition des composantes de substitution sur le chemin de N_0 à N . C'est à dire :

$$\left\{ \begin{array}{l} - \sigma_{N_0} = \emptyset \\ - N \xrightarrow{\sigma} N' \implies \sigma_{N'} = \sigma \sigma_N \end{array} \right.$$

Finalement, on définit :

$$\Sigma(A) = \{\sigma_N \mid N = "S" \text{ dans } A\}.$$

On exprime la correction de la méthode de construction par :

Théorème 8 : Soit A un AP quelconque de E et E' sur V .

$\Sigma(A)$ est un ECP de E et E' sur V .

Démonstration :

Elle se décompose en une preuve de cohérence et une preuve de complétude, correspondant aux deux conditions dans la définition d'un ECP. La cohérence suit du lemme 3.5 ci-dessous, et la complétude du lemme 3.6 ci-dessous. \square

Lemme 3.5. (de cohérence)

Soit A un AP de E et E' sur V , N un noeud succès quelconque de A .

Alors :

$$\sigma_N \in P(E, E').$$

Démonstration :

Notons la branche de N_0 , sommet de A , à N , par :

$$N_0 \xrightarrow{\sigma_1} N_1 \xrightarrow{\sigma_2} \dots \xrightarrow{\sigma_p} N_p = N = "S" \quad p \geq 0.$$

On définit des unificandes \bar{N}_i $i \leq 0$ par :

$$\begin{cases} \bar{N}_0 = \{<E, E'>\} \\ \bar{N}_i = \sigma_i N_{i-1} \quad 0 < i \leq p. \end{cases}$$

On a bien sûr $N_i = \text{SIMPL}(\bar{N}_i) \quad \forall i \quad 0 \leq i \leq p$.

Soit V_N défini comme en 3.4.1. Pour tout $i \geq 0$, on a $V[\bar{N}_i] \subset V_{N_i}$.

On définit par récurrence (descendante) $\xi_i \in S$ par

$$\begin{cases} \xi_p = \zeta N_{\bar{N}_p} \quad \text{avec } \zeta \text{ défini en 3.1.2.} \\ \xi_{i-1} = \xi_i \sigma_i \quad 0 < i \leq p. \end{cases}$$

Prouvons par récurrence que $\xi_i \in U(\bar{N}_i) \quad \forall i \quad 0 \leq i \leq p$.

La base ($i=p$) est donnée par le lemme 3.1.

Supposons que $\xi_i \in U(\bar{N}_i)$, $i > 0$. C'est à dire que, pour tout

$\langle e_1, e_2 \rangle$ de N_{i-1} :

$$\xi_i \sigma_i e_1 = \xi_i \sigma_i e_2 \text{ et donc } \xi_{i-1} \in U(N_{i-1}) = U(\bar{N}_{i-1})$$

par le lemme 3.3.

On en déduit donc, pour $i=0$:

$$\xi_0 = \xi_p \sigma_N \in U(E, E'), \text{ d'où le résultat. } \square$$

Lemme 3.6. (de complétude)

Soit $\rho \in P(E, E')$, A un AP de E et E' sur V . Il existe dans A un noeud terminal succès N tel que $\sigma_N \leq \rho$.

Démonstration :

Nous allons sélectionner dans A un chemin :

$$N_0 \xrightarrow{\sigma_1} N_1 \xrightarrow{\sigma_2} \dots \xrightarrow{\sigma_i} N_i \dots \quad i \geq 0,$$

et une séquence de substitutions $\xi_0, \xi_1, \dots, \xi_i$ telles que, en dénotant par \bar{N}_i les unificandes avant simplification, comme dans le lemme précédent, on ait :

$$\left| \begin{array}{ll} \forall i \geq 0 & \xi_i \in P(\bar{N}_i) \\ \forall i > 0 & \pi(\xi_i) < \pi(\xi_{i-1}) \\ \rho = \xi_i \sigma_{N_i} & \end{array} \right. \quad \begin{array}{l} (1) \\ (2) \\ (3) \end{array}$$

Nous prouvons ces propriétés par récurrence sur i . Pour $i=0$, elles sont aisément vérifiées en prenant $\xi_0 = \rho$.

Supposons-les vraies pour i . Si $N_i = "S"$, nous arrêtons, et on a bien par (3) : $\sigma_{N_i} \leq \rho$. Sinon, on ne peut pas avoir $N_i = "E"$, car $N_i = SIMPL(\bar{N}_i)$, et par le lemme 3.3 on aurait $U(\bar{N}_i) = \emptyset$, donc $P(\bar{N}_i) = \emptyset$, ce qui contredit $\xi_i \in P(\bar{N}_i)$. N_i est donc un unificande réduit, et $U(N_i) = U(\bar{N}_i)$ par le lemme 3.3. $\xi_i \mid_{V_{N_i}} \in U(N_i)$ entraîne $\xi_j \in P(N_i)$. Soit $\langle e_1, e_2 \rangle$ la paire sélectionnée dans N_i lors de la construction de A .

On sait par le lemme 3.4 que, puisque en particulier $\xi_i \in P(e_1, e_2)$, il existe dans $\Sigma = CHOIX(e_1, e_2, V_{N_i})$ un σ unique tel que $\exists n$:

$$\left| \begin{array}{l} \xi_i \models_{V_{N_i}} n\sigma \\ \pi(n) < \pi(\xi_i) \end{array} \right.$$

(Remarquons que cela implique $\Sigma \neq \emptyset$, et donc N_i ne peut être un noeud échec de seconde espèce).

Choisissons pour N_{i+1} le noeud successeur de N_i correspondant à σ , et pour $\xi_{i+1} : n$. (2) est immédiatement vérifié. Pour tout couple $\langle e_1, e_2 \rangle$ dans N_i : $\xi_i e_1 \sim \xi_i e_2$ par hypothèse, et donc $\xi_{i+1} \sigma e_1 \sim \xi_{i+1} \sigma e_2$ puisque $V[N_i] \subset V_{N_i}$, donc $\xi_{i+1} \in P(\bar{N}_{i+1})$ ce qui vérifie (1). Par définition

de V_{N_i} , il est facile de montrer que

$$I(\sigma_{N_i}) \subset V_{N_i}.$$

Comme $\sigma_{i+1} = \sigma$, on a $\xi_i \bar{V}_{N_i} \xi_{i+1} \sigma_{i+1}$ et, en appliquant le lemme 2.14 on obtient :

$$\xi_i \sigma_{N_i} \bar{V}_{N_i} \xi_{i+1} \sigma_{i+1} \sigma_{N_i} = \xi_{i+1} \sigma_{N_{i+1}}.$$

Par hypothèse $\rho \bar{V} \xi_i \sigma_{N_i}$, et on a donc $\xi_{i+1} \sigma_{N_{i+1}} \bar{V} \rho$, puisque $V \subset V_{N_i}$, ce qui vérifie (3).

A cause de la condition (2) le processus doit s'arrêter après p étapes. On a donc $N_p = "S"$, et par (3) $\rho \bar{V} \xi_p \sigma_{N_p}$. \square

Avant de poursuivre, illustrons la preuve ci-dessus par un exemple.

Exemple 4 :

$$\begin{cases} E = \lambda w \cdot f(\lambda v \cdot v(w)) \\ E' = \lambda v \cdot w \\ V = \{f\} \end{cases}$$

$\rho = \{\langle f, \lambda u \cdot u(\lambda w \cdot w) \rangle\}$, avec

$$\begin{cases} \tau(w) = \gamma \\ \tau(v) = (\gamma \rightarrow \gamma) \\ \tau(u) = ((\gamma \rightarrow \gamma) \rightarrow \gamma) \\ \tau(f) = (((\gamma \rightarrow \gamma) \rightarrow \gamma) \rightarrow \gamma) \circ \end{cases}$$

On vérifie que $\rho \in \mathcal{U}(E, F') \subset P(E, E')$. La branche correspondante construite dans l'AP de E et F' sur V est :

$$\begin{aligned} N_0 &= \{\langle \lambda w \cdot f(\lambda v \cdot v(w)), \lambda w \cdot w \rangle\} \\ \sigma_1 &= \downarrow \langle f, \lambda u \cdot u(h(u)) \rangle \\ &\quad \text{avec } \tau(h) = (((\gamma \rightarrow \gamma) \rightarrow \gamma) \times \gamma \rightarrow \gamma) \\ N_1 &= \{\langle \lambda w \cdot h(\lambda v \cdot v(w), w), \lambda w \cdot w \rangle\} \\ \sigma_2 &= \downarrow \langle h, \lambda u \cdot w \rangle \\ N_2 &= \boxed{S} \end{aligned}$$

Avec les notations de la preuve ci-dessus, on a :

$$\left| \begin{array}{lll} \xi_0 = \rho & \pi(\xi_0) = 2 & \sigma_{N_0} = \emptyset \\ \xi_1 = \{\langle h, \lambda u w \cdot w \rangle\} & \pi(\xi_1) = 1 & \sigma_{N_1} = \{\langle f, \lambda u \cdot u(h(u)) \rangle\} \\ \xi_2 = \emptyset & \pi(\xi_2) = 0 & \sigma_{N_2} = \{\langle h, \lambda u w \cdot w \rangle\} \cup \rho \end{array} \right.$$

et on a bien $\sigma_{N_2} \leq_V \rho$ (ici on a même $\sigma_{N_2} = \rho$). \square

Ceci achève la preuve du théorème 8. (en effet, si $\sigma_{N \cap V} \leq_V \rho$, alors $\sigma_{N \cap V} \leq_V \rho$ aussi). Comme nous l'avons remarqué en 3.1.1, si E et E' ne sont pas unifiables, alors \emptyset est l'unique ECP de E et E' sur V ; si E et E' sont unifiables, \emptyset n'est pas un FCP. Ceci nous permet donc de reconnaître l'unifiabilité de E et E' par la construction d'un AP. On peut alors interpréter le théorème 8 comme suit :

Corollaire :

Soit E et E' deux termes de même type, V un ensemble fini de variables contenant $V[F] \cup V[E']$, à un AP quelconque de E et E' sur V . E et E' sont unifiables si et seulement si A possède un noeud succès à un niveau fini.

De plus, tout noeud succès permet de construire un unificateur de E et E' . Cette méthode est effective, le degré de branchement étant fini à chaque noeud.

Si l'on désire simplement rechercher l'unifiabilité, il suffit de choisir $V_N = V[N] - \{v_N\}$: la démonstration du lemme 3.6 est toujours valide, sur les seules propriétés (1) et (2). Ceci permet d'autoriser des branches comme suit :

$$\left| \begin{array}{l}
 \{<f(x,y), A(B)>\} \xrightarrow{f, \lambda uv \cdot u} \{<x, A(B)>\} \xrightarrow{x, A(y)} \{y, B>\} \xrightarrow{y, B} \textcircled{S} \\
 \{<f(\lambda u \cdot y), A(B)>\} \xrightarrow{f, \lambda x \cdot x(h(x))} \{<y, A(B)>\} \xrightarrow{y, A(h)} \{<h, B>\} \xrightarrow{h, B} \textcircled{S} \\
 \{<x, A(B)>\} \xrightarrow{x, A(x)} \{<x, B>\} \xrightarrow{x, B} \textcircled{S}
 \end{array} \right.$$

Il n'y a en fait aucun intérêt à perdre sur la généralité des préunificateurs construits, car cela ne raccourtit en rien la recherche d'une solution. De fait, V_N est essentiellement un artifice technique pour faciliter les preuves. Dans la pratique, il n'est pas nécessaire de le calculer, et il suffit que CHOIX génère toujours des nouvelles variables.

La construction d'un AP peut être utilisée également pour générer un ECP. Remarquons que cela ne nous donne pas en général un ECU. En effet, nous ne connaissons pas toujours un ECU pour un noeud succès avant simplification. En fait, la recherche d'unificateurs dans le cas flexible-flexible est très générale et non dirigée, ce qui produit beaucoup de redondance dans les calculs. Nous prenons donc avantage de ce phénomène, en repoussant à la fin ces cas difficiles.

Nous allons voir dans le paragraphe suivant comment nous pouvons améliorer encore l'algorithme de construction d'un AF, de manière à ce que l'ensemble des solutions forme un ECPI.

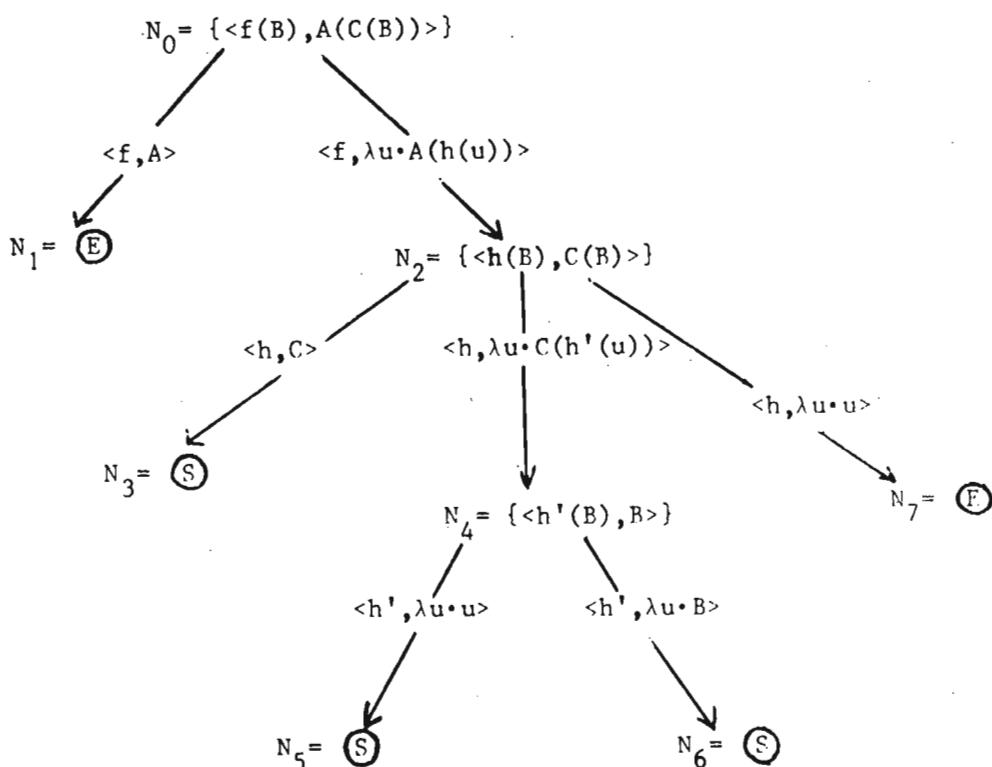
3.5. Arbres de préunification indépendants

Tout d'abord, montrons sur un exemple que nous générerons des noeuds inutiles dans un AP.

Exemple 5 :

Soit $E = f(B)$, $E' = A(C(B))$, $V = \{f\}$

avec $\begin{cases} \tau(f) = \tau(A) = (\beta \rightarrow \alpha) \\ \tau(B) = \beta \\ \tau(C) = (\beta \rightarrow \beta). \end{cases}$ Considérons l'AP A :



Du point de vue de la génération d'un ECP N_3 est redondant,

puisque $\sigma_{N_5} \equiv \sigma_{N_3}$.

Du point de vue de l'existence d'un unificateur pour E et E' ,

N_1 est également redondant, car $\sigma_{N_7}|_V = \{\langle f, \lambda u \cdot A(u) \rangle\}$ est une substitution plus générale que $\sigma_{N_1}|_V = \{\langle f, A \rangle\}$ dans le contexte $f(B)$. Pour exprimer cela précisément, nous avons besoin de nouvelles définitions.

3.5.1. Définitions

Soit $e = \lambda u_1 \cdots u_n \cdot @ (e_1, \dots, e_p)$ un terme quelconque en forme canonique.

On définit récursivement :

$$\text{STM}(e) = \{e\} \cup \bigsqcup_{i=1}^p \text{STM}(e_i).$$

Les éléments de $\text{STM}(e)$ sont appelés *sous-termes maximaux* de e .

Par exemple, avec $e = f(A, B)$, $\text{STM}(e) = \{f(A, B), A, B\}$; f et $f(A)$ sont des sous-termes non maximaux. Autrement exprimé, les sous-termes maximaux de e sont les sous-termes de e :

1) qui ne sont pas de la forme $\lambda u \cdot e'$

et 2) qui n'apparaissent pas en partie gauche d'une application.

Les sous-termes maximaux sont les sous-expressions de nos représentations abrégées, sans leurs en-têtes.

Soit $e \in T$, $f \in V[e]$. On appelle *degré de f dans e* l'entier :

$$\delta_f(e) = \min\{i \mid \exists e_1, \dots, e_i \quad f(e_1, e_2, \dots, e_i) \in \text{STM}(e)\}$$

3.5.2. Construction d'un arbre de préunification indépendant (API).

3.5.2.1. Modification de CHOIX

On rajoute à CHOIX un 4ème argument $\delta \geq 0$.

CHOIX (e_1, e_2, v, δ) construit les mêmes termes que l'algorithme donné en 3.3.1, sauf qu'il se restreint à générer des composantes

$$\langle f, e \rangle \quad \text{avec} \quad \bar{H}[e] \geq \delta$$

3.5.2.2. Construction d'un API.

Elle procède comme au 3.4.1. Pour chaque noeud N construit, avec v_N et v_n définis comme en 3.4.1, on construit

$$\Sigma = \text{CHOIX}(e_1, e_2, v_N, \delta_N), \text{ où } \delta_N \text{ est défini comme suit.}$$

-Si $v_N \in V$ (c'est à dire $v_N \in L = V[E] \cup V[E']$) alors $\delta_N = 0$.

-Sinon, soit N^{-1} le noeud ancêtre de N dans l'arbre le plus proche tel que $v_N \notin V[N^{-1}]$:

$$N_0 \rightarrow \dots \rightarrow N^{-1} \xrightarrow{< v_{N^{-1}}, e >} \dots \rightarrow N.$$

On doit alors avoir $v_N \in V[e]$. (C'est à dire qu'on a introduit v_N comme nouvelle variable, par CHOIX appelé en N^{-1}). On définit dans ce cas

$$\delta_N = \delta_{v_N}(e) = \bar{H}[e].$$

Le reste de la construction procède comme en 3.4.1.

Par exemple dans l'exemple 5 le noeud N_3 n'aurait pas été produit, car $\delta_{N_2} = 1$.

3.5.3. Preuve de correction

Définition :

Deux noeuds sont dits *indépendants* dans un arbre si l'un n'est pas un ancêtre de l'autre.

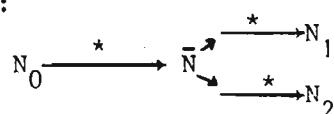
Lemme 3.7. d'indépendance

Soit A un API de E et E' sur V . Pour tout couple de noeuds indépendants N_1 et N_2 de A , on a

$$\sigma_{N_1} \mid \sigma_{N_2}, \text{ avec } L = V[E] \cup V[E'].$$

Démonstration :

Soit N_1 et N_2 deux noeuds indépendants de A , \bar{N} leur ancêtre commun le plus proche :



Pour tout noeud N dans la branche $N_0 \xrightarrow{*} \bar{N}$ on définit $\rho_1(N)$ (resp. $\rho_2(N)$) comme la composition des composantes qui étiquettent la branche de N à N_1 (resp. de N à N_2). On définit la propriété ϕ par :

$$\phi(N) \Leftrightarrow \rho_1(N) \underset{\{v_N\}}{\mid} \rho_2(N)$$

autrement dit :

$$\exists n_1, n_2 \quad n_1 \rho_1(N) v_N = n_2 \rho_2(N) v_N.$$

Nous allons prouver qu'il existe N entre N_0 et \bar{N} tel que :

$$(1) \quad \phi(N)$$

$$(2) \quad v_N \in L$$

Pour cela, soit N entre N_0 et \bar{N} tel que $v_N \notin L$ (i.e. $v_N \notin V$).

En définissant N^{-1} comme au 3.5.2.2 on prouve que $\phi(N) \Rightarrow \phi(N^{-1})$.

Définissons :

$$N^{-1} \xrightarrow{\sigma} N \xrightarrow{*} \bar{N} \quad \text{avec}$$

$$\sigma v_{N^{-1}} = e = \lambda w_1 \cdots w_n @ (\cdots, v_N(w_1, \dots, w_n), \cdots)$$

On suppose $\neg \phi(N^{-1})$, c'est à dire :

$$\exists n_1, n_2 \quad n_1 \rho_1(N^{-1}) v_{N^{-1}} = n_2 \rho_2(N^{-1}) v_{N^{-1}}$$

Par définition, $\rho_1(N^{-1}) = \rho_1(N) \xi \sigma$

et $\rho_2(N^{-1}) = \rho_2(N) \xi \sigma$. On obtient donc :

$n_1 \rho_1(N) \xi e = n_2 \rho_2(N) \xi e$. En remarquant que e est rigide, et en supposant que les w_i sont choisis de manière à ne pas créer de conflits, on a en particulier $n_1 \rho_1(N) \xi e' = n_2 \rho_2(N) \xi e'$ avec $e' = v_N(w_1, \dots, w_n)$.

Mais $v_N \in V[N]$ implique que $v_N \notin D(\xi)$ et donc que $\xi e' = e'$, d'où

$$n_1 \rho_1(N) e' = n_2 \rho_2(N) e', \quad \text{c'est à dire :}$$

$$[n_1 \rho_1(N) v_N]_{(w_1, \dots, w_n)} = [n_2 \rho_2(N) v_N]_{(w_1, \dots, w_n)}.$$

Soit maintenant σ_1 la composante initiale du chemin de N à N_1 .

Par définition, on a $\delta_N = n$. Par 3.5.2.1., on a donc $\bar{H}[\sigma_1 v_N] \geq n$, et puisque cette propriété se conserve par composition : $\bar{H}[\eta_1 \rho_1(N) v_N] \geq n$.

Puisque les w_i n'apparaissent pas dans $\eta_1 \rho_1(N) v_N$, on a alors

$$\lambda w_1 \cdots w_n [\bar{H}[\eta_1 \rho_1(N) v_N](w_1, \dots, w_n)] = \eta_1 \rho_1(N) v_N$$

par simple α -conversion. Par symétrie, on obtient donc :

$$\eta_1 \rho_1(N) v_N = \eta_2 \rho_2(N) v_N$$

c'est à dire $\bar{\eta}\phi(N)$. Par contraposition, on a bien $\phi(N) \Rightarrow \phi(N^{-1})$.

Ce résultat nous fournit le pas de récurrence. La base de la récurrence est $\phi(\bar{N})$, qui découle du fait que les composantes choisies par CHOIX sont indépendantes. Par récurrence, on obtient donc qu'il existe N entre N_0 et \bar{N} tel que $\phi(N)$ et $v_N \in L$. On a donc, pour cet N :

$$\sigma_{N_1} v_N = \rho_1(N) \sigma_N v_N = \rho_1(N) v_N \text{ et similairement}$$

$$\sigma_{N_2} v_N = \rho_2(N) v_N, \text{ d'où } \sigma_{N_1} \mid_L \sigma_{N_2}. \square$$

Ce lemme nous donne la condition d'indépendance des solutions obtenues aux noeuds terminaux d'un API. La condition de cohérence est bien sûr toujours valable, puisqu'on a seulement supprimé certaines solutions. Il reste à montrer que la condition de complétude est toujours vraie.

Lemme 3.8. de complétude des API

Soit $\rho \in P(E, E')$. Dans tout API A de E et E' sur V il existe un noeud terminal succès N tel que $\sigma_{N \mid V} \leq \rho$.

Démonstration :

C'est la même que pour le lemme 3.6, à laquelle on ajoute l'hypothèse de récurrence :

$$\forall x \in V_{N_i} \cap D(\xi_i) \quad \bar{H}[\xi_i x] \geq \Delta_x, \text{ avec } \Delta_x \text{ défini par :}$$

- si $x \in V$ alors $\Delta_x = \bar{H}[\rho x]$
- sinon, x a été introduit en N_j , $j < i$, par la composante $\langle v_{N_j}, e \rangle$; alors $\Delta_x = \bar{H}[e]$.

Pour la base, avec $\xi_0 = \rho$ et $V_{N_0} = V$, c'est évident. Pour le pas de récurrence, puisque la propriété est supposée vraie pour i on a, en particulier pour $x = v_N$:

$$\bar{H}[\xi_i v_N] \geq \Delta_x \geq \delta_N.$$

Le lemme 3.4 peut alors s'appliquer, et par construction :

$$\left\{ \begin{array}{l} \forall x \in (V_{N_i} \cap D(\xi_i)) - \{v_{N_i}\} \quad \xi_{i+1}x = \xi_i x, \\ \forall x \in V_{N_{i+1}} - V_{N_i} \quad \bar{H}[\xi_{i+1}x] \geq \bar{H}[\xi_i v_{N_i}] \geq \Delta_x. \end{array} \right.$$

Pour cette dernière inégalité, si on a $x = h_j$ (avec les notations de la preuve du lemme 3.4), alors

$$\bar{H}[\xi_i v_{N_i}] = k \quad \text{et} \quad \bar{H}[\xi_{i+1}x] = k + \bar{H}[E'_j].$$

Ceci permet de conclure le pas de récurrence, le reste de la preuve étant identique à celle du lemme 3.6. \square

Nous pouvons maintenant énoncer notre théorème d'adéquation de la construction :

Théorème 9 : Soit A un API quelconque de F et F' sur V . $\Sigma(A)$ est un FCP1 de F et F' sur V .

Nous pouvons donc récapituler les propriétés de $\Sigma(A)$ par :

$$\left\{ \begin{array}{ll} \forall \sigma \in \Sigma(A) & \sigma E \sim \sigma E' \\ \forall \rho \quad \rho E \sim \rho E' & \Rightarrow \exists ! \sigma \quad \sigma \leq \rho \\ \forall \sigma_1, \sigma_2 \in \Sigma(A) & \sigma_1 \mid_L \sigma_2 \quad \text{avec } L = V[E] \cup V[E']. \\ \sigma_1 \neq \sigma_2 & \end{array} \right.$$

Considérons par exemple E, E' et V comme donnés dans l'exemple 5 ci-dessus. On a $\delta_{N_2} = \bar{H}[\lambda u \cdot A(h(u))] = !$ et la composante $\langle h, C \rangle$ n'est donc pas générée. Soit A' l'arbre A sans le noeud N_3

A' est un API. Ici $\Sigma(A')$ contient deux éléments :

$$\begin{aligned} \sigma_5 \upharpoonright V &= \{\langle f, \lambda u \cdot A(C(u)) \rangle\} \\ \sigma_6 \upharpoonright V &= \{\langle f, \lambda u \cdot A(C(B)) \rangle\}. \end{aligned}$$

Dans ce cas, on obtient immédiatement des unificateurs, et $\Sigma(A')$ est donc un ECUM.

Le théorème 9 nous fournit un théorème d'existence d'un FCPI ; en le combinant avec le lemme 3.2., on obtient :

Corollaire :

Pour toute paire E, E' de termes de même type, et pour tout ensemble V fini de variables, avec $V[E] \cup V[E'] \subset V$, il existe un ECPI de E et E' sur V , qui est unique modulo \equiv_V .

En fait, $\Sigma(A)$ possède une propriété d'unicité qui est plus forte que l'équivalence \equiv_V ; soit A et A' deux API de E et E' sur V . Alors il existe une substitution ρ qui est une bijection dans $V-V$ (C'est à dire une permutation de variables non dans V) telle que $\Sigma(A') = \rho \Sigma(A)$. Autrement dit, $\Sigma(A)$ et $\Sigma(A')$ ne diffèrent que par le nom des variables h_i introduites par CHOIX. Nous ne prouverons pas ici cette propriété.

Cette propriété d'unicité de $\Sigma(A)$ peut être interprétée comme une propriété Church-Rosser de la construction des AP : choisir une paire $\langle e_1, e_2 \rangle$ plutôt qu'une autre n'affecte pas l'ensemble des solutions. Ce choix n'affecte même pas la hauteur dans l'arbre des différents noeuds succès. Par contre il peut affecter l'apparition des noeuds échec, et donc la terminaison dans le cas où $U(E, E') = \emptyset$.

Considérons l'exemple suivant :

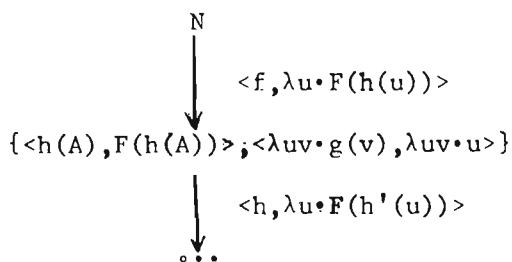
Soit dans un AP le noeud :

$$N = \{ \langle f(A), F(f(A)) \rangle, \langle \lambda u v. g(v), \lambda u v. u \rangle \}$$

avec

$\tau(A) = \tau(u) = \alpha$
$\tau(v) = \beta$
$\tau(f) = (\alpha \rightarrow \beta)$
$\tau(F) = (\beta \rightarrow \beta)$
$\tau(g) = (\beta \rightarrow \alpha).$

Si l'on décide de choisir en N la deuxième paire, CHOIX retourne immédiatement \emptyset (aucune règle n'est applicable) et la branche aboutissant à N s'arrête sur cet échec. Si l'on choisit la première paire, on peut aboutir à la construction d'une branche infinie :



Pour éviter ce genre de phénomène, il est souhaitable de représenter les unificandes comme des piles, et de donner à CHOIX la paire <flexible, rigide> la plus ancienne dans la pile.

Lorsque l'on est intéressé uniquement à l'existence d'unificateurs, nos API ne sont pas tout à fait minimaux. Par exemple, dans l'arbre A' pour l'exemple 5, le noeud N_1 est redondant. Nous allons montrer maintenant comment supprimer cette dernière redondance.

3.6. Arbres d'unifiabilité

3.6.1. Construction

Les arbres d'unifiabilité sont des API simplifiés. CHOIX est comme défini en 3.5.2.1.

Soit N le noeud courant de la construction, où l'on choisit la paire $\langle e_1, e_2 \rangle$, soit v_N la variable de tête de e_1 .

On définit $\delta'_N \geq 0$ par :

$$\delta'_N = \min_{v_N} \{\delta_{v_N}(e) \mid v_N \in V[e] \text{ et } (\langle e, e' \rangle \in N \vee \langle e', e \rangle \in N)\}.$$

(On a donc en particulier δ'_N inférieur ou égal au nombre d'arguments de e_1).

La définition de δ'_N n'est qu'apparemment compliquée ; intuitivement, c'est $\delta_{v_N}(N)$, avec N considéré comme un terme. δ'_N peut se calculer par simple inspection de N , alors que δ_N comme défini en 3.5. nécessite de remonter dans l'arbre jusqu'à N^{-1} .

On peut maintenant construire les successeurs de N , suivant l'ensemble de solutions de $\Sigma = \text{CHOIX } (e_1, e_2, V[N], \delta'_N)$.

Le reste de la construction est le même qu'en 3.4.1.

Un arbre construit de cette façon est appelé *arbre d'unifiabilité de E et E'* (abrégé AU). Remarquons qu'il n'apparaît plus d'ensemble V dans la construction.

En reprenant comme exemple notre exemple 5 de 3.5., les noeuds N_1 et N_3 n'auraient pas été produits, car $\delta'_{N_0} = \delta'_{N_2} = 1$.

3.6.2. Correction

Le théorème 8 n'est pas vrai des AU. Mais tout ce que nous désirons ici, c'est un équivalent de son corollaire.

Pour la cohérence, c'est immédiat puisqu'un AU est un AP dans lequel on a supprimé certaines branches (le lecteur vérifiera que le lemme 3.5. est toujours valable si on remplace V_N par $V[N]$). En fait, nous pouvons prouver davantage : les AU sont des API simplifiés. Pour cela, il nous suffit de montrer que, pour tout noeud N et pour tout choix de v_N , on a $\delta'_N \geq \delta_N$.

Lemme 3.9. $\delta'_N \geq \delta_N$.

Démonstration :

Soit E et E' des termes de même type, V un ensemble fini de variables contenant $V[E] \cup V[E']$, A un API de E et F' sur V . Soit N un noeud quelconque de A , v_N la variable choisie en N .

Si $v_N \in V$, alors le résultat est évident car $\delta_N = 0$. Sinon, soit N^{-1} l'ancêtre de N dans A où v_N a été introduite :

$$N^{-1} \xrightarrow{\sigma_1} N_1 \longrightarrow \dots \xrightarrow{\sigma_p} N_p = N$$

avec $\sigma_1 = \langle x, \lambda u_1 \dots u_n @ (\dots, v_N(u_1, \dots, u_n), \dots) \rangle$

Soit $\bar{N}_1 = \sigma_1 N^{-1}$. Dans \bar{N}_1 , tous les sous-termes maximaux de tête v_N ont au moins n arguments, et comme SIMPL laisse intacts les termes de tête v_N (puisque'ils sont flexibles) cette propriété est vraie de N_i . On prouve par récurrence qu'elle est vraie de N_i avec $i \geq p$.

Supposons la propriété vraie de N_i , avec $i \leq p$. Comme $v_{N_i} \neq v_N$, on a $\sigma_{i+1}[v_N(e_1, \dots, e_m)] = v_N(\sigma_{i+1}e_1, \dots, \sigma_{i+1}e_m)$; σ_{i+1} ne peut pas introduire v_N , par définition de N^{-1} . Le nombre d'arguments de v_N ne peut donc qu'augmenter, et la propriété est donc vraie de N_{i+1} . Elle est donc vraie de N , par récurrence. Comme il existe au moins une occurrence de v_N dans N (la tête du premier terme de la paire sélectionnée), cela prouve bien que $\delta'_N \geq n = \delta_N$. \square

La preuve que nous venons de donner n'est pas complètement formalisée. Pour faire une preuve rigoureuse, il nous faudrait démontrer plusieurs lemmes "techniques" sur $\delta_x(e)$ qui sont intuitivement triviaux.

Le résultat n'étant pas essentiel pour la suite, nous avons préféré donner une preuve plus informelle.

Pour la complétude, nous prouvons un lemme plus faible que 3.6.
Donnons d'abord quelques définitions.

Définitions :

Pour tout type $\alpha \in T$, on définit l'*arité composée* de α par :

$$\#(\alpha) = \begin{cases} \text{si } \alpha \in T_0 & \text{alors } 0 \\ \text{sinon } \alpha = (\alpha_1 \times \alpha_2 \times \dots \times \alpha_n \rightarrow \beta) \text{ avec } \beta \in T_0 \text{ alors } n + \sum_{i=1}^n \#(\alpha_i). \end{cases}$$

Pour tout terme $e \in T$, on définit la *complexité* $\chi(e)$ de e par :
 $\chi(\lambda u_1 u_2 \dots u_p \cdot @ (e_1, \dots, e_p)) = \sum_{i=1}^p \chi(e_i) + \sum_{i=p+1}^q \#(\alpha_i) + q$, où
 $\tau(@) = (\alpha_1 \times \alpha_2 \times \dots \times \alpha_q \rightarrow \beta)$.

La complexité $\chi(e)$ d'un terme est une mesure qui croît avec la taille $\pi(e)$, mais qui est plus fine car elle tient compte des types apparaissant dans e .

Exemple 6 : Soit $e = f(x)$, avec $\tau(x) = (\gamma \times (\gamma \rightarrow \gamma) \rightarrow \gamma)$

et $\tau(f) = ((\gamma \times (\gamma \rightarrow \gamma) \rightarrow \gamma) \times (\gamma \rightarrow \gamma) \rightarrow \gamma)$. On a $\pi(e) = 1$,

mais $\chi(e) = \chi(x) + \#((\gamma \rightarrow \gamma)) + 2 = 3 + 1 + 2 = 6$. \square

Définition :

On étend χ à S de la même manière que π :

Soit $\sigma = \{<x_i, e_i> \mid 1 \leq i \leq n\}$, $n = |\mathcal{D}(\sigma)|$. On définit :

$$\chi(\sigma) = n + \sum_{i=1}^n \chi(e_i).$$

La précaution $n = |\mathcal{D}(\sigma)|$ est importante ici, car on n'a pas toujours $\chi(x) = 0$, pour $x \in V$. En fait :

Lemme 3.10 :

$$\forall x \in V_\alpha \quad \chi(x) = \#(\alpha).$$

Démonstration :

Soit $x \in V_\alpha$, avec $\alpha = (\alpha_1 \times \alpha_2 \times \dots \times \alpha_n \rightarrow \beta)$. Alors, par définition :

$$\chi(x) = \sum_{i=1}^n \#(\alpha_i) + n = \#(\alpha). \quad \square$$

Définition :

Soit $e = \lambda u_1 \dots u_n \cdot @ (e_1, \dots, e_p)$, avec

$\tau(e) = (\alpha_1 \times \dots \times \alpha_n \times \dots \times \alpha_m \rightarrow \beta)$, $m \geq n$, $\beta \in T_0$.

Pour tout k tel que $n \leq k \leq m$, on définit le terme e prolongé à l'arité k par :

$$e \uparrow k = \lambda u_1 \dots u_n u_{n+1} \dots u_k \cdot @ (e_1, \dots, e_p, u_{n+1}, \dots, u_k),$$

où les $u_{n+1} \dots u_k$ sont des variables de types respectivement $\alpha_{n+1} \dots \alpha_k$ n'apparaissant pas dans e .

Par construction, on a $H[e \uparrow k] = k$. De plus :

Lemme 3.11.

$\forall x, e, E$ tels que $\tau(x) = \tau(e)$, $\forall n$ tel que $x \notin D(n)$, $\forall k :$

$$\bar{H}[e] \leq k \leq \delta_x(E) \implies \sigma E = \sigma' E,$$

avec $\sigma = n \cup \{<x, e>\}$ et $\sigma' = n \cup \{<x, e+k>\}$.

Démonstration :

Par récurrence structurelle sur E .

Soit $e = \lambda u_1 \dots u_n @ (e_1, \dots, e_p)$.

On suppose $E = \lambda v_1 \dots v_m @' (E_1, \dots, E_q)$,

avec $\sigma E_i \neq \sigma' E_i \quad 1 \leq i \leq q$.

$$\begin{aligned} (i) \quad @' &\neq x. \text{ Alors } \sigma E = \lambda v_1 \dots v_m @' (\sigma E_1, \dots, \sigma E_q) \\ &= \lambda v_1 \dots v_m @' (\sigma' E_1, \dots, \sigma' E_q) \\ &= \sigma' E. \end{aligned}$$

$$(ii) \quad @' = x.$$

On a $q \geq \delta_x(E)$ par définition de δ

$\geq k$ par hypothèse.

$\geq n$ par hypothèse.

En utilisant le lemme 2.3., et en supposant qu'il n'y a pas de conflits,

on a : $\sigma E = \lambda v_1 \dots v_m @ (\rho e_1, \dots, \rho e_p, \sigma F_{n+1}, \dots, \sigma F_q)$

et : $\sigma' E = \lambda v_1 \dots v_m @' (\rho' e_1, \dots, \rho' e_p, \rho' u_{p+1}, \dots, \rho' u_k, \sigma' E_{k+1}, \dots, \sigma' E_q)$

avec $\rho = \{<u_i, \sigma E_i> \mid 1 \leq i \leq n\}$

et $\rho' = \{<u_j, \sigma' E_i> \mid 1 \leq i \leq k\}$.

(a) $1 \leq i \leq p$ Puisque u_j n'apparaît pas dans e_j $n < j \leq k$ par construction de $e+k$, on a :

$$\rho' e_i = \rho'' e_i \quad \text{avec} \quad \rho'' = \rho' \setminus \{u_1, \dots, u_n\}$$

$$= \rho e_i \quad \text{car} \quad \sigma F_\ell = \sigma' E_\ell \quad 1 \leq \ell \leq n \text{ par hypothèse}$$

de récurrence.

(b) $p < i \leq k$ $\rho' u_i = \sigma' E_i = \sigma E_i$ par hypothèse de récurrence.

(c) $k < i \leq q$ $\sigma' E_i = \sigma E_i$ par hypothèse de récurrence.

ce qui conclut $\sigma E = \sigma' E$ dans le cas (ii). \square

Lemme 3.12 : $\forall e$ et k tels que $e \uparrow k$ ait un sens : $\chi(e \uparrow k) = \chi(e)$.

Démonstration :

Immédiat par la définition de $e \uparrow k$, en utilisant le lemme 3.10. \square

Nous sommes maintenant en mesure de prouver notre lemme de complétude.

Lemme 3.13 de complétude des AU.

Soit E et E' deux termes de même type unifiables. Tout AU de E et E' possède un noeud succès à distance finie.

Démonstration :

Soit $\rho \in U(E, E')$, A un AU de E et E' . La démonstration procède comme pour le lemme 3.6 , mais avec des hypothèses de récurrence moins fortes. C'est à dire qu'on cherche dans A un chemin

$$N_0 \xrightarrow{\sigma_1} N_1 \rightarrow \dots \rightarrow \xrightarrow{\sigma_i} N_i \dots$$

et des substitutions ξ_0, ξ_1, \dots , telles que, avec \bar{N}_i dénotant N_i avant simplification, on ait :

$$\left\{ \begin{array}{ll} \forall i \geq 0 & \xi_i \in U(\bar{N}_i) \\ \forall i > 0 & \chi(\xi_i) < \chi(\xi_{i-1}) \end{array} \right. \quad \begin{array}{l} (1) \\ (2) \end{array}$$

Pour $i=0$, prenons $\xi_0 = \rho$.

Supposons maintenant (1) et (2) vrais pour i .

Si $N_i = "S"$, on arrête et le lemme est prouvé.

Sinon N_i ne peut pas être un noeud échec de première espèce, par le lemme 3.3, puisque $\xi_i \in U(\bar{N}_i)$. Posons $N_j = SIMPL(\bar{N}_i)$.

$U(N_i) = U(\bar{N}_i)$ implique $\xi_i \in U(N_i)$. Définissons :

- $x = v_{N_i}$
- $e = \xi_i x = \lambda u_1 \cdots u_n @ (E'_1, \dots, E'_p)$,
- $\bar{\xi}_i = \xi_i \upharpoonright (\mathcal{D}(\xi_i) - \{x\})$.
- $k = \delta'_{N_i}$
- $\rho_i = \begin{cases} \text{si } k \leq n \text{ alors } \xi_i \\ \text{sinon } \bar{\xi}_i \cup \{<x, e \uparrow k>\} \end{cases}$

Pour tout $<e_1, e_2>$ dans N_i , prouvons $\rho_i e_1 = \xi_i e_1$. Si $k \leq n$, c'est immédiat. Sinon $H[e] = n \leq k \leq \delta_x(e_1)$ par définition de $\delta'_{N_i} = k$. On peut donc appliquer le lemme 3.11 pour prouver $\rho_i e_1 = \xi_i e_1$. De même $\rho_i e_2 = \xi_i e_2$. Ceci entraîne donc que $\rho_i \in U(N_i)$. Mais maintenant $H[\rho_i x] \geq k$ implique, comme pour le lemme 3.3, que CHOIX va retourner une composante solution σ telle que $H[\sigma x] = H[\rho_i x]$. Ceci entraîne en particulier que N_i n'est pas un noeud échec de seconde espèce, et on peut prendre $\sigma_{i+1} = \sigma$, et N_{i+1} comme le successeur correspondant de N_i dans A . Examinons les deux cas :

$$(i) \quad k \leq n \quad \sigma x = \lambda u_1 \cdots u_n @ (E_1, \dots, E_p)$$

avec $E_i = h_i(u_1, \dots, u_n)$, $h_i \notin V[N_i]$.

$$\text{Soit } \eta = \bar{\xi}_i \cup \{<h_i, \lambda u_1 \cdots u_n @ E'_i>\}$$

$$\text{On a } \rho_i = \xi_i = \eta \sigma, \quad \text{d'où } \eta \in U(\bar{N}_{i+1}),$$

$$\text{et } x(\eta) = x(\bar{\xi}_i) + \sum_{i=1}^p x(E'_i) + p > x(\xi_i).$$

$$(ii) \quad k = n+m \quad m > 0 \quad \sigma x = \lambda u_1 \cdots u_k @ (E_1, \dots, E_{p+m}) \text{ pour respecter le type de } x,$$

avec $E_i = h_i(u_1, \dots, u_k)$, $h_i \notin V[N_i]$.

Soit $n = \bar{\xi}_i \cup \{<h_i, \lambda u_1 \cdots u_k \cdot E'_i> \mid 1 \leq i \leq p\}$.

$$\cup \{<h_{p+j}, \lambda u_1 \cdots u_k \cdot u_{n+j}> \mid 1 \leq j \leq m\}.$$

De même qu'au cas (i), nous avons ici

$$\underset{V[N_i]}{=} n\sigma, \text{ d'où } n \in U(\bar{N}_{i+1}),$$

et $\chi(n) > \chi(p_i) = \chi(\xi_i)$ par le lemme 3.12.

Dans les deux cas on peut donc choisir $\xi_{i+1} = n$, ce qui prouve les deux hypothèses de récurrence pour $i+1$.

A cause de (2) la construction doit donc s'arrêter sur un noeud succès, ce qui termine la preuve. \square

Pour illustrer la preuve ci-dessus, donnons tout de suite un exemple.

Exemple 7 :

$$\text{Soit } \begin{cases} E = f(\lambda v w \cdot w(A), B) \\ E' = B(A) \end{cases}$$

$$\text{avec } \begin{cases} \tau(A) = \tau(v) = \gamma \\ \tau(B) = \tau(w) = (\gamma \rightarrow \gamma) \\ \tau(f) = ((\gamma \times (\gamma \rightarrow \gamma) \rightarrow \gamma) \times (\gamma \rightarrow \gamma) \rightarrow \gamma) \end{cases} \quad \gamma \in T_0.$$

Considérons une portion d'un API de E et E' sur $\{f\}$:

$$N_0 = \{<f(\lambda vw \cdot w(A), B), B(A)>\}$$

$$<f, \lambda u \cdot u(h(u))>$$

$$\sigma_1 = <f, \lambda uw \cdot u(h_1(u, w), h_2(u, w))>$$

$$N_4 = \textcircled{S}$$

$$N_1 = \{<h_2(\lambda vw \cdot w(A), B, A), B(A)>\}$$

$$<h_2, \lambda uw \cdot w>$$

$$\sigma_2 = <h_2, \lambda uwv \cdot w(h_3(u, w, v))>$$

$$N_5 = \textcircled{S}$$

$$N_2 = \{<h_3(\lambda vw \cdot w(A), B, A), A>\}$$

$$\sigma_3 = <h_3, \lambda uwv \cdot w>$$

$$N_3 = \textcircled{S}$$

Nous ignorons délibérément les autres branches de l'arbre. En fait, nous sommes intéressés en l'unificateur

$$\xi_0 = \{<f, \lambda u \cdot u(C)>\} \text{ de } E \text{ et } E', \text{ avec } \tau(C) = \gamma. \xi_0 \text{ est obtenu}$$

directement par composition avec σ_{N_4} , mais N_4 ne serait pas généré si on se restreint aux AU. Avec les notations de la preuve ci-dessus,

$$\text{on a : } \rho_0 = \{<f, \lambda uw \cdot u(C, w)>\}, \chi(\rho_0) = \chi(\xi_0) = 4$$

$$\xi_1 = \{<h_1, \lambda uw \cdot C>, <h_2, \lambda uw \cdot w>\}, \chi(\xi_1) = 3$$

Ici de même, N_5 n'est pas générée, car $\delta_{N_1}^! = 3$. On doit donc construire $\rho_1 = \{\langle h_1, \lambda uw \cdot C \rangle, \langle h_2, \lambda uwv \cdot w(v) \rangle\}$, $x(\rho_1) = 3$.

$$\xi_2 = \{\langle h_1, \lambda uw \cdot C \rangle, \langle h_3, \lambda uwv \cdot v \rangle\}, \quad x(\xi_2) = 2.$$

$$\text{Maintenant } \rho_2 = \xi_2, \text{ et } \xi_3 = \{\langle h_1, \lambda uw \cdot C \rangle\}, \quad x(\xi_3) = 1.$$

Comme $N_3 = \emptyset$, la construction s'arrête. Le préunificateur (qui est ici un unificateur) Δ E et E' trouvé est donc :

$$\sigma_{N_3} \Delta \{f\} = \{\langle f, \lambda uw \cdot u(h_1(u, w), \lambda v \cdot w(v)) \rangle\}. \square$$

Cet exemple nous montre qu'un AU peut avoir son premier noeud succès plus profond qu'un API. Par contre, les AU permettent en général une convergence plus rapide, car le degré de branchement des noeuds est plus petit, ce qui est très important car c'est de lui dont dépend le coefficient exponentiel. L'exemple ci-dessus est bien sûr "pathologique", et non représentatif de ce qui se passe en pratique.

Nous pouvons maintenant énoncer notre théorème de correction des AU, conséquence immédiate du lemme 3.13.

Théorème 10 :

Soit E et E' des termes de même type, A un AU quelconque de E et E' . E et E' sont unifiables si et seulement si A possède un noeud succès à distance finie.

Remarques :

1) On pourrait penser que la construction d'un API comme définie en 3.5. est trop compliquée, et en s'inspirant des résultats de ce paragraphe définir une autre construction, où au noeud N on appellerait $\text{CHOIX}(e_1, e_2, v_N, \delta''_N)$,

avec :

$$\delta''_N = \begin{cases} \text{si } v_N \in V \text{ alors } 0 \\ \text{sinon } \delta'_N. \end{cases}$$

Mais cette construction est incomplète, en ce sens que l'ensemble des solutions obtenues ne forme pas un ECP. L'exemple 6 ci-dessus nous fournit un contre-exemple : le noeud N_5 serait rejeté, car en $N_1 : \delta''_{N_1} = \delta'_{N_1} = 3 > \bar{H}[\lambda uw \cdot w] = 2$. Le préunificateur $\rho = \{\langle f, \lambda uw \cdot u(A, w) \rangle\}$ n'est minoré par aucune des solutions conservées.

2) $e \# k$ se déduit de e par une série de η -expansions. Lorsque nous autorisons la η -conversion comme règle de conversion supplémentaire, on a donc $e \# k = e$. Il semble donc que, dans ce langage, notre construction va se simplifier parce que beaucoup de solutions vont se factoriser en une solution unique. C'est ce que nous verrons au chapitre 4.

3.7. Cas particuliers

Nous allons nous intéresser à quelques cas particuliers du problème de l'unification, et montrer que l'on peut conclure plus précisément dans ces cas. En particulier, nous montrerons en 3.7.2 comment la demi-unification peut se ramener à un cas particulier de l'unification.

3.7.1. Cas d'un terme fermé

Nous allons supposer ici que E' est fermé, c'est à dire que $V[E'] = \emptyset$.

Remarquons tout d'abord que le problème du filtrage de E vers F' est identique ici à l'unification de E et E' ; tout filtre est un unificateur, et réciproquement :

$$F(E, E') = U(E, E').$$

De plus, tout pré-unificateur est un unificateur, comme le montre le lemme suivant.

Lemme 3.14

Si $V[E'] = \emptyset$, alors $\forall e \quad e \sim E' \implies e = E'$.

Démonstration :

Par récurrence sur F' . \square

Corollaire

Si $V[E'] = \emptyset$, alors $\forall F \quad P(F, E') = U(F, E') = F(F, E')$.

Démonstration :

Soit $\sigma \in P(E, F')$. Faire $e = \sigma F$ dans le lemme 3.14. \square

Dans ce cas, le théorème 9 nous permet de conclure.

Définissons un ensemble complet d'unificateurs indépendants (FCUI)

comme un ECUM Σ vérifiant la condition d'indépendance :

$$\forall \sigma, \sigma' \in \Sigma \quad \begin{matrix} \sigma \mid \sigma' \\ \sigma \neq \sigma' \end{matrix} \quad L \quad \text{avec } L = V[E] \cup V[E'].$$

On définit de même des ECDI.

On peut maintenant énoncer :

Lemme 3.15 Soit E et E' deux termes de même type, avec $V[E'] = \emptyset$.

Soit A un API quelconque de E et E' sur V , avec $V[E] \subset V$. Alors $\Sigma(A)$ est un ECUI de E et E' sur V , et donc aussi un ECDI de E vers E' sur V . (Rappelons que par définition $\forall \sigma \in \Sigma(A) \quad D(\sigma) \subset V[E] \cup V[E']$).

Nous allons voir dans le prochain paragraphe que l'on peut toujours utiliser un ensemble complet de pré-unificateurs pour définir un ensemble complet de demi-unificateurs même lorsque $V[E'] \neq \emptyset$. Mais la condition d'indépendance sera remplacée, dans le cas général, par la condition plus faible de minimalité.

3.7.2. Demi-unification

Rappelons qu'un demi-unificateur σ de E vers E' est un filtre laissant invariantes les variables de E' :

$$\sigma E = E' \quad \text{avec} \quad D(\sigma) \subset V[F] - V[E'].$$

Il est facile de ramener le problème de la demi-unification au cas particulier d'unification avec un terme fermé, en "gelant" les variables de E' .

Plus précisément, soit $V[E'] = \{x_1, \dots, x_k\}$.

Introduisons n nouvelles constantes x_1, \dots, x_k , de types $\tau(x_i) = \tau(x'_i)$, et considérons la substitution :

$$\xi = \{<x_i, x'_i> \mid 1 \leq i \leq k\}.$$

Une telle substitution peut naturellement s'inverser.

C'est à dire, pour tout terme e construit sur

$C' = C \cup \{x_1, \dots, x_k\}$ avec $V[e] \cap V[F'] = \emptyset$, il existe un e' unique construit sur C , et tel que $\xi e' = e$. En effet, on peut définir un tel e' comme étant $\xi^{-1}[e]$, avec :

$$\xi^{-1}[\lambda u_1 \dots u_n @ (e_1, \dots, e_p)] = \lambda u_1 \dots u_n @' (\xi^{-1}[e_1], \dots, \xi^{-1}[e_p]),$$

$$\text{avec } @' = \begin{cases} @ & \text{si } @ \in V \cup C \\ x_i & \text{si } @ = x_i \quad 1 \leq i \leq k \end{cases}$$

Les crochets servent à insister sur le fait que ξ^{-1} n'est pas une substitution. Il est facile de montrer, par récurrence sur e , que :

$$\begin{cases} \xi \xi^{-1}[e] = e & \text{pour tout } e \text{ tel que } V[E] \cap V[F'] = \emptyset \\ \xi^{-1}[\xi e] = e & \text{pour tout } e \text{ construit sur } C. \end{cases}$$

Nous allons maintenant montrer comment il est possible de construire un ECDM de E vers E' à partir d'un FCPI de ξE et $\xi E'$.

Lemme 3.16

Soit E et E' deux termes de même type, et soit $V = V[F] - V[F']$, $L = V[E] \cup V[E']$.

Soit Σ un FCPI de ξE et $\xi E'$ sur V , tel que pour tout σ dans Σ on ait $I(\sigma) \cap V[F'] = \emptyset$.

Pour tout σ dans Σ , on définit $\sigma' = \{\langle x, \xi^{-1}[\sigma x] \rangle \mid x \in V\}$.

Alors $\Sigma' = \{\sigma' \mid \sigma \in \Sigma\}$ est un ECDM de E vers F' sur L .

Démonstration :

Nous montrons successivement que Σ' a les propriétés de cohérence, complétude, et minimalité.

(i) cohérence.

Soit $\sigma \in \Sigma$. Par le corollaire au lemme 3.14, on a

$$\sigma\xi E = \sigma\xi E' = \xi E'.$$

$$\text{d'où } E' = \xi^{-1}[\sigma\xi E]. \quad (!)$$

$$\begin{aligned} \text{D'autre part, } \sigma\xi E &= (\sigma \setminus V)\xi E \quad \text{car } V[\xi E] = V \\ &= (\sigma \setminus V \cup \xi)E \quad \text{car } D(\xi) \cap V = \emptyset \text{ et } I(\xi) = \emptyset. \end{aligned}$$

Pour tout x dans $V[E]$:

$$- \text{ si } x \in V \text{ alors } \xi\sigma'x = \sigma x \text{ par définition de } \sigma'$$

$$- \text{ si } x \in V[E] \cap V[E'] \text{ alors } \xi\sigma'x = \xi x \text{ car } D(\sigma') \subset V$$

donc dans tous les cas $(\sigma \setminus V \cup \xi)E = \xi\sigma'E$ d'où : $\sigma'E = \xi^{-1}[\sigma\xi E]$. En combinant avec (1) on obtient bien $E' = \sigma'E$, d'où $\sigma' \in D(E, F')$.

(ii) complétude.

Soit $\rho \in D(E, F')$; i.e. $\rho E = E'$ avec $D(\rho) \subset V$.

Nous supposons de plus que $\forall x \in D(\rho) \rho x$ est construit sur C , naturellement. On a :

$$\xi\rho E = \xi\rho\xi E \text{ car } D(\rho) \subset V \text{ et } I(\xi) = \emptyset.$$

Donc : $\xi\rho\xi E = \xi E'$, i.e. $\xi\rho \in P(\xi E, \xi F')$.

Σ étant un ECP de ξE et $\xi E'$ sur V , il existe σ dans Σ tel que

$$\exists_{\eta} \eta\sigma = \xi\rho, \text{ c'est à dire : } \forall x \in V \eta\sigma x = \xi\rho x. \quad (2)$$

$$\text{On définit } \sigma' = \{\langle x, \xi^{-1}[\sigma x] \rangle \mid x \in V\}, \text{ et } V' = I(\sigma') \cup (V - D(\sigma')).$$

Considérons la substitution $\bar{\eta}$ définie par :

$$\bar{\eta} = \{\langle x, \xi^{-1}[\xi\eta\xi x] \rangle \mid x \in V'\}.$$

Pour tout x dans V' , on a donc $\xi\bar{\eta}x = \xi\eta\xi x$ (car pour tout e , $V[\xi e] \cap V[E'] = \emptyset$).

Donc, pour tout terme e tel que $V[e] \subset V'$, on a aussi

$$\xi\bar{\eta}e = \xi\eta\xi e.$$

En particulier, en prenant $e = \sigma'x$, avec x quelconque dans L , on obtient : $\forall x \in L \quad \bar{\eta}\sigma'x = \xi\eta\xi\sigma'x$.

Par définition $\sigma'x$ est construit sur C , et de même $\bar{\eta}\sigma'x$ (par définition de ξ^{-1}). On peut donc écrire :

$\forall x \in L \quad \bar{\eta}\sigma'x = \xi^{-1}[\xi\eta\xi\sigma'x]$. Il y a maintenant deux cas :

- $x \in V \quad \sigma'x = \xi^{-1}[\sigma x] \quad$ par définition de σ'

$$\text{d'où } \bar{\eta}\sigma'x = \xi^{-1}[\xi\eta\xi\xi^{-1}[\sigma x]]$$

$$= \xi^{-1}[\xi\eta\sigma x] \quad \text{car } I(\sigma) \cap V[E'] = \emptyset$$

$$= \xi^{-1}[\xi\xi\rho x] \quad \text{en utilisant (2)}$$

$$= \xi^{-1}[\xi\rho x] \quad \text{car } \xi \text{ est idempotent}$$

$$= \rho x \quad \text{car } \rho x \text{ est construit sur } C \text{ par hypothèse.}$$

- $x \in V[E'] \quad \sigma x' = x$

$$\text{d'où } \bar{\eta}\sigma'x = \xi^{-1}[\xi\eta\xi x] = \xi^{-1}[\xi\eta X] = \xi^{-1}[X] = x$$

$$= \rho x \quad \text{car } D(\rho) \subset V.$$

Donc dans tous les cas, on trouve

$$\bar{\eta}\sigma'x = \rho x, \quad \text{et donc}$$

$$\rho \leq_{L} \sigma', \quad \text{avec } \sigma' \in \Sigma'.$$

(iii) minimalité.

Soit σ_1, σ_2 dans Σ , $\sigma_1 \neq \sigma_2$.

Supposons qu'il existe ρ tel que $\sigma_2' = \rho\sigma_1'$.

C'est à dire :

$$- \forall x \in V \quad \xi^{-1}[\sigma_2 x] = \rho\xi^{-1}[\sigma_1 x] \quad (3)$$

$$\text{et} \quad - \forall x \in V[E'] \quad x = \rho x \quad (4).$$

On montre facilement par récurrence sur e , que pour tout e tel que $V[e] \cap V[E'] = \emptyset$, on a :

$$\xi\rho\xi^{-1}[e] = \xi\rho e.$$

En effet, avec $e = \lambda u_1 \cdots u_n \cdot @ (e_1, \dots, e_p)$ si $@ \in C \cup V$ pas de problème. Si $@ = x_i$, alors

$$\begin{aligned} \xi\rho\xi^{-1}[e] &= \xi\rho(\lambda u_1 \cdots u_n \cdot x_i(\dots, \xi^{-1}[e_j], \dots)) \\ &= \xi(\lambda u_1 \cdots u_n \cdot x_i(\dots, \rho\xi^{-1}[e_j], \dots)) \quad \text{en utilisant (4)} \\ &= \lambda u_1 \cdots u_n \cdot x_i(\dots, \xi\rho\xi^{-1}[e_j], \dots) \\ &= \lambda u_1 \cdots u_n \cdot x_i(\dots, \xi\rho e_j, \dots) \quad \text{par hypothèse de récurrence} \\ &= \xi\rho e. \end{aligned}$$

En particulier, en prenant $e = \sigma_1 x$, pour x quelconque dans V , on obtient : $\forall x \in V \quad \xi\rho\xi^{-1}[\sigma_1 x] = \xi\rho\sigma_1 x$ et, en reportant dans (3), on a : $\forall x \in V \quad \xi\xi^{-1}[\sigma_2 x] = \xi\rho\sigma_1 x$,

$$\text{c'est à dire} \quad \sigma_2 x = \xi\rho\sigma_1 x.$$

Donc $\sigma_1 \leq_V \sigma_2$, ce qui est contraire à la condition d'indépendance de Σ .
(Remarquer que $V = V[\xi E] \cup V[\xi E']$). \square

Exemple :

$$\text{Soit } \begin{cases} E = f(A) \\ E' = g(A) \end{cases} \quad \text{avec} \quad \begin{cases} \tau(A) = \alpha \\ \tau(f) = \tau(g) = (\alpha \rightarrow \alpha) \end{cases}$$

On a ici $V = \{f\}$, $L = \{f, g\}$, $\xi = \{\langle g, G \rangle\}$.

On peut montrer que $\Sigma = \{\sigma_1, \sigma_2, \sigma_3\}$ est un ECPI de ξE et $\xi E'$ sur V ,

avec :

$$\begin{cases} \sigma_1 = \{\langle f, \lambda u \cdot G(u) \rangle\} \\ \sigma_2 = \{\langle f, \lambda u \cdot G(A) \rangle\} \\ \sigma_3 = \{\langle f, G \rangle\}. \end{cases}$$

D'où $\Sigma' = \{\sigma'_1, \sigma'_2, \sigma'_3\}$ est un ECDM de E vers F' sur L, avec :

$$\left| \begin{array}{l} \sigma'_1 = \{<f, \lambda u \cdot g(u)>\} \\ \sigma'_2 = \{<f, \lambda u \cdot g(A)>\} \\ \sigma'_3 = \{<f, g>\}. \square \end{array} \right.$$

Remarque :

Cet exemple nous montre que la condition d'indépendance de Σ , si elle implique la condition de minimalité de Σ' , n'en implique pas pour autant une condition d'indépendance.

En effet, on a par exemple $n\sigma'_1 = n\sigma'_2$, avec

$$n = \{<g, \lambda u \cdot g(A)>\}.$$

Il ne faut donc pas pouvoir espérer construire un ECDI, lorsque $V[E'] \neq \emptyset$, et ceci même en se restreignant au second ordre.

Théorème 11 : Soit E et E' deux termes de même type, et soit

$L = V[E] \cup V[E']$. Il existe un ECDM Σ de E vers E' sur L, qui est unique à un isomorphisme près.

Démonstration :

Soit A un API de ξE et $\xi E'$ sur L. D'après le théorème 11, $\Sigma(A)$ est un ECPI de E et E' sur L, et donc aussi sur $V = V[E] - V[E']$. De plus, par construction, pour tout σ dans $\Sigma(A)$, on a

$$I(\sigma) \cap L = \emptyset.$$

Par le lemme 3.16, $\Sigma(A)'$ est donc un ECDM de E vers E' sur L.

L'unicité d'un tel ECDM se démontre comme le lemme 2.16. \square

Remarques :

- 1) En pratique, il n'est pas besoin de faire intervenir ξ explicitement. Il suffit de "geler" les occurrences libres des variables dans $V[E']$, de manière à ce que SIMPL et CHOIX les traitent comme des constantes. Il est clair que l'ensemble $\Sigma(A)$ ainsi obtenu sera directement $\Sigma(A)'$.
- 2) Nous conjecturons que l'unifiabilité de E et E' est décidable, lorsque $V[E'] = \emptyset$. Ce résultat entraînerait naturellement la décidabilité de la demi-unification.

3.7.3. Cas d'un terme variable

Nous allons maintenant examiner le cas opposé au 3.7.1, où $E' = x$.

Remarquons tout d'abord que le problème du filtrage de x vers E est trivial, car $F(x, E) = \{\sigma \mid \sigma x = E\}$.

$$\text{De même } D(x, E) = \begin{cases} \emptyset & \text{si } x \in V[E] \\ \{\{\langle x, E \rangle\}\} & \text{sinon.} \end{cases}$$

Pour l'unification, remarquons que ce problème se ramène à la recherche d'un point fixe du terme $\lambda x. E$. Plus précisément :

Appelons *point fixe* du terme $\lambda x. E$, avec $\tau(E) = \tau(x)$, tout terme \bar{E} de type $\tau(x)$ tel que

$$N[(\lambda x. E) \bar{E}] = \bar{E}.$$

Si \bar{E} est un point fixe de $\lambda x. E$, alors $\{\langle x, \bar{E} \rangle\}$ est un unificateur de E et de x . Réciproquement, soit σ un unificateur de E et de x , et soit $\bar{\sigma} = \sigma \upharpoonright (V - \{x\})$. Alors σx est un point fixe de $\bar{\sigma}(\lambda x. E)$. En effet,

soit $\mathcal{D}(\sigma) - \{x\} = \{y_1, \dots, y_n\}$. Par définition, on a :

$$\begin{aligned}\sigma E &= N[\lambda y_1 \dots y_n x \cdot E](\sigma y_1, \dots, \sigma y_n, \sigma x) = \\ &= N[\bar{\sigma}(\lambda x E)\sigma x] = \sigma x \text{ par hypothèse.}\end{aligned}$$

Exemple :

Soit $E = \lambda u \cdot u(x(\lambda v w \cdot v), f(x))$

avec $\left\{ \begin{array}{l} \tau(v) = \tau(w) = \alpha \\ \tau(u) = (\alpha \times \alpha \rightarrow \alpha) \\ \tau(x) = ((\alpha \times \alpha \rightarrow \alpha) \rightarrow \alpha) \\ \tau(f) = (((\alpha \times \alpha \rightarrow \alpha) \rightarrow \alpha) \rightarrow \alpha). \end{array} \right.$

Alors $\sigma = \{<x, \lambda u \cdot u(y, z)>, <f, \lambda x \cdot z>\}$, avec $\tau(y) = \tau(z) = \alpha$

est un unificateur de E et de x .

Donc $\lambda u \cdot u(y, z)$ est un point fixe de $\lambda x u \cdot u(x(\lambda v w \cdot v), z)$.

Nous ne connaissons pas de méthode générale pour trouver les points fixes d'un terme quelconque. Dans le cas particulier où $x \notin V[E]$, alors la solution triviale $<x, E>$ est solution principale :

Lemme 3.17

Si $x \notin V[E]$, alors $\sigma = \{<x, E>\}$ est un unificateur principal de E et de x .

Démonstration :

Soit $\rho \in U(E, x)$. Prouvons que $\rho = \rho\sigma$.

Soit $\bar{\rho} = \rho \upharpoonright (V - \{x\})$. On a $\rho\sigma = \{<x, \rho E>\} \cup \bar{\rho} = \rho$, puisque $\rho E = \rho x$ par hypothèse. \square

Ce lemme est une généralisation d'une propriété bien connue dans les langages de premier ordre.

Ce résultat pourrait être incorporé dans l'algorithme SIMPL, qui pourrait simplifier directement un unificande comprenant une paire $\langle x, E \rangle$, avec $x \notin V[E]$. Cela éviterait la génération d'une branche d'arbre de pré-unification, de longueur $\pi(E)$, obtenue par une chaîne d'imitations.

Remarquons toutefois qu'ici, contrairement aux langages de premier ordre, la réciproque du lemme 3.17 n'est pas vraie. Par exemple, $E = f(x)$ est unifiable avec x , par $\{f, \lambda u \cdot x\}$ ou $\{f, \lambda u \cdot u\}$.

Définition :

Appelons *chemin rigide vers x dans E* une séquence de termes :

$$E = e_1, e_2, \dots, e_n, e_{n+1} \quad n \geq 1 \quad \text{telle que :}$$

- i) $\forall i \leq n \quad e_{i+1}$ est un argument de e_i .
- ii) $\forall i \leq n \quad$ la tête de e_i est une constante, ou une variable liée dans le lieur d'un e_j , avec $j \leq i$.
- iii) la tête de e_{n+1} est une occurrence de x libre dans E .

S'il n'existe pas de chemin rigide vers x dans E , alors E et x sont facilement unifiables, par exemple par $\zeta \upharpoonright (V - \{x\}) \cup \{\langle x, \zeta E \rangle\}$.

La réciproque n'est pas vraie, comme le montre l'exemple suivant :

$$E = \lambda u \cdot u(x(\lambda v \cdot v)) \quad \text{avec} \quad \begin{cases} \tau(v) = \gamma \\ \tau(u) = (\gamma \rightarrow \gamma) \\ \tau(x) = ((\gamma \rightarrow \gamma) \rightarrow \gamma), \end{cases}$$

puisque $\sigma = \{\langle x, \lambda u \cdot u(y) \rangle\}$ unifie E et x , avec $\tau(y) = \gamma$.

Voir de même l'exemple 3.

Il nous est possible de conclure dans deux cas particuliers, dont nous laisserons la preuve au lecteur :

- i) Si $\bar{H}[e] = 0$ et s'il existe un chemin rigide vers x dans E , alors E et x ne sont pas unifiables.
- ii) S'il existe un chemin rigide vers x dans E finissant sur un terme sans arguments, i.e. $e_{n+1} = \lambda u_1 \dots u_p^* x$, alors E et x ne sont pas unifiables.

En particulier, on obtient le résultat bien connu qu'au premier ordre $U(E, x) = \emptyset$ si $x \in V(E)$.

Ces cas particuliers risquant d'être assez fréquents en pratique, il est important de les incorporer dans l'algorithme SIMPL, qui saura simplifier un unificande contenant une telle paire en un noeud échec.

CHAPITRE 4 : Extension au $\lambda\text{-}\eta$ -calcul

Nous allons considérer dans ce chapitre le problème de l'unification dans le $\lambda\text{-}\eta$ -calcul. Nous montrerons que tous nos résultats s'étendent sans difficulté, et même que les algorithmes se simplifient considérablement.

4.1. Forme extensionnelle

4.1.1. η -réduction

Soit $e = E<\lambda x(e_1 x)>$, avec $x \notin V[e_1]$; $e' = E<e_1>$ est dit dériver de e par η -réduction, et on note :

$$e \xrightarrow{\eta} e'.$$

La règle de η -réduction est une règle du λ -calcul, qui exprime que les termes $\lambda x(e_1 x)$ et e_1 dénotent la même fonction : appliqués à tout terme e_2 , ils donneront par β -conversion le terme $(e_1 e_2)$.

La η -réduction exprime donc une forme faible du principe d'extensionnalité. Plus précisément, pour tous $e, e' \in T$ et pour tout $x \notin V[e] \cup V[e']$,

(ex) $\equiv (e'x) \Rightarrow e \equiv e'$, pour toute équivalence \equiv plus grossière que \rightarrow_η .

La η -réduction est compatible avec la β -réduction : les théorèmes 1 et 2 restent vrais quand on remplace $\overset{*}{\beta}$ par $\overset{*}{\beta} \eta$.

De plus, la η -réduction ne sert pas à faire de nouvelles β -réductions.

En particulier, on a :

Lemme 4.1 du retard de la règle η

$$\forall e, e' \quad e \xrightarrow[\beta, \eta]^* e' \Rightarrow \exists e'' \quad e \overset{\beta}{\not\rightarrow} e'' \xrightarrow[\eta]^* e'.$$

Nous ne prouverons pas ici ce lemme, non plus que l'analogie des théorèmes 1 et 2, qui sont démontrés dans Curry et Feys [CF1].

Dénotons par \equiv_η et $\equiv_{\beta\eta}$ les équivalences engendrées respectivement par \rightarrow_η et $\overset{*}{\beta} \cup \rightarrow_\eta$. Nous pouvons maintenant énoncer :

Lemme 4.2

$$\forall e, e' \quad e \equiv_{\beta\eta} e' \Leftrightarrow N[e] \equiv_\eta N[e']$$

Démonstration

\Leftarrow est évident.

\Rightarrow Soit $e \equiv_{\beta\eta} e'$. Comme $e \xrightarrow[\beta]^* N[e]$ et $e' \xrightarrow[\beta]^* N[e']$, on a donc $N[e] \equiv_{\beta\eta} N[e']$.

Par l'analogie du corollaire du théorème 2 pour la β - η -réduction, il existe e'' tel que :

$$N[e] \xrightarrow[\beta\eta]^* e'' \text{ et } N[e'] \xrightarrow[\beta\eta]^* e''.$$

Par le lemme du retard de la règle η ci-dessus, et $N[e]$ et $N[e']$ étant irréductibles par la règle β , on a bien

$$N[e] \xrightarrow[\eta]^* e'' \text{ et } N[e'] \xrightarrow[\eta]^* e''. \quad \square$$

Ce lemme justifie l'emploi de la η -convertibilité, non par une règle de calcul supplémentaire, mais par une mise sous forme normale, que nous appelerons forme extensionnelle.

4.1.2 Définition

Soit $e = \lambda x_1 x_2 \cdots x_n @ (e_1, e_2, \dots, e_p)$ un terme en forme normale, avec

$$\tau(e) = (\alpha_1 \times \alpha_2 \times \cdots \times \alpha_n \times \alpha_{n+1} \times \cdots \times \alpha_{n+k} \rightarrow \beta)$$

où $k \geq 0$, $\beta \in T_0$.

Soit y_1, \dots, y_k variables distinctes n'apparaissant pas dans e , avec $\tau(y_i) = \alpha_{n+i}$ $1 \leq i \leq k$.

On définit récursivement la forme extensionnelle $\eta[e]$ de e par :

$$\eta[e] = \lambda x_1 x_2 \cdots x_n y_1 \cdots y_k @ (\eta[e_1], \eta[e_2], \dots, \eta[e_p], \eta[y_1], \dots, \eta[y_k]).$$

Remarquons que le terme $\eta[e]$ est en forme β -normale, que $\tau(\eta[e]) = \tau(e)$, et que $\eta[e] \xrightarrow{\eta} e$. Enfin, montrons que la forme extensionnelle de formes β -normales η -convertibles est unique.

Lemme 4.3

Pour tous e et e' en forme β -normale,

$$e \xrightarrow{\eta} e' \implies \eta[e] = \eta[e'].$$

Démonstration : par récurrence sur la structure de e

Soit $e = \lambda x_1 \cdots x_n @ (e_1, \dots, e_p)$. Il y a deux cas :

i) $e_p = x_n$ et $e' = \lambda x_1 \cdots x_n @ (e_1, \dots, e_{p-1})$. Alors $\eta[e] = \eta[e']$ trivialement.

ii) sinon on doit avoir $e' = \lambda x_1 \cdots x_n @ (e_1, \dots, e_{k-1}, e'_k, e_{k+1}, \dots, e_p)$, et $e_k \xrightarrow{\eta} e'_k$. Alors $\eta[e_k] = \eta[e'_k]$ par hypothèse de récurrence, d'où le résultat. \square

Corollaire :

$$\forall e, e' \quad e \underset{\beta\eta}{=} e' \Leftrightarrow n[N[e]] = n[N[e']]$$

La preuve découle directement des lemmes 4.2 et 4.3.

La forme extensionnelle de e ne s'obtient pas par η -réduction, mais au contraire par la η -expansion maximale compatible avec le type de e . Nous allons voir que cette forme extensionnelle possède une importante propriété de fermeture respectivement aux règles de formation des termes et à la substitution que ne possède pas la forme η -réduite. Pour cela, nous allons revenir aux termes du λ -calcul typé, sous forme non réduite.

Définitions :

Pour tout terme e , de type $\tau(e) = (\alpha_1 \times \alpha_2 \times \dots \times \alpha_n \rightarrow \beta)$, on définit $\bar{\tau}(e) = n$.

On définit T_n comme l'ensemble des termes e tels que, pour tout contexte $E < >$, pour tout atome \emptyset , $e = E < \emptyset >$ entraîne $E < > = E' < (\dots ((< > e_1) e_2) \dots e_n) >$, avec $n = \bar{\tau}(\emptyset)$. Autrement dit, tous les atomes apparaissent appliqués autant de fois que leur type le leur permet.

Lemme 4.4 Pour tous e, e', x de types convenables :

- a) $e, e' \in T_n \Rightarrow (ee') \in T_n$
- b) $e \in T_n \Rightarrow \lambda x e \in T_n$
- c) $E < e > \in T_n, e' \in T_n \Rightarrow E < e' > \in T_n$
- d) $E < ee' > \in T_n \Rightarrow e' \in T_n$
- e) $E < \lambda x e > \in T_n \Rightarrow e \in T_n$
- f) $e, e' \in T_n \Rightarrow S_e^x[e'] \in T_n$

Démonstration :

Ces relations de fermeture sont aisées à montrer ; nous en laissons la preuve au lecteur. \square

Lemme 4.5

T_η est fermé par β -conversion.

Démonstration :

Soit $e = E<(\lambda x M N)> \beta e' = E<S_N^x[M]>$, avec $e \in T_\eta$. En utilisant le lemme 4.4 ci-dessus :

$$\begin{aligned} M &\in T_\eta && \text{par e)} \\ N &\in T_\eta && \text{par d)} \\ S_N^x[M] &\in T_\eta \text{ par f)} \\ \text{d'où } e' &\in T_\eta && \text{par c). } \square \end{aligned}$$

Définition :

On définit l'ensemble T_x des termes de T_η en forme β -normale.

Lemme 4.6

T_x est l'ensemble des formes extensionnelles.

Démonstration :

Par définition, toute forme extensionnelle est dans T_x . Pour la réciprocité, il est aisément de montrer par récurrence sur e , que $\forall e \in T_x \quad n[e] = e$. \square

Lemme 4.7 $\forall e, e' \in T_x :$

- a) $\lambda x e \in T_x$
- b) $N\Gamma(ee') \in T_x$.

Démonstration :

- a) découle du lemme 4.4 b).

b) découle du lemme 4.4 a) et du lemme 4.5. \square

Ces relations de fermeture sont importantes. C'est leur validité qui a motivé notre usage des formes extensionnelles plutôt que des formes n -réduites, qui sont traditionnellement prises comme formes normales dans le λ - n -calcul. En effet, le lemme 4.7 ne s'applique pas à ces formes n réduites, comme on peut le constater en prenant :

- pour a) : $e = A(x)$
- pour b) : $e = \lambda u x . A(u(x)) \quad e' = \lambda x . x.$

Le lemme 4.7 nous permet d'utiliser tous nos résultats précédents : en considérant au départ des termes sous forme extensionnelle, tous les termes générés seront sous forme extensionnelle également, sans avoir à employer la règle n .

Un inconvénient des formes extensionnelles est que ces formes peuvent être de taille sensiblement supérieure aux formes n -réduites. En fait, la forme extensionnelle de e n'est autre que l'itérée de e k comme défini au chapitre 3. Il est facile de constater que la taille de $n[e]$ n'est autre que la complexité de e , comme définie en 3.6 :

Lemme 4.8

$$\forall e \in T \quad \pi(n[e]) = \chi(e).$$

Démonstration :

Immédiat par récurrence sur e , en utilisant la définition de $\chi(e)$ en 3.6 et le lemme 3.10. \square

Exemple :

En reprenant l'exemple 6 de 3.6 :

$$e = f(x) \text{ avec } \begin{cases} \tau(x) = (\gamma \times (\gamma \rightarrow \gamma) \rightarrow \gamma) \\ \tau(f) = ((\gamma \times (\gamma \rightarrow \gamma) \rightarrow \gamma) \times (\gamma \rightarrow \gamma) \rightarrow \gamma), \end{cases}$$

on a

$$\eta[e] = \lambda u \cdot f(\lambda v w \cdot x(v, \lambda t \cdot w(t)), \lambda y \cdot u(y))$$

$$\text{avec } \begin{cases} \tau(v) = \tau(t) = \tau(y) = \gamma \\ \tau(u) = \tau(w) = (\gamma \rightarrow \gamma). \end{cases}$$

$$\pi(e) = 1, \quad \pi(\eta[e]) = \chi(e) = 6.$$

Sur cet exemple, le terme e a perdu en lisibilité, en passant en forme extensionnelle. Mais sa simplicité d'origine n'était qu'apparente, et nous n'avons fait qu'amener à la surface la complexité de e , implicite dans les types complices de f et de x .

Pour certaines applications, on peut ne pas désirer l'extensionnalité des termes, et donc la η -réduction. Par exemple, la théorie des types de Church [CA1] est un système logique d'ordre ω ayant comme langage le λ -calcul typé et autorisant comme règles d'inférence les α et β conversions, mais pas la η -conversion.

C'est pour cette raison que nous avons formulé tous nos résultats dans T muni de la seule β -conversion, aux chapitres 2 et 3.

Nous montrerons dans les paragraphes suivants comment ces résultats s'appliquent également à T_x . D'une manière générale, tous les résultats s'étendent facilement, et sont souvent plus simples sur les formes extensionnelles.

4.2. Substitutions extensionnelles

Nous allons maintenant considérer les substitutions dans le $\lambda\text{-}\eta$ -calcul.

Comme nous l'avons vu en 4.1, il suffit de s'intéresser aux formes extensionnelles. On définit donc une composante de substitution extensionnelle comme une composante de substitution $\langle x, e \rangle$ telle que $e \in T_x$. Une *substitution extensionnelle* est un ensemble fini de composantes de substitutions extensionnelles se rapportant à des variables distinctes. On dénote par S_x l'ensemble des substitutions extensionnelles.

L'égalité dans S_x est définie par :

$$\sigma = \sigma' \iff (\sigma - \sigma') \cup (\sigma' - \sigma) \subset \{\langle x, n[x] \rangle \mid x \in V\}$$

Maintenant, en effet, $D(\sigma) = \{x \mid \sigma x \neq n[x]\}$.

Les substitutions extensionnelles sont des morphismes de T_x , en conservant la même définition d'application que ci-dessus. En effet :

Lemme 4.9

$$\forall e \in T_x \quad \forall \sigma \in S_x \quad \sigma e \in T_x$$

Démonstration :

Directement à partir du lemme 4.7, et de la définition de σe . \square

4.3. Unification extensionnelle

Le lecteur vérifiera que les lemmes 2.4 et 2.5 restent valables. L'arité d'un terme extensionnel ne dépendant que de son type, le lemme 2.6 peut être rendu plus fin :

$$\forall e \in T_f \quad \bar{H}[\sigma e] = \bar{H}[e].$$

Les lemmes 2.7 et 2.8 restent inchangés.

Les notions de filtres et d'unificateurs sont les mêmes ici, en se restreignant aux substitutions extensionnelles.

Il est facile de vérifier que tous les lemmes et théorèmes restent vrais. Leurs preuves sont identiques, ou quelquefois plus simples, car il y a moins de cas à examiner. Par exemple, pour la preuve du théorème 4, on se limite au cas $\bar{H}[e] = 1$, et on supprime dans Σ la solution non extensionnelle $\langle f, F \rangle$. De même, dans la preuve du théorème 5 la solution σ_0 disparaît, et les termes E et E' deviennent :

$$\begin{cases} E = f(\lambda v \cdot x(v), A) \\ E' = f(\lambda v \cdot x(v), B). \end{cases}$$

Dans la preuve du théorème 6, remplacer dans E
 $f(u, u, \dots, u)$ par $f(\lambda z \cdot u(z), \dots, \lambda z \cdot u(z))$
et dans E' $u(g(u))$ par $u(g(\lambda z \cdot u(z)))$.

Tous nos résultats s'étendent donc sans difficulté au $\lambda\eta$ -calcul.

4.4. Arbres de pré- η -unification

Nous allons maintenant nous intéresser à la construction d'arbres de pré-unification pour le $\beta\eta$ -calcul.

Nous supposons dorénavant que les termes sont en forme extensionnelle, comme définie en 4.1.

Rappelons que si e est en forme extensionnelle, alors en particulier on a :

$$\bar{H}[e] = \bar{\tau}(e).$$

4.4.1. Construction d'arbres de pré- η -unification - (AP η).

La principale modification consiste en une simplification considérable de l'algorithme CHOIX.

Tout d'abord (en utilisant les notations du 3.3.1) on a toujours $n_1 = n_2$, d'où $n = 0$. Cela permet de supprimer les cas 3.3.1.1.1 et 3.3.1.2.2. Enfin, dans les cas 3.3.1.1.2 et 3.3.1.2.1, on se restreint à considérer la solution $k = p_1$. Dans chaque cas, on remplace E_j par $n[E_j]$, de sorte que CHOIX retourne des substitutions extensionnelles.

Ceci diminue sensiblement le nombre de solutions retournées par CHOIX (au plus 1 imitation et p_1 projections).

Il s'ensuit une diminution du degré de branchement des arbres d'unification correspondants, d'où un gain exponentiel sur l'espace de recherche.

Le reste de la construction est identique à celle donnée en 3.4.1.

Par exemple, pour l'exemple 5 donné en 3.5, on ne construirait pas les branches menant à N_1 et N_3 .

4.4.2. Correction de la construction.

Le lemme 3.4 reste valide. Pour le vérifier il suffit de constater, avec les notations de la preuve, que pour tout ρ la forme extensionnelle de ρf a un en-tête de longueur p_1 . La preuve du lemme 3.6 de complétude est inchangée. Le lemme 3.5 de cohérence est bien sûr toujours valide (on ne fait que supprimer certaines solutions). Remarquons tout de même que nous pouvons maintenant avoir des solutions supplémentaires du fait du passage en forme extensionnelle. Par exemple, en prenant

$$e_1 = \lambda u \cdot f(u)$$

$$\text{et } e_2 = A$$

avec $\tau(u) = \gamma$, $\tau(f) = \tau(A) = (\gamma \rightarrow \gamma)$, l'algorithme CHOIX donné en 3.3 ne rentrait aucune solution, alors que maintenant, e_2 étant mis sous sa forme extensionnelle $\lambda u \cdot A(u)$, on a la solution $\langle f, \lambda u \cdot A(h(u)) \rangle$, complétée ultérieurement par $\langle h, \lambda u \cdot u \rangle$. L'algorithme CHOIX simplifié n'est donc ni cohérent ni complet lorsque la règle de n -conversion n'est pas autorisée.

Lorsque la règle de η -conversion est valide, on a donc l'équivalent du théorème 8. Mais nous avons de plus la propriété d'indépendance, sans modification supplémentaire. En effet, grâce à la forme extensionnelle, toute variable a maintenant un degré maximum dans tout terme. La condition du 3.5.2.1 est donc toujours remplie, et le théorème 9 est donc valide. On peut donc l'énoncer :

Théorème 12 :

Soit A un AP η quelconque de E et F' sur V . $\Sigma(A)$ est un ECPI de E et E' sur V , dans le λ - η -calcul.

Enfin remarquons que nous avons supprimé ici toute redondance, y compris celle qui nous avait fait considérer en 3.6 les arbres d'unifiabilité, car

$$\delta'_N = \delta_N = \bar{\tau}(v_N).$$

Du point de vue pratique de l'implémentation d'un algorithme d'unification en λ -calcul typé, il y a donc tout intérêt à se restreindre à générer des AP η , lorsque la règle de η -conversion est admise. Ceci est une hypothèse raisonnable, la η -conversion étant une forme faible du principe de l'extensionnalité de l'égalité.

4.5. L'algorithme de Jensen-Pietrzykowski

Jensen et Pietrzykowski ont défini dans [JP1] un algorithme qui énumère un ensemble complet d'unificateurs pour des termes du λ - η -calcul. Cet algorithme peut être décrit d'une manière similaire à notre construction d'APU. L'équivalent de l'algorithme CHOIX engendre des composantes de substitution suivant

cinq règles :

- 1 - élimination
- 2 - itération
- 3 - projection
- 4 - imitation
- 5 - identification.

De ces règles, seules 1, 3 et 4 sont dirigées. Les règles 2 et 5 engendrent des termes arbitraires. La règle d'itération est particulièrement prolifique : elle utilise des variables en nombre et en type arbitraires. A cause de cette règle, les arbres d'unification de Jensen et Pietrzykowski ne sont pas finiment engendrés. Cette approche ne semble donc pas pratiquement implémentable.

La raison principale de cette difficulté réside naturellement dans la difficulté de l'unification de deux termes flexibles, et de la redondance obligée qui est impliquée par le théorème 5.

Le but initial des deux méthodes était commun : une mécanisation de la théorie des types par une extension de la méthode de résolution. Comme nous l'avons montré [HG3], on peut baser une méthode de preuve complète sur la génération d'ECP, ce qui semble pratiquement plus raisonnable. De plus, si l'on veut générer un ECU, on peut toujours passer par l'intermédiaire d'un AP, en utilisant l'algorithme de Jensen-Pietrzykowski pour continuer le traitement des unificandes aux noeuds succès. Cette méthode de retard éviterait de générer beaucoup de branches inutiles.

CHAPITRE 5 : Unification dans les langages de premier ordre

Nous allons reprendre dans ce chapitre la plupart des définitions, car les objets sont beaucoup plus simples.

Il n'y a plus de λ -abstractions, donc plus de variables liées, ni de conversions.

5.1. Termes de premier ordre

On se donne un ensemble dénombrable de *variables* :

$$V = \{v_1, v_2, \dots\},$$

et un ensemble fini ou dénombrable de symboles de fonctions *constantes* :

$$C = \{F_1, F_2, \dots\},$$

On utilisera *x,y,z,u,v,w* pour dénoter des variables, *F,G,H,A,B,C*, pour dénoter des constantes.

Chaque constante *F* est affectée d'une arité $\alpha(F) \geq 0$.

(On pourrait plus généralement affecter chaque atome d'un type ; nous ne le ferons pas ici pour simplifier la notation).

Nous supposons que notre langage n'est pas entièrement monadique, c'est à dire qu'il existe au moins une constante d'arité supérieure à 1.

L'ensemble des termes T est le plus petit ensemble contenant V et fermé par les opérations :

$$e_1, \dots, e_{\alpha(F)} \longrightarrow F e_1 \cdots e_{\alpha(F)} \quad F \in C.$$

$$\text{i.e. } V \subset T \quad \text{et} \quad e_1, \dots, e_{\alpha(F)} \in T \Rightarrow F e_1 \cdots e_{\alpha(F)} \in T.$$

Autrement dit T est le magma libre $M(C, V)$ de base V engendré par les opérateurs de C . T peut également être considéré comme le langage engendré à partir de l'axiome ξ par la grammaire :

$$\left\{ \begin{array}{ll} \xi \rightarrow v_i & \forall v_i \in V \\ \xi \rightarrow F \underbrace{\xi \xi \xi \xi}_{\alpha(F)} & \forall F \in C \end{array} \right.$$

(Ce langage est algébrique si C est fini).

Les éléments de T seront dénotés e, e_1, \dots, e', \dots .

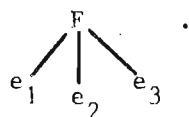
On réserve le signe $=$ pour l'égalité dans T .

Nous ferons usage constant de la propriété fondamentale de décomposition unique sur le magma libre :

$$F e_1 \cdots e_{\alpha(F)} = G e'_1 \cdots e'_{\alpha(G)} \Leftrightarrow F = G \text{ et } \forall i \leq \alpha(F) \quad e_i = e'_i.$$

Nous ferons librement usage de la notation parenthésée plus usuelle que cette "forme polonaise", et à l'occasion nous utiliserons des représentations par arbres orientés. Par exemple, $F e_1 e_2 e_3$ pourra être représenté par

$F(e_1, e_2, e_3)$ ou par



Nous pourrions considérer T comme une algèbre hétérogène, en rajoutant des types. Tous les résultats qui suivent restent valables. Nous ne traitons ici que le cas homogène, pour que les notations soient plus lisibles.

L'ensemble des variables d'un terme e , noté $V(e)$, est défini récursivement par :

$$\begin{cases} V(x) = \{x\} & \forall x \in V \\ V(F e_1 \cdots e_{\alpha(F)}) = \bigsqcup_{i=1}^{\alpha(F)} V(e_i) & \forall F \in C. \end{cases}$$

L'ouverture d'un terme est son nombre de variables :

$$v(e) = |V(e)|.$$

Si $v(e) = 0$, on dit que le terme e est fermé.

La taille $\theta(e)$ d'un terme e est définie récursivement par :

$$\begin{cases} \theta(x) = 0 & \forall x \in V \\ \theta(F e_1 \cdots e_{\alpha(F)}) = 1 + \sum_{i=1}^{\alpha(F)} \theta(e_i) & \forall F \in C \end{cases}$$

(Remarquer que $\theta(e)$ n'est pas l'analogue, pour les termes de premier ordre, de $\pi(e)$ comme défini au chapitre 1).

Finalement on définit, pour $x \in V$ et $e \in T$, le nombre d'occurrences de x dans e , noté $\#(x, e)$, par :

$$\begin{cases} \#(x, x) = 1 \\ \#(x, y) = 0 \quad \text{si } y \neq x \\ \#(x, F e_1 \cdots e_{\alpha(F)}) = \sum_{i=1}^{\alpha(F)} \#(x, e_i) \end{cases}$$

On a bien sûr $x \in V(e) \Leftrightarrow \#(x, e) > 0$.

5.2. Substitutions, filtrage

Une substitution est une application de V dans T , identité presque partout (c'est à dire sauf sur un sous-ensemble fini de V).

Nous utiliserons les lettres grecques σ, ρ, η pour dénoter les substitutions.

Comme auparavant : $D(\sigma) = \{x \mid \sigma x \neq x\}$, $I(\sigma) = \bigsqcup_{x \in D(\sigma)} V(\sigma x)$.

Les substitutions sont étendues en des endomorphismes de T , par les règles :

$$\sigma F(\dots, e_i, \dots) = F(\dots, \sigma e_i, \dots) \quad \forall F \in C.$$

(On utilise σe comme abréviation de $\sigma(e)$).

La composition de σ avec ρ est notée $\rho\sigma$. (i.e. $\rho\sigma e = \rho(\sigma(e))$).

Les automorphismes de T sont notés ξ, ξ', \dots ; on les appelle des permutations.

La limitation à des substitutions de domaine fini n'est pas une restriction, car à tout moment on ne s'intéresse vraiment qu'à un nombre fini de variables, comme le suggère le lemme suivant, qui est l'équivalent du lemme 2.4.

Lemme 5.1

Pour tout ensemble V de variables, pour tout terme e et pour toute substitution σ :

$$V(e) \subset V \implies \sigma e = (\sigma|_V)e$$

Démonstration :

Immédiate par récurrence sur la structure de e . \square

Par contre ici nous avons une version plus forte du lemme 2.5, parce que sa réciproque est vraie aussi :

Lemme 5.2

$$\forall e, \sigma \quad D(\sigma) \cap V(e) = \emptyset \iff \sigma e = e.$$

Démonstration :

\Rightarrow : par le lemme précédent, en prenant $V = V(e)$.

\Leftarrow : par une récurrence structurelle sur e . \square

Définition :

On dit que e schématisse e' , et on note $e \leq e'$, s'il existe une substitution σ telle que $e' = \sigma e$.

$e \leq e'$ signifie que e "contient moins d'information" que e' , ce qui explique la direction de notre inégalité. L'autre direction pourrait également être motivée par la remarque que e est "plus général" que e' , ou de manière équivalente que e' est un cas particulier de e .

Le problème du *filtrage* consiste, étant donnés e et e' , à trouver la substitution σ telle que $e' = \sigma e$. Si σ existe, $\sigma \setminus V(e)$ est unique, comme le montre le lemme suivant. On dit que $\sigma \setminus V(e)$ est le *filtre de e vers e'*.

Lemme 5.3

$$\forall e, e', \sigma, \sigma' \quad e' = \sigma e \text{ et } e' = \sigma' e \implies \sigma \setminus V(e) = \sigma' \setminus V(e).$$

Démonstration :

Par récurrence structurelle sur e ;

i) $e = x$ alors $\sigma x = e' = \sigma' x$.

ii) $e = Fe_1 \cdots e_n$ alors $e' = F\sigma e_1 \cdots \sigma e_n = F\sigma' e_1 \cdots \sigma' e_n$.

Par hypothèse de récurrence, avec $V_i = V(e_i)$, on a $\sigma \setminus V_i = \sigma' \setminus V_i$.
Comme $V(e) = \bigsqcup_{i=1}^n V_i$, on a bien le résultat. \square

Contrairement aux langages d'ordre supérieur, un algorithme très simple de descente récursive permet de résoudre le problème du filtrage sans retours en arrière.

Algorithme de filtrage

Soit e, e' dans T , $V = V(e)$. Initialement $\sigma x = \perp$ pour tout x dans V , où \perp est un symbole spécial non dans T ("indéfini").

Filtre (e,e') :

```

- si      e = Fe1...en alors
          - si e' = Fe'1...e'n alors
              [Filtre (e1,e'1) ;
               ...
               Filtre (en,e'n)]
          - sinon exit échec

- sinon   (e ∈ V) alors
          - si σe = ⊥ alors σe ← e'
          - sinon si σe ≠ e' alors exit échec ; fin de Filtre. □

```

Il est clair que si l'algorithme Filtre s'arrête en échec alors e ne schématise pas e'. Sinon, la substitution σ finale est telle que e' = σe. Le temps d'exécution de l'algorithme est proportionnel à θ(e').

La relation \leq est un préordre. On dit que e et e' sont des *variantes*, et on note $e \equiv e'$, si $e \leq e'$ et $e' \leq e$.

Les variantes sont identiques, au renommage de leurs variables près, comme le montre le lemme suivant :

Lemme 5.4

$$e \equiv e' \Leftrightarrow \exists \xi \quad e = \xi e'$$

(où ξ est une permutation).

Démonstration :

- \Leftarrow : évident, car toute permutation admet un inverse, par définition.
- \Rightarrow : Supposons qu'il existe σ et σ' telles que :

$$e' = \sigma e \quad \text{et} \quad e = \sigma' e'.$$

Alors $e' = \sigma\sigma'e'$ et $e = \sigma'\sigma e$.

Par le lemme 5.2 : $\forall x \in V(e') \sigma\sigma'x = x$

$$\forall x \in V(e) \sigma'\sigma x = x.$$

σ (resp. σ') est donc une injection de $V(e)$ dans $V(e')$ (resp. de $V(e')$ dans $V(e)$), et par conséquent

$$v(e) = v(e').$$

Soit ϕ une bijection quelconque de $V(e') - V(e)$ dans $V(e) - V(e')$.

Les substitutions ξ et ξ' définies par :

$$\xi x = \begin{cases} \sigma x & \text{si } x \in V(e) \\ \phi x & \text{si } x \in V(e') - V(e) \\ x & \text{sinon} \end{cases}$$

$$\xi'x = \begin{cases} \sigma'x & \text{si } x \in V(e') \\ \phi^{-1} & \text{si } x \in V(e) - V(e') \\ x & \text{sinon} \end{cases}$$

sont des permutations inverses, et par le lemme 5.2 on a bien :

$$e' = \xi e \quad \text{et} \quad e = \xi'e'. \square$$

Ce lemme nous fournit une caractérisation pour les variantes, qui est équivalente ici, pour les variables libres, à l'égalité par α -conversion des variables liées dans le λ -calcul typé. Mais remarquons que ce lemme n'est plus vrai dès le second ordre, puisqu'on a par exemple

$$x \equiv f(A) \equiv g(x).$$

On peut étendre la relation \leq aux substitutions par extensionnalité, en définissant : $\sigma \leq \sigma' \iff \forall e \in T \quad \sigma e \leq \sigma' e$.

(on dit que σ est une substitution plus générale que σ').

De même : $\sigma \equiv \sigma' \iff \forall e \in T \quad \sigma e \equiv \sigma' e$.

Le lemme suivant nous montre que cette définition est équivalente à celle que nous avons utilisée jusqu'à présent :

Lemme 5.5

$$\sigma \leq \sigma' \Leftrightarrow \exists \rho \quad \sigma' = \rho \sigma.$$

Démonstration :

Soit σ et σ' deux substitutions quelconques, soit

$$V = D(\sigma) \cup D(\sigma') \cup I(\sigma), \text{ et}$$

soit E un terme quelconque tel que $V \subset V(E)$. Posons :

$$(1) \quad \sigma \leq \sigma'$$

$$(2) \quad \sigma E \leq \sigma' E$$

$$(3) \quad \exists \rho \forall x \in V \quad \rho \sigma x = \sigma' x$$

$$(4) \quad \exists \rho' \quad \sigma' = \rho' \sigma$$

On prouve successivement :

$$(4) \Rightarrow (1) : \text{trivial}$$

$$(1) \Rightarrow (2) : \text{par définition de } \sigma \leq \sigma'$$

$$(2) \Rightarrow (3) : \text{en prenant } \rho \text{ tel que } \rho \sigma E = \sigma' E, \text{ en utilisant le lemme 5.3.}$$

$$(3) \Rightarrow (4) : \text{soit } \rho \text{ tel que } \forall x \in V \quad \rho \sigma x = \sigma' x.$$

Prenons $\rho' = \rho|_V$.

On a : $\forall x \in V \quad \rho' \sigma x = \sigma' x \quad \text{car } I(\sigma) \subset V$.

$\forall x \notin V \quad \sigma' x = x \quad \text{car } D(\sigma') \subset V$

$\sigma x = x \quad \text{car } D(\sigma) \subset V$

et $\rho' x = x \quad \text{par définition de } \rho'$

donc $\rho' \sigma x = \sigma' x = x$

ce qui donne bien $\sigma' = \rho' \sigma$.

L'équivalence de (1) et de (4) termine la preuve. Le seul point délicat concerne l'existence de F . Elle découle directement de notre hypothèse concernant l'existence d'un symbole de fonction non monadique. En fait, le lemme

est faux dans les langages monadiques, comme il est facile de constater. \square

Remarque : Si nous supposons les substitutions non restreintes à un domaine fini, le lemme 5.5 reste vrai. La démonstration est plus compliquée, car au lieu du terme E on doit considérer une séquence croissante de termes E_i , et construire ρ par un passage à la limite.

Lemme 5.6 $\sigma \equiv \sigma' \Leftrightarrow \exists \xi \quad \sigma' = \xi\sigma$, où ξ est une permutation.

Démonstration :

Similaire à celle du lemme 5.4, en prenant maintenant

$$V = D(\sigma) \cup D(\sigma') \cup I(\sigma) \cup I(\sigma'). \quad \square$$

Remarque : Par contre, ce lemme n'est plus vrai pour les substitutions infinies, car une injection de V dans V n'est pas toujours une bijection lorsque V est infini. Par exemple, en définissant σ par :

$$\forall i \quad \sigma v_i = v_{2i},$$

et en prenant pour σ' l'identité : $\sigma' = \emptyset$, alors on a $\sigma \equiv \sigma'$, mais on ne peut obtenir σ' de σ par une permutation.

Nous allons maintenant étudier plus en détail la structure de l'ensemble quotient $\hat{T} = T/\equiv$. On dénote par $[e]$ la classe d'équivalence de e , c'est à dire l'ensemble de ses variantes. \leq définit un ordre partiel sur \hat{T} , que nous noterons également \leq . Vis à vis de cet ordre, \hat{T} possède l'élément minimum $\perp = [x] = V$.

Avant de poursuivre, faisons un bref rappel algébrique.

Nous utiliserons les abréviations "inf" (resp. "sup") pour borne inférieure (resp. borne supérieure).

5.3. Inf demi-treillis bien fondés

Définition :

On dit que la structure $\langle \hat{T}, \leq, \wedge \rangle$ est un *inf demi-treillis bien fondé* (en abrégé ITF) ssi :

1) \leq est une relation d'ordre sur \hat{T}

2) $\forall t_1, t_2 \in \hat{T}$ il existe dans \hat{T} un plus grand minorant de t_1 et t_2 , appelé *inf* de t_1 et t_2 , et dénoté par $t_1 \wedge t_2$.

$$\text{c.a.d. : } t_1 \wedge t_2 \leq t_1$$

$$t_1 \wedge t_2 \leq t_2$$

$$\forall t \in \hat{T} (t \leq t_1 \text{ et } t \leq t_2) \Rightarrow t \leq t_1 \wedge t_2$$

3) \leq est un ordre bien fondé. C'est à dire, en dénotant

$$t < t' \Leftrightarrow t \leq t' \text{ et } t \neq t',$$

il n'existe pas de chaîne infinie décroissante

$$t_1 > t_2 > \dots > t_n > \dots$$

Nous allons maintenant énoncer quelques propriétés des ITF.

On désignera abusivement un ITF $\langle \hat{T}, \leq, \wedge \rangle$ par \hat{T} .

Lemme 5.7

Soit \hat{T} un ITF. Tout sous-ensemble A non vide de \hat{T} admet un *inf* $\cap A$.

Démonstration :

Soit A non vide. Soit \bar{A} le plus petit sous-ensemble de et fermé par \wedge . Nous montrons que \bar{A} possède un minimum \bar{a} , c.a.d.

$$\exists \bar{a} \in \bar{A} \quad \forall b \in \bar{A} \quad \bar{a} \leq b.$$

En effet, \bar{A} est non vide et possède un élément a_1 . Si \bar{A} ne possédait pas de minimum, alors on construirait une chaîne infinie décroissante :

$$a_1 > a_2 > \dots \quad \text{comme suit.}$$

Soit a_i le dernier élément construit. Par hypothèse a_i n'est pas minimum, donc $\exists b_i \in \bar{A} \quad a_i \neq b_i$.

Soit $a_{i+1} = a_i \wedge b_i$; $a_{i+1} \in \bar{A}$, par construction de \bar{A} .

Il est clair que $a_{i+1} \leq a_i$. Si $a_{i+1} = a_i$ alors on aurait $a_i \leq b_i$, contraire à l'hypothèse. Donc $a_{i+1} < a_i$.

L'existence d'une chaîne infinie décroissante étant exclue par la définition d'ITF, \bar{A} possède bien un minimum \bar{a} . Puisque $A \subset \bar{A}$, on a bien $\forall a \in A \quad a \leq \bar{a}$. Soit maintenant b un minorant quelconque de A . Il est clair que b minore aussi \bar{A} , et donc $b \leq \bar{a}$. L'élément \bar{a} a donc bien les propriétés d'un inf, et on définit $\inf A = \bar{a}$. \square

En particulier, on dénote : $\perp = \inf \hat{T}$.

Lemme 5.8.

Soit \hat{T} un ITF. Tout sous-ensemble A de \hat{T} majoré admet un sup $\sup A$.

Démonstration :

Soit A un sous-ensemble de \hat{T} majoré. Soit \hat{A} l'ensemble des majorants de A . Par hypothèse \hat{A} n'est pas vide. Par le lemme 5.7, \hat{A} admet donc un inf : $\hat{a} = \inf \hat{A}$. \hat{A} étant fermé par \wedge , \hat{a} est en fait un minimum, i.e. $\hat{a} \in \hat{A}$. Il est clair que \hat{a} est le sup de A , et on définit donc $\sup A = \hat{a}$. \square

Les lemmes 5.7 et 5.8 nous indiquent dans quelle mesure la condition d'ordre bien fondé implique des propriétés de complétude d'un ITF. En particulier ils expriment que si \hat{T} est un ITF, alors on peut le compléter par un élément T en un treillis complet. En effet, soit $\hat{\hat{T}} = \hat{T} \cup \{T\}$. On étend \leq à $\hat{\hat{T}}$ en ajoutant :

$\forall x \in \hat{\hat{T}} \quad x \leq T$. Alors il est clair que T est l'inf du sous-ensemble vide de $\hat{\hat{T}}$, que tout sous-ensemble de $\hat{\hat{T}}$ est majoré par T , qu'enfin tout sous-ensemble $A \cup \{T\}$ de $\hat{\hat{T}}$ contenant T admet pour inf $\inf A$ et pour sup T . Ceci montre bien que $\hat{\hat{T}}$ est un treillis complet. La réciproque bien sûr

n'est pas vraie, c'est à dire qu'il existe des treillis complets qui ne sont pas des ITF.

Remarquons enfin que nous n'utilisons aucune propriété de cardinalité de \hat{T} .

5.4. \hat{T} est un ITF.

Le but de ce paragraphe est de montrer que notre ensemble \hat{T} des classes d'équivalences de termes de 1^{er} ordre possède une structure d'inf demi-treillis bien fondé, vis à vis de l'ordre \leq défini en 5.2. L'élément minimum de \hat{T} , est bien sûr V .

La preuve se décompose en deux parties :

- 1) \leq est un ordre bien fondé sur \hat{T} .
- 2) existence de l'inf.

Tout d'abord, remarquons que θ et \vee sont bien définis dans \hat{T} , car

$$e \equiv e' \implies (\theta(e) = \theta(e') \text{ et } \vee(e) = \vee(e')).$$

Nous ferons les preuves dans T plutôt que dans \hat{T} , toutefois. Une des difficultés ici est que, dans T , \leq n'est pas compatible avec la structure des termes, i.e.

$$e_i \leq e'_i \not\implies F(\dots, e_i, \dots) \leq F(\dots, e'_i, \dots).$$

De plus, les substitutions ne sont pas des fonctions monotones, i.e.
 $e \leq e' \not\implies \sigma e \leq \sigma e'$.

Nous allons tout d'abord définir une opération p de T dans \hat{T} , appelée projection.

Définition :

On dit que les permutations ξ_i séparent les termes e_i , avec $1 \leq i \leq n$, ssi :

$$\forall i, j \leq n \quad V(\xi_i e_i) \cap V(\xi_j e_j) = \emptyset.$$

Soit $e \in T$. On définit la *projection* $p(e)$ de e dans \hat{T} par la définition récursive :

$$(i) \quad \forall x \in V \quad p(x) = l$$

$$(ii) \quad \forall F \in C \quad p(F(\dots, e_i, \dots)) = [F(\dots, \xi_i p_i, \dots)],$$

où $p_i \in p(e_i)$, et les ξ_i séparent les p_i .

Il est clair que dans (ii), le résultat ne dépend pas de l'élément p_i pris dans la classe $p(e_i)$. Notons également que p est compatible avec \equiv , i.e. $e \equiv e' \Rightarrow p(e) = p(e')$.

Par exemple, en prenant $e = F(x, G(y, x))$, on a

$$p(e) = [F(x, G(y, z))].$$

Intuitivement, p distingue toutes les occurrences de variables.

Le nom de projection est suggéré par les deux propriétés :

$$\begin{cases} p(e) \leq \lceil e \rceil \\ \forall e' \in p(e) \quad p(e') = p(e). \end{cases}$$

Nous allons maintenant montrer que pour croître dans T , il faut soit augmenter θ , soit diminuer v . Par exemple :

$$F(x, y) \leq F(x, G(x, y)) \leq F(x, G(x, x)).$$

Lemme 5.9

$$\forall e, \sigma \quad \theta(\sigma e) = \theta(e) + \sum_{x \in V} \#(x, e) \theta(\sigma x)$$

Démonstration :

Par récurrence sur e :

$$(i) \quad e = x \text{ trivial}$$

$$(ii) \quad e = Fe_1 \dots e_n$$

$$\begin{aligned} \theta(\sigma e) &= \theta(F\sigma e_1 \dots \sigma e_n) = 1 + \sum_{i=1}^n \theta(\sigma e_i) \\ &= 1 + \sum_{i=1}^n \theta(e_i) + \sum_{i=1}^n \sum_{x \in V} \#(x, e_i) \theta(\sigma x) \text{ par hyp. de récurrence} \\ &= \theta(e) + \sum_{x \in V} \#(x, e) \theta(\sigma x) \text{ par définition. } \square \end{aligned}$$

Remarque : $\sum_{x \in V} \#(x, e) \theta(\sigma x) = \sum_{x \in V} \#(x, e) \theta(\sigma x)$, où V est l'ensemble fini $\mathcal{D}(\sigma) \cap V(e)$, car en dehors de V au moins un des facteurs du produit est nul.

Corollaire :

$$\forall e, \sigma \quad \theta(\sigma e) = \theta(e) \iff \forall x \in V \quad \sigma x \in V.$$

Lemme 5.10

$$\forall e, \sigma \quad \theta(e) = \theta(\sigma e) \implies v(e) \geq v(\sigma e).$$

Démonstration : par le corollaire ci-dessus, si $\theta(e) = \theta(\sigma e)$, alors

$$\forall x \in V(e) \quad \sigma x \in V. \text{ On a donc}$$

$$V(\sigma e) = \bigsqcup_{x \in V(e)} V(\sigma x) = \{\sigma x \mid x \in V(e)\}, \text{ et donc}$$

$$v(\sigma e) \leq v(e). \square$$

Remarquons que l'égalité n'est obtenue que lorsque $\sigma|V(e)$ est une injection, qui peut alors être complétée en une permutation, i.e. $e \equiv \sigma e$.

Autrement dit :

Lemme 5.11

$$\forall e, e' \quad e < e' \text{ et } \theta(e) = \theta(e') \implies v(e) > v(e').$$

(où $e < e'$ est bien sûr défini comme : $e < e' \iff e \leq e' \text{ et } e' \neq e$).

Lemme 5.12

$$\forall e, e' \quad e \leq e' \text{ et } \theta(e) = \theta(e') \implies p(e) = p(e').$$

Démonstration : par récurrence structurelle sur e :

(i) si $e = x$ alors $\theta(e') = 0$ entraîne $e' \in V$, et donc $p(e') = p(e) = V$.

(ii) si $e = F(\dots, e_i, \dots)$, soit σ tel que $e' = \sigma e$. Par le corollaire au lemme 5.9, on a $\forall x \in V(e) \quad \sigma x \in V$, et donc aussi $\theta(e_i) = \theta(\sigma e_i)$, ce qui implique $p(e_i) = p(\sigma e_i)$ par hypothèse de récurrence, et donc aussi $p(e) = p(\sigma e)$. \square

Il est facile de montrer que $\theta(p(e)) = \theta(e)$, et que $\forall e' \in p(e)$ $e' \leq e$. Par le lemme 5.10, ceci entraîne $v(e) \leq v(p(e))$. En combinant ceci avec les lemmes 5.11 et 5.12, on obtient :

Lemme 5.13

$$\forall e, e' \quad e < e' \text{ et } \theta(e) = \theta(e') \Rightarrow v(e') < v(e) \leq v(p(e')).$$

Lemme 5.14

Il n'y a pas de chaîne infinie décroissante dans T .

Démonstration :

Soit une chaîne infinie décroissante :

$$e_1 > e_2 > \dots > e_i > e_{i+1} \dots .$$

D'après le lemme 5.9, on a $\theta(e_i) \geq \theta(e_{i+1})$. Donc, à partir d'un rang k on doit avoir des termes de taille identique : $\forall i \geq k \quad \theta(e_i) = \theta(e_k)$. Mais alors, on devrait avoir, par le lemme 5.13, une chaîne infinie croissante : $v(e_k) < v(e_{k+1}) < \dots$ bornée par $v(p(e_k))$. Ceci n'est pas possible, ce qui prouve le théorème. \square

Naturellement, le résultat s'étend immédiatement à \hat{T} .

Remarque : En fait, nous pourrions montrer davantage :

$\forall t \in \hat{T}, \quad \{t' | t' \leq t\}$ est fini. Les chaînes décroissantes issues d'un terme donné sont donc de longueur bornée.

Nous allons maintenant montrer l'existence de $t \wedge t'$, pour tous t et t' dans \hat{T} . Nous nous intéressons aux classes plutôt qu'aux termes, car l'inf est unique dans les ordres, et non les pré-ordres. Mais nous continuons de raisonner sur les termes, afin de faire "passer" les récurrences.

Définitions :

Soit ϕ une bijection quelconque entre T^2 et V , π^1 et π^2 ses fonctions inverses :

$$\forall e, e' \in T \quad \phi(e, e') \in V, \quad e = \pi^1 \phi(e, e'), \quad e' = \pi^2 \phi(e, e').$$

L'inf de deux termes e et e' est défini récursivement par :

- (i) $F(\dots, e_i, \dots) \wedge F(\dots, e'_i, \dots) = F(\dots, e_i \wedge e'_i, \dots) \quad \forall F \in C$
- (ii) $e \wedge e' = \phi(e, e')$ dans tous les autres cas.

Exemple :

$$\text{Soit } e = F(G(A), G(A), A, A, C)$$

$$e' = F(G(B), G(B), B, A, B).$$

$$\text{Soit } x = \phi(A, B), \quad y = \phi(C, B). \quad \text{On a :}$$

$$\begin{aligned} e \wedge e' &= F(G(A) \wedge G(B), G(A) \wedge G(B), A \wedge B, A \wedge A, C \wedge B) \\ &= F(G(A \wedge B), G(A \wedge B), x, A, y) \\ &= F(G(x), G(x), x, A, y). \end{aligned}$$

Prouvons maintenant que $e \wedge e'$ est bien l'inf de e et e' (modulo une permutation).

Lemme 5.15

$$\forall e, e' \quad e \wedge e' \leq e \quad \text{et} \quad e \wedge e' \leq e'.$$

Démonstration : Soit $e'' = e \wedge e'$.

\wedge étant symétrique en e et e' , il suffit de montrer $e'' \leq e$.

Pour cela, on montre que $e = \pi^1 e''$, par récurrence structurelle sur e'' :

- (i) si $e'' = x$ alors $e'' = \phi(e, e')$ par cas (ii) de la définition de \wedge , et $e = \pi^1 e''$ par définition de π^1 .
- (ii) si $e'' = F(\dots, e''_i, \dots)$ alors par le cas (i) de la définition de \wedge ,

On doit avoir $e = F(\dots, e_i, \dots)$,

$$e' = F(\dots, e'_i, \dots),$$

et $e'' = e_i \wedge e'_i$. D'où :

$$\pi^1 e'' = F(\dots, \pi^1 e''_i, \dots) = F(\dots, e_i, \dots) \text{ par hypothèse de récurrence.}$$

$\vdash e.$ \square

Nous voulons maintenant prouver que

$$\forall e, e', e'' \quad (e \leq e' \text{ et } e \leq e'') \implies e \leq e' \wedge e''.$$

Mais ici la récurrence évidente sur e ne "passe pas", toujours à cause du fait que \leq n'est pas compatible avec la structure des termes. C'est pourquoi nous devons employer une hypothèse de récurrence plus forte, afin de "paralléliser" les substitutions.

Lemme 5.16

Soit $e_i, e'_i, e''_i \in T$, $1 \leq i \leq n$ tels qu'il existe deux substitutions σ' et σ'' telles que :

$$\begin{cases} e'_i = \sigma' e_i & 1 \leq i \leq n \\ e''_i = \sigma'' e_i. & \end{cases}$$

Alors il existe une substitution σ telle

que $e'_i \wedge e''_i = \sigma e_i$. $1 \leq i \leq n$.

Démonstration : par récurrence sur $\theta = \sum_{i=1}^n \theta(e_i)$.

(i) si $\theta = 0$ alors $\forall i \leq n \quad e_i \in V$ et on peut prendre

$$\sigma = \{<e_i, e'_i \wedge e''_i>\}$$

(si $e_i = e_j$, alors $e'_i = \sigma' e_i = \sigma' e_j = e'_j$, de même $e''_i = \sigma'' e_i = \sigma'' e_j = e''_j$, et donc $e'_i \wedge e''_i = e'_j \wedge e''_j$).

(ii) si $\theta > 0$ alors il existe $k \leq n$ tel que $\theta(e_k) > 0$,

$$\text{c.a.d. } e_k = F(\dots, e_k^j, \dots) \text{ avec } F \in \mathcal{C}.$$

$$\text{Alors : } e'_k = \sigma' e_k = F(\dots, \sigma' e_k^j, \dots)$$

$$e''_k = \sigma'' e_k = F(\dots, \sigma'' e_k^j, \dots).$$

On applique maintenant l'hypothèse de récurrence à :

$$\left\{ \begin{array}{l} e_1, \dots, e_{k-1}, e_k^1, \dots, e_k^p, e_{k+1}, \dots, e_n \\ e'_1, \dots, e'_{k-1}, \sigma' e_k^1, \dots, \sigma' e_k^p, e'_{k+1}, \dots, e'_n \\ e''_1, \dots, e''_{k-1}, \sigma'' e_k^1, \dots, \sigma'' e_k^p, e''_{k+1}, \dots, e''_n \end{array} \right.$$

et on obtient :

$$\begin{aligned} \forall i \leq n \quad i \neq k \quad \sigma e_i &= e'_i \wedge e''_i \\ \forall j \leq p \quad \sigma e_k^j &= \sigma' e_k^j \wedge \sigma'' e_k^j \text{ et donc} \\ e'_k \wedge e''_k &= F(\dots, \sigma' e_k^j \wedge \sigma'' e_k^j, \dots) = F(\dots, e_k^j, \dots) = e_k. \square \end{aligned}$$

Théorème 13 : \hat{T} est un ITF.

Démonstration :

Découle directement des lemmes 5.14, 5.15 et 5.16. \square

Remarquons que le passage au quotient par \equiv relativise la bijection ϕ : on peut prendre un différent ϕ pour tout couple $\langle e, e' \rangle$; il n'a alors besoin d'être défini que pour les sous termes de e et e' . En fait tout ce qui importe est de distinguer les occurrences de sous termes communs.

On sait reconnaître en temps linéaire les occurrences de sous termes identiques. On peut donc calculer $t \wedge t'$ en un temps proportionnel à $\theta(t) + \theta(t')$. L'étude détaillée des coefficients et de la mémoire nécessaire dépend des structures de données utilisées ; nous ne la ferons pas ici.

T a bien sûr la même structure que \hat{T} , sauf que \leq y est seulement un préordre. On peut donc qualifier T d'inf-demi-pré-treillis bien fondé.

La complétude de \hat{T} découle des lemmes 5.7 et 5.8. En particulier, remarquons que nous avons prouvé l'existence, pour tous t et t' dans \hat{T} possédant un majorant, d'un $\sup t \vee t'$, sans avoir recours à un algorithme d'unification. C'est la démarche contraire que nous allons employer : nous allons définir un algorithme d'unification, et montrer qu'il est correct, c'est à dire qu'il calcule le sup de deux termes séparés.

De même nous avons donné pour l'inf directement une définition récursive, sans passer par un algorithme d'anti-unification comme le font Plotkin [PG2] et Reynolds [RJ1]. Remarquons que le treillis complet considéré par Reynolds n'est autre que T , mais "renversé", auquel on a adjoint deux éléments supplémentaires Λ et Ω . Λ est totalement inutile, puisque $V = [x]$ remplit son rôle dans \hat{T} . Quant à Ω , il représente la réponse "incompatible" à l'algorithme d'unification. Nous allons voir plus loin qu'en fait il est possible de compléter \hat{T} d'une manière plus intéressante.

Avant d'aborder l'unification, remarquons que nos résultats nous permettent de caractériser simplement la relation de couverture de notre ITF.

Définitions : Soit $e_1, e_2 \in T$

On dit que e_1 couvre e_2 ssi $e_1 > e_2$ et $\forall e_3 \quad e_1 > e_3 > e_2$.

On définit la relation \rightarrow dans T^2 par $e_1 \rightarrow e_2$ ssi :

Soit (i) $e_1 = \sigma e_2$ avec $\sigma = \{<x, F(y_1, \dots, y_p)>\}$
où $x \in V(e_2)$ et $y_i \notin V(e_2) \quad 1 \leq i \leq p$.

Soit (ii) $e_1 = \sigma e_2$ avec $\sigma = \{<x, y>\}$ où x et y sont des variables distinctes de $V(e_2)$.

Lemme 5.17

$$e_1 \rightarrow e_2 \implies e_1 \text{ couvre } e_2.$$

Démonstration :

D'après les lemmes 5.9 et 5.13 :

$$e_1 > e_2 \implies \begin{cases} \theta(e_1) > \theta(e_2) \\ \text{ou } \theta(e_1) = \theta(e_2) \text{ et } v(e_1) < v(e_2). \end{cases}$$

Si $e_1 \rightarrow e_2$, alors $e_1 \geq e_2$, avec deux cas :

$$(i) \quad e_1 = \sigma e_2 \text{ avec } \sigma = \{<x, F(y_1, \dots, y_p)>\}.$$

Dans ce cas, soit $m = \#(x, e_2) \geq 1$. $\theta(e_1) = \theta(e_2) + m$, d'où $e_1 > e_2$.

Soit e_3 quelconque tel que $e_1 \geq e_3 > e_2$, avec $e_3 = \rho e_2$, $e_1 = n e_3$.

$\sigma e_2 = n \rho e_2$ entraîne en particulier, par le lemme 5.3 :

$$\forall y \in V(e_2) - \{x\} \quad n \rho y = y, \text{ d'où } \rho y \in V.$$

a) $\rho x \in V$. Alors $\theta(e_3) = \theta(e_2)$, et $e_2 > e_2$ entraîne $v(e_3) < v(e_2)$.

Donc $\exists y_1, y_2 \in V(e_2)$, $y_1 \neq y_2$ tels que $\rho y_1 = \rho y_2$, et donc $n \rho y_1 = n \rho y_2$.

Soit par exemple $y_1 \neq x$. Alors $n \rho y_1 = y_1$.

Si $y_2 \neq x$ alors impossible car $n \rho y_2 = y_2 \neq y_1$.

Si $y_2 = x$ alors impossible car $n \rho x = \sigma x \neq y_1$.

Ce cas ne peut donc pas se produire.

b) $\rho x = F(t_1, \dots, t_p)$ (car $n \rho x = F(y_1, \dots, y_p)$).

Alors soit $n' = \rho \setminus V(e_2) \cup \{<y_i, t_i> \mid 1 \leq i \leq p\}$.

On a $\forall y \in V(e_2) \quad n' \sigma y = \rho y$, d'où $e_3 = n' e_1 \geq e_1$, ce qui entraîne

$e_3 \equiv e_1$, et donc e_1 couvre e_2 .

$$(ii) \quad e_1 = \sigma e_2 \text{ avec } \sigma = \{<x, y>\}, \quad x \neq y, \quad x, y \in V(e_2).$$

On a alors $\theta(e_1) = \theta(e_2)$ et $v(e_1) = v(e_2) - 1$, ce qui entraîne bien

$e_1 > e_2$, et $\exists e_3 \quad e_1 > e_3 > e_2$. \square

Lemme 5.18

$$e_1 \text{ couvre } e_2 \implies \exists e_3 \in e_1 \quad e_3 \rightarrow e_2.$$

Démonstration :

Soit σ tel que $e_1 = \sigma e_2$. Il y a deux cas :

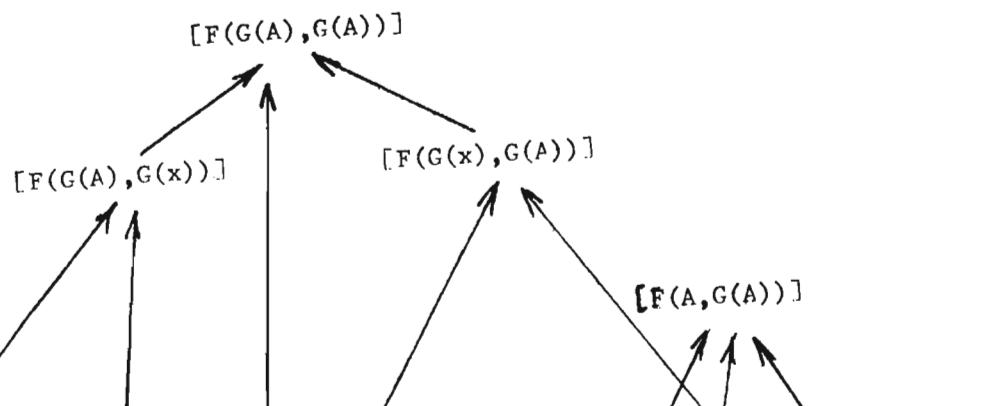
(i) $\exists x \in V(e_2) \theta(\sigma x) \geq 1$, c.a.d. $\sigma x = F(\dots, F_i, \dots)$.

Alors soit $\{y_i\}$ des variables distinctes non dans $V(e_2)$, et soit $\rho = \{\langle x, F(\dots, y_i, \dots) \rangle\}$. On a $e_1 \geq \rho e_2$, car $\sigma = \rho \sigma$, avec $\sigma = \{\langle y_i, F_i \rangle\} \cup \sigma \setminus V(e_2)$. Or $\rho e_2 \rightarrow e_2$ par définition de \rightarrow (i), donc $\rho e_2 \geq e_2$ par le lemme 5.17, donc $e_1 \equiv \rho e_2$ par hypothèse e_1 couvre e_2 .

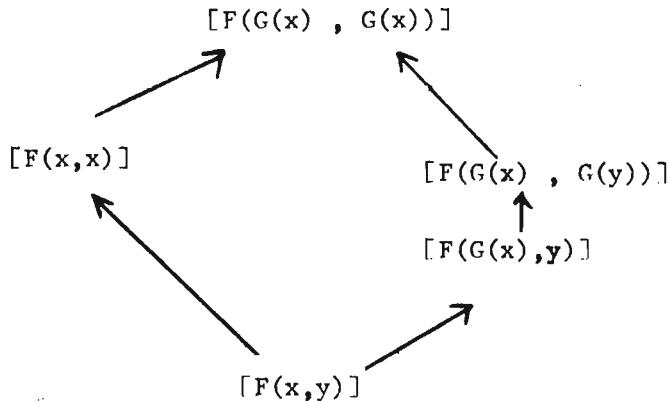
(ii) $\forall x \in V(e_2) \theta(\sigma x) = 0$, c.a.d., $\sigma x \in V$. Donc $\theta(e_2) = \theta(e_1)$, ce qui implique $v(e_1) < v(e_2)$ donc $\exists x_1, x_2 \in V(e_2) \sigma x_1 = \sigma x_2$. Soit $\rho = \{\langle x_1, x_2 \rangle\}$. De même que plus haut, on a $e_1 \geq \rho e_2 \rightarrow e_2$, ce qui prouve $e_1 \equiv \rho e_2$. \square

Les lemmes 5.17 et 5.18 nous montrent donc que \rightarrow/\equiv est la relation de couverture dans l'ITF \hat{T} .

A titre d'exemple nous donnons une portion de \hat{T} , avec la relation \rightarrow/\equiv pour C contenant une constante A , un symbole de fonction unaire C et un symbole de fonction binaire F .

$\theta = 5$  $\theta = 4$ $[F(G(A), A)]$ $\theta = 3$ $F(G(A), x)$, $[F(G(x), A)]$ $[F(G(x), G(y))]$ $[F(A, G(x))]$ $[F(x, G(A))]$ $\theta = 2$ $[F(G(x), y)]$ $\theta = 1$ $[A]$ $[F(x, y)]$ $[C(x)]$ $[C(F(x, y))]$ $[C(G(x))]$ $\theta = 0$ $[x] = V$

Comme on peut le constater sur la figure ci-dessus, \hat{T} n'est pas modulaire. En effet, on a la sous-structure :



Finalement, remarquons que \hat{T} admet pour élément minimum $[x] = V$, et pour éléments maximaux tous les termes fermés (uniques dans leurs classes).

Dénotons par F l'ensemble de ces termes fermés :

$$F = \{e \in T \mid v(e) = 0\}.$$

F est non vide si et seulement si C contient au moins une constante C d'arité $\alpha(C) = 0$. Dans ce cas, il est facile de montrer que tout terme est l'inf des termes fermés qui le majorent :

Lemme 5.19 Si C contient une constante C , avec $\alpha(C)=0$, alors $\forall e \in T$

$$e \equiv \inf \{e' \in F \mid e' \geq e\}.$$

Démonstration :

Soit $V(e) = \{x_1, \dots, x_n\}$, et soit $F \in C$ avec $\alpha(F) \geq 1$.

Soit e_1, \dots, e_n n termes fermés distincts commençant par F (par exemple, si F est unaire prendre $e_i = F^i(C)$).

On considère $\sigma_1 = \{\langle x_i, e_i \rangle \mid 1 \leq i \leq n\}$ et $\sigma_2 = \{\langle x_i, C \rangle \mid 1 \leq i \leq n\}$.

En choisissant une bijection ϕ telle que $\forall i \leq n \quad \phi(e_i, C) = x_i$, on montre facilement par récurrence sur e que $e = \sigma_1 e \wedge \sigma_2 e$. \square

5.5. Unification dans T

5.5.1. Relations entre l'unification et le sup

Soit $e_1, e_2 \in T$. *Unifier* e_1 et e_2 , c'est résoudre dans T l'équation $e_1 = e_2$. On appelle *unificateur* de e_1 et e_2 toute solution, c'est à dire toute substitution σ telle que $\sigma e_1 = \sigma e_2$.

Tout unificateur σ de e_1 et e_2 détermine une instance commune, c'est à dire un majorant commun de e_1 et e_2 . Inversement, si e_1 et e_2 sont des termes séparés, i.e. $V(e_1) \cap V(e_2) = \emptyset$, alors tout majorant commun détermine un unificateur : soit $e = \sigma_1 e_1 = \sigma_2 e_2$; en prenant $\sigma = \sigma_1 \upharpoonright V(e_1) \cup \sigma_2 \upharpoonright V(e_2)$, on a bien $\sigma e_1 = \sigma e_2$.

Si e_1 et e_2 ne sont pas séparés, alors soit ξ une permutation séparant e_2 de e_1 : $V(e_1) \cap V(\xi e_2) = \emptyset$.

Tout majorant commun $\sigma_1 e_1 = \sigma_2 e_2$ détermine un unificateur $\sigma_1 \upharpoonright V(e_1) \cup (\sigma_2 \xi^{-1}) \upharpoonright V(\xi e_2)$ de e_1 et ξe_2 . On peut donc trouver les majors communs à e_1 et e_2 par unification de e_1 et ξe_2 . Inversement, soit à unifier e_1 et e_2 .

Soit $V = \{x_1, x_2, \dots, x_n\} = V(e_1) \cap V(e_2)$.

Soit F un symbole de fonction binaire ; on considère les termes :

$$E_1 = F(x_1, F(x_2, \dots, F(x_n, e_1) \dots))$$

$$\text{et } E_2 = F(x_1, F(x_2, \dots, F(x_n, e_2) \dots)).$$

Soit F un majorant commun à E_1 et E_2 , s'il en existe :

$$F = F(x_1, F(x_2, \dots, F(x_n, e) \dots)).$$

On a $E = \sigma_1 E_1 = \sigma_2 E_2$, avec $\sigma_1 \upharpoonright V = \sigma_2 \upharpoonright V = \{<x_i, X_i> \mid 1 \leq i \leq n\}$.

On peut donc former $\sigma = \sigma_1 \nabla(e_1) \cup \sigma_2 \nabla(e_2)$, et on a bien $e = \sigma e_1 = \sigma e_2$. De plus, σ est unique par le lemme 5.3.

La recherche d'unificateurs est donc identique à la recherche de majorants communs. A l'existence du sup de deux classes dans \hat{T} va correspondre l'existence d'un unificateur principal de deux termes quelconques. Ce résultat est un résultat fondamental des langages de premier ordre. Il a des implications profondes en théorie de la démonstration, comme l'a pressenti pour la première fois Herbrand (voir la propriété A dans sa thèse [HJ1]).

Le problème a été repris par Prawitz [PD1], puis par Robinson [RJA1], qui utilise un algorithme d'unification comme clé dans la définition d'un système déductif complet utilisant une règle d'inférence unique, la résolution. La règle de résolution est une "coupure modulo unification", sur des propositions (clauses) mises sous forme normale conjonctive, après élimination des quantificateurs par la méthode de Skolem. Cette règle sert de base à la plupart des programmes de démonstration automatique utilisés à ce jour.

Nous allons maintenant décrire un algorithme d'unification similaire à celui décrit originellement par Robinson, puis nous prouverons sa correction, en montrant qu'il calcule effectivement l'unificateur principal de deux termes unifiables.

On peut donc utiliser cet algorithme pour trouver le sup de deux classes t_1 et t_2 dans \hat{T} ; soit $e_1 \in t_1$, $e_2 \in t_2$, ξ_1 et ξ_2 deux permutations qui séparent e_1 et e_2 , σ l'unificateur principal, s'il existe, de $\xi_1 e_1$ et $\xi_2 e_2$. Alors on définit $t_1 \wedge t_2 = [\sigma \xi_1 e_1]$.

5.5.2. Unificandes

Nous allons utiliser des notations similaires à celles définies au chapitre 3.

On définit un *unificande* comme un ensemble fini de paires de termes $N = \{e_i, e'_i \mid 1 \leq i \leq n\}$.

On dénote par N l'ensemble des unificandes.

L'ensemble des variables de N est :

$$V(N) = \bigsqcup_{i \leq n} V(e_i) \cup \bigsqcup_{i \leq n} V(e'_i).$$

L'ouverture de N est : $v(N) = |V(N)|$.

On dit que la substitution σ *unifie* N ssi

$$\forall i \leq n \quad \sigma e_i = \sigma e'_i.$$

On dénote par $U(N)$ l'ensemble des unificateurs de N . (Remarquer que $U(\emptyset) = S$, l'ensemble de toutes les substitutions).

Nous allons tout d'abord donner un algorithme de simplification, SIMPL, qui prend comme argument un unificande, et retourne comme résultat un unificande ou la réponse " \top " (échec). On identifie ici la réponse succès avec l'unificande vide \emptyset .

Nous décrirons ensuite l'algorithme d'unification comme énumérant l'analogue d'un AP. Mais ici tout arbre se réduit à une branche finie, et il y a donc une solution unique qu'on obtient en un temps fini.

On dénote $\bar{N} = N \cup \{\top\}$, et par convention $U(\top) = \emptyset$ et $V(\top) = \emptyset$.

5.5.3. L'algorithme SIMPL

Description algorithmique de $\text{SIMPL}(N)$, pour tout $N \in \mathbb{N}$.

étape 1 : S'il n'existe pas dans N de paire $\langle e, e' \rangle$ telle que $\theta(e) > 0$ et $\theta(e') > 0$, alors aller à l'étape 2.

Sinon, soit $\langle e, e' \rangle$ une telle paire : $N = \tilde{N} \cup \{\langle e, e' \rangle\}$,

$$\text{avec } e = Fe_1 \cdots e_n$$

$$e' = Ge'_1 \cdots e'_n$$

(i) si $F \neq G$ alors sortir avec la réponse "T".

(ii) sinon, on doit avoir $n' = n$. Alors, avec $N \leftarrow \tilde{N} \cup \{\langle e_i, e'_i \rangle \mid 1 \leq i \leq n\}$, retourner à l'étape 1.

étape 2 : Remplacer dans N toutes les paires $\langle e, x \rangle$, avec $\theta(e) > 0$, par les paires $\langle x, e \rangle$ correspondantes.

étape 3 : Supprimer dans N toutes les paires $\langle x, x \rangle$.

étape 4 : S'il existe dans N une paire $\langle x, e \rangle$ avec $x \in V(e)$, sortir avec la réponse "T" ; sinon N est le résultat. \square

Il est évident que l'algorithme SIMPL termine toujours. Pour l'étape 1, par récurrence sur $\sum_{\langle e, e' \rangle \in N} \theta(e)$, pour les autres étapes c'est trivial. Il est à remarquer que, du point de vue algorithmique, il est possible de faire les trois étapes finales en parallèle en une seule passe sur la liste représentant N .

La correction de SIMPL se ramène au lemme suivant.

Lemme 5.20

Pour tout unificande N :

$$a) \quad U(\text{SIMPL}(N)) = U(N)$$

$$b) \quad V(\text{SIMPL}(N)) \subset V(N)$$

Démonstration :

Laissée au lecteur. Le seul point intéressant consiste à remarquer que si $x \in V(e)$ et $e \neq x$, alors $\exists \sigma \quad \sigma x = \sigma e$, car pour tout σ $\theta(\sigma x) < \theta(\sigma e)$ par le lemme 5.9, et donc que $SIMPL(N) = \tau \Rightarrow U(N) = \emptyset$. \square

5.5.4. Unification d'un unificande N

L'algorithme d'unification proprement dit consiste en la construction d'une suite :

$$N_0 \xrightarrow{\sigma_1} N_1 \cdots \xrightarrow{\sigma_p} N_p \quad p \geq 0$$

telle que :

- $N_0 = SIMPL(N)$
- $N_i \in N \quad \forall i < p$
- $N_p = \emptyset$ (succès)
- ou $= \tau$ (échec)
- $\sigma_i \in N_{i-1} \quad i > 0$
- $N_i = SIMPL(\{<\sigma_i e_i, \sigma_i e'_i> \mid <e_j, e'_j> \in N_{i-1}\}) \quad i > 0$

Remarquons qu'on a le choix de la paire $\sigma_i = <x, e>$ choisie dans l'unificande N_{i-1} . Cette paire est alors considérée comme composante de substitution, et appliquée à N_{i-1} . Remarquons tout de suite qu'on peut éviter des calculs de simplification inutiles en posant : $N_{i-1} = \{\sigma_i\} \cup \bar{N}_i$, et en définissant plutôt $N_i = SIMPL(\sigma_i \bar{N}_i)$. La paire $\langle \sigma_i x, \sigma_i e \rangle$, avec $\sigma_i = \langle x, e \rangle$ sera éliminée par les étapes 1 et 3 de SIMPL, puisqu'on est sûr que $x \notin V(e)$ par la passe de simplification précédente. On a donc $V(\sigma_i \bar{N}_i) = V(N_{i-1}) - \{x\}$, et en utilisant le lemme 5.20 on a donc $v(N_i) < v(N_{i-1})$, ce qui assure la terminaison en au plus $v(N)$ étapes.

5.5.4. Preuve de correction

Soit $N_0 \xrightarrow{\sigma_1} N_1 \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_p} N_p = \emptyset$ une suite d'unificandes construite par l'algorithme précédent. On définit une séquence ξ_0, \dots, ξ_p des substitutions par la récurrence descendante :

$$\begin{cases} \xi_p = \emptyset \\ \xi_{i-1} = \xi_i \sigma_i & 0 < i \leq p. \end{cases}$$

La correction de l'algorithme s'exprime par le théorème suivant.

Théorème 14 :

Pour tout unificande $N \in \mathcal{N}$, $U(N)$ est vide ou possède un élément minimum. L'algorithme d'unification retourne T dans le premier cas, dans le second cas il s'arrête en p étapes avec $N_p = \emptyset$, et $\xi_0 = \sigma_p \cdots \sigma_1$ est un unificateur principal de N .

Démonstration :

Supposons que l'algorithme d'unification s'arrête avec succès en p étapes : $N_p = \emptyset$. Avec ξ_i défini comme plus haut, on prouve par récurrence descendante sur i que $\forall i \leq p \quad \xi_i \in U(N_i)$.

(i) pour $i=p$ c'est trivial, car $U(\emptyset) = S$.

(ii) $i > 0$; supposons $\xi_i \in U(N_i) = U(\sigma_i \bar{N}_i)$ pour le lemme 5.20.

On a donc $\xi_{i-1} = \xi_i \sigma_i \in U(\bar{N}_i)$.

Soit $\sigma_i = \langle x, e \rangle$. On a $\sigma_i x = e = \sigma_i e$ car $x \notin U(e)$, puisque N_{i-1} a été obtenu par SIMPL. Donc $\xi_i \sigma_i x = \xi_i \sigma_i e$, ce qui achève de montrer que ξ_{i-1} est unificateur de $N_{i-1} = \{ \langle x, e \rangle \} \cup \bar{N}_i$. On obtient donc pour $i=0$: $\xi_0 \in U(N_0) = U(N)$.

Réiproquement, soit $\sigma \in U(N_0) = U(N)$.

On montre que, pour tout $i \leq p$ $\exists n_i \in U(N_i)$ $\sigma = n_i \sigma_i \sigma_{i-1} \dots \sigma_1$, par récurrence sur i .

i) $i = 0$ on prend $n_0 = \sigma$

ii) on suppose que la propriété est vraie pour $i-1$.

Soit σ_i la paire sélectionnée dans N_{i-1} :

$$N_{i-1} = \{\sigma_i\} \cup \bar{N}_i \text{ avec } \sigma_i = \langle x, e \rangle.$$

Par hypothèse $n_{i-1} \in U(\bar{N}_{i-1})$, et donc

$$n_{i-1}x = n_{i-1}e = n_{i-1}\sigma_i x.$$

Considérons

$$n_i = n_{i-1} \setminus V - \{x\}.$$

On a

$$n_{i-1}x = n_i \sigma_i x \quad \text{car} \quad x \notin V(\sigma_i x)$$

$$\text{et} \quad \forall y \neq x \quad n_{i-1}y = n_i y = n_i \sigma_i y \quad \text{car} \quad \sigma_i y = y.$$

$$\text{Donc} \quad n_{i-1} = n_i \sigma_i.$$

$$n_{i-1} \in U(N_{i-1}) \Rightarrow n_i \sigma_i \in U(\bar{N}_i) \Rightarrow n_i \in U(\sigma_i \bar{N}_i) = U(N_i)$$

par le lemme 5.20. Ceci termine la récurrence.

Pour $i=p$, on obtient $\sigma = n_p \xi_0$, i.e. $\xi_0 \leq \sigma$.

$(U(N_i)) \neq \emptyset$ pour tout i impose que l'arrêt se fasse par succès, c'est à dire avec $N_p = \emptyset$.

Donc $\xi_0 \in U(N) : \xi_0$ est un unificateur principal de N . \square

Remarquons que par le lemme 5.6, le résultat ξ_0 de l'algorithme d'unification est unique modulo une permutation.

Exemple :

$$N = \{<F(x, F(G(z), F(u, y))), F(G(y), F(y, F(z, G(u))))>\}$$

$$N_0 = \{<x, G(y)>, <y, G(z)>, <u, z>, <y, G(u)>\}$$

$$\sigma_1 = <x, G(y)>$$

$$N_1 = \{<y, G(z)>, <u, z>, <y, G(u)>\}$$

$$\sigma_2 = <y, G(z)>$$

$$N_2 = \{<z, u>, <u, z>\}$$

$$\sigma_3 = <z, u>$$

$$N_3 = \emptyset.$$

Avec les notations de la preuve :

$$\xi_2 = \{<z, u>\}$$

$$\xi_1 = \{<y, G(u)>, <z, u>\}$$

$$\xi_0 = \{<x, G(G(u))>, <y, G(u)>, <z, u>\}.$$

Avec $\sigma = \{<x, G(G(z))>, <y, G(z)>, <u, z>\}$, on obtiendrait :

$$\eta_0 = \sigma$$

$$\eta_1 = \{<y, G(z)>, <u, z>\}$$

$$\eta_2 = \{<u, z>\}$$

$$\eta_3 = \eta_2.$$

Remarquons que σ est, comme ξ_0 , un unificateur principal de N .

Remarques :

- 1) L'existence d'un unificateur principal est une propriété plus forte que l'existence du sup dans \hat{T} . En particulier, remarquons qu'une substitution unifiant deux termes e_1 et e_2 en une instance la plus générale e n'est pas forcément un unificateur principal, même si on réduit son domaine

aux variables apparaissant dans e_1 et e_2 . Par exemple, considérons

$\left\{ \begin{array}{l} e_1 = x \\ e_2 = F(z) \end{array} \right.$

minimale de e_1 et e_2 ; $e = F(y)$ est une instance commune

$e = \sigma e_1 = \sigma e_2$ avec $\sigma = \begin{cases} <x, F(y)> \\ <z, y> \end{cases}$. Pourtant σ est strictement moins générale que l'unificateur principal $\rho = \{<x, F(z)>\}$ produit par l'algorithme d'unification :

$$\sigma = n\rho \text{ avec } n = \{<z, y>\}$$

$$\text{mais } \exists n' \rho = n'\sigma, \text{ donc } \rho < \sigma.$$

Ce qui importe ici, c'est que l'unificateur principal nous donne une instance commune minimale de e_1 et e_2 qui ne possède pas de variables en dehors de $V(e_1) \cup V(e_2)$.

D'un autre côté, le théorème 14 ne nous permet de conclure à l'existence de sups que pour des ensembles finis de termes, alors que le théorème 13 nous donne l'existence d'un sup même pour des ensembles infinis, à condition qu'ils soient majorés.

2) Comparaison avec l'algorithme original de Robinson. On peut comparer nos unificandes aux "disagreement sets" de Robinson [RJA1]. Il y a pourtant deux différences mineures.

Tout d'abord, le test de l'étape 4 de SIMPL est effectué lors de la simplification, et non lors de la sélection d'une composante de substitution. Ensuite, on choisit cette composante librement, si bien que la descente récursive peut se faire dans une direction quelconque, pas seulement de gauche à droite. Par exemple, si on cherche à unifier

$$e = F(F_1, x)$$

et $e' = F(F_2, G(x))$, où F_1 et F_2 sont des expressions

unifiables très grosses, alors on obtient directement $N_0 = "T"$.

De toutes façons ces deux algorithmes sont peu efficaces, car ils font des recopies coûteuses inutiles. Par exemple, si on cherche à unifier les termes

$$e_1 = F(x, x, \dots, x)$$

$$\text{et } e_2 = F(F, y, \dots, y), \text{ où } F \text{ est un terme très grand,}$$

alors on fera inutilement les comparaisons $\langle E, F \rangle$ pour les $\alpha(F)-2$ derniers arguments de F . Il y a perte de place, puisqu'on fait des copies du terme E , et de temps.

Le problème de l'espace peut être résolu grâce à une implémentation par pointeurs, comme celle proposée dans un algorithme amélioré de Robinson [RJA2]. Mais ceci ne résout nullement le problème du temps. En fait, le temps requis est exponentiel en fonction de la taille des termes d'entrée, ne serait-ce que pour effectuer le test d'occurrence, comme on peut le constater aisément sur l'exemple :

$$\begin{cases} e_3 = F(x_1, x_2, x_3, \dots, x_n) \\ e_4 = F(G(x_0, x_0), G(x_1, x_1), \dots, G(x_{n-1}, x_{n-1})). \end{cases}$$

Pour remédier à ce problème, Baxter [BL1] a proposé un algorithme d'unification qui consiste en deux phases :

- une phase "transformationnelle" qui construit un unificateur potentiel.
- une phase "de tri", qui teste une condition de circularité qui équivaut au test d'occurrence précédent.

Malheureusement, la phase transformationnelle de l'algorithme de Baxter effectue des opérations inutiles, comme on peut le constater sur l'exemple $\langle e_1, e_2 \rangle$ ci-dessus. Plus grave, cette phase peut ne pas terminer, par exemple avec :

$$\begin{cases} e_5 = F(x, y, x) \\ e_6 = F(G(y), G(x), y). \end{cases}$$

Nous allons proposer en 5.7 un algorithme d'unification rapide, qui utilise une structure de données à partage par pointeurs, et qui effectue également une phase de tri terminale.

La phase transformationnelle est basée sur la construction de classes d'équivalences de termes, représentées à la Fisher-Galler [FG]. Cet algorithme a été découvert en collaboration avec Gilles Kahn.

L'algorithme d'unification rapide sera présenté, prouvé et analysé en 5.7. Le prochain paragraphe présente des propriétés de base de certaines congruences de termes, qui sont nécessaires pour sa bonne compréhension.

5.6. Congruences rationnelles

5.6.1. Equivalences rationnelles

Nous allons nous intéresser à des relations d'équivalence définies sur l'ensemble des termes T . Toutes les relations d'équivalence \sim que nous considérerons auront la propriété de *finitude* suivante :

$$[x]_{\sim} = \{x\} \text{ presque partout dans } V.$$

(On dénote par $[e]_{\sim}$ l'ensemble $\{e' \in T \mid e \sim e'\}$).

On note $\sim_1 \subset \sim_2$ pour $\forall e, e' \quad e \sim_1 e' \Rightarrow e \sim_2 e'$, et on dit que \sim_1 est plus fine que \sim_2 .

On appelle *congruence* une équivalence \sim compatible avec la structure de magma, c'est à dire telle que :

$$e_i \sim e'_i \quad 1 \leq i \leq n \Rightarrow F(e_1, \dots, e_n) \sim F(e'_1, \dots, e'_n),$$

Réciiproquement, une équivalence \sim est dite *simplifiable* ssi :

$$F(e_1, \dots, e_n) \sim F(e'_1, \dots, e'_n) \Rightarrow e_i \sim e'_i \quad 1 \leq i \leq n.$$

Une équivalence \sim est dite *cohérente* ssi

$$F(e_1, \dots, e_n) \neq G(e'_1, \dots, e'_p) \text{ pour } F \neq G.$$

Remarquons que la congruence engendrée par une équivalence \sim est finie (resp. simplifiable, cohérente) ssi \sim est finie (resp. simplifiable, cohérente).

On appelle *équivalence rationnelle* une équivalence finie simplifiable et cohérente.

Soit \sim une équivalence rationnelle. On définit une relation \rightarrow_{\sim} dans T/\sim par :

$$[Fe_1 \dots e_n]_{\sim} \rightarrow_{\sim} [e_i]_{\sim} \quad 1 \leq i \leq n.$$

On dit que $[e_i]_{\sim}$ est sous classe immédiate de $[e]_{\sim}$ dans \sim .

Soit $\overset{*}{\rightarrow}_{\sim}$ la fermeture réflexive et transitive de \rightarrow_{\sim} .

$SC(e, \sim) = \{[e']_\sim \mid [e]_\sim \xrightarrow{*} [e']_\sim\}$ est l'ensemble des sous-classes de $[e]_\sim$ dans \sim .

Remarque : Si \sim est une congruence rationnelle, alors \rightarrow_\sim ne dépend pas de \sim et est donc une relation dans $P(T)$, car la connaissance de $[e]_\sim$ en tant que partie de T suffit à déterminer de manière unique les classes $[e_i]_\sim$ telles que $[e]_\sim \rightarrow_\sim [e_i]_\sim$. Dans ce cas, les sous classes d'une classe $[e]_\sim$ ne dépendent pas de la congruence \sim particulière utilisée pour la définir.

Lemme 5.21

Soit \sim une équivalence rationnelle. Pour tout e dans T , $SC(e, \sim)$ est fini.

Démonstration :

Supposons qu'il existe une classe $[e_1]_\sim$ admettant un nombre infini de sous-classes dans \sim .

Toute classe n'a qu'un nombre fini de sous-classes immédiates, à cause de la condition de cohérence, et l'arité d'un symbole de fonction étant finie. Par le lemme de König, il existe donc une suite infinie de classes distinctes :

$$[e_1]_\sim \rightarrow_\sim [e_2]_\sim \rightarrow_\sim \cdots \rightarrow_\sim [e_n]_\sim \rightarrow_\sim \cdots$$

Définissons : $\theta_i = \min \{\theta(e) \mid e \sim e_i\}$. On a :

$$\theta_i > 0 \Rightarrow \theta_{i+1} < \theta_i. \text{ En effet, si } \theta_i > 0,$$

soit $e \sim e_i$ avec $\theta(e) = \theta_i$: $e = F(\hat{e}_1, \dots, \hat{e}_n)$. Il existe k tel que $\hat{e}_k \sim e_{i+1}$, avec $\theta(\hat{e}_k) = \theta(e)-1$.

On peut donc sélectionner une sous-suite infinie de classes distinctes :

$$[e_{k_1}]_\sim \xrightarrow{*} [e_{k_2}]_\sim \xrightarrow{*} [e_{k_3}]_\sim \rightarrow_\sim \cdots$$

avec $\theta_{k_i} = 0$. Cela détermine donc une suite de variables distinctes v_1, v_2, \dots , avec $[v_i]_\sim = \{v_i\} \neq \emptyset$. Cela contredit la propriété de finitude de \sim . \square

Soit \sim une équivalence rationnelle et $e \in T$. On définit $\chi(e, \sim) \in \mathbb{N} \cup \{\infty\}$ comme la longueur de la plus longue \rightarrow_{\sim} -chaîne à partir de $[e]_{\sim}$. On dit que \sim est acyclique en e si $\chi(e, \sim) < \infty$. On dit que \sim est acyclique si elle est acyclique en tout $e \in T$.

Lemme 5.22 : Soit \sim une équivalence rationnelle, $\hat{\sim}$ sa fermeture par congruence, \sim est acyclique en e si et seulement si $\hat{\sim}$ est acyclique en e .

Démonstration :

Comme $\hat{\sim} \subset \sim$, il est évident que si \sim est cyclique, alors $\hat{\sim}$ l'est aussi.

Réciprocement, définissons

$$\sim' = \sim \cup \{<Fe_1 \dots e_n, Fe'_1 \dots e'_n> \mid e_i \sim e'_i \quad 1 \leq i \leq n\}.$$

On a bien sûr $\hat{\sim} = \sim \cup \sim' \cup (\sim')' \cup \dots$, et il nous suffit donc de prouver que \sim' cyclique entraîne \sim cyclique. Si \sim' est cyclique en e'_1 , alors il existe une suite e'_1, \dots, e'_p telle que :

$$[e'_1]_{\sim'}, \rightarrow_{\sim'}, [e'_2]_{\sim'}, \rightarrow \dots \rightarrow, [e'_p]_{\sim'}, \rightarrow_{\sim'}, [e'_1]_{\sim'}, \dots$$

Nous allons construire un cycle analogue dans T/\sim comme suit..

On définit une suite e_1, \dots, e_p, e_{p+1} par :

i) $e_1 = e'_1$.

ii) Soit e_i déjà construit, avec $e_i \in [e'_i]_{\sim'}$.

Proposition : $\exists \bar{e}_i \in [e_i]_{\sim}$ tel que $\theta(\bar{e}_i) > 0$. En effet,

$[e_i]_{\sim} \subset V \implies [\bar{e}_i]_{\sim} = [e'_i]_{\sim'} \subset V$ ce qui est contraire à l'hypothèse que e'_i possède une sous-classe dans \sim' . Soit $\bar{e}_i = F\bar{e}_i^1 \dots \bar{e}_i^m$, et soit k tel que $e'_{i+1} \sim' \bar{e}_i^k$. On choisit $e_{i+1} = \bar{e}_i^k$ dans $[e'_{i+1}]_{\sim'}$. (Si $i=p$, remplacer $i+1$ par 1).

Par construction, on a bien :

$$[e_1]_{\sim} \rightarrow_{\sim} [e_2]_{\sim} \rightarrow \cdots \rightarrow_{\sim} [e_p]_{\sim} \rightarrow_{\sim} [e_{p+1}]_{\sim}.$$

Si $e_1 \sim e_{p+1}$, c'est terminé. Sinon, on a $e_1 \sim' e_{p+1}$, et par définition de \sim' cela implique que $[e_{p+1}]_{\sim} \rightarrow_{\sim} [e_2]_{\sim}$, ce qui prouve bien que \sim est cyclique. \square

Exemple illustrant la preuve précédente.

Soit \sim défini par $x \sim y \sim G(z)$ et $z \sim F(y)$.

\sim' est cyclique en $F(x)$, avec la chaîne de longueur $p=2$:

$$e'_1 = F(x) \xrightarrow{\sim'} e'_2 = x.$$

\sim est cyclique en $F(x)$, mais avec la chaîne de longueur 3 :

$$e_1 = F(x) \rightarrow e_2 = x \xrightarrow{\sim} e_3 = z. \square$$

5.6.2. Congruences d'unification

Soit σ une substitution quelconque. On appelle *congruence d'unification* de σ la congruence \sim_{σ} définie par :

$$e \sim_{\sigma} e' \Leftrightarrow \sigma e = \sigma e'.$$

Les substitutions étant des morphismes identité presque partout, il est clair que toute congruence d'unification est rationnelle.

Lemme 5.23

$$\sigma \leq \rho \implies \sim_{\sigma} \subseteq \sim_{\rho} \quad (\text{i.e. } \sim_{\sigma} \text{ est plus fine que } \sim_{\rho}).$$

Démonstration Par définition :

$$\sigma \leq \rho \Leftrightarrow \exists \eta \quad \rho = \eta \sigma \quad \text{d'où :}$$

$$e_1 \sim_{\sigma} e_2 \implies \sigma e_1 = \sigma e_2 \implies \rho e_1 = \rho e_2 \implies e_1 \sim_{\rho} e_2. \square$$

La réciproque de ce lemme n'est pas vraie. En effet, certaines substitutions "circulaires" ne concourent à aucune unification.

Par exemple, avec $\sigma_1 : \sigma_1 x = Fx$

et $\sigma_2 : \sigma_2 x = Gx$

on a $\sim_{\sigma_1} = \sim_{\sigma_2} = \text{Identité.}$

Une réciproque plus faible sera donnée au lemme 5.27.

Lemme 5.24 Pour toute substitution σ , \sim_σ est une congruence rationnelle acyclique.

Démonstration : Soit $e \in T$ quelconque.

$[e]_\sim \xrightarrow[\sigma]{} [e']_\sim$ entraîne $e \sim_\sigma F(\dots, e', \dots)$, d'où $\sigma e = \sigma F(\dots, e', \dots) = F(\dots, \sigma e', \dots)$, et donc $\theta(\sigma e) > \theta(e')$. La longueur d'une \sim_σ -chaîne issue de $[e]_\sim$ est donc bornée par $\theta(\sigma e)$, ce qui prouve que \sim_σ est une congruence rationnelle acyclique. \square

Définition : Soit \sim une équivalence rationnelle acyclique. Pour tout x dans V , on définit $\tilde{x} \in T$ comme un représentant de la classe $[x]_\sim$, avec les propriétés :

$$\text{a)} \quad \tilde{x} \in [x]_\sim$$

$$\text{b)} \quad \tilde{x} \in V \Leftrightarrow [x]_\sim \subset V.$$

C'est à dire que $\theta(\tilde{x}) > 0$ s'il existe un tel terme dans $[x]_\sim$.

On définit maintenant la substitution σ_\sim par :

$$\sigma_\sim x = \begin{cases} \tilde{x} & \text{si } \tilde{x} \in V \\ \sigma_\sim \tilde{x} & \text{sinon.} \end{cases}$$

Cette définition a un sens, car si $\tilde{x} \notin V$:

$$\tilde{x} = Fe_1 \cdots e_n \text{ alors } \chi(x, \sim) > \chi(e_i, \sim) \quad 1 \leq i \leq n,$$

et $\sigma_\sim e$ est donc bien défini par récurrence sur $\chi(e, \sim)$.

Lemme 5.25 Soit \sim une équivalence rationnelle acyclique, $\hat{\sim}$ sa fermeture par congruence.

$$\forall e \in T \quad \sigma_{\sim} e \hat{\sim} e.$$

Démonstration : par récurrence sur $\chi(e, \sim)$.

Si $\chi(e, \sim) = 0$, alors :

-soit $[e]_{\sim} \subset V$, et alors par définition $\sigma_{\sim} e = \tilde{e} \sim e$.

-soit e est une constante (symbole de fonction d'arité 0)

et alors $\sigma_{\sim} e = e$.

Si $\chi(e, \sim) \neq 0$, alors $e \sim F(\dots, e_i, \dots)$. Comme pour tout i , $\chi(e_i, \sim) < \chi(e, \sim)$, on a $\sigma_{\sim} e_i \hat{\sim} e_i$ par hypothèse de récurrence. Donc

$$\begin{aligned} F(\dots, e_i, \dots) &\hat{\sim} F(\dots, \sigma_{\sim} e_i, \dots) \text{ car } \hat{\sim} \text{ est une congruence} \\ &= \sigma_{\sim} F(\dots, e_i, \dots) \\ &= \sigma_{\sim} e. \square \end{aligned}$$

Lemme 5.26 Soit \sim une équivalence rationnelle acyclique, $\hat{\sim}$ sa fermeture par congruence.

$$\sim_{\sigma_{\sim}} = \hat{\sim}.$$

Démonstration :

Si $\sigma_{\sim} e = \sigma_{\sim} e'$, alors $e \hat{\sim} e'$ par le lemme 5.25.

Réiprocalement, montrons que $e \sim e'$ implique $\sigma_{\sim} e = \sigma_{\sim} e'$.

La preuve se fait par récurrence sur $\chi(e, \sim) = \chi(e', \sim)$.

Si $\chi(e, \sim) = 0$, alors :

-soit $[e]_{\sim} \subset V$, et alors $\sigma_{\sim} e = \sigma_{\sim} e' = \tilde{e}$.

-soit il existe une constante A dans $[e]_{\sim}$, et alors

-si $e = A$ $\sigma_{\sim} e = A$

-si $e = x$ $\sigma_{\sim} e = \sigma_{\sim} \tilde{x} = \sigma_{\sim} A = A$.

Si $\chi(e, \sim) > 0$, alors :

-si $\theta(e) > 0$ et $\theta(e') > 0$, $e \sim e'$ implique qu'il existe

$F \in \mathcal{C}$, tel que :

$$\begin{cases} e = F(\dots, e_i, \dots) \\ e' = F(\dots, e'_i, \dots) \end{cases} \quad \text{avec } e_i \sim e'_i$$

et $\sigma_{\sim} e = F(\dots, \sigma_{\sim} e_i, \dots)$

$$= F(\dots, \sigma_{\sim} e'_i, \dots) \quad \text{par hypothèse de récurrence,}$$

$$\text{car } \chi(e_i, \sim) < \chi(e, \sim)$$

$$= \sigma_{\sim} e'$$

-si $\theta(e) = 0$, remplacer ci-dessus e par \tilde{e} .

-si $\theta(e') = 0$, remplacer ci-dessus e' par \tilde{e}' .

On a donc $\sim \subset \sim_{\sigma_{\sim}}$. $\tilde{\sim}$ étant la plus petite congruence contenant \sim ,
on a donc bien aussi $\tilde{\sim} \subset \sim_{\sigma_{\sim}}$. \square

Théorème 15.

Une congruence rationnelle est une congruence d'unification si et seulement si elle est acyclique.

Démonstration : Directement du lemme 5.24 et du lemme 5.26. \square

Lemme 5.27 Pour toute équivalence rationnelle acyclique \sim , pour toute substitution ρ :

$$\sim \subset \sim_{\rho} \implies \sigma_{\sim} \leq \rho.$$

Démonstration :

Par le lemme 5.25 : $\forall x \in V \quad \sigma_{\sim} x \tilde{\sim} x$.

Pour tout ρ tel que $\sim \subset \sim_{\rho}$, on a $\tilde{\sim} \subset \sim_{\rho}$, et donc

$$\sigma_{\sim} x \sim_{\rho} x$$

donc $\rho\sigma_{\sim}x = \rho x$

i.e. $\rho = \rho\sigma_{\sim}$

et $\sigma_{\sim} \leq \rho$ par le lemme 5.5. \square

Remarque : S'il est vrai que $\sigma_{\sim}x \sim x$, il n'est pas vrai en général que $\rho x \sim_p x$. Par exemple si $x \in V(\rho x)$ et $\rho x \neq x$, on aura $x \not\sim_p \rho x$ pour toute congruence rationnelle acyclique \sim . Un autre exemple est ρ défini par $\rho x = y$, $\rho y = x$, car $\sim_{\rho} = \text{Identité}$.

Le lemme 5.27 nous indique en particulier que σ_{\sim} est la substitution la plus générale dont la congruence d'unification égale $\hat{\sim}$.

σ_{\sim} pourrait être défini différemment. L'avantage de notre définition est que l'on peut calculer $\sigma_{\sim}x$ récursivement à partir d'une équivalence \sim complétée par simplification "vers le bas", sans avoir à compléter \sim par congruence "vers le haut".

Théorème 16 d'unification

Soit E un ensemble fini de termes, \sim_F la plus petite équivalence contenant les paires de termes dans E .

Soit \sim la fermeture simplifiable de \sim_F . Si \sim est cohérente et acyclique, alors σ_{\sim} est un unificateur principal de E , sinon E n'est pas unifiable.

Démonstration :

E étant un ensemble fini, la relation d'équivalence \sim_E remplit la condition de finitude : $[x]_{\sim_E} = \{x\}$ presque partout. Soit \sim la fermeture simplifiable de \sim_F . Si \sim n'est pas cohérente, il n'y a pas de congruence rationnelle contenant \sim_F , et E n'est donc pas unifiable. Si \sim est cohérente, c'est l'équivalence rationnelle la plus fine contenant \sim_F . Si elle est cyclique, alors sa

fermeture congruente $\hat{\sim}$ est cyclique par le lemme 5.22, et, $\hat{\sim}$ étant la congruence rationnelle la plus fine contenant \sim_E , E n'est pas unifiable par le lemme 5.24. Si elle est acyclique, alors σ_\sim unifie E par le lemme 5.26.

Pour tout unificateur ρ de E , \sim_ρ est une congruence rationnelle acyclique contenant \sim_E . Donc $\sim \subseteq \sim_\rho$, et $\sigma_\sim \leq \rho$ par le lemme 5.27, σ_\sim est donc un unificateur principal de E . \square

Le théorème 16 suggère un algorithme d'unification. Nous allons voir au prochain paragraphe que \sim est calculable de façon finie et rapide à partir de \sim_E , et que l'on sait tester rapidement si \sim est cyclique ou pas. De plus l'information dans \sim suffit à calculer σ_\sim , et nous n'aurons donc à aucun moment à construire explicitement une fermeture par congruence, ce qui correspondrait du point de vue algorithmique à effectuer des recopies.

5.7. Un algorithme d'unification rapide.

Nous allons maintenant présenter un algorithme d'unification rapide, basé sur la construction du théorème 16.

Soit E un ensemble fini de termes à unifier. La relation d'équivalence induite \sim_E est représentée sous la forme d'un unificande N , ensemble fini de paires de termes. L'algorithme PATIO donné ci-dessous génère l'équivalence rationnelle \sim la plus fine, si elle existe, qui contienne \sim_E . En cas de succès, l'algorithme CYCLE teste si \sim est acyclique. Dans ce cas, σ_\sim est l'unificateur principal de E .

5.7.1. Structure de données utilisée

Comme l'algorithme explore récursivement les sous-termes des termes de la partition, il est pratique d'avoir une notation pour ces sous-termes permettant de décider rapidement si deux tels termes sont dans la même classe. On peut pour ce faire utiliser des variables auxiliaires, et ne représenter

un terme non variable que par son premier niveau, les sous-termes internes étant "liés" aux variables correspondantes par l'intermédiaire de la partition. Remarquons qu'il y a plusieurs manières d'effectuer une telle décomposition. Par exemple, le terme $F(A,A)$ peut être représenté par la variable x dans la partition :

$$\{\{x, F(u, v)\}, \{u, v, A\}\},$$

ou dans la partition :

$\{\{x, F(y, y)\}, \{y, A\}\}.$ (d'ailleurs remarquons que $F(u, v) \sim F(u, u)$ dans la congruence engendrée par la première partition).

Ces deux partitions représentent deux formes normales extrêmes de décomposition :

- 1) décomposition totalement séparée, ne contenant que des termes $F(x_1, \dots, x_n)$ avec les x_i distincts
- 2) décomposition à partage maximal.

Les algorithmes donnés ci-dessous peuvent utiliser l'une ou l'autre des décompositions, et il y a donc intérêt à utiliser la deuxième, qui peut être calculée en temps linéaire en la taille du terme. De toute manière, cette décomposition peut être effectuée initialement, et toute les manipulations ultérieures utilisant l'unification effectuées sur une représentation interne décomposée.

Les partitions représentées ont donc la propriété que chaque classe $[e]$ comporte :

- au moins une variable
- au plus un terme non variable, de taille 1.

Dénotons T_1 l'ensemble des termes de taille 1 :

$$T_1 = \{t \in T \mid \theta(t) = 1\}.$$

On appelle *environnement* une application ρ d'un sous ensemble fini $\mathcal{D}(\rho)$ de l' dans T_1 . Soit V un ensemble fini de variables tel que :

$$\mathcal{D}(\rho) \cup I(\rho) \subset V,$$

avec $I(\rho) = \{y_i \mid \exists x \in \mathcal{D}(\rho) \text{ et } \rho x = Fy_1 \dots y_n\}.$

Soit ρ et V donnés. Toute partition cohérente \sim compatible avec ρ , c'est à dire telle que $\forall x \in D(\rho) \quad x \sim \rho x$, et telle que $[x]_\sim = \{x\} \quad \forall x \notin V$, peut être représentée par une application de V dans V :

$$\begin{aligned} x &\leftrightarrow \bar{x} \\ \text{telle que : i)} \quad x \sim y &\implies \bar{x} = \bar{y} \\ \text{ii)} \quad \bar{x} &\sim x \\ \text{iii)} \quad x \in D(\rho) &\implies \bar{x} \in D(\rho) \text{ et } \rho x \text{ et } \rho y \text{ ont même} \end{aligned}$$

symbole de fonction. La donnée de cette application définit naturellement \sim comme l'équivalence engendrée par $x \sim \bar{x} \quad \forall x \in V$ et $x \sim \rho x \quad \forall x \in D(\rho)$.

En définissant :

$$\tilde{x} = \begin{cases} \rho \bar{x} & \text{si } x \in V \text{ et } \bar{x} \in D(\rho) \\ \bar{x} & \text{si } x \in V \text{ et } \bar{x} \notin D(\rho) \\ x & \text{sinon} \end{cases}$$

on vérifie que \tilde{x} est un représentant canonique de $[x]_\sim$ ayant les propriétés requises en 5.6.

Nous avons donc maintenant un moyen uniforme de représenter les termes et les partitions cohérentes. La représentation concrète de ρ et de \bar{x} sera décrite lors de l'analyse de l'algorithme.

5.7.2. Description algorithmique de RATIO et de CYCLE

On suppose donnés V et ρ comme au paragraphe précédent. Ces données resteront inchangées : on n'introduira pas de variables supplémentaires, et ρ est un tableau utilisé seulement en lecture.

Un unificande N sera maintenant un sous-ensemble fini de V^2 . Initialement N contient les noms des paires de termes que l'on désire unifier.

A chaque variable x de V on associe une variable \bar{x} de V . Initialement $\bar{x} = x$ pour tout x dans V .

RATIO(N) ;

Etape 1 : Si $N = \emptyset$ alors arrêt succès.

Sinon, soit $N = \{<x,y>\} \cup \bar{N}$; $N \leftarrow \bar{N}$;

Si $\bar{x} = \bar{y}$ alors retourner à l'étape 1 ;

Etape 2 : Si $\bar{x} \in D(\rho)$ et $\bar{y} \in D(\rho)$ alors

[soit $\rho\bar{x} = Fx_1 \dots x_n$, $\rho\bar{y} = Gy_1 \dots y_p$;

si $F \neq G$, alors arrêt échec (retourner T),

sinon $N \leftarrow N \cup \{<x_i, y_j> \mid 1 \leq i \leq n\}$;

Réunir les classes de x et y , avec comme représentant \bar{x} si

$\bar{x} \in D(\rho)$, \bar{y} sinon.

Aller à l'étape 1 ;

Fin de RATIO. \square

CYCLE

Soit $\bar{V} = \{\bar{x} \mid x \in V\}$ l'ensemble des représentants de la classe d'équivalence rationnelle construite par RATIO, en cas de succès.

On définit la relation \rightarrow dans V par :

$\bar{x} \rightarrow \bar{y}_i$ ssi $x \in D(\rho)$ et $\rho x = Fy_1 \dots y_n$.

Si le graphe $\langle \bar{V}, \rightarrow \rangle$ est acyclique, alors arrêt succès, sinon arrêt échec. \square

Exemple :

Soit à unifier :

$$e_1 = F(x, F(u, x))$$

$$\text{et } e_2 = F(F(y, A), F(z, F(B, z))).$$

On décompose e_1 et e_2 en respectivement w_1 et w_2 , dans l'environnement :

$$\rho = \{ \langle w_1, Fxw_3 \rangle, \langle w_2, Fw_4w_5 \rangle, \langle w_3, Fux \rangle, \\ \langle w_4, Fyw_7 \rangle, \langle w_5, Fzw_6 \rangle, \langle w_6, Fw_8z \rangle, \\ \langle w_7, A \rangle, \langle w_8, B \rangle \}.$$

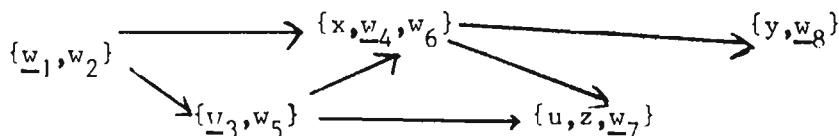
On a donc $V = \{x, y, z, u, w_1, w_2, \dots, w_8\}$ et on prend initialement $\bar{v} = v$ pour tout v dans V , et $N = \{ \langle w_1, w_2 \rangle \}$.

Exécutions RATIO(N) :

$$\begin{array}{ll} N \leftarrow \{ \langle x, w_4 \rangle, \langle w_3, w_5 \rangle \} & \bar{w}_2 \leftarrow w_1 \\ N \leftarrow \{ \langle w_3, w_5 \rangle \} & \bar{x} \leftarrow w_4 \\ N \leftarrow \{ \langle u, z \rangle, \langle x, w_6 \rangle \} & \bar{w}_5 \leftarrow w_3 \\ N \leftarrow \{ \langle x, w_6 \rangle \} & \bar{u} \leftarrow z \\ N \leftarrow \{ \langle y, w_8 \rangle, \langle w_7, z \rangle \} & \bar{w}_6 \leftarrow w_4 \\ N \leftarrow \{ \langle w_7, z \rangle \} & \bar{y} \leftarrow w_8 \\ N \leftarrow \emptyset & \bar{z}, \bar{u} \leftarrow w_7 \end{array}$$

Arrêt succès.

La partition finale de V obtenue est donc :



(on a souligné le représentant canonique \bar{x} de chaque classe $[x]$).

L'algorithme CYCLE vérifie alors que le graphe obtenu n'est pas cyclique. En définissant \tilde{x} , pour tout x de V , par :

$$\tilde{x} = \begin{cases} \bar{x} & \text{si } \bar{x} \notin D(\rho) \\ \rho \bar{x} & \text{sinon} \end{cases}$$

la méthode du 5.6 nous permet d'obtenir l'unificateur :

$$\sigma = \{\langle x, F(B, A) \rangle, \langle u, A \rangle, \langle z, A \rangle, \langle y, B \rangle\}.$$

Remarquons que dans l'exécution de RATIO, nous avons supposé implicitement que l'unificande N était représenté par une liste, accédée comme pile "FIFO". En fait l'algorithme peut sélectionner librement dans N , cette hypothèse n'est nécessaire ni à la correction ni à la terminaison de l'algorithme.

5.7.3 Preuve de correction et analyse des algorithmes

Soit \sim_E l'équivalence définie par l'unificande initial N . Il est clair que RATIO(N) construit la fermeture simplifiable de \sim_E , en testant la cohérence à chaque étape. Si RATIO termine en échec, il n'existe donc pas d'équivalence rationnelle contenant \sim_E , sinon l'équivalence construite est la telle équivalence la plus fine. Il nous reste donc seulement à montrer que RATIO(N) termine pour tout unificande N .

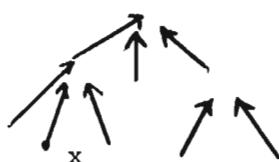
Soit $n = |V| = |\sim|$ initialement, $p = |N|$ initialement. Chaque passage à l'étape 1 fait décroître $|N|$, chaque passage à l'étape 2 fait décroître $|\sim|$. RATIO termine donc toujours, par récurrence sur $w \cdot |\sim| + |N|$.

Pour évaluer l'efficacité de RATIO, il nous faut analyser plus précisément le nombre des passages à chaque étape. Le pire des cas est naturellement en cas de sortie par succès. A ce moment, on est passé au plus $n-1$ fois à l'étape 2. Soit k l'arité maximale des symboles de fonctions présents dans l'environnement ρ .

A chaque passage à l'étape 2, on a placé dans N au maximum k nouvelles paires. Le nombre m de passages à l'étape 1 étant le nombre total de paires placées dans N , il est donc au plus de $kn+p$, et donc linéaire en la taille du problème. Si on cherche à unifier deux termes e_1 et e_2 , alors leur décomposition

produira au plus $|D(p)| = \theta(e_1) + \theta(e_2)$, et on peut donc prendre $n = \theta(e_1) + \theta(e_2) + v(e_1) + v(e_2)$, et $p=1$.

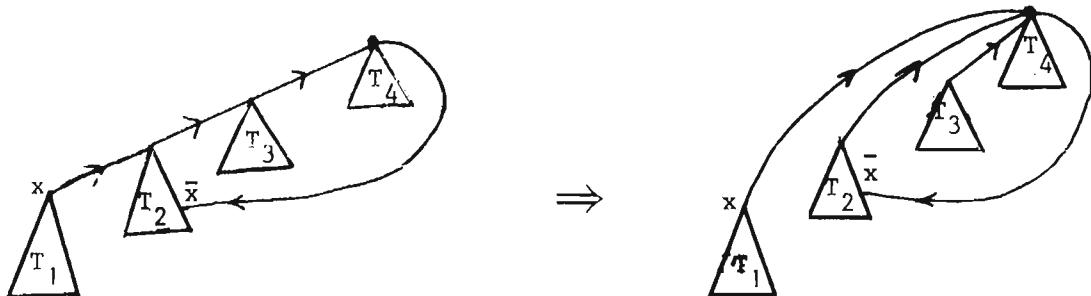
Pour déterminer précisément la complexité de RATIO , il nous faut décrire plus en détail la structure de données utilisée pour représenter les partitions de V. Cette structure doit permettre d'accéder rapidement au représentant \bar{x} , et de faire rapidement l'opération "réunir les classes de x et y ". Une telle structure a été étudiée pour le "set union algorithm" de Caller et Fisher [GF1], où les classes sont représentées par des structures de données arborescentes, le sommet de chaque arborescence étant choisi comme représentant de la classe :



L'opération de réunion est réalisée en faisant pointer l'arbre de la classe ayant le moins d'éléments vers le sommet de l'autre arbre. Pour ce faire, on maintient des compteurs, associés avec chaque variable. Malheureusement, cette règle ne nous permet pas de garantir que le sommet de l'arbre soit dans $D(p)$ lorsque cela est possible. Il faut donc associer au sommet de l'arbre un pointeur en arrière vers l'élément \bar{x} . (Ceci ne nécessite pas de champ supplémentaire, le champ d'adressage des sommets d'arbre étant libre. Il suffit donc de repérer les sommets avec une variable booléenne).

L'opération de recherche de \bar{x} est effectuée par cheminement vers le sommet de l'arbre, et une deuxième passe fait la recopie des pointeurs le long

du chemin parcouru, afin de rendre l'arbre le plus plat possible :



L'analyse complète de ces algorithmes est donnée par Tarjan [TA1].

Le nombre d'opérations de recherche de \bar{x} étant $2m \leq kn+p$, et le nombre de réunions étant $\leq n-1$, le résultat de Tarjan nous permet de conclure que le temps d'exécution de RATIO est de l'ordre de $nG(n)$, où $G(n) \leq 3$ en pratique,

Nous pouvons donc conclure que RATIO est un algorithme pratiquement linéaire. L'espace nécessaire est lui aussi linéaire ; les coefficients de proportionnalité peuvent être considérablement réduits par des méthodes de compaction dans lesquelles nous ne rentrerons pas. Pour l'implémentation pratique de RATIO, il peut se révéler plus efficace en moyenne de ne pas utiliser la règle de réunion du plus petit arbre au plus grand. Cela permet de ne pas garder de compteurs, et d'utiliser directement \bar{x} comme racine. La complexité théorique de l'algorithme serait alors de $n \log(n)$.

L'algorithme CYCLE cherche l'existence d'un cycle dans le graphe déterminé par ρ/\sim . On peut l'implémenter en recherchant un cycle dans le graphe défini sur V par les arborescences de classes construites par RATIO plus les relations $\bar{x} \rightarrow y_i$ pour tous les représentants \bar{x} dans $\mathcal{D}(\rho)$, avec $\rho\bar{x} = Fy_1 \cdots y_n$. Ce graphe a un nombre de noeuds égal à n , et un nombre d'arêtes borné supérieurement par kn . On peut donc déterminer en temps $\mathcal{O}(n)$ si un tel graphe a un cycle, par exemple par tri topologique (voir Knuth [KNI]).

En résumé, l'algorithme d'unification de termes de premier ordre défini par RATIO et CYCLE est donc de complexité :

- $\mathcal{O}(n \cdot G(n))$ en temps
- $\mathcal{O}(n)$ en mémoire.

Il est possible d'effectuer l'unification en temps exactement linéaire. Nous reviendrons sur ce point en 5.9. Mais auparavant nous allons montrer comment compléter l'ensemble des termes en introduisant des termes infinis, solutions d'équations récursives simples. L'algorithme RATIO peut alors être interprété comme effectuant l'unification sur l'ensemble des termes finis et infinis.

5.8. Unification sur les arbres

Soit \sim une équivalence rationnelle et e un terme quelconque.

Comme nous l'avons vu en 5.6, si \sim est acyclique en e , il existe dans la classe $[e]_\sim$ un terme maximal $\sigma_\sim e$. Si \sim est cyclique en e , il est possible de définir de la même manière une suite croissante de termes, qui sont des approximations d'un arbre infini. En effet, choisissons pour tout x dans V un représentant $\tilde{x} \in T$, tel que :

$$1) \quad \tilde{x} \sim x$$

$$2) \quad \tilde{x} \in V \Leftrightarrow [x]_\sim \subset V.$$

Soit maintenant la substitution $\tilde{\sigma}$ définie par :

$$\tilde{\sigma} = \{<x, \tilde{x}> \mid x \in V\}.$$

Alors pour tout e dans T la suite

$$<e_0, e_1, \dots, e_n, \dots>$$

$$\text{avec } e_0 = e$$

et $e_n = \tilde{\sigma} e_{n-1}$ $n \geq 1$ est une suite croissante

(i.e. $e_n \geq e_{n-1}$) dans $[e]_\sim$. Elle définit un arbre infini rationnel, similaire aux arbres définis par des schémas de programme récursifs zéro-adiques (Courcelle [CB1]).

Par exemple, en prenant \sim définie par :

$x \sim Fy zu$ et $y \sim z \sim Gy$, à la classe $[x]_\sim$ correspond la suite

$$<x, Fy zu, FGyGyu, FGGy GGyu, \dots>$$

qui définit l'arbre infini "FG $^\infty$ G $^\infty$ u".

L'ordre \leq s'étend naturellement aux arbres infinis. Par exemple,

$$FG^\infty G^\infty u \leq FG^\infty G^\infty Gv \leq FG^\infty G^\infty G^\infty.$$

Nous allons montrer que nous pouvons résoudre les équations aux termes sur ces arbres et que si de telles équations admettent une solution, alors elles admettent une solution minimum, qui peut être calculée par l'algorithme RATIO.

Le prochain paragraphe définit les arbres, finis ou infinis. Ensuite nous montrerons comment représenter ces arbres grâce à des congruences cohérentes simplifiables. Enfin nous montrerons comment résoudre les équations aux arbres par des substitutions généralisées, appelées greffes.

5.8.1. Arbres

Définitions

Dénotons par N^* l'ensemble des n-uples d'entiers. On dénotera les éléments de N^* par u, v, w .

Si $u, v \in N^*$, uv dénote la concaténation des n-uples.

ϵ dénote l'élément neutre zéro-uple.

On appelle *arbre* une application A de N_A dans $C \cup V$, où N_A est un sous-ensemble de N^* qui satisfait :

$$1) \epsilon \in N_A$$

$$2) \forall u \in N^* \quad \forall i \in N$$

$$ui \in N_A \Leftrightarrow u \in N_A \text{ et } i \leq \begin{cases} \alpha(A(u)) & \text{si } A(u) \in C \\ 0 & \text{si } A(u) \in V \end{cases}$$

On appelle N_A ensemble des noeuds de A .

L'arbre A est fini si N_A est fini, infini sinon.

Aux arbres finis correspondent bijectivement les termes T , de la manière évidente.

En effet, si A est un arbre alors

-soit $N_A = \{\epsilon\}$, et $A(\epsilon) \in V \cup \{F \in C \mid \alpha(F) = 0\}$.

On notera un tel arbre élémentaire $\langle A(\epsilon) \rangle$.

-soit $N_A = \{\epsilon\} \cup \{i \mid u \in N_{A_i}, i \leq n = \alpha(F)\}$ avec $F = A(\epsilon) \in C$,

et alors l'application A_i de N_{A_i} dans $C \cup V$ définie par :

$A_i(u) = A(iu)$ est un arbre. On note alors $A = \langle FA_1 \cdots A_n \rangle$.

Un arbre fini A correspond donc un terme $t_A \in T$ par l'isomorphisme :

-si $N_A = \{\epsilon\}$ alors $t_A = A(\epsilon)$

-sinon, soit $A = \langle FA_1 \cdots A_n \rangle$ alors $t_A = F(t_{A_1}, \dots, t_{A_n})$.

L'arbre A est dit *rationnel* ssi il existe un automate fini

$\text{Aut}(A)$ ayant pour ensemble d'états un ensemble fini K et pour

alphabet N tel que $\forall u \in N^*$ $\text{Aut}(A)$ reconnaît u dans l'état q ssi $u \in N_A$ et

$A(u) = \phi(q)$, où $\phi : K \rightarrow C \cup V$.

Prenons que K fini entraîne en particulier que l'arité des $A(u)$ est bornée par un entier k , et donc qu'on peut limiter l'alphabet à l'ensemble fini $\{1, 2, \dots, k\}$.

On dénote par \mathcal{A} l'ensemble des arbres, et par \mathcal{R} l'ensemble des arbres rationnels.

5.8.2. Greffes

Définition :

On définit une *graffe* comme une application totale de V dans \mathcal{A}_q .

Nous dénoterons les greffes par γ, δ .

Les greffes sont la généralisation aux arbres des substitutions.

Si γ est une greffe, on l'étend en un endomorphisme de A par :

- si $A = \langle x \rangle$ alors $\gamma A = \gamma x$

- si $A = \langle FA_1 \dots A_n \rangle$ alors $\gamma A = \langle F\gamma A_1 \dots \gamma A_n \rangle$.

Les greffes peuvent également être étendues en homomorphismes de T dans A .

Si γ et δ sont des greffes, on définit la composition $\gamma\delta$ par :

$$\gamma\delta x = \gamma(\delta x).$$

On définit un préordre \leq dans g par :

$$\gamma \leq \gamma' \iff \exists \delta \quad \gamma' = \delta\gamma.$$

Nous allons maintenant montrer comment définir des arbres et des greffes à l'aide d'équivalences cohérentes simplifiables sur T .

5.8.3. Equivalences d'arbre

Définition :

On appelle *équivalence d'arbre* toute équivalence sur T qui est cohérente et simplifiable (cf 5.6.1).

Soit \sim une équivalence d'arbre. Pour tout x dans V , on se donne un représentant canonique $\tilde{x} \in C$ de la classe $[x]_\sim$, ayant les propriétés :

a) $\tilde{x} \in [x]_\sim$

b) $\tilde{x} \in V \iff [x]_\sim \subset V$.

On définit maintenant, pour tout $e \in T$, un arbre $A = A_\sim(e) \in A$ appelé arbre associé à e dans \sim par :

i) si $[e]_\sim \subset V$ alors $A = \langle \tilde{e} \rangle$

ii) sinon soit $Fe_1 \dots e_n \sim e$ alors

$$A = \langle FA_1 \dots A_n \rangle \text{ avec } A_i = A_\sim(e_i).$$

Le lemme suivant montre que cette définition a un sens.

Lemme 5.26

Soit \sim une équivalence d'arbre. Pour tout $e \in T$, l'arbre $A_{\sim}(e)$ existe, et $e \sim e' \Rightarrow A_{\sim}(e) = A_{\sim}(e')$.

Démonstration :

Montrons que $A(w)$ est uniquement déterminé par $[e]_{\sim}$, par récurrence sur $|w|$, pour tout $A = A_{\sim}(e)$.

- si $w = \epsilon$: dans le cas i) $A(\epsilon) = \tilde{e}$.

dans le cas ii) $A(\epsilon) = F$ unique car \sim est cohérente.

- si $w = iu$ dans le cas i) $w \notin N_A$

dans le cas ii) $A(iu) = A_i(u)$, qui est uniquement déterminé par hypothèse de récurrence, la classe $[e_i]_{\sim}$ étant unique car \sim est simplifiable. \square

Remarque : Si \sim est une équivalence rationnelle, alors $A_{\sim}(e) \in R$. En effet, définissons $SC(e, \sim)$ comme en 4.6.1. A toute classe $[e_i]_{\sim}$ de $SC(e, \sim)$ on peut associer de manière unique un élément de $C \cup V$:

$$\phi([e_i]_{\sim}) = \begin{cases} - \tilde{e}_i & \text{si } [e_i]_{\sim} \subset V \\ - F & \text{si } e_i \sim FE_1 \dots E_n. \end{cases}$$

Considérons le graphe fini $\langle SC(e, \sim), \xrightarrow{i} \rangle$, où $[FE_1 \dots E_n]_{\sim} \xrightarrow{i} [e_i]$.

Ce graphe détermine de manière unique un automate fini prenant ses états dans $SC(e, \sim)$ et d'alphabet N , et reconnaissant l'arbre $A_{\sim}(e)$. Ceci justifie notre terminologie d'équivalence rationnelle.

Si de plus \sim est acyclique, alors il est facile de vérifier que $A_{\sim}(e)$ est l'arbre fini isomorphe à $\sigma_{\sim}e$.

Définition : Si \sim est une équivalence d'arbre, on dénote par $\hat{\sim}$ sa fermeture par congruence.

Nous laissons au lecteur le soin de montrer que pour tous e et \sim ,
 $A_{\sim}(e) = A_{\hat{\sim}}(e)$.

Nous allons maintenant montrer un lemme qui jouera ici le même rôle que 5.23 pour le cas fini.

Si $e \in T$ et $w \in N^*$, on utilise l'abus de notation $e(w)$ pour dénoter $A(w)$, où A est l'arbre fini isomorphe à e .

Lemme 5.27 des approximants

Soit \sim une équivalence d'arbre, $e \in T$, et $A = A_{\sim}(e)$.

Pour tout w dans N^* , si $w \in N_A$ et $A(w) = @$, alors il existe e' dans T , avec $e \hat{\sim} e'$, tel que $e'(w) = @$.

Démonstration :

Par récurrence sur $|w|$.

(i) si $w = \varepsilon$ on peut prendre :

$$e' = \begin{cases} \hat{x} & \text{si } e = x \\ e & \text{sinon.} \end{cases}$$

ii) si $w = iu$ on suppose la propriété vraie pour u , pour tout $e \in T$.

- si $[e]_{\sim} \subset V$ alors $N_A = \{\varepsilon\}$ et donc $w \notin N_A$.

- sinon soit $F e_1 \cdots e_n \sim e$, et soit $A_i = A_{\sim}(e_i)$.

Si $w \in N_A$, alors $u \in N_{A_i}$, et $A(w) = @ \implies A_i(u) = @$.

Par hypothèse de récurrence, il existe $e'_i \hat{\sim} e_i$ tel que $e'_i(u) = @$.

Alors $e' = F e_1 \cdots e_{i-1} e'_i e_{i+1} \cdots e_n$ est bien tel que $e' \hat{\sim} e$, avec $e'(w) = @$. \square

Ce lemme exprime que, pour tout noeud de l'arbre $A_{\sim}(e)$, il existe un approximant fini e' congru à e par \sim qui couvre ce noeud.

Définition : Soit \sim une équivalence d'arbre. On définit la *greffe canonique* γ_{\sim} de \sim comme l'application de V dans A définie par :

$$\gamma_{\sim}x = A_{\sim}(x) \quad \forall x \in V.$$

Remarque : Si \sim est une équivalence rationnelle, alors γ_{\sim} est une application de V dans R , telle que $\gamma_{\sim}x = \langle x \rangle$ presque partout. On dit alors que γ_{\sim} est une *greffe rationnelle*.

Lemme 5.28 Soit \sim une équivalence d'arbre.

$$\forall e \in T \quad \gamma_{\sim}e = A_{\sim}(e).$$

Démonstration : Facile par récurrence sur e . \square

Corollaire : $\forall e, e' \in T \quad e \sim e' \Rightarrow \gamma_{\sim}e = \gamma_{\sim}e'$.

Définition :

Soit γ une greffe quelconque. On définit la *congruence de greffe* \sim_{γ} associée à γ comme la congruence sur T définie par :

$$e \sim_{\gamma} e' \Leftrightarrow \gamma e = \gamma e'.$$

Il est clair que \sim_{γ} est une congruence d'arbre. De plus, si γ est une greffe rationnelle, alors \sim_{γ} est une congruence rationnelle.

Remarque : Le lemme 5.28 exprime que s'il existe une équivalence d'arbre \sim telle que $e \sim e'$, alors la greffe canonique unifie e et e' dans A . De ce lemme on déduit sans peine $\sim \subset \sim_{\gamma_{\sim}}$. Par contre ici la réciproque n'est plus vraie, même si \sim est une équivalence rationnelle. Par exemple, l'équivalence rationnelle \sim définie par les seules paires $x \sim Fx$ et $y \sim Fy$ est telle que $\gamma_{\sim}x = \gamma_{\sim}y = \overline{F}$. Mais on n'a pas $x \sim y$. Bien sûr si \sim est acyclique, alors le lemme 5.24 est toujours vrai.

Lemme 5.29 De minimalité de γ_{\sim} .

Soit \sim une équivalence d'arbre, γ une greffe quelconque.

Alors $\sim \subset \sim_{\gamma} \Rightarrow \gamma_{\sim} \leq \gamma$.

Démonstration :

Soit \sim et γ telles que $\sim \subset \sim_{\gamma}$. \sim_{γ} étant une congruence, on a $\sim \subset \hat{\sim} \subset \sim_{\gamma}$. Montrons que, pour tout e dans T , $\gamma e = \gamma A$, avec $A = A_{\sim}(e)$.

Pour cela on montre que, pour tout $w \in N^*$, $[\gamma e](w) = [\gamma A](w)$.

Il y a deux cas :

- i) $w \in N_A$, avec $A(w) = F \in C$. Alors par le lemme 5.27 il existe $e' \in T$, avec $e' \hat{\sim} e$, tel que $e'(w) = F$. $e' \hat{\sim} e \Rightarrow e' \sim_{\gamma} e \Rightarrow \gamma e = \gamma e'$, et donc $[\gamma e](w) = [\gamma e'](w) = [\gamma A](w) = F$.
- ii) $w = uv$, avec $A(u) = x \in V$, et donc $[\gamma A](w) = [\gamma x](v)$.

Alors par le lemme 5.27 il existe $e' \in T$, avec $e' \hat{\sim} e$, tel que $e'(u) = x$, et de même qu'en (i) $\gamma e = \gamma e'$, d'où $[\gamma e](w) = [\gamma e'](w) = [\gamma x](v) = [\gamma A](w)$ (dans le cas où $v \notin N_{\gamma x}$, alors $w \notin N_{\gamma A}$ et $w \notin N_{\gamma e}$).

On a donc, en particulier, pour tout x dans V :

$$\gamma x = \gamma A_{\sim}(x) = \gamma \gamma_{\sim} x, \text{ d'où } \gamma_{\sim} \leq \gamma. \quad \square$$

Corollaire : Soit \sim et \sim' deux équivalences d'arbres. Si $\sim \subset \sim'$, alors $\gamma_{\sim} \leq \gamma_{\sim'}$.

Démonstration :

Il suffit de faire $\gamma = \gamma_{\sim'}$ dans le lemme 5.29, et d'appliquer $\sim' \subset \sim_{\gamma_{\sim'}}$, qui découle du lemme 5.28. \square

Remarque : De même que pour le cas fini, il n'est pas vrai qu'en général, pour une greffe γ quelconque, on ait $\gamma = \gamma_{\sim}$.

Par exemple , avec γ définie par $\left\{ \begin{array}{l} \gamma x = \langle F(x) \rangle \\ \dots \\ \gamma y = \langle y \rangle \quad \forall y \neq x, \end{array} \right.$

on a $\sim_\gamma =$ Identité, d'où $\gamma_{\sim_\gamma} = \gamma'$, avec $\gamma'x = \langle x \rangle \quad \forall x \in V$.

Le lemme 5.29 étant l'analogue du lemme 5.25 dans le cas fini, on a maintenant l'analogue du théorème 16, mais concernant maintenant l'unification dans A .

5.8.4. Unification dans A

Soit E un ensemble fini de termes, \sim_E la plus petite équivalence contenant les paires de termes dans E . Soit \sim la congruence simplifiable de \sim_E . Si \sim est cohérente, alors γ_\sim est la greffe minimum qui unifie E dans A ; sinon E n'est pas unifiable dans A .

De plus, \sim_E , remplaçant la condition de finitude, est une équivalence rationnelle, et sa fermeture simplifiable \sim l'est donc également.

La greffe minimum est donc toujours rationnelle. On obtient γ_\sim (ou la réponse échec) par l'algorithme RATIO décrit en 5.7. On peut alors interpréter un environnement comme un système d'équations récursives simples (analogues aux schémas de programme zéro-adiques).

Remarquons qu'on peut se donner au départ un système d'équations quelconques, et résoudre ce système en un système explicite d'équations récursives. Ceci donne donc un moyen de résoudre des équations dans R . D'où le théorème :

Théorème 17 d'unification dans A .

Soit $E = \{e_i = e'_i \mid e_i, e'_i \in T, 1 \leq i \leq n\}$ un système quelconque d'équations entre éléments de T . Soit \sim_E la plus petite équivalence telle que $\forall i \leq n \quad e_i \sim_E e'_i$. Soit \sim la fermeture simplifiable de \sim_E . Si \sim est cohérente,

alors E possède des solutions dans A . De plus, il existe une solution minimum γ_{\sim} , qui est solution dans R . Sinon, F n'a pas de solution dans A .

Nous ne ferons pas la démonstration de ce théorème, qui est similaire à celle du théorème 16.

Exemple :

Soit à unifier

$$F(G(x, x'), w) \text{ et } F(G(u, F(w, y')), G(w, x'))$$

où

$$\left\{ \begin{array}{l} x = F(F(x, y), z) \\ y = G(x, z) \\ u = F(u, v) \\ v = G(u, v). \end{array} \right.$$

Initialement, \sim_E contient les cinq paires de l'énoncé.

Après fermeture par simplification, on obtient l'équivalence rationnelle \sim ayant pour classes :

$$C_1 = \{F(G(x, x'), w), F(G(u, F(w, y')), G(w, x'))\}$$

$$C_2 = \{G(x, x'), G(u, F(w, y'))\}$$

$$C_3 = \{F(F(x, y), z), x, u, F(u, v), F(x, y)\}$$

$$C_4 = \{x', F(w, y')\}$$

$$C_5 = \{w, G(w, x')\}$$

$$C_6 = \{y, v, G(x, z), z, G(u, v)\}.$$

On a donc une solution minimale γ_{\sim} , définie par :

$$\left\{ \begin{array}{l} \gamma_{\sim} x = \gamma_{\sim} u = A_3 \\ \gamma_{\sim} x' = A_4 \\ \gamma_{\sim} w = A_5 \\ \gamma_{\sim} y = \gamma_{\sim} v = \gamma_{\sim} z = A_6, \text{ où} \end{array} \right.$$

$$A_3 \equiv \langle F A_3 A_6 \rangle$$

$$A_4 \equiv \langle F A_5 \langle y' \rangle \rangle$$

$$A_5 \equiv \langle G A_5 A_4 \rangle$$

$$A_6 \equiv \langle G A_3 A_6 \rangle.$$

Toutes les autres solutions s'en déduisent en substituant un arbre quelconque à y' .

5.8.5. Complétion de \hat{F} par les classes d'arbres infinis

De même que dans le cas fini, on peut définir dans l'ensemble des arbres A un préordre \leq de filtrage par :

$$A \leq A' \Leftrightarrow \exists \gamma \quad A' = \gamma A.$$

Soit \equiv l'équivalence associée à \leq :

$$A \equiv A' \Leftrightarrow A \leq A' \text{ et } A' \leq A.$$

On considère maintenant $\hat{R} = R/\equiv$ et $\hat{A} = A/\equiv$, munis de l'ordre \leq .

\hat{F} étant isomorphe à une partie de \hat{A} , on peut considérer \hat{A} comme la complétion de \hat{F} par des termes infinis.

Le théorème 17 nous donne immédiatement l'existence du sup, sous la forme :

Lemme 5.30

Dans la structure d'ordre partiel $\langle \hat{A}, \leq \rangle$, deux classes finies ou rationnelles quelconques bornées supérieurement ont un sup, qui est dans \hat{R} .

Par exemple, on a maintenant :

$[Fx]_{\equiv} \vee [FxGx]_{\equiv} = [F\overset{\infty}{\overbrace{G}}]_{\equiv}$, où ce terme infini rationnel est défini par $[Fzz]_{\sim}$, avec $z \sim Gz$.

Plus généralement, on peut montrer que deux arbres quelconques A et A' compatibles admettent un sup $A \vee A'$. Supposons A et A' définis par e et e' dans une même équivalence d'arbre \sim . (On suppose naturellement que les variables

définissant A et A' sont distinctes). Alors A et A' sont compatibles si et seulement si la relation $\sim \cup \{e, e'\}$ peut s'étendre en une équivalence d'arbre \sim' , et alors $A \vee A' \equiv A_{\sim'}(e)$. On obtient \sim' , comme dans le cas rationnel, en formant la fermeture simplifiable de \sim , et en testant sa cohérence.

Mais nous sommes davantage intéressés en les arbres rationnels, car ils sont finiment représentables, et le lemme 5.30 nous indique que le sup de deux tels arbres est finiment calculable par la procédure RATIO.

Nous allons maintenant donner un algorithme d'inf, qui construit l'inf $A \wedge A'$ de deux classes d'arbres quelconques dans \hat{A} .

Algorithme d'inf dans \hat{A}

Soit A et A' deux arbres quelconques dans \hat{A} . On peut supposer, sans perte de généralité, que A et A' sont déterminés par une même équivalence d'arbre \sim . C'est à dire qu'il existe e et e' tels que $A \equiv A_{\sim}(e)$ et $A' \equiv A_{\sim}(e')$. Soit $C = SC(e, \sim)$, $C' = SC(e', \sim)$, $V = \{x \in V \mid [x]_{\sim} \notin C \cup C'\text{ et }[x]_{\sim} = \{x\}\}$, et soit ϕ une bijection quelconque de $C \times C'$ dans V . (Même si $C \cup C'$ est infini, on peut toujours supposer que V est infini).

On construit l'équivalence \sim' contenant \sim et toutes les paires $\langle x, Fy_1 \dots y_n \rangle$ telles que $x = \phi(c_1, c_2)$, où $c_1 \in C, c_2 \in C'$ sont deux classes telles qu'il existe $Fe_1 \dots e_n$ dans c_1 et $Fe'_1 \dots e'_n$ dans c_2 , et où $y_i = \phi([e_i]_{\sim}, [e'_i]_{\sim})$ pour $i \leq n$. On vérifie sans peine que \sim' est une équivalence d'arbre, et que si \sim est une équivalence rationnelle, alors \sim' est rationnelle aussi.

Maintenant on définit

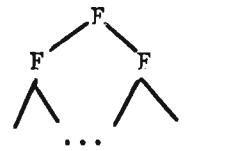
$$A \wedge A' = A_{\sim'}(z),$$

$$\text{avec } z = \phi([e]_{\sim}, [e']_{\sim}). \square$$

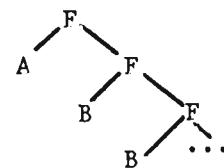
Nous pensons qu'il est facile pour le lecteur de se persuader que $A \wedge A'$ est bien l'inf de A et A' dans la structure $\langle A, \leq \rangle$. Nous en omettrons la preuve, qui est fastidieuse et sans intérêt, et nous donnons plutôt quelques exemples d'ufs d'arbres rationnels.

Exemple 1 :

$$\text{Soit } A = A_\sim(u_1) =$$



$$\text{et } A' = A_\sim(v_1) =$$



avec \sim définie par :

$$u_1 \sim Fu_1u_1$$

$$v_1 \sim FA v_2$$

$$v_2 \sim FR v_2.$$

$$\text{Soit } \phi \text{ tel que } \phi([u_1]_\sim, [v_1]_\sim) = z_1$$

$$\phi([u_1]_\sim, [v_2]_\sim) = z_2$$

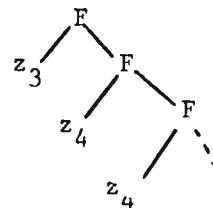
$$\phi([u_1]_\sim, [A]_\sim) = z_3$$

$$\phi([u_1]_\sim, [B]_\sim) = z_4$$

On construit \sim' à partir de \sim , en ajoutant :

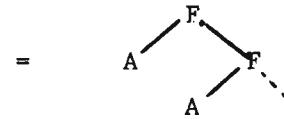
$$\left\{ \begin{array}{l} z_1 \sim' Fz_3z_2 \\ z_2 \sim' Fz_4z_2 \end{array} \right.$$

$$\text{et } A \wedge A' = A_{\sim'}(z_1) =$$

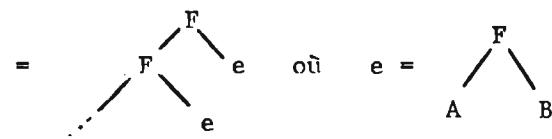


Exemple 2 :

$$\text{Soit } A = A_{\sim}(u_1)$$



$$\text{et } A' = A_{\sim}(v_1)$$



avec \sim définie par :

$$u_1 \sim FAu_1$$

$$v_1 \sim Fv_1 v_2$$

$$v_2 \sim FAB$$

$$\text{Avec } \phi \text{ tel que } \phi([u_1]_{\sim}, [v_1]_{\sim}) = \phi u_1 v_1 = z_1$$

$$\phi u_1 v_2 = z_2$$

$$\phi u_1 B = z_3$$

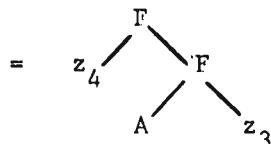
$$\phi A v_1 = z_4$$

$$\phi A A = z_5$$

on ajoute à \sim :

$$\left\{ \begin{array}{l} z_1 \sim^* Fz_4 z_2 \\ z_2 \sim^* Fz_5 z_3 \\ z_5 \sim^* A \end{array} \right.$$

$$\text{et } A \wedge A' = A_{\sim^*}(z_1)$$



Nous pouvons résumer nos résultats sur la structure des arbres par le lemme suivant.

Lemme 5.31

$\langle \hat{\mathcal{A}}, \leq \rangle$ est un treillis sous condition, c'est à dire que :

- deux arbres quelconques admettent un inf
- deux arbres majorés admettent un sup.

De plus, si A et A' sont rationnels, alors $A \wedge A'$ et $A \vee A'$ sont rationnels aussi.

La structure $\langle \hat{R}, \leq \rangle$ est donc également un treillis sous condition. Mais cette structure est plus compliquée que l'ITF $\langle \hat{T}, \leq \rangle$. En particulier, il existe des chaînes infinies décroissantes dans \hat{R} . Par exemple, considérons les arbres rationnels définis par :

$$\begin{cases} A_1 = \langle F < x_1 > A_1 \rangle \\ A_n = \langle F < x_n > A_{n-1} \rangle \end{cases} \text{ avec } n > 1.$$

On a bien $A_{n-1} = \gamma_n A_n$, avec γ_n défini par :

$$\gamma_n x_i = \langle x_{i-1} \rangle \quad 1 < i \leq n.$$

Par contre, on n'a pas $A_n = \gamma A_{n-1}$, comme il est facile de montrer. On a donc ici une chaîne infinie décroissante

$$A_1 > A_2 > \dots > A_n > \dots$$

Tous les A_n sont rationnels. Mais la limite de cette chaîne est l'arbre non rationnel $\langle F < u_1 \rangle < F < u_2 \rangle < F < u_3 \rangle \dots \rangle \gg$.

On ne peut donc espérer avoir une condition de complétude de $\langle \hat{R}, \leq \rangle$, comme nous en avions pour $\langle \hat{T}, \leq \rangle$.

5.9 Unification linéaire

Récemment M. Paterson et M. Wegman ont présenté un algorithme d'unification de termes de premier ordre linéaire [PW1].[†] Leur algorithme peut être expliqué en termes d'équivalences rationnelles acycliques.

Donnons brièvement une description de cet algorithme.

Soit e et e' les termes à unifier, et soit initialement

$$\sim_0 = \{<e, e'>\}.$$

L'algorithme procède, comme plus haut, par raffinements successifs de la relation \sim_i par itération de la condition de simplification, et test de la cohérence.

Mais ici il y a une restriction supplémentaire sur la classe sélectionnée pour être simplifiée.

Définitions

Soit \sim une équivalence cohérente, c une classe de \sim .

On dit que c est *simplifiable* ssi

$$Fe_1 \cdots e_n \in c \text{ et } Fe'_1 \cdots e'_n \in c \implies e_i \sim e'_i \quad 1 \leq i \leq n.$$

On dit que la classe c est *permise* ssi il n'existe pas de classe c' non simplifiable, telle que $c' \rightarrow_{\sim} c$.

Lemme 5.32

Soit \sim une équivalence finie cohérente acyclique. Si toutes les classes permises de \sim sont simplifiables, alors \sim est simplifiable.

Nous omettons la démonstration de ce lemme, qui découle directement des définitions.

[†] U. Montanari a découvert cet algorithme indépendamment. (communication personnelle).

Soit maintenant \sim_i l'équivalence obtenue après la i -ème itération.

Si \sim_i n'est pas cohérente, on arrête avec échec. Si \sim_i n'est pas acyclique, on arrête également avec échec. Sinon, si toutes les classes permises de \sim_i sont simplifiables, alors \sim_i est une équivalence rationnelle acyclique, et e et e' sont unifiables par σ_{\sim_i} . Dans le cas contraire, on sélectionne une classe permise de \sim_i non simplifiable, et \sim_{i+1} est obtenue à partir de \sim_i en simplifiant cette classe.

L'intérêt de la méthode réside dans le fait que si une classe est permise, on sait qu'on ne rajoutera aucun nouveau terme dans cette classe par des simplifications ultérieures. Il est donc possible de ne plus considérer cette classe, dès qu'on l'a simplifiée.

Nous renvoyons à l'article de Paterson et Wegman pour plus de détails, en particulier pour la description de la structure de données utilisée pour pouvoir effectuer les opérations nécessaires de gestion des classes d'équivalence de termes en un temps linéaire en $\theta(e) + \theta(e')$.

Remarquons toutefois qu'ici la condition "acyclique" est essentielle. On ne peut donc étendre l'algorithme linéaire pour faire l'unification d'arbres rationnels, comme nous avons pu le faire avec l'algorithme plus simple RATIO, où la sélection de la classe à simplifier était arbitraire.

Du point de vue pratique, l'algorithme linéaire est considérablement plus compliqué que notre algorithme, et il semble donc que la coefficient de proportionnalité plus élevé rende l'algorithme linéaire moins performant en moyenne.

D'un autre côté, l'algorithme linéaire aura l'avantage de détecter plus rapidement les termes non unifiables dans T , mais unifiables dans R .

CHAPITRE 6

Unification dans les langages de second ordre

Nous considérons dans ce chapitre le cas des langages de second ordre, où l'on autorise des variables de fonction.

De même que dans le chapitre 5, nous allons reprendre ici la plupart des définitions, car tout l'arsenal du λ -calcul ne nous est pas nécessaire. Néanmoins nous ne ferons généralement pas de preuves complètes, en particulier pour les lemmes techniques dont nous avons déjà établi des versions plus générales aux chapitres 1 à 4.

Nous allons tout d'abord généraliser l'ensemble des termes de premier ordre du chapitre 5 en un ensemble T de termes de second ordre restreint, où l'on autorise des variables de fonctions, mais pas l'opérateur λ .

Le préordre \leq du chapitre 5 peut se généraliser de deux manières différentes :

- en une relation notée \leq qui est la restriction aux termes de second ordre du préordre \leq du chapitre 4.
- en une relation notée Δ , qui impose aux fonctions solutions d'un

filtrage d'être strictes. Le problème de l'unification, avec cette relation, est celui des équations aux arbres. Dans le cas monadique, on se ramène au problème des équations dans un monoïde libre avec constantes.

L'analogue de Δ , pour le λ -calcul, serait de se restreindre aux λ -I-termes, c'est à dire aux termes construits à l'aide d'une règle d'abstraction restreinte :

$$\lambda u e \text{ est un } \lambda\text{-I-terme ssi } u \in V[e].$$

Nous montrerons que les structures $\langle T, \leq \rangle$ et $\langle T, \Delta \rangle$ sont plus complexes que dans le premier ordre, et en particulier qu'il n'y a ni infini sup, et qu'il existe des chaînes infinies décroissantes strictement. Ces résultats sont vrais même si on se limite aux termes monadiques.

Nous compléterons ensuite T en \bar{T} en autorisant de lier des variables avec l'opérateur λ . \bar{T} est la restriction au second ordre de l'ensemble T_x du chapitre 4. Nous étudierons les problèmes d'unification et de demi-unification dans le langage \bar{T} .

6.1. Définitions générales

6.1.1. Termes de second ordre restreints

Pour tout $i \geq 0$, on se donne un ensemble V_i dénombrable de variables de fonctions d'arité i . On suppose les V_i disjoints deux à deux. L'ensemble des variables est :

$$V = \bigsqcup_{i \geq 0} V_i.$$

Soit f une variable de V_i . On dénote son arité par :

$$\alpha(f) = i.$$

On se donne également un ensemble fini ou dénombrable

$$\mathcal{C} = \{F_1, F_2, \dots\}$$

de constantes de fonctions données avec leur arité $\alpha(F_i) \geq 0$.

On utilisera f, g, h pour dénoter des variables, x, y, z, \dots

pour des variables d'arité 0, F, G, H, A, B, \dots pour des constantes, $@, @', \dots$

pour des variables ou constantes.

L'ensemble des termes restreints T est défini par :

$$@ \in V \cup \mathcal{C} \text{ et } t_1, t_2, \dots, t_{\alpha(@)} \in T \implies @_1 \dots t_{\alpha(@)} \in T.$$

(Remarquer que $V_0 \subset T$). Comme pour le premier ordre, nous emploierons quelquefois la notation parenthésée $@(t_1, \dots, t_n)$.

Si $t \in T$, l'ensemble des variables $V(t)$ de t est défini par :

$$V(@ t_1 \dots t_n) = \bigcup_{i=1}^n V(t_i) \cup (\{\@\} \cap V).$$

Soit $t \in T$. On définit le degré de t $\delta(t)$ par :

$$\delta(t) = \min \{i \mid V(t) \subset V_i\}.$$

On définit : $T_i = \{t \in T \mid \delta(t) = i\}$.

T_0 , l'ensemble des termes de premier ordre, a les mêmes propriétés que l'ensemble T du chapitre 5.

T_1 est l'ensemble des termes monadiques.

Dans le cas particulier où $\forall F \in \mathcal{C} \quad \alpha(F) \leq 1$, on désignera T_1 par L , ensemble des termes linéaires.

Nous allons montrer que T a une structure beaucoup plus complexe qu'au premier ordre. Nous étendrons ensuite T en l'ensemble complet \bar{T} , en introduisant des variables liées, où nous étudierons les problèmes de filtrage et d'unification.

Remarque :

De même qu'au premier ordre, on pourrait raffiner l'arité des atomes en un type, comme pour le λ -calcul. Il est évident que cela ne pose pas de problème supplémentaire, et que tous nos résultats sont valables sur des langages typés.

6.1.2. Substitutions

On appelle *terme mutificateur relatif à f*, avec $f \in V_n$, tout terme

$$fx_1 \cdots x_n,$$

les x_i étant des variables distinctes de V_0 . On dénote par $m_f, m'_f \dots$ un terme **mutificateur relatif à f**.

Remarquons que si $x \in V_0$, x est terme mutificateur relatif à lui-même.

Une *substitution* est un ensemble fini de composantes

$$\sigma = \{m_f, t_f\}$$

où les m_f sont des termes mutificateurs relatifs à des variables distinctes, et les t_f des termes.

Soit σ une substitution, t un terme de T . On définit récursivement σt par :

$$\sigma @ t_1 \cdots t_n =$$

- (i) - s'il existe dans σ une paire $\langle @x_1 \cdots x_n, t @ \rangle$, alors $\sigma t @$, où ρ est la substitution de 1er ordre $\{x_i, \sigma t_i | 1 \leq i \leq n\}$
- (ii) - sinon $@ \sigma t_1 \cdots \sigma t_n$.

Remarques :

- 1) Si $@ \in C$ on est toujours dans le cas (ii).
- 2) Cette définition est équivalente pour les termes de second ordre, à celle donnée au chapitre 2, si l'on remplace la paire $\langle fx_1 \cdots x_n, t_f \rangle$ par la paire $\langle f, \lambda x_1 \cdots x_n \cdot t_f \rangle$, les β -réductions étant appliquées de l'intérieur vers l'extérieur, et la η -réduction étant permise.
- 3) Nous n'avons pas ici de problème de renommage de variables liées,

comme le montre l'exemple suivant.

Exemple :

Soit $t = f(x)$,

$$\sigma = \{<x, A>, <f(x), G(x, x)>\}$$

$$\begin{aligned} \text{Alors } \sigma t &= \rho G(x, x) \quad \text{avec } \rho = \{<x, A>\} \\ &= G(A, A). \end{aligned}$$

Définitions :

Soit σ, ρ des substitutions, V un sous-ensemble fini de U .

On définit :

$$\mathcal{D}(\sigma) = \{f \mid \exists <m_f, t_f> \in \sigma \quad t_f \neq m_f\}$$

Soit $<fx_1 \dots x_n, t_f> \in \sigma$. On définit : $I(\sigma, f) = V(t_f) - \{x_1, \dots, x_n\}$.

$$\text{et : } I(\sigma) = \bigsqcup_{f \in \mathcal{D}(\sigma)} I(\sigma, f).$$

Enfin : $\sigma \upharpoonright V = \{<m_f, t_f> \in \sigma \mid f \in V\}$.

On définit l'égalité entre substitutions par :

$$\sigma = \rho \iff \mathcal{D}(\sigma) = \mathcal{D}(\rho) \quad \text{et}$$

$$\forall f \in \mathcal{D}(\sigma) \quad \text{avec } <fx_1 \dots x_n, t> \in \sigma \text{ et}$$

$$<fy_1 \dots y_n, t'> \in \rho,$$

alors $t = \xi t'$ et $t' = \xi' t$, avec $\xi = \{<y_i, x_i> \mid i \leq n\}$ et $\xi' = \{<x_i, y_i> \mid i \leq n\}$.

De même que précédemment, on définit :

$$\sigma \underset{V}{=} \rho \iff \sigma \upharpoonright V = \rho \upharpoonright V.$$

Lemme 6.1

$$\sigma = \rho \iff \forall t \in T \quad \sigma t = \rho t.$$

Démonstration :

- \Rightarrow par récurrence sur t , en utilisant la définition de σt .
- \Leftarrow en choisissant pour t un terme mutificateur dans σ
ou dans ρ . \square

Définitions :

La composante de substitution $\langle fx_1 \dots x_n, t_f \rangle$ est dite *stricte* ssi $\forall i \leq n \quad x_i \in V(t_f)$.

Une substitution est dite *stricte* ssi toutes ses composantes sont strictes.

On dénote par S l'ensemble des substitutions, et par \tilde{S} l'ensemble des substitutions strictes.

On définit deux préordres \leq et Δ dans T par :

$$\begin{aligned} t \leq t' &\iff \exists \sigma \in S \quad t' = \sigma t \\ t \Delta t' &\iff \exists \sigma \in \tilde{S} \quad t' = \sigma t. \end{aligned}$$

Dans le premier cas on dit que t schématise t' , dans le second que t schématise t' strictement.

Prenons que les lemmes 5.3 et 5.4 ne sont pas vrais ici, pour \leq comme pour Δ . En effet, on a par exemple :

$$\begin{aligned} f(x) &\Delta F(A) \quad \text{car} \quad F(A) = \sigma f(x), \text{ avec} \\ \sigma &= \{\langle f(y), y \rangle, \langle x, F(A) \rangle\} \\ \text{ou} \quad \sigma &= \{\langle f(y), F(y) \rangle, \langle x, A \rangle\}. \end{aligned}$$

Enfin, $x \Delta f(x) \Delta x$.

On définit la *composition* de σ et ρ comme la substitution :

$$\sigma\rho = \{ \langle m_f, \rho t_f \rangle \mid \langle m_f, t_f \rangle \in \sigma \} \cup \{ \langle m_f, t_f \rangle \in \rho \mid f \notin D(\sigma) \}.$$

Dans cette notation, nous supposons qu'il n'y a pas de conflits de variables, et qu'en particulier les variables liées par m_f n'appartiennent pas à $D(\rho)$ ou $I(\rho)$. Sinon, on effectue les renommages nécessaires.

Par exemple, avec $\sigma = \{ \langle f(x), F(x,y) \rangle \}$

et $\rho = \{ \langle x, A \rangle, \langle y, G(x) \rangle \}$,

on a $\rho\sigma = \{ \langle f(z), F(z, G(x)) \rangle \} \cup \rho$.

Avec ces précautions, on peut prouver :

Lemme 6.2

$$\forall t \in T \quad \forall \sigma, \rho \in S \quad (\rho\sigma)t = \rho(\sigma t).$$

Démonstration :

Par récurrence sur t . Les détails sont laissés au lecteur. []

En combinant les lemmes 6.1 et 6.2, il est aisément d'obtenir l'associativité de la composition de substitutions, comme au chapitre 2.

Remarquons que si σ est stricte, alors $V(t) \subset V(\sigma t)$.

Il s'ensuit que si $\sigma, \rho \in \bar{S}$, alors $\rho\sigma \in \bar{S}$.

Remarque :

\leq et Δ sont deux généralisations possibles du préordre du chapitre 5. Aux ordres supérieurs, \leq est l'analogie de la relation étudiée dans le chapitre 4, sur les termes de λ -K-calcul munis des règles β et η . L'analogie de Δ consisterait à se restreindre aux termes du λ -I-calcul, où la construction $\lambda x e$ n'est admise que si $x \in V[e]$.

Dans le prochain paragraphe, nous commencerons par étudier l'ensemble L des termes linéaires, c'est à dire entièrement composés de fonctions au plus monadiques. Nous étudierons plus particulièrement la structure de L relativement aux préordres \leq et Δ .

6.2. Termes linéaires

Nous supposons dans ce paragraphe que $\forall @ \in C \cup V \quad \alpha(@) \leq 1$.

Dénotons :

$$F = \{@ \in C \cup V \mid \alpha(@) = 1\} \quad (\text{atomes fonctionnels})$$

$$I = \{@ \in C \cup V \mid \alpha(@) = 0\} \quad (\text{atomes individuels}).$$

L'ensemble des termes linéaires est

$$L = F^* I.$$

6.2.1. Filtrage strict

6.2.1.1. Réduction au filtrage dans F^*

Soit $t, t' \in L$. Nous cherchons les solutions à $t \wedge t'$, c'est à dire les substitutions strictes σ telles que $\sigma t = t'$.

Les substitutions strictes sont ici des morphismes, prolongeant des applications de $F \cap V$ dans F^* et de $I \cap V$ dans $F^* I$.

Soit $t = w @$, $t' = w' @'$ avec $@, @' \in I$.

On doit avoir, en particulier, $\sigma @ = u @'$, avec $u \in F^*$.

Il y a deux cas :

i) $@ \in C$; alors si $@' \neq @$, pas de solutions,

sinon on se ramène à $\sigma w = w'$.

ii) $@ \in V$; alors on introduit une variable fonctionnelle h n'apparaissant pas dans tt' , et on se ramène à $\sigma wh = w'$.

Dans tous les cas, on se ramène donc à une équation du type $w \Delta w'$, avec $w, w' \in F^*$.

Toute solution à cette équation donnera bien sûr une solution au problème de filtrage original (dans le cas ii, avec $u = \sigma h$, on prendra $\sigma x = u @'$).

6.2.1.2. Recherche des solutions en longueurs

Soit l'équation $w \Delta w'$, avec $w, w' \in F^*$.

Soit $V(w) = \{f_1, \dots, f_n\}$, et soit o_i le nombre d'occurrences de f_i dans w , $1 \leq i \leq n$. Finalement, soit α le nombre d'occurrences de constantes dans w , et β la longueur de w' . L'équation en $\lambda_1, \dots, \lambda_n \geq 0$:

$$\sum_{i=1}^n o_i \lambda_i + \alpha = \beta \quad \text{admet un nombre fini de solutions.}$$

Une telle solution sera dite *solution en longueurs* de $w \Delta w'$.

A toute solution en longueurs $\langle \lambda_1, \dots, \lambda_n \rangle$ correspond au plus une solution σ de $w \Delta w'$, telle que $|\sigma f_i| = \lambda_i$. Cette solution est obtenue en divisant w' suivant les segments correspondants à la solution en longueurs.

Par exemple, l'équation $fg \Delta H$ donne deux solutions en longueurs :

- i) $|f| = 0 \quad |g| = 1$, qui donne la solution $\sigma f = \varepsilon$, $\sigma g = H$
- ii) $|f| = 1 \quad |g| = 0$, qui donne la solution $\sigma f = H$, $\sigma g = \varepsilon$.

6.2.1.3 Cet algorithme permet donc de générer un nombre fini de solutions à un problème de filtrage strict.

Réciproquement, ce sont là toutes les solutions.

Par exemple, soit $t = fx$

et $t' = HA$.

en choisissant la nouvelle variable fonctionnelle g , on réduit au problème dans F^* : $fg \Delta H$, qui a les deux solutions trouvées plus haut.

On trouve donc deux solutions à $t \Delta t'$:

$$\begin{cases} \sigma_1 = \{\langle fy, y \rangle, \langle x, HA \rangle\} \\ \sigma_2 = \{\langle fy, Hy \rangle, \langle x, A \rangle\}. \end{cases}$$

6.2.2 Filtrage

6.2.2.1 Réduction au filtrage strict

Nous cherchons maintenant des solutions à l'équation

$$t \leq t'.$$

Soit $V(t) \cap F = \{f_1, \dots, f_n\}$, où on suppose que les f_i sont pris dans leur ordre d'occurrence dans t (i.e. la première occurrence de f_1 est à gauche de la première occurrence de f_2 , etc...). Soit w_1, \dots, w_n les parties initiales de t , précédant la première occurrence de f_1, \dots, f_n .

Soit $x \in V_0 - (V(t) \cup V(t'))$. On se ramène aux $n+1$ problèmes :

$$\begin{cases} w_1 x \Delta t' & (1) \\ \dots \dots \dots \\ w_n x \Delta t' & (n) \\ t \Delta t' & (n+1). \end{cases}$$

Toute solution ρ au problème $(n+1)$ est une solution de $t \leq t'$.

Soit ρ une solution au problème (p) , avec $p \leq n$. Alors

$\sigma = \rho \cup \{\langle f_p x, \rho x \rangle\}$ est une solution de $\sigma t = t'$.

Inversement, toute solution σ de $t \leq t'$ qui soit stricte en f_1, \dots, f_{p-1} et non stricte en f_p est solution du problème (p) , avec $p \leq n+1$.

6.2.2.2 Exemple

$$\text{soit } \begin{cases} t = fx \\ t' = HA. \end{cases}$$

On réduit ici le problème $t \leq t'$ à 2 problèmes :

(1) $y \Delta HA$, qui a une solution unique

$$\rho = \{<y, HA>\}$$

(2) $fx \Delta HA$.

Dans le premier cas, on trouve la solution :

$$\sigma_3 = \{<fy, HA>\}.$$

Dans le second, on trouve les solutions σ_1 et σ_2 ci-dessus.

6.2.2.3 On trouve donc toujours un ensemble complet fini de filtres dans L . Nous verrons plus loin comment ce résultat s'étend aux termes de second ordre généraux.

6.2.2.4 Existence de filtre

Supposons que l'on s'intéresse simplement à l'existence d'une solution à $t \leq t'$. Nous avons vu que nous pouvions nous ramener à $(n+1)$ filtrages De plus, remarquons que si le problème $(j+1)$ a une solution σ , alors le problème (j) a une solution également. En effet, σ solution de $(j+1)$ impose en particulier que $t' = ww'$, avec $w = \sigma w_j$.

La substitution $\sigma \setminus \{f_1, \dots, f_{j-1}\} \cup \{<x, w>\}$, est bien une solution de (j) .

En particulier, $t \leq t'$ admet une solution ssi $t \in C^*$ et $t' = t$ ou $w_j x \Delta t'$ admet une solution, c'est à dire si et seulement si t' commence par la partie initiale dans C^* de t .

La condition de filtrage dans L est donc très simple.

Nous allons voir de même qu'il existe un inf et un sup dans L , relativement à \leq , et qu'ils se calculent très simplement.

6.2.3 Inf et sup relativement à \leq

Nous donnons ci-dessous un algorithme de calcul de l'inf $t \wedge t'$ et du sup $t \vee t'$ de deux termes linéaires quelconques t et t' , relativement à l'ordre induit par \leq .

Scit $t, t' \in L$, et soit w le mot initial commun à t et t' maximal.:

$$t = ww_1 \quad t' = ww_2$$

- (i) si $w_1 = \epsilon$ alors $w \in L$, d'où $w_2 = \epsilon$, et $t \wedge t' = t \vee t' = t = t' = w$.
- (ii) si $w_1 = x \in V_0$ ou $w_1 = fw_3$ avec $f \in V_1$, alors $t \leq t'$, et on prend donc $t \wedge t' = t$, $t \vee t' = t'$.
- (iii) sinon, si $w_2 = x$ ou $w_2 = fw_4$, alors cas symétrique.
- (iv) sinon, on prend $t \wedge t' = wx$, avec $x \in V_0$; t et t' n'ont pas de majorant commun.

La preuve de correction de cet algorithme est triviale, nous ne la ferons pas.

Le préordre Δ est singulièrement plus compliqué. Nous allons voir dans les prochains paragraphes que l'on ne peut définir ni un inf ni un sup vis à vis de l'ordre induit par Δ .

6.2.4 Non existence du sup dans $\langle L, \Delta \rangle$

Considérons

$$\begin{cases} t = fFA \\ t' = FgA \end{cases}$$

Il est facile de montrer que l'ensemble des majorants communs de t et t' relativement à Δ est :

$$\{FA, FwFA\}, \text{ où } w \in F^* \text{ quelconque.}$$

Cet ensemble n'ayant pas de plus petit élément ($FfFa \triangleleft FA$), t et t' sont des termes linéaires majorés sans sup.

6.2.5 Non existence de l'inf dans $\langle L, \Delta \rangle$

Considérons

$$\begin{cases} t = FGA \\ t' = GFA \end{cases} \quad \begin{cases} t_1 = fFgA \\ t_2 = fGgA \end{cases}$$

t_1 et t_2 sont deux minorants communs à t et t' .

Soit t_3 un majorant commun à t et t' , $t_3 = \sigma t_1$.

σ doit introduire G soit par f , soit par g . Dans le premier cas t_3 ne peut minorer t , dans second t_3 ne peut minorer t' . Ceci montre bien que t et t' n'admettent pas d'inf relativement à Δ .

Ceci peut se produire même s'il n'y a qu'une fonction constante F , et même si t et t' ont le même nombre d'occurrences de F , comme le montre l'exemple suivant :

$$\begin{cases} t = fFFFgA \\ t' = fFgFgA. \end{cases}$$

Considérons

$$\begin{cases} t_1 = f_1 f_2 F f_2 f_3 F f_3 f_4 A. \\ t_2 = f_1 f_2 F f_3 f_2 F f_3 f_4 A. \end{cases}$$

On a bien : $t = \sigma t_1 = \sigma t_2$

$$\text{et } t' = \sigma' t_1 = \sigma' t_2$$

avec $\sigma = \{\langle f_1 x, x \rangle, \langle f_2 x, f x \rangle, \langle f_3 x, x \rangle, \langle f_4 x, g x \rangle\}$

$$\text{et } \sigma' = \{\langle f_1 x, f x \rangle, \langle f_2 x, x \rangle, \langle f_3 x, g x \rangle, \langle f_4 x, x \rangle\}.$$

Les majorants de t_1 et t_2 qui minorent t et t' ne peuvent substituer à f_1, f_2, f_3, f_4 que ϵ ou les atomes f et g .

Soit t_3 un tel majorant de t_1 et t_2 : $t_3 = \sigma_1 t_1 = \sigma_2 t_2$.

Les conditions initiales et finales imposent

$$\begin{cases} \sigma_1 f_1 f_2 = \sigma_2 f_1 f_2 \\ \sigma_2 f_3 f_4 = \sigma_2 f_3 f_4 \end{cases} \quad \text{d'où } \sigma_1 \equiv \sigma_2 \text{ avec } V = \{f_1, f_2, f_3, f_4\}.$$

Donc au milieu : $\sigma_1 f_2 f_3 = \sigma_1 f_3 f_2$. Il y a 3 cas :

- 1) $\sigma_1 f_2 = \epsilon$; alors t_3 ne peut minorer t .
- 2) $\sigma_1 f_3 = \epsilon$; alors t_3 ne peut minorer t' .
- 3) $\sigma_1 f_2 = \sigma_1 f_3 = f$ ou g ; alors t_3 ne peut minorer ni t ni t' .

Ceci achève bien de prouver que t et t' ne possèdent pas

d'inf (remarquer que $fFgFhA$ est strictement inférieur à t_1 et t_2).

6.2.6 $\langle L, \Delta \rangle$ n'est pas bien fondé

Nous allons maintenant montrer que L possède des chaînes infinies décroissantes strictement, avec l'ordre de filtrage strict.

Considérons la suite de termes :

$$\left\{ \begin{array}{l} t_0 = A \\ t_1 = f_1 f_1 A \\ t_2 = f_2 f_1 f_1 f_2 A \\ \dots \\ t_n = w_n \tilde{w}_n A \quad \text{avec } w_n = f_n f_{n-1} \dots f_1, \\ \dots \\ \tilde{w}_n = f_1 f_2 \dots f_n. \end{array} \right.$$

On a bien $t_n \Delta t_{n-1}$. En effet $t_{n-1} = \sigma_n t_n$,

avec $\sigma_n = \{\langle f_n, x \rangle\}$.

Montrons, par récurrence sur n , que $\forall i < n \quad t_i \not\Delta t_n$.

C'est vrai (trivialement) pour $n=0$. Supposons la propriété vraie pour $n-1$, et montrons la pour n , par récurrence sur i .

C'est vrai pour $i=0$, car $\forall \sigma \sigma A = A$.

Supposons la propriété fausse pour i ; i.e.

$$\exists \rho \in \bar{S} \quad \rho t_i = t_n.$$

Comme $|\rho w_i| = |\rho \tilde{w}_i|$, on doit avoir : $\begin{cases} \rho w_i = w_n \\ \rho \tilde{w}_i = \tilde{w}_n \end{cases}$ (1)

(2).

$$(1) \text{ entraîne : } \rho f_i = \begin{cases} f_n \cdots f_j & j \leq n \\ \epsilon & \end{cases}$$

(2) réduit le premier cas à $j=n$.

$$\text{Mais alors (1) entraîne } \begin{cases} \rho w_{i-1} = w_{n-1} \\ \tilde{\rho} \tilde{w}_{i-1} = \tilde{w}_{n-1} \end{cases} \text{ d'où } \rho t_{i-1} = t_{n-1},$$

ce qui est contraire à l'hypothèse de récurrence sur n .

Dans le deuxième cas : $\rho f_i = \epsilon$, (1) entraîne

$$\begin{cases} \rho w_{i-1} = w_n \\ \tilde{\rho} \tilde{w}_{i-1} = \tilde{w}_n \end{cases} \text{ d'où } \rho t_{i-1} = t_n,$$

ce qui est contraire à l'hypothèse de récurrence sur i .

Ceci complète le pas de récurrence sur i , ce qui complète la récurrence sur n .

On a donc une chaîne infinie strictement décroissante :

$$t_0 \tilde{\Delta} t_1 \tilde{\Delta} \cdots \tilde{\Delta} t_n \tilde{\Delta} \cdots$$

où $t \tilde{\Delta} t' \Leftrightarrow t' \Delta t$ et $t \not\Delta t'$.

Par contre, toute chaîne strictement décroissante avec le préordre \leq issue de t est de longueur au plus $|w_1|$, où w_1 est la chaîne initiale de t maximale dans C^* . $\langle L, \leq \rangle$ est donc un ITF particulièrement simple, l'ordre étant l'ordre préfixiel sur les segments constants initiaux.

6.2.7 Structure des termes linéaires par l'ordre de filtrage

Nous pouvons rassembler nos résultats sur la structure de L muni des deux ordres de filtrage par les lemmes :

Lemme 6.3 Dans L muni de \leq :

- deux termes quelconques admettent un inf.
- deux termes majorés admettent un sup.
- il n'y a pas de chaîne infinie strictement décroissante.

Lemme 6.4 Dans L muni de Δ :

- certains termes t, t' n'admettent pas d'inf.
- certains termes t, t' majorés n'admettent pas de sup.
- il existe des chaînes infinies strictement décroissantes.

Nous allons maintenant étudier le problème de l'unification des termes linéaires, d'abord dans le cas strict, puis dans le cas non strict. Nous montrons tout d'abord que l'unification stricte est un problème interréductible au problème de Markov.

6.2.8 Unification stricte

Soit à unifier strictement deux termes quelconques de L t et t' . Nous allons montrer comment ramener ce problème à un problème dans F^* . On commence donc à examiner les atomes terminaux de t et t' .

Il y a 6 cas :

1) $t = wA \quad t' = w'B \quad A$ et B distincts.

t et t' ne sont pas unifiables.

2) $t = wA \quad t' = w'A$.

Alors $\sigma t = \sigma t'$ avec $\sigma \in \overline{S}$ si et seulement si

$\sigma w = \sigma w'$, où w est maintenant considéré comme un

endomorphisme du monoïde F^* .

(i.e. $\sigma f = w_f$ correspondant à $\langle fu, w_f u \rangle \in \sigma$)

3) $t = wx \quad t' = w'A$.

Alors soit f_x une variable dans $F \cap V$ n'apparaissant pas dans t ou t' .

On a $\sigma t = \sigma t'$ si et seulement si

$\sigma = \rho \cup \{\langle x, (\rho f_x) A \rangle\}$, où ρ est maintenant solution de $wf_x = w'$.

4) $t = wA \quad t' = w'y$ cas symétrique à 3).

5) $t = wx \quad t' = w'x$ comme en 2).

6) $t = wx \quad t' = w'y$. On se ramène à 5) en substituant d'abord f_y à x ,

puis f_y à y , ce qui fait deux cas séparés.

Dans tous les cas, on se ramène donc à résoudre une équation $w = w'$, avec $w, w' \in F^*$, une solution étant un endomorphisme ρ de F^* tel que $\rho w = \rho w'$. (Plus exactement, ρ est le prolongement unique par morphisme de monoïde d'une application de F dans F^* , identité sur C).

La réduction inverse est immédiate (avec 2) ci-dessus).

La résolution d'équation dans le monoïde libre avec constantes est un problème très difficile. La décidabilité de l'existence de solutions à de telles équations est encore un problème ouvert (généralement connu sous le nom de problème de Markov). On connaît des algorithmes de décision dans quelques cas particuliers.

Soit c le nombre de constantes distinctes apparaissant dans ww' , v le nombre de variables. Les cas particuliers suivants sont décidables :

- 1) $c = 0$ la solution triviale ϵ est toujours solution.
- 2) $c = 1$ (cas commutatif) il suffit de résoudre l'équation aux longueurs.
- 3) $v = 1$ on peut montrer que l'ensemble des solutions est un langage rationnel (voir [RG1]).
- 4) $v = 2$ un algorithme a été découvert par Markov [MA1].
- 5) $v = 3$ un algorithme a été publié par Ju.I. Hmelevskii (voir [HJ1], [HJ2], [HJ3]).

Le cas $c > 2$ avec $v > 3$ est encore ouvert.

Nous donnerons en 6.4.2 un algorithme d'unifiabilité, dérivé de celui défini au chapitre 3. Cet algorithme peut être utilisé pour la semi-décision de telles équations, en se restreignant aux solutions strictes.

6.2.9 Unification

Soit à unifier deux termes quelconques de L t et t' .

Nous allons tout d'abord montrer comment on peut se ramener au problème de l'unification stricte.

6.2.9.1 Réduction à l'unification stricte

L'idée est la même que celle utilisée en 6.2.2.1.

Soit $L = V(t) \cup V(t')$, $L \cap F = \{f_1, \dots, f_n\}$.

On considère les 2^n substitutions :

$$\sigma_I = \{\langle f_i x, z_i \rangle \mid i \in I\},$$

où I est un sous-ensemble quelconque de $\{1, 2, \dots, n\}$, et les z_i sont des variables dans $V_0 - L$.

Pour chacune de ces substitutions, on considère le problème d'unification stricte de $\sigma_I t$ et $\sigma_I t'$.

Toute solution au I ème problème, composée avec σ_I , donnera un unificateur de t et t' . Réciproquement, tout unificateur σ de t et t' qui est strict sur $\{f_i \mid i \notin I\}$ et non strict sur $\{f_i \mid i \in I\}$ peut s'écrire sous la forme :

$$\sigma_L = \rho \sigma_I.$$

Ceci achève bien la preuve de la réduction de l'unification de t et t' à 2^n problèmes d'unification stricte.⁽⁺⁾

On obtient en particulier que si le problème d'équations dans les monoïdes est décidable, alors l'existence d'unificateur à des termes linéaires l'est aussi.

Nous n'avons pas réussi à effectuer la réduction inverse.

La difficulté réside dans l'impossibilité apparente de forcer une condition du genre "f est stricte" par une contrainte d'unification linéaire. Il est par contre aisément d'obtenir l'équivalence du problème d'unification et du problème de résolution d'équations dans le monoïde libre complété par un élément "zéro" (absorbant à droite).

(+) En fait, le nombre de problèmes distincts est au plus de

$$\binom{n}{0} + \binom{n}{1} + \binom{n}{2} = 1 + \frac{n(n+1)}{2}$$

Le problème d'unification peut donc être décidable, sans que les équations dans le monoïde le soient. Mais nous allons donner dans le prochain paragraphe une indication de la complexité du problème d'unification, en montrant que la longueur minimale d'un unificateur ne peut être bornée par un polynôme en la longueur des termes à unifier.

6.2.9.2 Complexité minimale des solutions

Définition : Soit $t, t' \in L$, $\sigma \in U(t, t')$. On appelle complexité de σ l'entier : $\max \{ |w| \mid \langle f_x, w \rangle \in \sigma \text{ et } f \in V(t) \cup V(t')\}$.

Lemme 6.5

Il n'existe pas d'entiers p et q tels que, pour tous termes t et t' dans L , si t et t' sont unifiables, alors il existe un unificateur σ de complexité inférieure à $p\Lambda^q$, avec $\Lambda = |t| + |t'|$.

Démonstration

Considérons :

$$\begin{cases} t = f_1 A f_2 A \cdots f_m A f_1 B f_2 B \cdots f_{m-1} C \\ t' = A f_1 A f_2 \cdots A f_m f_2^n B f_3^n B \cdots f_m^n C \end{cases}$$

où par exemple $f_2^n = \underbrace{f_2 f_2 \cdots f_2}_n$. Les termes

t et t' sont unifiables, par exemple par la substitution
 $\sigma = \{\langle f_1 x, A^{n^{m-1}-1} x \rangle\} \cup \{\langle f_i x, A^{n^{m-i}} x \rangle \mid 1 < i \leq m\}$.

Il est facile de montrer que σ est l'unificateur de t et t' de plus petite complexité. En effet, soit ρ un unificateur quelconque de t et t' . On prouve tout d'abord que $\forall i \leq m \quad \rho$ est stricte en f_i .

[†] $U(t, t')$ est l'ensemble des unificateurs, stricts ou non, de t et t' .

En effet, si ρ n'était pas stricte en f_1 , c'est à dire si : $\langle f_1 x, t_1 \rangle \in \rho$, avec $x \notin V(t_1)$, alors $\rho t = t_1 \neq A t_1 = \rho t'$, en contradiction avec l'hypothèse $\rho \in U(t, t')$. Donc $\langle f_1 x, w x \rangle \in \rho$, et $\rho t = \rho t'$ entraîne en particulier $wA = Aw$. Ceci implique $w \in A^*$, par récurrence sur $|w|$. Mais $\rho t = \rho t'$ entraîne aussi $\rho(f_2 A \cdots) = \rho(A f_2 \cdots)$ et de proche en proche on obtient bien $\langle f_i x, w_i x \rangle \in \rho$, avec $w_i \in A^*$.

Le reste de l'équation entraîne :

$$\begin{cases} w_i = w_{i+1}^n & 1 \leq i < m \\ Aw_1 = w_2^n \end{cases}$$

La solution $w_i = \epsilon$ ne convenant pas, la plus petite solution est : $w_m = A$, $w_{m-1} = A^n, \dots$, i.e. $\rho = \sigma$.

La complexité de σ est $C = |w_1 x| = n^{m-1}$. Ici

$$\Lambda = |t| + |t'| = 4m + (n+3)(m-1) + 1.$$

Il suffit de choisir $m = q+2$, et n tel que

$$n^{q+1} > p [(q+1)n + 7q + 12]^q, \text{ pour avoir } C > p \Lambda^q. \square$$

Ce résultat indique en particulier qu'il n'est pas possible de borner la longueur des branches d'un arbre d'unifiabilité de t et t' par un polynôme en $|t| + |t'|$.

Nous venons de voir que, même dans le cas des termes linéaires, l'unification était un problème très difficile.

Nous allons maintenant considérer l'ensemble T_1 des termes monadiques, en supposant l'existence d'au moins une constante d'arité > 1 .

6.3 Termes monadiques

Nous allons tout d'abord montrer que T_1 , muni de l'ordre de filtrage \leq , a une structure aussi compliquée que $\langle L, \Delta \rangle$.

6.3.1 Lemme 6.6

Soit $f, g \in V_1$, A et B deux constantes zéro-aires distinctes. Pour toutes substitutions σ et σ' :

$$\text{si } \sigma fA = \sigma' gA$$

$$\text{et } \sigma fB = \sigma' gB,$$

$$\text{alors } \forall t \in T \quad \sigma ft = \sigma' gt.$$

Démonstration

Soit σ et σ' répondant à l'hypothèse, et soit $t \in T$. σ doit posséder une composante relative à f , soit $\langle fu, t_1 \rangle \in \sigma$.

De même, soit $\langle g, v, t_2 \rangle \in \sigma'$.

Par définition,

$$\left\{ \begin{array}{l} \sigma fA = \rho_1 t_1 \quad \text{avec } \rho_1 = \{\langle u, A \rangle\} \\ \sigma fB = \rho_2 t_1 \quad \text{avec } \rho_2 = \{\langle u, B \rangle\} \\ \sigma' gA = \eta_1 t_2 \quad \text{avec } \eta_1 = \{\langle v, A \rangle\} \\ \sigma' gB = \eta_2 t_2 \quad \text{avec } \eta_2 = \{\langle v, B \rangle\}. \end{array} \right.$$

Il est facile de montrer, par récurrence sur t_1 et t_2 , que t_1 et t_2 sont identiques, au renommage près de u par v :

$$t_1 = \sigma_1 t_2 \quad \text{et} \quad t_2 = \sigma_2 t_1 \quad \text{avec } \sigma_1 = \{\langle v, u \rangle\} \quad \sigma_2 = \{\langle u, v \rangle\}.$$

On en tire facilement que pour tout t $\sigma ft = \sigma' gt$. Nous ne faisons pas ici les preuves complètes qui seraient plus aisées dans la notation du λ -calcul. \square

Ce lemme nous permet de ramener un problème d'existence de sup dans T_1 à un problème d'unification, comme nous allons le voir.

6.3.2 Non existence du sup dans $\langle T_1, \leq \rangle$

Nous allons nous inspirer de l'exemple de 6.2.4.

Considérons :

$$\begin{cases} t = H(f_A, f_B, f_{FC}) \\ t' = H(g_A, g_B, Fg_C) \end{cases}$$

En utilisant le lemme 6.6, on montre facilement que l'ensemble des majorants de t et t' est :

$$\{H(w_A, w_B, Fw_C) \mid w \in F^*\}.$$

Cet ensemble n'ayant pas de plus petit élément, t et t' sont des termes majorés sans sup.

6.3.3 Non existence de l'inf dans $\langle T_1, \leq \rangle$

Nous nous inspirons de l'exemple du 6.2.5, en le compliquant pour éliminer les solutions non strictes.

Soit :

$$\begin{cases} t = H(FGA, FGA, GFA) \\ t' = H(GFA, FGA, GFA) \end{cases}$$

$$\begin{cases} t_1 = H(f_1 Fg_1 A, Ff_1 g_1 A, f_1 g_1 FA) \\ t_2 = H(f_2 Gg_2 A, f_2 g_2 GA, Gf_2 g_2 A) \end{cases}$$

t_1 et t_2 sont des minorants de t et t' :

$$\begin{cases} t = \sigma_1 t_1 = \sigma_2 t_2 \\ t' = \sigma'_1 t_1 = \sigma'_2 t_2 \end{cases} \quad \text{avec}$$

$$\sigma_1 = \{<f_1 u, u>, <g_1 u, Gu>\}$$

$$\sigma_2 = \{<f_2 u, Fu>, <g_2 u, u>\}$$

$$\sigma'_1 = \{<f_1 u, Gu>, <g_1 u, u>\}$$

$$\sigma'_2 = \{<f_2 u, u>, <g_2 u, Fu>\}.$$

t_1 peut être considéré comme une "abstraction en F" de t et t' , et t_2 comme une "abstraction en G". Ces deux abstractions sont incompatibles, comme nous allons le montrer.

Soit $t'' = \rho_1 t_1 = \rho_2 t_2$ un majorant commun à t_1 et t_2 .

On montre, par une analyse par cas fastidieuse mais facile, que ρ_1 et ρ_2 doivent être strictes (il suffit de regarder les conditions aux longueurs).

Posons :

$$\rho_1 = \{<f_1 u, w_1 u>, <g_1 u, w_2 u>\}$$

$$\rho_2 = \{<f_2 u, w_3 u>, <g_2 u, w_4 u>\},$$

et écrivons $\rho_1 t_1 = \rho_2 t_2$. On doit avoir :

$$\begin{cases} w_1^F w_2 = w_3^G w_4 & (1) \\ F w_1 w_2 = w_3 w_4^G & (2) \\ w_1 w_2^F = G w_3 w_4 & (3) \end{cases}$$

(1) impose $w_1 w_3 \neq \epsilon$. Si $w_3 \neq \epsilon$, alors (2) impose que w_3 commence par F, donc $t'' = H(F \dots, \dots, \dots)$ ne minore pas t' . Si $w_1 \neq \epsilon$, alors (3) impose que w_1 commence par G, donc $t'' = H(G \dots, \dots, \dots)$ ne minore pas t . Il n'existe donc pas de majorant commun à t_1 et t_2 qui minore t et t' . Ceci achève de montrer que t et t' ne possèdent pas d'inf relativement à \leq .

Remarquons que t et t' n'admettent pas d'inf dans T , et non seulement dans T_1 . Cet exemple montre que dans T il y a des manières incompatibles de généraliser (ou abstraire, ou induire) à partir de deux termes. Donnons un autre exemple, sans preuve cette fois :

$$\begin{cases} t = H(FFA, FA) \\ t' = H(GGB, GB) \end{cases} \text{ sont deux termes de } T_0$$

qui possèdent comme minorants communs, en particulier :

$$t_1 = H(ffx, fx) \quad \text{dans } T_1$$

et $t_2 = H(h(FA, GB), h(A, B))$ dans T_2 (avec $h \in V_2$) qui sont incompatibles.

6.3.4 $\langle T_1, \leq \rangle$ n'est pas bien fondé

Nous allons maintenant montrer que T_1 , muni de l'ordre \leq de filtrage, possède des chaînes infinies strictement décroissantes.

Soit $\{f_1, f_2, \dots\} \subset V_1$, A et $F \in C$, avec $\alpha(A) = 0$ et $\alpha(F) = 2$.

On considère la suite de termes de T_1 :

$$\left\{ \begin{array}{l} t_0 = F(A, A) \\ t_1 = F(f_1 A, f_1 f_1 A) \\ t_2 = F(f_2 f_1 A, f_2 f_1 f_1 f_2 A) \\ \dots \\ t_n = F(w_n A, \tilde{w}_n \tilde{w}_n A) \quad \text{avec } w_n = f_n f_{n-1} \dots f_1, \\ \dots \\ \tilde{w}_n = f_1 \dots f_n. \end{array} \right.$$

On a bien $t_n \leq t_{n-1}$. En effet $t_{n-1} = \sigma_n t_n$, avec

$$\sigma_n = \{\langle f_n x, x \rangle\}.$$

Montrons maintenant que $t_n \neq t_{n+1}$. Pour cela, supposons qu'au contraire il existe σ tel que $t_{n+1} = \sigma t_n$.

σ ne peut être une substitution stricte, car sinon on aurait

$$w_n \tilde{w}_n A \triangleq w_{n+1} \tilde{w}_{n+1} A, \text{ ce qui est impossible (voir preuve en 6.2.6).}$$

Donc , en posant

$$t = \sigma w_n A, \text{ on a}$$

$$\sigma t_n = F(t, t), \text{ qui ne peut donc être égal à } t_{n+1}.$$

Ceci achève de prouver que $t_n \neq t_{n+1}$, et donc

$$t_0 > t_1 > \dots > t_n > \dots$$

Nous pouvons maintenant rassembler nos résultats sur la structure de $\langle T_1, \leq \rangle$:

6.3.5 Lemme 6.7

Dans T_1 muni de \leq :

- certains termes t, t' n'admettent pas d'inf.
- certains termes t, t' majorés n'admettent pas de sup.
- il existe des chaînes infinies strictement décroissantes.

Nous voyons donc que la structure de $\langle T, \leq \rangle$ est compliquée, même en se limitant aux termes monadiques.

En ce qui concerne l'unification, la situation ici n'est guère plus compliquée que dans L , et guère plus simple que dans l'ensemble complet des termes de second ordre. Nous allons donc passer directement à la situation générale, au paragraphe 6.4.

Avant cela, montrons que l'unification stricte dans T_1 permet de résoudre l'unification dans un langage de premier ordre, en présence d'un axiome d'associativité.

6.3.6 Relations avec l'unification associative

Soit A un langage de termes de premier ordre, dans lequel on distingue une constante binaire H . Nous supposons que H obéit à un axiome d'associativité :

$$H(x, H(y, z)) = H(H(x, y), z). \quad (!).$$

Le problème de l'unification associative consiste à rechercher les solutions d'une équation

$$t = t' \quad \text{dans } A,$$

en présence de l'axiome (!).

Ce problème est un cas particulier de l'unification dans des langages de premier ordre munis d'un ensemble d'identités, problème étudié par Plotkin dans [PG2].

Nous allons maintenant montrer que ce problème se réduit à l'unification stricte dans un langage de second ordre monadique \mathcal{B} , que nous allons construire.

Supposons que A soit construit à partir d'un ensemble V de variables, et d'un ensemble C de constantes. On construit \mathcal{B} à partir de :

$$\begin{cases} V_0 = \{u\} \\ V_1 = \{f_x \mid x \in V\} \\ \bar{C} = \{\bar{F} \mid F \in C - \{H\}\} \cup \{K\}, \end{cases}$$

$$\text{avec } \alpha(\bar{F}) = \alpha(F) + 1 \text{ et } \alpha(K) = 0.$$

Pour tout t dans \mathcal{B} , on dénote par \tilde{t} le terme fermé de \mathcal{B} défini par : $\tilde{t} = S_K^u[t]$.

On définit une application ϕ de A dans \mathcal{B} , par :

- (i) $\phi(x) = f_x u \quad \forall x \in V$
- (ii) $\phi(F(t_1, \dots, t_n)) = \bar{F}(\tilde{\phi}(t_1), \dots, \tilde{\phi}(t_n), u) \quad \forall F \in C - \{H\}$
- (iii) $\phi(H(t_1, t_2)) = S_{\phi(t_2)}^u[\phi(t_1)].$

Par exemple :

$$\phi(H(F(A), B)) = \bar{F}(\bar{A}(K), \bar{B}(u))$$

$$\phi(H(A, H(B, C))) = \phi(H(H(A, B), C)) = \bar{A}(\bar{B}(\bar{C}(u))).$$

D'une manière générale, on peut prouver que

$\phi(t) = \phi(t') \iff t \simeq t'$, où \simeq est la congruence engendrée par (i).

Les termes $\phi(t)$ sont des termes particuliers de \mathcal{B} : ils possèdent une et une seule occurrence de u , dans la position la plus à droite la plus interne.

L'idée de la construction est naturellement de prendre en compte l'associativité de la fonction H par l'associativité de la composition des fonctions monadiques.

On étend ϕ aux substitutions sur A comme suit :

Soit σ une substitution de A , de domaine $D(\sigma)$. On définit :

$$\phi(\sigma) = \{ \langle f_x u, \phi(\sigma x) \rangle \mid x \in D(\sigma) \}.$$

Montrons maintenant que ϕ est un isomorphisme de A/\sim sur $\phi(A) \subset \mathcal{B}$. Il nous faut montrer que le diagramme suivant commute :

$$\begin{array}{ccc} t & \xrightarrow{\phi} & \phi(t) \\ \downarrow \sigma & & \downarrow \phi(\sigma) \\ t' & \xrightarrow{\phi} & \phi(t') \end{array}$$

c'est à dire que $\forall t, \sigma \quad \phi(\sigma) \phi(t) = \phi(\sigma t)$.

Nous laissons au lecteur la preuve de cette propriété, qui se montre facilement par récurrence sur t .

Soit maintenant t et t' des termes de A quelconques, σ une substitution telle que $\sigma t \simeq \sigma t'$. Alors $\phi(\sigma)$ est un unificateur de $\phi(t)$ et $\phi(t')$. Réciproquement, soit ρ un unificateur de $\phi(t)$ et $\phi(t')$ tel qu'il existe σ , avec $\phi(\sigma) = \rho$. Alors on a $\sigma t \simeq \sigma t'$.

On a donc bien ramené le problème d'unification dans A , modulo l'axiome (!), à un problème d'unification dans le langage de second ordre monadique B .

Il faut naturellement prendre garde à ce que les solutions dans B soient inversibles par ϕ . En particulier, remarquons que $\phi(\sigma)$ est toujours une substitution stricte.

Plus précisément, si on désire unifier $\phi(t)$ et $\phi(t')$ en construisant un API de ces deux termes, alors on peut montrer qu'on peut imposer à la procédure CHOIX de ne construire que les composantes :

i) d'imitation

$$\langle fu, \bar{F}(z_1, \dots, z_n, hu) \rangle$$

ii) de projection

$\langle hu, u \rangle$ que l'on peut restreindre aux variables h introduites.

Nous ne donnons pas ici les détails de la construction, Plotkin ayant donné dans [PC2] un algorithme qui permet d'énumérer un ECTM de deux termes quelconques dans A/\simeq . Notre intention était simplement de montrer la liaison entre les deux problèmes, qui sont en fait deux versions voisines du problème des équations dans le monoïde libre avec constantes.

Remarquons enfin que ce problème est sous-jacent à de nombreux problèmes informatiques. Par exemple, le problème de l'appel des

procédures par "pattern matching" dans le langage QA4, en présence de variables de listes, est une instance de ce problème.

Lorsque l'on cherche à unifier des termes modulo un ensemble arbitraire d'axiomes, on aboutit généralement à résoudre des équations dans la structure algébrique libre correspondante. Par exemple, si la fonction H ci-dessus obéit également à l'axiome de commutativité, on doit résoudre des équations dans le semi-groupe commutatif libre. On peut alors toujours trouver un ensemble complet fini de solutions minimales [SM1].

Des problèmes analogues sont traités dans la thèse de M. Stickel [SM2].

6.4 Unification dans \bar{T}

6.4.1 L'ensemble complet \bar{T} .

Nous complétons T en \bar{T} en introduisant des variables liées. Plus précisément, \bar{T} est le plus petit ensemble contenant T et fermé par l'opération :

$$t \longrightarrow \lambda x t \quad x \in V.$$

Comme pour le λ -calcul, nous utiliserons l'abréviation $\lambda x_1 \cdots x_n \cdot t$ pour $\lambda x_1 \lambda x_2 \cdots \lambda x_n t$. De fait, \bar{T} est la restriction au second ordre, de l'ensemble T du chapitre 4⁽⁺⁾.

L'ensemble des substitutions S est inchangé. Mais nous écrirons de même $\langle f, \lambda u_1 \cdots u_n \cdot t \rangle$ au lieu de $\langle f u_1 \cdots u_n, t \rangle$.

(+) Ceci n'est pas tout à fait vrai. Pour obtenir les termes de second ordre du genre $A(\lambda x \cdot e)$, il faudrait définir des types, l'arité des atomes ne suffisant pas pour déterminer si un terme est bien formé. Nous ne le ferons pas ici par simplicité, mais tous les résultats qui suivent restent vrais.

Nous allons tout d'abord montrer comment la construction d'arbres de préunification, comme définie au chapitre 3, peut se simplifier au second ordre.

6.4.2 Arbres de préunification

6.4.2.1

Nous allons définir et construire des arbres de préunification, à l'aide des procédures SIMPL et CHOIX données au chapitre 3. Mais ici un certain nombre de simplifications s'imposent.

Tout d'abord, nous sommes dans le cas où la n -règle est valide, ce qui nous permet d'utiliser les simplifications déjà définies en 4.4.

Ensuite, la règle de projection de CHOIX est considérablement simplifiée. En effet, les projections sont ici de la seule forme

$$\langle f, \lambda w_1 \dots w_{p_1} . v_i \rangle \quad \text{avec } 1 \leq i \leq p_1.$$

(i.e. il n'y a plus d'arguments E_j).

Enfin, un certain nombre d'heuristiques sont applicables.

Une heuristique particulièrement importante concerne le cas où le deuxième terme e_2 de la paire donnée à CHOIX est de la forme

$$\lambda v_1 \dots v_n . v_m (\dots) \quad \text{avec } 1 \leq m \leq n.$$

Dans ce cas l'imitation n'est pas possible, seule la règle de projection est applicable. En itérant cette condition, on voit donc que cette paire ne sera simplifiée que si la variable liée de e_1 correspondante u_m apparaît dans l'argument de e_1 sur lequel on se projette.

Autrement dit, il est suffisant de se limiter aux projections

$\langle f, \lambda w_1 \dots w_i p_j \cdot w_i \rangle$ où i est tel que $u_m \in V(e_i^1)$ (avec les notations de 3.3.1).

Cette règle permet dans certains cas de limiter considérablement le nombre de noeuds engendrés.

Une autre heuristique concerne le cas où la variable de tête du premier terme e_1 appartient à V_0 , i.e.

$$e_1 = \lambda u_1 \dots u_n \cdot x.$$

Dans ce cas seule l'imitation est possible, et dans le noeud suivant on est dans la même situation pour toutes les places d'arguments de la tête de e_2 . On peut exécuter en une fois une "imitation itérée" comme suit. Ecrivons e_2 sous la forme :

$$e_2 = \lambda v_1 \dots v_n \cdot E \langle e_1, \dots, e_k \rangle,$$

où E est un contexte ne contenant que des constantes, et où

$$E_j = f_j E_j^1 \dots E_j^{\alpha(f)} \quad \text{avec} \quad f_j \in V, \quad 1 \leq j \leq k.$$

Maintenant, si

$\{f_1, \dots, f_k\} \cap \{v_1, \dots, v_n, x\} \neq \emptyset$, alors retourner échec ('E'), sinon retourner la composante

$$\langle x, E \langle y_1, \dots, y_k \rangle \rangle,$$

où les y_j sont des variables distinctes dans $V_0 - V$.

La justification de cette règle est évidente :

- si $f_j = v_k$ il y aurait un noeud échec dans l'AP après un certain nombre d'imitations.
- si $f_j = x$ il y aurait une branche infinie d'imitations.

(les deux phénomènes ci-dessus pouvant se combiner entre eux et avec des échecs dus à d'autres causes)

- sinon il y aurait une cascade d'imitations, dont la composition des composantes formerait justement la composante ci-dessus.

Cette heuristique permet en particulier d'éviter les bouclages dûs uniquement aux variables élémentaires (dans V_0).

Donnons maintenant la description algorithmique complète de l'algorithme CHOIX simplifié, que nous appellerons CHOIX 2.

6.4.2.2 Description algorithmique de CHOIX 2 (e_1, e_2, V).

Soit

$$\begin{cases} e_1 = \lambda u_1 \cdots u_n \cdot f(e_1^1, \dots, e_{p_1}^1) \\ e_2 = \lambda v_1 \cdots v_n \cdot @ (e_1^2, \dots, e_{p_2}^2) \quad n, p_1, p_2 \geq 0 \end{cases}$$

(i) si $f \in V_0$ (et donc $p_1=0$), alors effectuer l'imitation itérée décrite au paragraphe précédent (1 ou 0 solution), sinon aller en (ii).

(ii) imitation ; cette règle est applicable seulement si $@ \in C$:

si $@ = v_i \quad 1 \leq i \leq n$, aller en (iii).

Cette règle donne une solution, la composante :

$\langle f, \lambda w_1 \cdots w_{p_1} \cdot @ (E_1, \dots, E_{p_2}) \rangle$
 avec $E_i = h_i(w_1, \dots, w_{p_1}) \quad 1 \leq i \leq p_2$, où les h_i sont des variables de V_{p_1} non dans V .

(iii) projection ; cette règle est applicable seulement si $p_1 > 0$.

Elle donne p_1 solutions, toutes les composantes

$\langle f, \lambda w_1 \cdots w_{p_1} \cdot w_i \rangle, \text{ avec } 1 \leq i \leq p_1$.

Toutefois, si $@ = v_m$, on se limite aux solutions telles que $u_m \in V(e_i^1)$.

Fin de CHOIX 2. \square

Cet algorithme est considérablement plus simple que l'algorithme CHOIX original de 3.3. Il est de plus beaucoup plus dirigé (et donc efficace) que l'algorithme de Pietrzykowski pour le même langage ([PT1]) ; ceci est dû à la même raison que précédemment : la recherche de préunificateurs est beaucoup moins complexe que la recherche d'un ensemble complet d'unificateurs.

La construction d'un arbre de préunifications (AP) s'opère exactement comme en 3.4. On peut de même définir des arbres de préunification indépendants (API) comme en 3.5. Il n'y a pas lieu ici de considérer la construction d'arbres d'unifiabilité comme en 3.6, car, comme dans le cas du λ - η -calcul, nous avons supprimé la redondance correspondante au niveau de l'algorithme CHOIX.

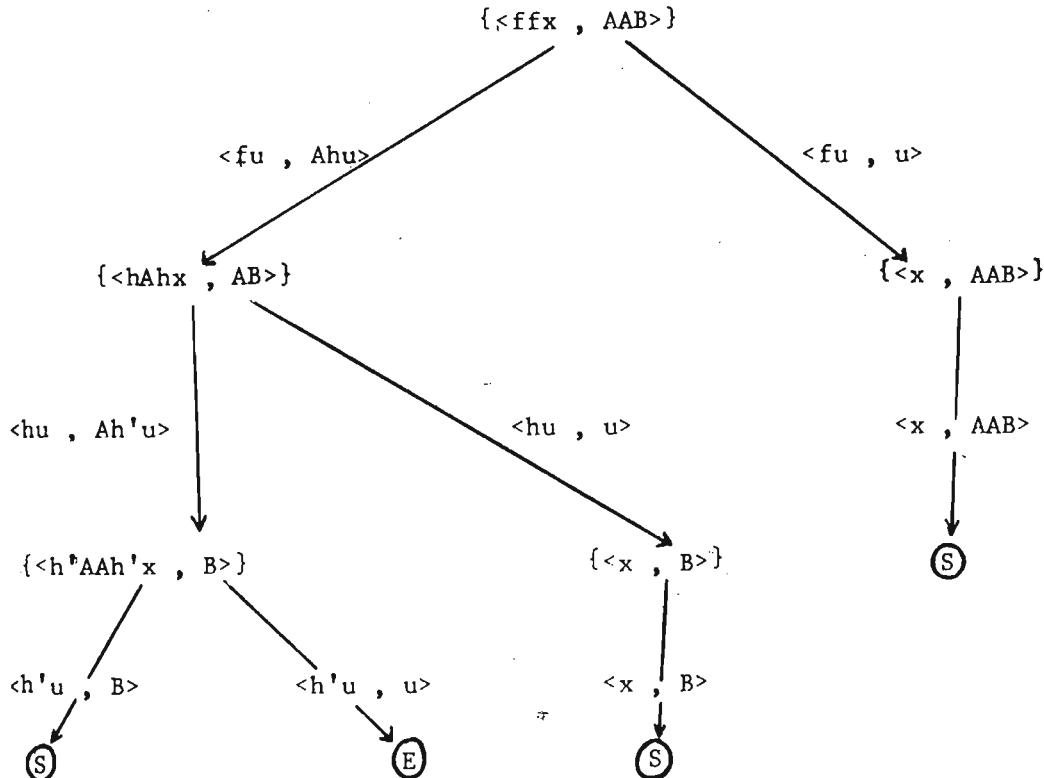
L'algorithme CHOIX 2 étant obtenu comme la restriction aux termes de second ordre de l'algorithme CHOIX pour le λ - η -calcul, l'équivalent du théorème 12 nous permet d'affirmer que l'ensemble des solutions $\Sigma(A)$ obtenues aux noeuds terminaux succès d'un API de t et t' construit à l'aide de la procédure CHOIX 2 est un ECPI de t et t' .

(Il suffit de justifier séparément l'heuristique concernant le cas (i) comme indiqué en 6.4.2.1, sans difficultés).

6.4.2.3 Exemple

Utilisons l'exemple 1 de 3.4.2.1 pour montrer un arbre de préunification construit à l'aide de la procédure CHOIX 2.

$$t = ffx, t' = AAB.$$



6.4.3. Demi-unification

Nous allons montrer que, pour deux termes quelconques t et t' de \bar{T} , il existe toujours un ECDM fini de t et t' . Pour cela, nous commençons par examiner le problème de l'unification de t et t' , lorsque $V(t') = \emptyset$.

6.4.3.1 Unification de t et t' , lorsque $V(t') = \emptyset$.

Nous sommes ici dans le même cas qu'en 3.7.1. Le lemme 3.15 est toujours valable : tout API de t et t' détermine un ECUI de t et t' . Mais ici nous pouvons montrer de plus que la construction termine toujours, et qu'on obtient donc un ECUI fini.

La preuve de terminaison est simple ; elle est basée sur la forme simplifiée des composantes de projection.

Définissons $\pi(t)$, pour $t \in \bar{T}$, comme en 1.8. Pour un unificande N , définissons de même $\pi_1(N)$ et $\pi_2(N)$ comme en 3.1.1. Soit A un API quelconque de t et t' , N un noeud quelconque de A , et N' un successeur de N dans A . Il y a deux cas :

(i) N' est obtenu de N par la règle d'imitation (ou d'imitation itérée). Alors il est aisément de constater que $\pi_2(N') < \pi_2(N)$. (En effet, l'atome de tête $@$ du terme e_2 imité sera éliminé par la procédure SIMPL).

(ii) N' est obtenu de N par la règle de projection. Alors dans ce cas on constate que $\pi_1(N') < \pi_1(N)$, alors que $\pi_2(N') \leq \pi_2(N)$.

Par récurrence sur $\pi_1(N) + \omega \cdot \pi_2(N)$ on montre donc facilement que toute branche de A doit terminer. En combinant avec le lemme 3.15, on obtient donc :

Lemme 6.8 Soit t et t' dans \bar{T} , avec $V(t') = \emptyset$. Soit A un API quelconque de E et E' sur V , avec $V(t) \subset V$. Alors A est fini, et $\Sigma(A)$ est un ECUI fini de t et t' sur V .

Ce lemme a une application informatique importante concernant les transformations de programme. Il est possible d'utiliser les termes de \bar{T} pour décrire des schémas de programmes, les termes fermés correspondant aux programmes. On peut décrire des transformations de programmes par des paires

de schémas équivalents $\langle t, t' \rangle$. Un programme P peut se transformer par une telle paire, en un programme P' si et seulement si

$$P = E\langle t \rangle \text{ et } \exists \sigma (t'' = \sigma t \text{ et } P' = E\langle \sigma t' \rangle).$$

Le lemme ci-dessus nous indique que pour tout programme P et pour toute paire $\langle t, t' \rangle$, il y a un nombre fini de transformations que l'on peut effectuer, qui peuvent être effectivement déterminées en construisant les API correspondants.

6.4.3.2 Demi-unification de t et t'

De même qu'en 3.7.2, on peut construire, pour tous termes t et t' , un ensemble complet de demi-unificateurs minimaux (ECDM) de t vers t' , en utilisant le résultat précédent.

L'équivalent du lemme 3.16 est toujours vrai, et le théorème 11 peut donc s'énoncer :

Théorème 18

Soit $t, t' \in T$, et $L = V(t) \cup V(t')$. Il existe un ECDM Σ de t vers t' sur L , qui est fini, et unique à un isomorphisme près.

La construction de Σ étant effective, on obtient naturellement que la demi-unification est décidable dans \bar{T} .

6.5. Conclusion

L'unification dans les langages de second ordre est un problème difficile, qui a des rapports étroits avec le problème de la résolution d'équations dans un monoïde libre avec des constantes.

La décidabilité de l'existence d'unificateurs est encore un problème ouvert, sauf dans des cas particuliers assez restreints. Par contre, la demi-unification est décidable, et on peut toujours trouver un ensemble fini de solutions minimales.

En ce qui concerne la recherche d'un ensemble complet d'unificateurs, il est conjecturé qu'on peut toujours imposer à un tel ECU la condition de minimalité. Remarquons que Plotkin a donné dans [PG1] un algorithme qui génère l'ECUM de deux termes quelconques de premier ordre, en présence d'un axiome d'associativité, problème voisin, comme nous l'avons vu. Par contre, l'algorithme donné par Pietrzykowski [PT1] génère des solutions redondantes.

CONCLUSION

1. Résultats obtenus, problèmes ouverts

Nous avons étudié dans ce travail trois types d'équations aux termes :

- l'unification : $U = \{\sigma \mid \sigma E = \sigma E'\}$
- la demi-unification : $D = \{\sigma \mid \sigma E = E'\}$
- la pré-unification : $P = \{\sigma \mid \sigma E \sim \sigma E'\}.$

Pour ces trois problèmes, nous avons étudié les propriétés d'ensembles complets de solutions, en particulier l'existence d'ensembles minimaux, indépendants ou principaux, l'existence d'ensemble complet de solutions fini et enfin la décidabilité de chacun des problèmes. Les résultats obtenus sont résumés dans le tableau ci-dessous :

équation pro- priétés d'un ensem- ble com- plet de solu- tions.	$U : \sigma E = \sigma E'$			$D : \sigma E = E'$			$P : \sigma E \sim \sigma E'$		
	1	2	≥ 3	1	2	≥ 3	1	2	≥ 3
principal	0	<i>N</i>	<i>N</i>	0	<i>N</i>	<i>N</i>	0	<i>N</i>	<i>N</i>
fini	0	<i>N</i>	<i>N</i>	0	0	<i>N</i>	0	<i>N</i>	<i>N</i>
indépendant	0	<i>N</i>	<i>N</i>	0	<i>N</i>	<i>N</i>	0	0	0
minimal	0	?3	<i>N</i>	0	0	0	0	0	0
décidable	0	?1	<i>N</i>	0	0	?2	0	?1	<i>N</i>

0 : OUI

N : NON

? : ouvert

On remarque trois niveaux de difficulté : 1er ordre, 2ème ordre, 3ème ordre et au delà.

Au premier ordre, il existe une solution principale pour chaque type d'équations. En particulier, il existe un unificateur principal de tout ensemble fini de termes. De plus, nous avons montré comment calculer cet unificateur (ou reconnaître la non-existence de solutions) en temps linéaire en fonction de la taille de l'équation à résoudre.

Par contre, dès le 3ème ordre, il n'existe pas en général d'ensemble complet fini. De plus, pour l'unification, il n'existe pas toujours d'ensemble complet de solutions minimales. Enfin l'unifiabilité y est indécidable. On ne peut donc espérer, pour générer un ensemble complet d'unificateurs, qu'une procédure qui énumère un ensemble infini contenant des solutions redondantes. Cependant, si nous sommes seulement intéressés par l'unifiabilité, alors on peut se limiter à générer un ensemble complet de préunificateurs, toute équation étant préunifiable si et seulement si elle est unifiable. L'intérêt de cette méthode est qu'alors on peut garantir aux solutions générées d'être non seulement minimales, mais aussi indépendantes. Ceci permet d'obtenir un algorithme d'unifiabilité qui termine si une solution existe et qui ne fait pas de recherches redondantes. Cet algorithme a été présenté au chapitre 3, pour le langage d'ordre ω que constitue le λ -K-calcul typé muni de la β -réduction. Il est montré au chapitre 4 comment cet algorithme se simplifie lorsqu'on autorise aussi la règle de n -conversion. Enfin le chapitre 6 donne un algorithme particulièrement simple pour la restriction aux termes de second ordre.

En ce qui concerne la demi-unification, il est montré au chapitre 3 qu'on peut toujours définir un ensemble complet de solutions minimales. Cet ensemble est unique à un isomorphisme près, et nous montrons au chapitre 6 qu'à

l'ordre 2 il est fini, et peut être engendré par un algorithme de filtrage particulièrement simple.

Du point de vue de la structure de ces ensembles de solutions, nous laissons ouverts principalement trois problèmes, notés \mathcal{P}_1 , \mathcal{P}_2 et \mathcal{P}_3 dans le tableau précédent.

\mathcal{P}_1 concerne la décidabilité de l'unifiabilité à l'ordre 2.

Ce problème, qui a des rapports étroits avec le problème d'existence de solutions à un système d'équations dans le monoïde libre, est probablement très difficile.

\mathcal{P}_2 concerne la décidabilité du filtrage à l'ordre 3 et plus.

Nous conjecturons ce problème décidable, mais la preuve en est probablement difficile.

\mathcal{P}_3 concerne l'existence d'un ensemble complet d'unificateurs minimaux à l'ordre 2. Nous conjecturons également une réponse positive.

Le deuxième type de résultats obtenus concerne la structure de l'ensemble des termes, ordonnés par l'ordre de filtrage.

Nous avons montré au chapitre 5 qu'au premier ordre, le magma libre constituant l'ensemble des termes possède vis-à-vis de cet ordre une structure d'inf demi treillis bien fondé, ce qui implique que

- 1) tout sous ensemble non vide de termes admet un inf (plus grand minorant)
- 2) tout sous ensemble de termes majoré supérieurement admet un sup (plus petit majorant)
- 3) il n'existe pas de chaînes infinies strictement décroissantes.

Nous montrons de plus comment on peut étendre l'ensemble des termes finis en leur adjoignant les arbres infinis rationnels, définissables par des équations récursives simples. L'existence d'un unificateur principal est conservée.

En fait deux arbres finis ou infinis compatibles admettent un unificateur principal qui est rationnel si les deux arbres sont rationnels. L'algorithme d'unification rapide peut être interprété comme calculant cet unificateur principal sur les arbres, ou d'un autre point de vue, comme unifiant des termes modulo une forme simple de récursion.

Par contre la structure des arbres rationnels est plus compliquée que dans le cas fini : le treillis sous condition n'est plus complet, ni pour le sup ni pour l'inf, et il existe des chaînes infinies strictement décroissantes.

Dans le chapitre 6 nous montrons qu'au second ordre on ne peut même plus garantir l'existence d'un inf ou d'un sup de deux termes, et qu'il existe des chaînes infinies strictement décroissantes. Ceci est vrai même en se limitant à des symboles de fonctions monadiques. Nous tombons dans ce cas sur des problèmes ouverts de la théorie des monoïdes libres. En particulier, l'unifiabilité de deux termes par des fonctions monadiques strictes est équivalente au problème de Markov.

Les résultats négatifs obtenus pour l'unification à partir du 3ème ordre suggèrent que la structure des termes y est encore plus complexe.

2. Applications

Les langages de termes considérés dans ce travail sont les outils de base du logicien et de l'informaticien.

Notre langage de premier ordre, le magma libre ayant pour opérateurs l'ensemble des symboles de fonction, est précisément l'ensemble des termes d'un calcul des prédictats du premier ordre.

Ces termes sont utilisés par l'informaticien pour décrire la syntaxe abstraite des programmes, les structures de données arborescentes et l'enchaînement des calculs.

De même au second ordre, le logicien reconnaîtra l'ensemble des termes d'un calcul des prédictats du second ordre. Ces termes servent à l'informaticien en particulier à exprimer les schémas de programmes récursifs, et les organigrammes contenant des méta-variables ou variables de procédures.

Enfin, à l'ordre ω , notre langage est le λ -calcul utilisé par Church comme syntaxe de sa théorie des types. L'informaticien est en train de reconnaître à ce langage un rôle croissant en théorie de la programmation.

Nous allons maintenant présenter rapidement quelques applications particulières de nos résultats en informatique théorique.

Après une présentation succincte, nous renverrons le lecteur intéressé aux articles cités en référence.

Démonstration automatique en calcul des prédictats du premier ordre

La démonstration automatique a pour objet la réalisation de programmes qui vérifient (ou réfutent) une proposition dans une certaine théorie mathématique, par la génération d'une démonstration formelle.

La plupart des programmes de démonstration automatique en logique de premier ordre utilisés à ce jour sont basés sur la règle de résolution. La résolution est une règle d'inférence unique complète pour les propositions mises sous forme normale conjonctive. Les quantificateurs sont éliminés par la méthode de Skolem, et on montre qu'une proposition P peut se déduire dans la théorie d'axiomes A en réfutant par résolution $A \cup \{\neg P\}$. La méthode est exposée complètement dans Robinson [RJA1].

La résolution combine les règles de coupure et de substitution, par l'intermédiaire de l'unification. Plus précisément, soit C une clause (ensemble de disjonctions) contenant une formule atomique L :

$$C = \{L\} \cup \bar{C}.$$

Soit D une clause contenant une négation de formule atomique $\neg M$:

$$D = \{\neg M\} \cup \bar{D}.$$

Nous supposons de plus que les variables libres apparaissant dans D ont été renommées, de manière à être distinctes des variables apparaissant dans C (ces variables sont en effet quantifiées implicitement au niveau de chaque clause).

Si L et M s'unifient, avec un unificateur principal σ , alors on peut dériver par résolution la clause :

$$\sigma \bar{C} \cup \sigma \bar{D}.$$

(Remarquer que c'est le résultat de la coupure de σC et σD). Plus généralement, on peut couper $\{L_1, \dots, L_n\}$ avec $\{\neg M_1, \dots, \neg M_p\}$. Donnons maintenant un exemple de preuve par résolution.

Soit à prouver : "Tout demi-groupe simplifiable à droite et à gauche possède un élément neutre à droite". On utilise le prédictat ternaire P pour signifier :

$P(x, y, z) \iff x * y = z$, où $*$ est l'opération du demi-groupe. Les axiomes utilisés s'expriment par :

- associativité :

$$\forall xyzuv [(P(x, y, u) \wedge P(y, z, v)) \Rightarrow \forall w (P(x, v, w) \iff P(u, z, w))]$$

qui donne deux clauses en forme normale conjonctive, la seule nécessaire étant :

$$(1) \quad \neg P(x, y, u), \neg P(y, z, v), \neg P(x, v, w), P(u, z, w)$$

- solution à gauche :

$$\forall xy \exists z P(z, x, y) \text{ qui donne une clause où apparaît la fonction de Skolem } G :$$

$$(2) \quad P(G(x, y), x, y)$$

- de même pour la solution à droite :

$$(3) \quad P(x, D(x, y), y)$$

Nous désirons démontrer le théorème :

$\exists x \forall y P(y, x, y)$, dont la négation fournit la clause :

(4) $\neg P(Y(x), x, Y(x))$.

Appliquons la règle de résolution sur les clauses 2 et 1, où les formules à couper sont

$P(G(x, y), x, y)$, $P(x', y', u)$ et $P(x', v, w)$

(x et y de (1) étant renommées en x' et y'). Un unificateur principal de ces formules est :

$$\sigma = \{<x', G(x, y)>, <y', x>, <v, x>, <u, y>, <w, y>\}$$

qui donne la clause résolvante :

(5) $\neg P(x, z, x), P(y, z, y)$

La résolution de (4) et de (5) produit de même :

(6) $\neg P(x, z, x)$

Enfin la résolution de (3) et (6) produit la clause vide \square , qui signale qu'on a obtenu une réfutation.

La résolution est une combinaison élégante des règles d'inférences usuelles de coupure et de substitution. La complétude de cette méthode procède des mêmes raisons qui permettent l'élimination des coupures dans les systèmes de déduction naturelle de Gentzen. L'existence d'un unificateur principal est certainement l'une des propriétés fondamentales de la logique de premier ordre.

L'algorithme d'unification rapide présenté au chapitre 5 permet de calculer rapidement des inférences par résolution.

Nous espérons que cet algorithme servira de base à la construction de programmes de démonstration automatique futurs plus efficaces.

Extension à l'égalité

Robinson G. et Wos ont défini [RW1] une extension de la règle de résolution au calcul des prédictats du premier ordre avec égalité. Une règle de substitutivité de l'égalité est introduite, appelée paramodulation :

Soit une clause C, dans laquelle apparaît un terme t : $C = F(t)$.

Soit une clause D contenant une équation :

$$D = \{t_1 = t_2\} \cup \bar{D}.$$

On suppose ici aussi que les variables de C et D sont distinctes.

Si t et t_1 sont unifiables, avec unificateur principal σ , alors on peut dériver la clause : $\sigma[E(t_2) \cup \bar{D}]$.

(De même en échangeant les rôles de t_1 et t_2).

Un ensemble de clauses C est insatisfiable dans toute interprétation normale ssi on peut dériver la clause vide de $C \cup \{x=x\}$ par résolution et paramodulation.⁽⁺⁾

Donnons un exemple de paramodulations. Soit * une opération binaire vérifiant les deux axiomes :

$$(1) \quad x*(x*y) = y$$

$$(2) \quad (x'*y') * y' = x'$$

Montrons par paramodulation que * est commutative.

On commence par choisir $t = x'*y'$, $t_1 = x*(x*y)$, $t_2 = y$.

On obtient $\sigma = \{\langle x', x \rangle, \langle y', (x*y) \rangle\}$, d'où le paramodulant de (2) par (1) :

$$(3) \quad y*(x*y) = x.$$

En paramodulant (1) par (3), on obtient bien :

$$(4) \quad y*x = x*y.$$

(Nous laissons deviner au lecteur les termes t et t_1 utilisés dans ce cas).

(+) Ceci est encore une conjecture ; il peut être nécessaire de rajouter à C des axiomes du type $x = x' \supset F(x) = F(x')$.

Extension aux logiques d'ordre supérieur

La méthode de résolution peut être étendue aux logiques d'ordre supérieur. Dans [HG1] nous avons défini une méthode de preuves par réfutations, complète pour la théorie des types de Church. La règle principale est une règle de résolution généralisée, qui n'effectue que les unifications faciles. Les unifications difficiles sont retardées, les équations correspondantes étant gardées explicitement.

Lorsqu'une réfutation potentielle est détectée, alors l'algorithme de recherche de pré-unificateurs décrit au chapitre 3 est appliqué. La production d'une solution valide la réfutation, en donnant un unificateur global pour tous les pas de résolution employés.

De nombreux exemples de la méthode sont donnés dans [HG3]. L'algorithme de pré-unification a été implémenté en ALCOL W, à l'IRIA, et la méthode de preuve par résolutions retardées a été partiellement implémentée, pour un langage d'ordre 2, par J. Darlington au GMD (Bonn). P. Andrews expérimente avec une implantation complète, à l'université Carnegie Mellon.

A titre d'exemple, considérons l'ensemble de clauses :

- (1) $P(f(x))$, $Q(z(C), C)$, $R(f(z))$
- (2) $\neg P(A)$
- (3) $\neg Q(y(A), y)$
- (4) $\neg R(B(w))$

où les atomes utilisés sont de type, respectivement :

w, A	α
y, B, C	$(\alpha \rightarrow \alpha)$
x, z	$((\alpha \rightarrow \alpha) \rightarrow \alpha)$
f	$((((\alpha \rightarrow \alpha) \rightarrow \alpha) \rightarrow \alpha)$

et les prédictats constants P, Q et R sont de types

$$\left\{ \begin{array}{ll} P, R & (\alpha \rightarrow 0) \\ Q & (\alpha \times (\alpha \rightarrow \alpha) \rightarrow 0) \end{array} \right.$$

où 0 est le type "valeur de vérité".

La séquence de résolutions qui consiste à couper ensemble les formules atomiques de même symbole de prédictat aboutit à l'unificande :

$$N_0 = \{<f(x), A>, <z(C), y(A)>, <C, y>, <f(z), B(w)>\}.$$

En développant un arbre d'unifiabilité pour N_0 , comme expliqué au chapitre 3, on trouve l'unificateur :

$$\sigma = \{<f, \lambda x \cdot x(B)>, <z, \lambda y \cdot y(A)>, <w, A>, <x, \lambda y \cdot A>\}.$$

Cet unificateur valide la réfutation. En effet, en reportant σ dans les clauses initiales, on trouve bien un ensemble de clauses propositionnellement contradictoire.

Complétude des règles de simplification

Soit R un ensemble de paires $<t_1, t_2>$, où t_1 et t_2 sont des termes de premier ordre. On définit une relation de réécriture \rightarrow entre termes par :

$t \rightarrow t'$ ssi $t = E<t'>$ (i.e. t' apparaît en sous terme de t) et $<t_1, t_2> \in R$, avec $\bar{t} = \sigma t_1$ et $t' = E<\sigma t_2>$.

On dit que l'ensemble R est complet si et seulement s'il possède la propriété de Church-Rosser, i.e. :

$t \rightarrow t_1$ et $t \rightarrow t_2 \Rightarrow \exists t' \quad t_1 \xrightarrow{*} t'$ et $t_2 \xrightarrow{*} t'$, où $\xrightarrow{*}$ dénote la fermeture réflexive et transitive de \rightarrow .

Dans [KB1], Knuth et Bendix ont montré comment, dans le cas des systèmes simplifiants (c'est à dire tels que la relation \rightarrow est bien fondée), on peut donner une condition nécessaire et suffisante de complétude basée sur la paramodulation mutuelle des paires de réécriture. De plus, lorsque l'ensemble de

règles n'est pas complet, l'examen de la condition permet d'engendrer de nouvelles règles compatibles avec l'ensemble de départ. Si le processus s'arrête, on obtient un système complet.

Les systèmes de simplification complets sont d'une grande importance en calcul symbolique. Ils permettent de définir des formes normales uniques, calculables en un temps fini. De plus, il est possible de définir des systèmes de démonstration automatique en calcul des prédictats avec l'égalité, où la règle de substitutivité est remplacée par des réécritures simplifiantes. Dans les théories où cela est possible, cette méthode produit une amélioration spectaculaire par rapport aux règles précédemment utilisées (paramodulation). Ici encore, l'algorithme d'unification rapide du chapitre 5 permet une implémentation efficace de cette méthode.

Donnons un exemple en théorie des groupes. Au départ, on se donne les trois règles de simplification suivantes, correspondant aux axiomes usuels de la structure de groupe :

- (1) $E*x \rightarrow x$
- (2) $x^{-1}*x \rightarrow E$
- (3) $(x*y)*z \rightarrow x*(y*z)$

Les deux premières règles forment un système complet, mais pas l'ensemble des trois. En effet, on peut paramoduler la troisième par la deuxième, en unifiant $x*y$ et $u^{-1}*u$.

Ceci détermine un "terme critique" $(x^{-1}*x)*z$ qui est simplifiable par (2) puis (1) en z , et par (3) en $x^{-1}*(x*z)$. Cette constatation suggère l'utilisation d'une règle de simplification supplémentaire :

- (4) $x^{-1}*(x*z) \rightarrow z$

Mais le système résultant n'est pas encore complet. L'unification du sous terme $x * z$ avec la partie gauche de la règle (1) produit le terme critique $E^{-1} * (E * z)$, simplifiable par (4) en z et par (1) en $E^{-1} * z$. D'où la règle :

$$(4.5) \quad E^{-1} * z \rightarrow z$$

Le processus continue ainsi, jusqu'à l'obtention des 6 règles supplémentaires :

- (5) $x * E \rightarrow x$
- (6) $E^{-1} \rightarrow E \quad [(4.5) \text{ est alors supprimée}]$
- (7) $(x^{-1})^{-1} \rightarrow x$
- (8) $x * x^{-1} \rightarrow E$
- (9) $x * (x^{-1} * y) \rightarrow y$
- (10) $(x * y)^{-1} \rightarrow y^{-1} * x^{-1}.$

Le système obtenu est maintenant complet, et peut servir à résoudre le problème des mots pour un groupe libre : $t = t'$ est une conséquence des axiomes de groupe si et seulement si t et t' possèdent la même forme normale, obtenue par des séquences arbitraires de simplifications par les règles (1) à (10).

Equivalence des schémas récursifs

L'algorithme d'unification rapide peut être étendu à l'unification sur des domaines d'arbres infinis, comme nous l'avons vu au chapitre 5. Cet algorithme permet alors de résoudre l'équivalence des schémas récursifs zéroadiques, comme définis par exemple dans [CKV].

Ces schémas interviennent dans de nombreux problèmes informatiques, comme par exemple pour définir récursivement des types dans un langage tel qu'ALGOL 68. Notre algorithme peut être ainsi utilisé pour décider rapidement de la compatibilité de types (modes) entre deux objets définis dans un tel langage.

Par exemple, supposons qu'un programme contienne les déclarations :

mode m = struct (ref m a, ref struct (ref m a, ref n b)b) ;

mode n = struct (ref n a, ref m b) ;

m i ; n j ;

Supposons maintenant qu'on ait à vérifier la compatibilité des modes des identificateurs *i* et *j*. On considère l'ensemble des termes construits sur les variables *m* et *n* par un symbole de fonction binaire S_{ab} . Sur cet ensemble de termes on considère l'équivalence \sim définie par :

$$\left\{ \begin{array}{l} m \sim S_{ab}(m, S_{ab}(m, n)) \\ n \sim S_{ab}(n, m) \\ m \sim n \end{array} \right.$$

On vérifie sans peine que \sim s'étend en une équivalence cohérente et simplifiable, ce qui prouve que les modes définis par *m* et *n* sont identiques.

Transformations schématiques

Il est possible de décrire certaines transformations de programmes, préservant le résultat des calculs, par des paires de schémas de programmes $\langle S_1, S_2 \rangle$, associés à un ensemble d'axiomes A qualifiant les variables des schémas S_1 et S_2 . Un programme $P = F\langle P' \rangle$ est transformable s'il existe σ , avec $P' = \sigma S_1$, tel que $M \models \sigma A$ dans tout modèle M du langage de programmation utilisée. Le programme résultant de cette transformation est défini comme étant $E\langle \sigma S_2 \rangle$.

Bien des optimisations de programmes sont exprimables de cette manière, par exemple pour transformer des récursions en itérations [DR1]. Un des problèmes qui se pose est naturellement de déterminer les filtres σ qui réalisent l'identification d'un programme et d'un schéma, les schémas en question étant généralement exprimés par des termes de second ordre.

Nous avons présenté au chapitre 6 un algorithme de filtrage de second ordre, qui génère un ensemble complet et fini de filtres minimaux. Cet algorithme est en cours d'implémentation à l'IRIA pour la transformation schématique de programmes PASCAL.

Donnons un exemple de transformation schématique.

Soit le schéma récursif défini par :

$$f(x) = \text{SI } a(x) \text{ ALORS } b(x) \text{ SINON } h(d(x), f(e(x))).$$

Nous pouvons représenter ce schéma par le terme de second ordre :

$$S_1 = \lambda f x \cdot C(a(x), b(x), h(d(x), f(e(x)))).$$

Avec les notations du chapitre 6, on a :

$$V(S_1) = \{a, b, d, e, h\}, \text{ avec}$$

$$\begin{cases} x \in V_0 \\ a, b, d, e, f \in V_1 \\ h \in V_2 \\ C \in C, \text{ avec } \alpha(C) = 3. \end{cases}$$

Le schéma ci-dessous, en notation pseudo-ALGOL, calcule itérativement une fonction f , avec résultat dans l'identificateur res .

$$S_2 = \left\{ \begin{array}{l} \text{fonction } f(x) ; \\ \text{SI } a(x) \text{ ALORS } res := b(x) \text{ SINON} \\ \quad \text{DEBUT} \\ \quad \quad res := d(x) ; \\ \quad \quad x := e(x) ; \\ \quad \text{TANT QUE NON } a(x) \text{ FAIRE} \\ \quad \quad \text{DEBUT} \\ \quad \quad \quad res := h(res, d(x)) ; \\ \quad \quad \quad x := e(x) ; \\ \quad \quad \quad \text{FIN} ; \\ \quad \quad res := h(res, b(x)) ; \\ \quad \quad \text{FIN} ; \end{array} \right.$$

On peut prouver que les schémas S_1 et S_2 calculent la même fonction, dans toute interprétation de a, b, d, e et h qui satisfait à l'ensemble d'axiomes :

$$A = \left\{ \begin{array}{l} \forall xyz \ h(h(x,y),z) = h(x,h(y,z)) \\ \forall x \ h(x,\omega) = \omega \end{array} \right.$$

où ω est interprété par "indéfini".

Tout programme récursif de la forme σS_1 peut être réécrit en σS_2 , si l'on peut prouver σA . Par exemple, considérons la définition récursive usuelle du programme LISP qui retourne une liste ; dans notre formalisme, on écrit :

$$P = \lambda \text{ rev } x . C(\text{NULL}(x), \text{NIL}, \text{APPEND}(\text{rev}(\text{CDR}(x)), \text{CONS}(\text{CAR}(x), \text{NIL}))).$$

Appliquons maintenant l'algorithme de filtrage du chapitre 6 à P et S_1 . Toutes les solutions trouvées ont en commun la substitution :

$$\sigma = \{ \langle a, \lambda x . \text{NULL}(x) \rangle, \langle b, \lambda x . \text{NIL} \rangle, \langle e, \lambda x . \text{CDR}(x) \rangle, \\ \langle h, \lambda u v . \text{APPEND}(v, g(u, v)) \rangle \}$$

Plus précisément, les solutions sont de la forme $\rho_i \sigma$ ($i \leq 3$), avec :

$$\begin{aligned}\rho_1 &= \{\langle g, \lambda u v . u \rangle, \langle d, \lambda x . \text{CONS}(\text{CAR}(x), \text{NIL}) \rangle\} \\ \rho_2 &= \{\langle g, \lambda u v . \text{CONS}(\text{CAR}(u), \text{NIL}) \rangle, \langle d, \lambda x . x \rangle\} \\ \rho_3 &= \{\langle g, \lambda u v . \text{CONS}(u, \text{NIL}) \rangle, \langle d, \lambda x . \text{CAR}(x) \rangle\}.\end{aligned}$$

Seule la première solution vérifie l'axiome d'associativité pour h , ce qui donne la traduction :

```

fonction f(x) ;
SI NULL(x) ALORS res := NIL SINON
    DEBUT
        res := CONS(CAR(x), NIL) ;
        x := CDR(x) ;
    TANT QUE NON NULL(x) FAIRE
        DEBUT
            res := APPEND(CONS(CAR(x), NIL), res) ;
            x := CDR(x) ;
        FIN ;
        res := APPEND(NIL, res) ;
    FIN ;

```

L'algorithme de filtrage de second ordre nous donne donc une méthode uniforme pour reconnaître qu'une transformation schématique peut s'appliquer à un programme. Remarquer sur cet exemple que le filtrage a changé l'appel récursif de place (du second argument de h dans le schéma il s'est déplacé dans le premier argument de APPEND dans le programme).

Langages de programmation évolués

Durant la dernière décennie sont apparus des langages de programmation développés par les centres de recherche en intelligence artificielle (PLANNER, QA4, QLISP). Parmi les traits communs à ces langages, on trouve l'appel des procédures par filtrage ("pattern invocated procedures"). Lors de l'appel d'une telle procédure, la procédure appelée est sélectionnée par filtrage de son en-tête et de la séquence d'appel.

Lorsque ces langages possèdent des structures de données, les problèmes de filtrage peuvent devenir très ardu. Par exemple en QA4, dans le cas des listes, on trouve des problèmes analogues au filtrage de termes de second ordre monadique, problème abordé au chapitre 6.

Lorsque de plus l'instantiation est permise dans les deux sens, le problème d'unification correspondant est équivalent au problème encore ouvert de la résolution d'équations dans le monoïde libre.

Nos résultats permettent donc d'évaluer les difficultés d'implémentation de telles facilités dans un langage de programmation. Des problèmes analogues sont étudiés dans la thèse de M. Stickel [SM2].

Notons enfin qu'à l'extrême on peut se restreindre à cet appel de procédures par unification comme unique structure de contrôle d'un langage de programmation, comme c'est le cas par exemple pour le langage PROLOG. L'unification est alors le mécanisme de base dirigeant l'interprétation des programmes écrits dans un tel langage.

Induction

Mentionnons enfin, comme application de l'anti-unification (i.e. la recherche de l'inf de deux termes vis à vis de l'ordre de filtrage, plutôt que leur sup) le problème de l'induction, ou généralisation à partir d'exemples. Cette application a été développée en particulier dans la thèse de G. Plotkin [PG3].

Nous arrêterons là cette liste d'applications. Il doit être clair maintenant que la résolution de diverses équations aux termes est un problème central en calcul symbolique, et que les applications en sont nombreuses en programmation et en intelligence artificielle.

Il ne fait pas de doute que nous aurons besoin de développer des algorithmes d'unification spécialisés à certaines théories. Par exemple, G. Plotkin a défini un algorithme d'unification en présence de simplifications arithmétiques [PG1]. Des résultats analogues sont présentés dans [SJ1] et [SM1].

Nous espérons que la terminologie et les méthodes utilisées dans cette thèse seront utiles à de telles études. On peut d'ailleurs considérer nos algorithmes d'unification en ordre supérieur comme effectuant l'unification modulo les axiomes de description utilisés implicitement par les règles de formation des λ -termes.

- [AP1] Andrews P.B. (1971)
Resolution in type theory.
Journal of Symbolic Logic 36,3 pp. 414-432.
- [AP2] Andrews P.B. (1974)
Provability in elementary type theory
Zeitschr.f.math . Logik und Grunlagen d. Math.
Bd.20,. S. 411-418.
- [BL1] Baxter Lewis D. (1973)
An efficient unification algorithm
Technical report CS-73-23
Applied analysis and Computer Science Dept,
University of Waterloo.
- [BL2] Baxter Lewis D. (1976)
Communication personnelle.
- [CA1] Church A. (1940)
A formulation of the simple theory of types.
Journal of Symbolic Logic 5,1 pp.56-68.
- [CA2] Church A. (1941)
The calculi of lambda-conversion
Annals of Mathematics Studies n°6,
Princeton University Press.
- [CB1] Courcelle B. (1975)
Ensembles Algébriques d'Arbres et Langages déterministes
Thèse Université de Paris.
- [CKV] Courcelle B., Kahn G. et Vuillemin J. (1974)
Algorithmes d'équivalence et de réduction à des expressions minimales dans une classe d'équations récursives simples.
Symposium on Automata, Languages and Programming,
Sarrebrück, édité par Springer-Verlag.
- [CF1] Curry H.B. et Feys R. (1968)
Combinatory Logic, Vol 1
North Holland, Amsterdam.
- [CHS1] Curry H.B., Hindley J.R. et Seldin J.P. (1972)
Combinatory Logic, Vol 2
North Holland, Amsterdam.
- [DJL1] Darlington J.L. (1971).
A partial mechanization of second-order logic.
Machine Intelligence 6, pp. 91-100.
American Elsevier, New-York.
- [DJL2] Darlington J.L. (1973)
Automatic program synthesis in second-order logic.
Proceedings of 3rd IJCAI, Stanford August 1973.
- [DB1] Darlington J. et Burstall R.M. (1973)
A system which automatically improves programs
Third International Joint Conference on Artificial Intelligence
Stanford California .

- [DB1] de Bruijn N.G. (1972)
Lambda-calculus notation with nameless dummies
Indagationes Math. 34,5
- [EG1] Ernst G.W. (1971)
A matching procedure for type theory.
Communication personnelle.
- [CF1] Galler B.A. et Fisher M.J. (1964)
An improved equivalence algorithm
CACM 7,5 pp. 301-303.
- [GW1] Gould W.E. (1966)
A matching procedure for ω -order logic.
Scientific Report N°4, AFCRL 66-781.
Contract AF (628)-3250.
AD 646-560.
- [GJ1] Guard J.R. (1964)
Automated logic for semi-automated mathematics.
Scientific Report n°1, AFCRL 64-411.
Contract AF 19 (628)- 3250
AD 602 710.
- [HT1] Hart Timothy P. (1965)
A useful algebraic property of Robinson's unification algorithm
Artificial Intelligence Project memo 91
Memorandum MAC-M-285
MIT, Cambridge, Massachussets
- [HJ1] Herbrand J. (1930)
Recherches sur la théorie de la démonstration
Thèse Université de Paris, publiée dans
Ecrits logiques de Jacques Herbrand
PUF, 1968.
- [HLS1] Hindley J.R., Lercher B. et Seldin J.P. (1972)
Introduction to Combinatory Logic.
London Mathematical Society Lecture Note Series 7
Cambridge University Press.
- [HJ1] Hmelevskii Ju.I.(1964)
The solution of certain systems of word equations
Soviet Math. Dokl. Vol 5 pp.724-726.
- [HJ2] Hmelevskii Ju.I.(1966)
Word equations without coefficients
Soviet Math. Dokl. Vol 7 N°6 pp. 1611-1613.
- [HJ3] Hmelevskii Ju.I.(1967)
Solution of word equations in three unknowns
Soviet Math. Dokl. Vol 8 N°6 pp. 1554-1556.

- [HG1] Huet G.P. (1972)
 Constrained Resolution : a complete method for type theory.
 Thèse de Ph. D.
 Jennings Computing Center Report 1117,
 Case Western Reserve University.
- [HG2] Huet G.P. (1973)
 The undecidability of unification in third order logic.
 Information and Control 22,3 pp. 257-267.
- [HG3] Huet G.P. (1973)
 A mechanization of type theory.
 Proceedings of 3rd IJCAI, Stanford August 1973.
- [HG4] Huet G.P. (1975)
 Unification in typed λ -calculus
 Proceedings of the Symposium on λ -calculus and computer
 Science theory, Roma
 Springer-Verlag Lecture Notes in Computer Science N°37
- [HG5] Huet G.P. (1975)
 A unification algorithm for typed λ -calculus.
 Theoretical Computer Science, 1.1 pp. 27-57.
- [JP1] Jensen D. et Pietrzykowski T. (1973)
 Mecyanizing ω -order type theory through unification
 Report CS-73-16, Dept. of applied Analysis and Computer Science,
 University of Waterloo.
 Theoretical Computer Science, à paraître.
- [KD1] Knuth Donald E. (1969)
 The art of computer programming
 Vol 1 : Fundamental algorithms
 Addison-Wesley, Reading Mass.
- [KB1] Knuth Donald E. et Bendix Peter B. (1970)
 Simple word problems in universal algebras
 in Computational Problems in Abstract Algebra,
 Leech J. Ed.
 Pergamon Press, Braunschweig
- [LA1] Lentin A. (1972)
 Equations dans les monoïdes libres
 Gauthier Villars, Paris
- [LC1] Lucchesi C.L. (1972)
 The undecidability of the unification problem for third
 order languages.
 Report CSRR 2059, Dept. of Applied Analysis and Computer Science,
 University of Waterloo.

- [MA1] Markov A.A.
Algorithme non publié.
- [MJ1] Morris J.H. (1968)
Lambda-calculus models of programming languages.
Ph. D. thesis, MIT.
MAC TR-57, December 68, 131 pp.
- [PW1] Pateřson M.S. et Wegman M.N. (1976)
Linear unification.
Communication personnelle.
- [PT1] Pietrzykowski T. (1971)
A complete mechanization of second order logic.
Journal of Assoc. for Comp. Math. 20,2 pp. 333-364.
- [PJ1] Pietrzykowski T. et Jensen B. (1972)
A complete mechanization of w-order type theory.
Association for Computing Machinery National Conference 1972,
vol. 1, pp. 82-92.
- [PG1] Plotkin G.D. (1972)
Building-in Equational Theories
Machine Intelligence 7, pp. 73-90.
American Elsevier, New-York.
- [PG2] Plotkin G.D. (1970)
Lattice-theoretic properties of subsumption
Memorandum MIP-R-77.
University of Edinburgh.
- [PG3] Plotkin G.D. (1971)
Automatic methods of inductive inference
Ph.D. thesis, University of Edinburgh.
- [PD1] Prawitz Dag (1960)
An Improved proof procedure
Theoria 26 pp.102-139.
- [RJ1] Reynolds J. (1970)
Transformational systems and the algebraic structure
of atomic formulas.
Machine Intelligence 5, pp.135-152.
American Elsevier, New-York.
- [RW1] Robinson G.A. et Wos L.T. (1969)
Paramodulation and theorem proving in first-order theories
with equality.
Machine Intelligence 4, pp. 135-150.
American Elsevier, New-York.

- [RJA1] Robinson J.A. (1965)
A machine-oriented logic based on the resolution principle.
Journal of Assoc. for Comp. Mach. 12,1 pp. 23-41.
- [RJA2] Robinson J.A. (1971)
Computational logic : the unification computation
in Machine Intelligence 6,
Eds B. Meltzer et D. Michie
American Elsevier, New-York.
- [RJA3] Robinson J.A. (1976)
Communication personnelle.
- [RG1] Rose G. (1972)
Communication personnelle.
- [SL1] Sanchis L.E. (1967)
Functionals defined by recursion.
Notre Dame Journal of Formal Logic Vol VIII, N°3
pp. 161-174.
- [SJ1] Slagle J.R. (1974)
Automated theorem-proving for theories with simplifiers,
commutativity, and associativity.
JACM 21,4, pp. 622-642.
- [SS1] Stenlund Sören (1972)
Combinators, λ -terms , and proof theory
D. Reidel, Dordrecht.
- [SM1] Stickel M.E. (1975)
A complete unification algorithm for
associative-commutative functions.
Proceedings IV-th IJCAI, Tbilisi.
- [SM2] Stickel M.E. (1976)
Unification algorithms for artificial intelligence languages.
Thèse de Ph. D.en cours, Carnegie Mellon U.
- [TW1] Tait W.W. (1967)
Intentional interpretations of functionals of finite type. I
Journal of Symbolic Logic 32, pp. 198-212.
- [TR1] Tarjan Robert F. (1975)
Efficiency of a good but not linear set union algorithm.
JACM 22,2 pp. 215-225.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique