N° d'ordre

THESE

présentée à
L'UNIVERSITE D'ORLEANS

pour obtenir le titre de
DOCTEUR

Spécialité :
INFORMATIQUE

par
Jacques **CHABIN**

Sujet de la thèse :

UNIFICATION GENERALE PAR SURREDUCTION ORDONNEE CONTRAINTE ET SURREDUCTION DIRIGEE

Thèse soutenue le 24 Janvier 1994
devant la commission d'examen composée de

J.-P. JOUANNAUD	Président
H. GANZINGER	Rapporteur
C. KIRCHNER	Rapporteur
S. ANANTHARAMAN	Examinateur
G. FERRAND	Examinateur
P. RETY	Examinateur

Remerciements

Ce travail de thèse a été effectué au sein du Laboratoire d'Informatique Fondamentale d'Orléans. Sa réalisation a été menée sous la direction scientifique de Siva Anantharaman, qui a su me transmettre avec un enthousiasme permanent sa passion pour la recherche. Je tiens à lui témoigner ici toute ma reconnaissance pour sa disponibilité quotidienne et ses conseils autant théoriques que pratiques.

J'adresse mes plus vifs remerciements à Jean-Pierre Jouannaud qui m'a fait l'honneur d'être le président du jury de cette thèse. Je lui exprime ici toute ma gratitude.

Je suis très reconnaissant envers Harald Ganzinger qui a accepté sans la moindre hésitation de rapporter cette thèse et de venir d'Allemagne pour participer à ce jury.

Je remercie chaleureusement mon rapporteur Claude Kirchner. Ses conseils et ses encouragements m'ont été d'une grande utilité tout au long de mon travail.

Je tiens aussi à remercier sincèrement tous les membres du LIFO. Ils m'ont acceuilli très chaleureusement pendant ces trois années et m'ont transmis leur passion pour le métier d'enseignant-chercheur. Je remercie plus particulièrement Gérard Ferrand qui m'a accepté dans son laboratoire et Pierre Réty qui m'a fait découvrir le domaine de la réécriture et avec lequel j'ai entretenu sans cesse des relations de travail et d'amitié. J'ai été très flatté qu'ils acceptent tous les deux de prendre part à ce jury.

Je voudrais dire aussi un grand merci à mes compagnons privilégiés, Wadoud Bousdira, Nirina Andrianarivelo, Dominique Archambault, Michel Braunwarth, et bien évidemment Christophe Léger du LESI qui est mon complice depuis le début de mes études à l'Université. Qu'ils trouvent tous ici l'expression de ma très sincère reconnaissance pour leurs encouragements, leurs conseils et les moments formidables que nous avons passés ensemble.

Pour conclure, je dédie cette thèse à tous les membres de ma famille, et plus particulièrement à mes parents, ma tante Paulette, mes frères et soeurs, et ma cousine Annie qui m'a accueilli chaque semaine pendant toute la durée de mes études.

Table des matières

Introduction

Les équations jouent un rôle très important dans la plupart des domaines scientifiques. Elles servent bien souvent à modéliser et à définir des situations concrètes, comme par exemple dans la vision par ordinateur, la compréhension du langage naturel, la résistance des matériaux, le positionnement des composants sur un circuit intégré, etc... Mais leur domaine privilégié restera toujours les mathématiques. Là elles permettent de définir axiomatiquement des structures algébriques classiques : les monoïdes, les groupes ou les algèbres. Les équations ont aussi une grande importance en informatique théorique. Elles sont utilisées pour décrire des types abstraits, les entiers, les booléens, et aussi des fonctions. Ainsi on peut définir des théories avec des ensembles d'équations, mais à partir de là, on a souvent besoin de connaître les théorèmes valides dans ces théories. Pour cela on utilise des méthodes basées sur la réécriture. L'un des mécanismes fondamentaux de la réécriture est l'unification, introduite par Herbrand [HER30], et étudiée par Robinson [ROB65] pour des besoins de résolution en logique du premier ordre. Pour résoudre un problème dans une théorie équationnelle, on part des hypothèses, sur lesquelles on applique les équations de la théorie, dans l'espoir d'aboutir à la conclusion. Le processus d'unification (ou de filtrage), sur lequel s'appuie le mécanisme dit de complétion, permet d'engendrer de nouvelles équations qui peuvent être utilisées, pour la résolution d'un problème.

Le problème qui nous intéresse dans cette thèse, est l'unification générale, c'est-à-dire la résolution d'équations dans des théories équationnelles quelconques. Considérons par exemple un symbole de fonction '+' à deux arguments, ayant la propriété d'idempotence et possédant un élément neutre à droite. Ceci peut se traduire par les deux équations suivantes :

$$\begin{array}{rcrcl}(e_1) & : & x+x & \doteq & x \\ (e_2) & : & x+0 & \doteq & x\end{array}$$

où '0' est un symbole de constante et 'x' représente une variable. Maintenant essayons de résoudre l'équation :

$$x + y \stackrel{?}{\doteq} a$$

où 'a' est une nouvelle constante. Il faut trouver les valeurs que doivent prendre les variables 'x' et 'y' pour que les deux membres de l'équation deviennent égaux, sachant que '+' est idempotent et qu'il a un élément neutre à droite. Nous avons comme solution '$x = a$' et '$y = a$'. On dit alors que la substitution $\theta_1 = \{x \leftarrow a, y \leftarrow a\}$ est une solution de l'équation '$x + y \stackrel{?}{\doteq} a$' ou encore que θ_1 est un unificateur des termes '$x + y$' et 'a' dans la théorie décrite par les équations (e_1) et (e_2). Les substitutions $\theta_2 = \{x \leftarrow a, y \leftarrow 0\}$ et $\theta_3 = \{x \leftarrow a + 0, y \leftarrow 0\}$ sont aussi des solutions. Mais θ_3 peut être 'simplifiée' puisque '$a + 0$' est égal à 'a', on dira que θ_3 n'est pas 'simple' ou 'normale'. On ne cherchera pas à décrire l'ensemble de toutes les solutions, car il est souvent infini, mais plutôt un ensemble générateur (on dit aussi : complet) des solutions normales. Sur notre exemple, $\{\theta_1, \theta_2\}$ est l'ensemble des solutions que l'on veut obtenir.

Maintenant que nous avons situé le problème, regardons comment on peut le résoudre de façon automatique. Il existe des algorithmes d'unification propres à des théories particulières : la commutativité, l'associativité-commutativité, etc. Mais on souhaiterait avoir des méthodes d'unification valables dans un cadre général. Claude Kirchner [KIR85, KIR86, KIR87] a cherché à systématiser la construction des algorithmes d'unification. Il a décomposé le processus en trois opérations : la décomposition, la fusion et la mutation, où seule la mutation dépend de la théorie considérée. La difficulté principale consiste alors à trouver une règle de mutation convenable.

Dans cette thèse nous allons développer une autre approche, basée sur des techniques de réécriture : la surréduction. La première chose à faire pour cela est d'orienter les équations et ainsi obtenir des règles de réécriture. En reprenant le même exemple que précédemment, on peut orienter les équations de la façon suivante :

$$\begin{array}{lcccc} (r_1) & : & x+x & \rightarrow & x \\ (r_2) & : & x+0 & \rightarrow & x \end{array}$$

Nous voulons toujours résoudre l'équation :

$$x + y \stackrel{?}{=} a$$

La surréduction, notée $\rightsquigarrow$, consiste à appliquer une règle de réécriture sur un des termes de l'équation, après avoir instancié, si nécessaire, ses variables, par exemple :

$$x + y \stackrel{?}{=} a \rightsquigarrow_{[(r_1),\{y \leftarrow x\}]} x \stackrel{?}{=} a$$

Pour appliquer la règle (r_1) sur '$x + y$', il faut d'abord changer 'y' en 'x' avec la substitution $\theta_1 = \{y \leftarrow x\}$, ce qui donne '$x+x$', lequel se réécrit avec (r_1) en 'x'. Nous obtenons une nouvelle équation '$x \stackrel{?}{=} a$'. La surréduction comprend donc deux étapes : instanciation des variables pour qu'une règle de réécriture puisse s'appliquer, puis réécriture par cette règle. La nouvelle équation à résoudre '$x \stackrel{?}{=} a$' a une solution syntaxique (sans faire intervenir (r_1) ou (r_2)) qui est $\theta_2 = \{x \leftarrow a\}$. Alors la composée des deux substitutions $\theta = \theta_2 \circ \theta_1 = \{x \leftarrow a, y \leftarrow a\}$ est une solution de '$x + y \stackrel{?}{=} a$' dans notre théorie. De la même façon on peut appliquer la règle (r_2) et on obtient :

$$x + y \stackrel{?}{=} a \rightsquigarrow_{[(r_2),\{y \leftarrow 0\}]} x \stackrel{?}{=} a$$

ce qui donne la solution $\theta' = \{x \leftarrow a, y \leftarrow 0\}$. On a bien calculé les deux solutions qui nous intéressent.

La surréduction a été introduite par Slagle [SLA74] et étudiée par Lankford [LB79]. Fay [FAY78, FAY79] et Hullot [HUL80a, HUL80c] ont prouvé, que l'ensemble des solutions obtenues en parcourant toutes les branches de surréduction issues de l'équation à résoudre est un ensemble complet de solutions, si le système de réécriture est confluent et noethérien. C'est-à-dire si toutes les étapes de réécriture issues d'un même terme confluent vers un même terme, et s'il n'existe pas de dérivation de réécriture infinie. Dans ce cas, on dit aussi que le système est canonique. Dans notre exemple, les deux règles forment un système de réécriture canonique.

Pour transformer un ensemble d'équations en un système de réécriture confluent et noethérien, Knuth et Bendix [KB70] ont proposé une procédure de complétion appelée *KB*. Cette procédure part d'un ensemble d'équations et d'un ordre pour orienter les équations et renvoie, lorsqu'elle termine, un système de réécriture canonique, équivalent à l'ensemble d'équations initial. Mais cet algorithme échoue lorsqu'il rencontre une équation inorientable. Pour contrer cette restriction, Bachmair, Dershowitz et Hsiang [BDH86] ont proposé la procédure de complétion sans échec appelée aussi procédure de complétion ordonnée : *UKB*. Elle permet d'obtenir dans les mêmes conditions que précédemment un système final d'équations, peut-être infini, mais qui est confluent et noethérien sur les termes clos (sans variable). La surréduction peut être combinée avec cette complétion ordonnée pour obtenir une méthode d'unification générale [OS90].

Tout serait parfait si on ne parlait pas d'efficacité. Un gros inconvénient de la surréduction est qu'elle créée un nombre de branches beaucoup trop important, et même implémentée sur des machines puissantes, elle est dans bien des cas inopérente. Pour améliorer cela, Hullot [HUL80a] a introduit la surréduction basique. Rappelons que la surréduction consiste à instancier les variables d'un terme, (que l'on veut surréduire), pour ensuite le réécrire. La surréduction basique consiste, dans une chaîne de surréduction, à interdire d'appliquer une étape de surréduction dans les instanciations apportées par les étapes précédentes. Hullot a montré que l'on conserve la complétude de la surréduction en se limitant aux positions basiques. Une notion similaire a été introduite récemment au niveau de la procédure de complétion [SL91, NR92a, BGLS92, BGSL93]. La notion de basicité peut être exprimée par marquage des positions interdites, ou grâce au formalisme des contraintes [KKR90].

Nous allons présenter dans cette thèse, une nouvelle méthode d'unification générale basée sur une combinaison de la complétion basique et de la surréduction basique, les deux exprimées dans un cadre contraint. Les contraintes nous servent à stocker les parties protégées. Nous avons reformulé la notion de complétion basique et défini une surréduction contrainte. Comme pour toute procédure de complétion, les étapes de simplification jouent un rôle très important pour l'efficacité de la méthode. Mais en complétion basique, la simplification, qui est basée sur une réécriture, ne doit ni protéger de nouvelles positions, sinon nous n'obtenons pas un système final confluent sur les termes clos; ni déprotéger des positions anciennement protégées, sinon nous perdons le bénéfice de la basicité. Pour cela, nous allons donner un algorithme calculant un filtre entre deux termes contraints, et définir une réécriture contrainte basée sur ce filtre, permettant de réécrire un terme contraint avec une équation contrainte tout en respectant les impératifs définis ci-dessus. Cette réécriture sera légèrement restreinte pour être utilisée dans les étapes de simplification.

Cette méthode a les avantages suivants :

- On applique les étapes de surréduction, pendant la complétion de l'ensemble d'équations, et ainsi la méthode calcule toute solution (close) donnée au bout d'un temps fini.
- Grâce aux contraintes, la surréduction peut terminer même quand la théorie équationnelle comporte des équations inorientables.
- Il nous parait intéressant de compléter le système d'équations initial par la complétion basique, plutôt que la complétion ordinaire : la surréduction est alors faite par des équations possédant des parties protégées, et ainsi, les équations générées par surréduction ont des parties protégées plus importantes. Ceci devrait diminuer le nombre de surréductions possibles.

Nous nous sommes aussi intéressés à l'amélioration de l'efficacité et de la terminaison de la surréduction en général. Pour cela nous avons repris une stratégie basée sur un graphe d'opérateurs [DS87], et nous avons introduit la notion de graphe de termes, qui est une optimisation plus forte. Dans les deux cas, l'idée est de faire un prétraitement sur l'ensemble des règles du système de réécriture et de l'équation à résoudre, en construisant un graphe, puis d'utiliser ce graphe pour éviter des branches inutiles dans les étapes de surréduction. Ces techniques généralisent la détection des collisions entre les constructeurs, et sont essentielles pour faire terminer la surréduction lorsque des fonctions sont définies récursivement. Nous avons intégré ces stratégies dans la surréduction contrainte, en considérant le graphe comme un mécanisme à part. Une méthode séduisante aurait été de voir les restrictions imposées par le graphe comme un nouveau type de contrainte. Ceci entre parfaitement dans le rôle général joué par les contraintes, mais impose une redéfinition de celles-ci et un mécanisme plus lourd pour leur manipulation. Ce travail reste un objectif pour nous dans l'avenir.

Le plan de cette thèse est le suivant : nous introduisons tout d'abord, dans le premier chapitre, les notions et les définitions de base, nécessaires à la bonne compréhension de la thèse. Le chapitre 2 présente la nouvelle méthode d'unification générale. Nous donnerons une définition complète de la réécriture contrainte avec un système de règles d'inférence permettant de calculer un filtre entre deux termes contraints. Nous présenterons ensuite, notre procédure combinant la complétion basique et la surréduction basique, puis nous détaillerons la preuve de complétude de notre méthode. Le chapitre 3 présente la stratégie de surréduction basée sur un graphe d'opérateurs puis sur un graphe de termes. Ces stratégies sont applicables à la surréduction dans des systèmes de réécriture canoniques. Nous présenterons une extension possible de ces stratégies à notre méthode d'unification. Le chapitre 4 décrit notre implantation de la complétion basique et de la surréduction basique dans un cadre contraint; cette implémentation n'incorpore pas encore l'optimisation par graphes.

Chapitre 1

Définitions et survol des principales méthodes existantes

Ce chapitre va se décomposer en deux parties. Dans un premier temps, nous allons présenter les différentes notations utilisées tout au long de ce mémoire ainsi que les définitions de base des objets et des méthodes utilisés en réécriture. Tout ceci a pour but de donner les bases théoriques et pratiques (les notations) nécessaires à la bonne compréhension de cette thèse. La deuxième partie sera consacrée à l'étude des différentes possibilités de résolution d'une équation dans une théorie équationnelle donnée. Après un survol du domaine, nous étudierons plus en détail la méthode de Gallier et Snyder, ainsi que celle de Dougherty et Johann.

1.1 Définitions générales

Définissons d'abord les notations de base. Il s'agit ici d'une présentation mono-typée, la notion de type n'interviendra donc pas dans cette thèse.

1.1.1 Symboles

On se donne deux ensembles de symboles dénombrables.

- Σ est appelé l'ensemble des symboles de fonctions : $\{f, g, h, ...\}$. A tout symbole de fonction, on associe un entier positif appelé arité de la fonction. On peut ainsi partitionner Σ en une réunion disjointe d'ensembles $\Sigma = \bigcup_{n \geq 0} \Sigma_n$, où Σ_n représente l'ensemble des symboles de fonctions d'arité n. Les fonctions d'arité 0 sont appelées constantes. De plus on supposera toujours que Σ contient au moins une constante. Les symboles de constantes seront désignés par les lettres $a, b, c, ...$.

- X est appelé l'ensemble des symboles de variables : $\{x, y, z, ...\}$. Cet ensemble est toujours infini. Ayant déjà considéré un ensemble fini de variables, on pourra toujours choisir de nouvelles variables en dehors de cet ensemble.

Une **signature** S est la donnée de ces deux ensembles : (Σ, X).

Passons maintenant à la définition d'un terme et aux différentes opérations réalisables sur les termes et qui sont nécessaires en réécriture.

1.1.2 Quelques notions de base en réécriture

Un **terme** est une expression, construite à partir d'une signature, qui respecte les arités des fonctions. Plus précisément, un terme est soit, une variable x, soit une expression de la forme $f(t_1,..,t_n)$ avec $f \in \Sigma_n$ et $t_1,..,t_n$ des termes. Le plus petit ensemble contenant tous les termes d'une signature, est noté $T(\Sigma, X)$. Les termes seront désignés par les lettres $l, r, s, t, \ldots$. L'ensemble des variables d'un terme t sera noté $V(t)$. Un terme t est dit clos s'il ne contient pas de variable, c'est-à-dire $V(t) = \emptyset$. L'ensemble des termes clos sera noté $T(\Sigma)$.

On utilise la notion d'**occurrence**, ou de **position** pour désigner, de façon formelle un sous-terme d'un terme. C'est une suite d'entiers positifs représentant le chemin allant de la racine d'un terme au symbole de tête du sous-terme considéré. La racine, ou la tête d'un terme est représentée par l'occurrence ϵ. L'ensemble des occurrences d'un terme t, noté $O(t)$, peut être défini de la manière suivante :

- $O(t) = \epsilon$ si t est une variable ou une constante.
- $O(t) = \bigcup_{i=1..n}\{i.u \;\; / \;\; u \in O(t_i)\}$ si $t = f(t_1,..,t_n)$.

On désignera les occurrences par les lettres p, u, v. La notation $t|_u$ représentera le sous-terme de t se trouvant à la position u, et $t[u \leftarrow t']$ le terme obtenu après avoir remplacé le sous-terme à l'occurrence u de t par le terme t'. La concaténation de deux occurrences u et v sera notée $u.v$. L'ensemble des occurrences non variables $\overline{O}(t)$ est défini par : $\overline{O}(t) = \{u \in O(t) \;\; / \;\; t|_u \notin X\}$. L'ensemble des occurrences est partiellement ordonné par l'ordre des préfixes $<$ défini par $u < v$ ssi il existe une occurrence p telle que $v = u.p$.

Exemple 1.1 Soit l'ensemble de symboles $\Sigma = \{a, b, h, f, g\}$. Les deux termes $s = f(a, g(x, b))$ et $t = h(y)$ appartiennent à $T(\Sigma, X)$. On a

- $V(s) = \{x\}$
- $O(s) = \{\epsilon, 1, 2, 2.1, 2.2\}$ et $\overline{O}(s) = \{\epsilon, 1, 2, 2.2\}$
- $s|_2 = g(x, b)$ et $s[2.2 \leftarrow t] = f(a, g(x, h(y)))$.

On omettra, dans les exemples qui vont suivre, de définir de façon explicite l'ensemble Σ. Il sera constitué des symboles apparaissant dans les termes.

Une **substitution** σ est une application de X dans $T(\Sigma, X)$ telle que $\sigma(x) = x$ excepté sur un ensemble fini de variables, appelé domaine de σ et noté $D(\sigma)$. L'ensemble des variables introduites par σ, appelé image de σ et noté $I(\sigma)$, est défini par $I(\sigma) = \bigcup_{x \in D(\sigma)} V(\sigma(x))$. Soit W un ensemble de variables, la définition de la restriction de la substitution σ à l'ensemble W est $\sigma|_W(x) = \sigma(x)$ si $x \in W$ et $\sigma|_W(x) = x$ si $x \notin W$. Une substitution peut être étendue de façon classique sur $T(\Sigma, X)$ en posant : $\sigma(f(t_1,..,t_n)) = f(\sigma(t_1),..,\sigma(t_n))$ pour tout terme $f(t_1,..,t_n)$. Les substitutions seront notées par les lettres $\sigma, \theta, \tau, \rho, \ldots$, et représentées par leur ensemble de définition $\{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$.

On dira qu'une substitution σ est **close** si $I(\sigma) = \emptyset$, qu'elle est **idempotente** si $\sigma \circ \sigma = \sigma$ (ce qui est équivalent à $D(\sigma) \cap I(\sigma) = \emptyset$), que c'est un **renommage** d'un ensemble de variables W si $\forall x \in W, \sigma(x) \in X$ et $\forall x, y \in W$, si $x \neq y$ alors $\sigma(x) \neq \sigma(y)$.

Une substitution σ est **plus générale** qu'une substitution τ sur un ensemble de variables W (on notera $\sigma \preceq \tau[W]$, on omettra $[W]$ si W est l'ensemble de toutes les variables) ssi il existe une substitution θ, telle que $\forall x \in W, \theta \circ \sigma(x) = \tau(x)$. ($\circ$ représente la composition des applications : $\theta \circ \sigma(x) = \theta(\sigma(x))$). Cet ordre est appelé **ordre de subsomption**.

Un terme t est une **instance** d'un terme s, ou s **filtre** t ssi il existe une substitution ρ, appelée filtre, telle que $t = \rho(s)$. Dans ce cas on notera $s \preceq t$ et on dira que s est plus général que t. Cet ordre sur les termes est aussi appelé ordre de subsomption, il peut être décomposé en ordre de subsomption strict $\prec$ et subsomption équivalence $\simeq$. On peut montrer que $s \simeq t$ s'il existe un renommage ρ tel que $s = \rho t$, et $s \prec t$ si $s \preceq t$ et $s \not\simeq t$. (On utilisera souvent l'abréviation ρt pour $\rho(t)$ et $\rho\theta$ pour $\rho \circ \theta$).

Un **unificateur** pour deux termes s et t est une substitution σ telle que, $\sigma s = \sigma t$. On dit que σ est l'unificateur le **plus général** ou l'unificateur **principal** de s et t si σ est un unificateur de s et t et si σ est le plus petit unificateur de s et t par rapport à l'ordre de subsomption sur les substitutions. Dans ce cas on notera $\sigma = mgu(s,t)$. Si deux termes sont unifiables, alors il existe toujours un unificateur le plus général unique (à un renommage de variables près). De plus cet unificateur pourra toujours être choisi idempotent.

Exemple 1.2 Soient les termes $s_1 = f(x_1, g(y_1))$, $s_2 = f(x_2, y_2)$ et $s_3 = f(a, x_3)$. On voit que s_1 et s_3 sont des instances de s_2, et que s_1 et s_3 sont unifiables par $\sigma = \{x_1 \leftarrow a, x_3 \leftarrow g(y_1)\}$ qui est l'unificateur principal, mais aussi par $\sigma_1 = \{x_1 \leftarrow a, x_3 \leftarrow g(a)\}$ et on a $\sigma \preceq \sigma_1$.

La notion d'ordre est très importante en réécriture, car elle permet toujours de se diriger vers quelque chose de plus simple. Un **ordre** est une relation binaire, irréflexive et transitive. A cette définition nous ajoutons d'autres propriétés qui nous permettent de travailler sur les termes.

Un ordre $>$ sur $T(\Sigma, X)$, sera dit ordre de **réduction** ssi il vérifie les trois propriétés suivantes :

1. $>$ est **noethérien**, (ou bien fondé, ou terminant), c'est-à-dire qu'il n'existe pas de chaîne infinie décroissante $t_1 > t_2 > t_3 > ...$ d'éléments de $T(\Sigma, X)$.
2. $>$ est **stable par substitution**, c'est-à-dire que pour tous termes s et t tels que $s > t$, et pour toute substitution σ on a $\sigma s > \sigma t$.
3. $>$ est **monotone**, (ou stable par contexte ou par plongement), c'est-à-dire que pour tous termes s et t tels que $s > t$, pour tout terme l et pour toute occurrence $u \in O(l)$ on a $l[u \leftarrow s] > l[u \leftarrow t]$.

Si en plus de ces trois propriétés, l'ordre possède la propriété dite du sous-terme, c'est-à-dire que pour tout terme s et pour toute occurrence $u \in O(s)$, différente de ϵ, $s > s|_u$, alors l'ordre $>$ sera dit ordre de **simplification**. Un ordre de simplification total sur les termes clos est appelé ordre de simplification complet, souvent noté en abrégé cso. Les principaux ordres de simplification sont rpo, lpo, kbo, l'ordre polynômial [DER82, KB70, BCL87].

Tout au long de cette thèse, nous allons utiliser différents symboles pour représenter différents types d'égalité, mais le symbole '=' seul, sera réservé uniquement pour représenter l'égalité syntaxique.

Une **équation** est un couple de termes, non ordonné, noté $s \doteq t$, où les variables de s et de t sont quantifiées universellement. L'équation $f(x,y) \doteq f(y,x)$ signifie que pour tous termes x et y, on a $f(x,y) \doteq f(y,x)$. Une **spécification**, (ou théorie équationnelle) S, est la donnée d'une signature et d'un ensemble d'équations (aussi appelées axiomes), et noté E. Une **règle** est un couple de termes ordonné, noté $s \rightarrow t$ et vérifiant $s > t$ pour un ordre $>$ donné. Un ensemble de telles règles constitue un **système de réécriture**, souvent noté R. De plus il est dit terminant si l'ordre $>$ est noethérien et est un ordre de réduction.

Une **congruence** sur les termes est une relation d'équivalence monotone. Soit E un ensemble d'équations, on notera par $\leftrightarrow_E$ la plus petite relation contenant E, symétrique, stable par substitution et monotone. Autrement dit, $s \leftrightarrow_E t$ ssi il existe une équation $l \doteq r$ de E, une substitution σ et une occurrence $u \in O(s)$ telles que $s|_u = \sigma l$ et $t = s[u \leftarrow \sigma r]$: on remplace un sous-terme d'un terme par un autre terme qui lui est égal dans la théorie équationnelle. On appelle **égalité modulo** E ou E**-égalité** la fermeture réflexive transitive de $\leftrightarrow_E$ notée $\leftrightarrow_E^*$ ou plus simplement $=_E$. C'est aussi la plus petite congruence stable par substitution et contenant E. On dit qu'un couple de termes (s,t) est un **théorème** valide dans E ssi $s =_E t$.

Exemple 1.3 Soit E l'ensemble d'équations suivant : $E = \{f(x,y) \doteq g(y),\ g(a) \doteq b\}$, on a $f(a,b) \leftrightarrow_E g(a) \leftrightarrow_E b$, donc $f(a,b) =_E b$.

Nous allons maintenant introduire la notion de réécriture ou de réduction, pour n'autoriser le remplacement via une équation dans une étape $\leftrightarrow_E$ que dans un sens.

Une relation $\rightarrow$ dans $T(\Sigma, X)$ est dite relation de **réécriture** ssi elle est noethérienne, stable par substitution et monotone. On peut définir une relation de réécriture par rapport à un ensemble de règles R et un ordre de réduction $>$ de la façon suivante : on dit qu'un terme s se réécrit, ou se réduit, en un terme t dans R et on note $s \rightarrow_R t$, ssi il existe une règle $l \rightarrow r$ dans R, $(l > r)$, une substitution σ et une occurrence $u \in \overline{O}(s)$ telles que $s|_u = \sigma l$ et $t = s[u \leftarrow \sigma r]$. On notera de façon détaillée $s \rightarrow_{[u, l \rightarrow r, \sigma]} t$. On appellera **dérivation** une chaîne composée de plusieurs étapes de réécriture, notée $\rightarrow_R^*$

On dit qu'un terme s est une forme normale d'un terme t par rapport à un système de réécriture R si $t \rightarrow_R^* s$, et s est irréductible; c'est-à-dire qu'il n'existe pas de terme s' tel que $s \rightarrow_R s'$.

La notion de réécriture fournit une méthode pour prouver la validité de théorèmes. Maintenant nous allons introduire la notion de **surréduction**. C'est une méthode qui permet de trouver les instanciations d'une équation pour lesquelles l'équation devient un théorème valide. Surréduire consiste à instancier pour pouvoir réécrire. Cette méthode se place dans le cadre d'un système de réécriture R. On dit qu'un terme s se surréduit en un terme t dans R et on note $s \rightsquigarrow_R t$ ssi il existe une règle $l \rightarrow r$ de R, une substitution σ et une occurrence $u \in \overline{O}(s)$ telles que $s|_u$ et l soient unifiables et σ soit l'unificateur principal, et $t = (\sigma s)[u \leftarrow \sigma r]$. On notera de façon détaillée $s \rightsquigarrow_{[u, l \rightarrow r, \sigma]} t$. On appellera **surdérivation** une chaîne composée de plusieurs étapes de surréduction, notée $\rightsquigarrow_R^*$

La notion de réécriture, comme celle de surréduction, peut être définie de façon axiomatique. On partage l'ensemble d'équations E en deux, d'un côté des règles R et de l'autre des équations appelées axiomes A. Pour faire de la réécriture (resp. de la surréduction) dans R modulo A, il suffit de remplacer le filtre (resp. l'unificateur) par un A-filtre (resp. un A-unificateur). La définition d'un A-filtre (resp. un A-unificateur) est la même que celle d'un filtre (resp. un unificateur) mais l'égalité syntaxique doit être remplacée par une A-égalité. Plus formellement, on aura $s \rightarrow_{R,A} t$ ssi il existe une règle $l \rightarrow r$ dans R, une substitution σ et une occurrence $u \in \overline{O}(s)$ telles que $s|_u =_A \sigma l$ et $t = s[u \leftarrow \sigma r]$. De même $s \rightsquigarrow_{R,A} t$ ssi il existe une règle $l \rightarrow r$ dans R, une substitution σ et un occurrence $u \in \overline{O}(s)$ telles que $\sigma(s|_u) =_A \sigma l$ et $t = (\sigma s)[u \leftarrow \sigma r]$, [KIR85].

Les méthodes de validation de théorèmes par réécriture, ou de résolution d'équations par surréduction, sont des méthodes prouvées correctes et complètes [1] à condition que notre système de réécriture soit *canonique*. Nous allons maintenant définir cette nouvelle notion, ainsi que les moyens utilisés pour aboutir à de tels systèmes.

[1]Les notions de correction et de complétude peuvent se comprendre de façon intuitive, d'un côté on n'obtiendra pas de chose fausse, et de l'autre, on trouvera la (ou les) solution(s). Des définitions plus formelles seront données au début de la deuxième partie de ce chapitre.

1.1.3 La complétion ordonnée

On utilisera la notation $s \downarrow_R t$ pour indiquer qu'il existe un terme u tel que $s \rightarrow^*_R u \; {}^*_R\!\leftarrow t$. On dira qu'un système de réécriture R est **confluent** ou **convergent** ssi pour tous termes s, t, u tels que $s \; {}^*_R\!\leftarrow u \rightarrow^*_R t$ alors $s \downarrow_R t$. Cette propriété est équivalente à la propriété de Church Rosser : si $s \leftrightarrow^*_R t$ alors $s \downarrow_R t$. Un système est dit **canonique** s'il est confluent et terminant. Dans un tel système, chaque terme a une forme normale unique.

Peu de systèmes ont cette propriété de canonicité, mais nous disposons d'une méthode de complétion dite **KB** (Knuth & Bendix), qui à partir d'un ensemble quelconque d'équations E et d'un ordre de réduction $>$, fournit un système canonique équivalent au premier, lorsque celui-ci existe. Cette procédure peut être décrite par un ensemble de quatre règles d'inférence travaillant sur deux ensembles : l'ensemble E d'équations du départ, et un ensemble R de règles, vide au départ, mais qui contiendra le système canonique si la procédure termine avec E vide. Nous donnons ici ces quatre règles d'inférence [BAC87] :

1) Orientation d'une équation

$$\frac{(E \cup \{s \doteq t\}, R)}{(E, R \cup \{s \rightarrow t\})} \qquad \text{si } s > t.$$

2) Ajout d'une conséquence équationnelle

$$\frac{(E, R)}{(E \cup \{s \doteq t\}, R)} \qquad \text{si } \exists \text{ un terme } u \text{ tel que } s \; {}^*_R\!\leftarrow u \rightarrow^*_R t.$$

3) Simplification d'une équation

$$\frac{(E \cup \{s \doteq t\}, R)}{(E \cup \{s' \doteq t\}, R)} \qquad \text{si } s \rightarrow_R s'.$$

4) Suppression d'une équation triviale

$$\frac{(E \cup \{s \doteq s\}, R)}{(E, R)}$$

L'exécution de cette procédure aboutit à l'un des trois cas suivants :

1. la procédure s'arrête et E est vide, c'est un succès, R contient le système canonique équivalent à E,
2. la procédure s'arrête, mais E n'est pas vide, c'est un échec, une équation n'a pas pu être orientée,
3. la procédure ne s'arrête pas.

L'ajout d'une conséquence équationnelle se fait par l'intermédiaire de paires critiques. On dit qu'il existe une **paire critique** à partir de la règle $l_1 \rightarrow r_1$ dans la règle $l_2 \rightarrow r_2$ ssi il existe une occurrence u de l_2 et un unificateur σ de l_1 et $l_2|_u$; la paire critique obtenue est alors l'équation $\sigma l_2[u \leftarrow \sigma r_1] = \sigma r_2$. L'ensemble des paires critiques de $l_1 \rightarrow r_1$ dans $l_2 \rightarrow r_2$ est noté $CP(l_1 \rightarrow r_1, l_2 \rightarrow r_2)$.

On peut se rendre compte facilement que, bien souvent, la procédure va terminer en échec. En effet '>' étant un ordre de réduction il ne peut pas comparer les variables et donc beaucoup d'équations, comme par exemple celle qui décrit la commutativité d'un opérateur, seront inorientables, ce qui produira un échec dans la procédure de complétion KB. Pour éviter cette impasse, nous disposons d'une nouvelle méthode, dite de complétion sans échec, ou UKB (Unfailing Knuth Bendix) ou Complétion ordonnée. Cette procédure ne provoque pas d'échec quand apparaissent des équations inorientables.

Comme la procédure de complétion KB, la procédure UKB est basée sur l'ajout de paires critiques et la simplification. Il faut donc maintenant définir une nouvelle notion de paire critique, ainsi que celle de réécriture ou de réduction, qui ne soit plus basée uniquement sur des règles mais aussi sur des équations.

Définition 1.4 Il existe une paire critique de l'équation $l_1 \doteq r_1$ dans l'équation $l_2 \doteq r_2$ ssi il existe une occurrence u non variable de l_2 et un unificateur principal σ de l_1 et $l_2|_u$ tels que $\sigma r_1 \not\geq \sigma l_1$ et $\sigma r_2 \not\geq \sigma l_2$. La paire critique ainsi obtenue, est l'équation $(\sigma l_2)[u \leftarrow \sigma r_1] \doteq \sigma r_2$. $\diamond$

Le processus de génération de paires critiques est aussi appelé superposition. Cette nouvelle définition englobe la précédente, en effet, lorsque $l_1 \doteq r_1$ et $l_2 \doteq r_2$ sont orientables, les nouvelles conditions sont automatiquement vérifiées.

Définition 1.5 Soit E un ensemble d'équations. On dit qu'un terme s se réécrit ou se réduit, avec une équation $l \doteq r \in E$ ssi il existe une occurrence u non variable de s et un filtre σ tels que $s|_u = \sigma l$ et $\sigma l > \sigma r$. On obtient le terme $s[u \leftarrow \sigma r]$. On notera $s \rightarrow_E s[u \leftarrow \sigma r]$ ou de façon plus précise $s \rightarrow_{[u, l \doteq r, \sigma]} s[u \leftarrow \sigma r]$. $\diamond$

On conserve la même définition de la notion de confluence (ou convergence) pour un ensemble d'équations E, en remplaçant la relation $\rightarrow_R$ par par la relation $\rightarrow_E$.

Donnons maintenant la description de la procédure de complétion UKB sous forme de règles d'inférences basées sur deux ensembles d'équations E, R et un ordre de réduction $>$ total sur les termes clos. Ce système d'inférence n'est pas celui de [BAC87] mais se rapproche plus d'un système implémentable. Il a été pris dans [MZA86]. Au départ E est un ensemble d'équations et R est vide. Au fur et à mesure de la procédure R va être rempli par des équations dont on a calculé toutes les conséquences équationnelles.

1) Choix et Superposition

$$\frac{(E \cup \{s \doteq t\}, R)}{(E \cup CP(s \doteq t, R), R \cup \{s \doteq t\})}$$

où $CP(s \doteq t, R)$ est l'ensemble des paires critiques obtenues par superposition entre $s \doteq r$ et chacune des équations de R.

2) Simplification d'une équation de E

$$\frac{(E \cup \{s \doteq t\}, R)}{(E \cup \{s' \doteq t\}, R)}$$

si $s \rightarrow_{E \cup R} s'$.

3) Simplification d'une équation de R

$$\frac{(E, R \cup \{s \doteq t\})}{(E \cup \{s' \doteq t\}, R)}$$

si $s \rightarrow_{E \cup R} s'$.

4) Suppression d'une équation triviale

$$\frac{(E \cup \{s \doteq s\}, R)}{(E, R)}$$

L'orientation d'une équation n'intervient pas de façon explicite dans ces règles d'inférences, mais du point de vue pratique, toute équation orientable sera transformée en une règle. De la même manière que pour KB, plusieurs éventualités sont possibles lorsque l'on applique la procédure UKB, sur un système d'équations avec un ordre de réduction total sur les termes clos.

1. E est vide, UKB s'arrête et toutes les équations de R sont orientées. Dans ce cas, le système de règles R est un système de réécriture canonique équivalent à E.

2. E est vide, UKB s'arrête, mais R contient des équations inorientables. Dans ce cas le système de règles R est convergent mais uniquement sur les termes clos.

3. UKB ne termine pas. Dans ce cas, étant donné deux termes clos E-égaux, il existe un système R_i de règles et d'équations, trouvées à une étape i de la procédure UKB, pour lequel ces deux termes ont la même forme normale.

Cette procédure est la base des méthodes de preuve par réfutation de [LAN79, HR87, BAC87]. Voulant prouver un théorème $s \doteq t$ dans une théorie équationnelle, on ajoute à l'ensemble d'équations, décrivant la théorie, la négation du théorème à prouver $s \neq t$, et l'on cherche à trouver une incohérence de la forme $x \neq x$ en appliquant un algorithme basé sur UKB. Une telle approche peut être étendue également au cadre de la réécriture conditionnelle [RUS87, GAN91, AND91]. Mais dans tous les cadres de travail, la règle primordiale pour l'efficacité de la méthode est la simplification. Des critères pour accroître les performances de la complétion usuelle ont été proposés dans [BG91]. Pour notre méthode de E-unification décrite au chapitre 2, nous utilisons la procédure de complétion dans un cadre contraint.

Nous allons maintenant passer à la deuxième partie de ce chapitre, à savoir, la description de différentes méthodes d'E-unification.

1.2 Survol des méthodes d'unification modulo une théorie équationnelle

Pour débuter cette section, nous allons définir ce que nous attendons d'une méthode de E-unification, c'est-à-dire qu'elle soit correcte et qu'elle fournisse toutes les solutions. Dans la suite de ce chapitre E représentera un ensemble d'équations décrivant une théorie équationnelle fixée.

1.2.1 Les objectifs d'une méthode de E-unification

Rappelons qu'un E-unificateur, ou unificateur modulo E de deux termes s et t, est une substitution σ telle que $\sigma s =_E \sigma t$. On notera $EU_E(s,t)$ l'ensemble, qui est très souvent infini, des E-unificateurs de s et t. Nous perdons la notion d'unificateur le plus général, car bien souvent il n'est pas unique, mais nous la remplaçons par la notion d'ensemble complet minimal d'E-unificateurs, [PLO72].

Nous avons introduit ici, un nouvel ordre sur les substitutions $\preceq_E$. C'est l'ordre de subsomption modulo E, $\sigma \preceq_E \tau[W]$ ssi $\exists\theta$ telle que $\forall x \in W,\ \ \theta \circ \sigma(x) =_E \tau(x)$, lorsque W est omis, on doit vérifier la relation pour toutes les variables.

Définition 1.6 Un ensemble complet de E-unificateurs de deux termes s et t, noté $ECU_E(s,t)$, est un ensemble Θ de substitutions tel que :

1. $\Theta \subseteq EU_E(s,t)$
2. $\forall\sigma \in EU_E(s,t), \exists\theta \in \Theta$ telle que $\theta \preceq_E \sigma$

Θ est minimal si en plus il possède la propriété suivante : $\forall\sigma,\theta \in \Theta, \sigma \preceq_E \theta \Rightarrow \sigma = \theta$. Un ensemble complet minimal de E-unificateurs de deux termes s et t sera noté $ECMU_E(s,t)$. $\diamond$

Fages et Huet [FH83] ont montré que si l'ensemble complet minimal des E-unificateurs de deux termes existe, alors il est unique à un renommage de variables près et modulo E. Cet ensemble peut être infini comme le montre l'exemple suivant.

Exemple 1.7 E est l'ensemble des trois équations suivantes :

$$\begin{array}{rcrcl} (e_1) & : & f(0) & \doteq & 0 \\ (e_2) & : & h(0) & \doteq & 0 \\ (e_3) & : & f(g(x)) & \doteq & h(f(x)) \end{array}$$

L'ensemble complet minimal des E-unificateurs de $f(x)$ et 0 est infini. En effet la substitution $\{x \leftarrow 0\}$ est solution, mais aussi
$\{x \leftarrow g(0)\}$ (car $f(g(0)) \overset{(e_3)}{=_E} h(f(0)) \overset{(e_1)}{=_E} h(0) \overset{(e_2)}{=_E} 0$) et
$\{x \leftarrow g(g(0))\}$ (car $f(g(g(0))) \overset{(e_3)}{=_E} h(f(g(0))) \overset{(e_3)}{=_E} h(h(f(0))) \overset{(e_1)}{=_E} h(h(0)) \overset{(e_2)}{=_E} h(0) \overset{(e_2)}{=_E} 0$). Pour tout entier n, $\{x \leftarrow g^n(0)\}$ est solution et les solutions sont incomparables entre elles par l'ordre de subsomption modulo E.

L'ensemble des E-unificateurs d'une équation, notée $s \overset{?}{=} t$, sera l'ensemble des E-unificateurs des deux termes s et t. Une procédure d'E-unification est une procédure qui prend en entrée un ensemble d'équations E qui décrit une théorie équationnelle, et une équation à résoudre $s \overset{?}{=} t$ et fournit en sortie un ensemble de substitutions. Cette procédure sera correcte si toute substitution fournie est un E-unificateur de l'équation $s \overset{?}{=} t$, elle sera complète si l'ensemble des substitutions fourni est un ensemble complet de E-unificateurs; si en plus il est minimal, on dira que la procédure est minimale. On peut classer les différents types de spécifications, en fonction de la cardinalité de l'ensemble complet minimal des E-unificateurs qu'elles provoquent [BS].

Une méthode d'E-unification a donc pour but de fournir une procédure correcte et complète d'E-unification. Si E est vide, une méthode d'unification doit trouver l'unificateur principal et s'appelle unification syntaxique.

1.2.2 L'unification syntaxique

Le premier algorithme d'unification est décrit dans la thèse de Herbrand [HER30], ensuite nous trouvons d'autres algorithmes, notamment celui de Robinson [ROB65] et Martelli Montanari [MM82]. Ce dernier peut être vu sous forme d'un système d'inférences travaillant sur un ensemble d'équations S, il apparaît en plus un nouveau symbole F qui indique qu'il n'y a pas de solution.

1) Suppression

$$\frac{S \cup \{s \overset{?}{=} s\}}{S}$$

2) Décomposition

$$\frac{S \cup \{f(s_1,..,s_n) \overset{?}{=} f(t_1,..,t_n)\}}{S \cup \{s_1 \overset{?}{=} t_1,..,s_n \overset{?}{=} t_n\}}$$

3) Collision

$$\frac{S \cup \{f(s_1,..,s_n) \overset{?}{=} g(t_1,..,t_n)\}}{F} \qquad \text{si } f \neq g.$$

4) Test d'Occurrence

$$\frac{S \cup \{x \overset{?}{=} f(s_1,..,s_n)\}}{F} \qquad \text{si } x \in V(f(s_1,..,s_n)).$$

5) Instanciation

$$\frac{S \cup \{x \stackrel{?}{=} s\}}{\{x \leftarrow s\}S \cup \{x \stackrel{?}{=} s\}} \qquad \text{si } x \in V(S), x \notin V(s).$$

S étant un ensemble d'équations, $\{x \leftarrow s\}S$ représente l'ensemble des équations obtenues à partir de S en remplaçant toutes les occurrences de la variable x dans S, par le terme s.

Cet ensemble de règles appliqué à une équation $s \stackrel{?}{=} t$ tant que c'est possible, retourne, ou bien F si s et t ne sont pas unifiables syntaxiquement, ou bien un ensemble d'équations de la forme $\{x_1 \stackrel{?}{=} r_1, .., x_n \stackrel{?}{=} r_n\}$ où les variables $x_1, .., x_n$ sont toutes différentes, et alors $\{x_1 \leftarrow r_1, .., x_n \leftarrow r_n\}$ est l'unificateur principal de s et t.

Si maintenant E est non vide, on peut séparer les méthodes d'E-unification en deux parties. D'un côté les méthodes définies pour un ensemble d'équations fixé, par exemple pour l'associativité ou l'associativité et la commutativité, [BS]. De l'autre les méthodes définies dans le cadre général, c'est-à-dire, quelque soit l'ensemble d'équations E; c'est ce cadre qui nous intéressera dans la suite de cette thèse.

Nous allons commencer par étudier différentes méthodes d'E-unification lorsque E est un système de réécriture canonique; ces méthodes sont basées sur la surréduction.

1.2.3 La surréduction

La surréduction, définie plus haut, est une méthode générale d'E-unification. Elle a été étudiée par Fay [FAY79] et prouvée correcte et complète dans le cadre des systèmes de réécriture canonique par Hullot [HUL80b]. Dans la suite de ce paragraphe, R représentera un système de réécriture canonique fixé.

Définition 1.8 Une substitution ρ est dite normalisée par rapport à un système de réécriture R, si pour toute variable $x \in D(\rho)$, le terme ρx est en forme normale par rapport à R. $\diamond$

Pour démontrer la complétude de la surréduction en tant que procédure d'E-unification, Hullot a commencé par établir une correspondance entre la réécriture et la surréduction.

Lemme 1.9 [Hullot] Soient t_0 un terme et ρ_0 une substitution normalisée. A toute dérivation [2] issue de $\rho_0 t_0$:

$$\rho_0 t_0 = t'_0 \rightarrow_{[p_0,\, g_0 \rightarrow d_0]} t'_1 \ldots \rightarrow_{[p_{n-1},\, g_{n-1} \rightarrow d_{n-1}]} t'_n$$

on peut associer une surdérivation issue de t_0 où les étapes de surréduction se font aux mêmes occurrences que les étapes de réécriture et avec les même règles :

$$t_0 \rightsquigarrow_{[p_0, g_0 \rightarrow d_0, \sigma_0]} t_1 \ldots \rightsquigarrow_{[p_{n-1}, g_{n-1} \rightarrow d_{n-1}, \sigma_{n-1}]} t_n$$

et pour tout $i \in [1..n]$ une substitution ρ_i, normalisée, telle que :

- $\rho_i t_i = t'_i$
- $\rho_0 = \rho_i \circ \sigma_{i-1} \circ \ldots \circ \sigma_0 [V(t_0)]$

La réciproque est triviale.

[2] A chaque étape i, la règle $g_i \rightarrow d_i$ utilisée sera une règle de R préalablement renommée avec des variables n'étant pas encore apparues dans aucune étape précédente.

La surréduction, comme la réécriture, définie jusqu'à présent sur des termes, peut être définie sur des équations, en supposant que le symbole $\stackrel{?}{=}$ est un nouveau symbole de fonction d'arité 2.

La correction de la méthode s'établit de la façon suivante. Si $(t \stackrel{?}{=} t') \rightsquigarrow^*_{[\sigma]} (s \stackrel{?}{=} s')$ où σ est la composée des substitutions utilisées pour les différentes étapes de surréduction, et si s et s' sont syntaxiquement unifiables par ν, alors $\nu \circ \sigma(t \stackrel{?}{=} t') \to^* \nu(s \stackrel{?}{=} s')$; mais puisque $\nu(s) = \nu(s')$ il résulte que $\nu \circ \sigma(t) =_R \nu \circ \sigma(t')$ et donc $\nu \circ \sigma$ est un R-unificateur de $t \stackrel{?}{=} t'$.

La complétude se base aussi sur le lemme de correspondance. Soit θ un R-unificateur de $t \stackrel{?}{=} t'$, comme le système de réécriture R est confluent et noethérien (canonique), on a $\theta(t \stackrel{?}{=} t') \to^* s \stackrel{?}{=} s$. D'après le lemme précédent [Hullot], on obtient $t \stackrel{?}{=} t' \rightsquigarrow^*_{[\sigma]} t_n \stackrel{?}{=} t'_n$ où σ est obtenue comme composée des substitutions utilisées pour la surréduction, et il existe ρ_n tel que $\theta = \rho_n \circ \sigma[V(t \stackrel{?}{=} t')]$.

La surréduction fournit donc une méthode complète et correcte d'unification dans les systèmes de réécriture canonique, ce qui se traduit par le théorème suivant :

Théorème 1.10 (Hullot) *Soient R un système de réécriture canonique, t_0 et t'_0 deux termes quelconques. L'ensemble des substitutions σ telles que :*

1. *il existe une surdérivation :*
$$t_0 \stackrel{?}{=} t'_0 \rightsquigarrow^*_{[\sigma_1]} \dots \rightsquigarrow^*_{[\sigma_n]} t_n \stackrel{?}{=} t'_n$$
telle que t_n et t'_n soient syntaxiquement unifiables par l'unificateur principal θ,

2. *$\sigma = \theta \circ \sigma_n \circ \dots \circ \sigma_1$ et σ est normalisée,*

est un ensemble complet de R-unificateurs de $t_0 \stackrel{?}{=} t'_0$.

Exemple 1.11 Soit R le système de réécriture canonique suivant :

$$\begin{array}{lcrcl} (r_1) & : & f(g(y)) & \to & g(y) \\ (r_2) & : & f(0) & \to & 0 \\ (r_3) & : & g(0) & \to & 0 \end{array}$$

La résolution de $f(x) \stackrel{?}{=} 0$ donne les étapes de surréduction suivantes

$f(x) \stackrel{?}{=} 0 \rightsquigarrow_{[(r_2)\{x \leftarrow 0\}]} 0 \stackrel{?}{=} 0$

$\rightsquigarrow_{[(r_1)\{x \leftarrow g(y)\}]} g(y) \stackrel{?}{=} 0 \rightsquigarrow_{[(r_3)\{y \leftarrow 0\}]} 0 \stackrel{?}{=} 0$

On obtient deux solutions $\{x \leftarrow 0\}$ et $\{x \leftarrow g(0)\}$, la deuxième étant non normalisée, on en déduit que $\{\{x \leftarrow 0\}\}$ est un ensemble complet de R-unificateurs de $f(x)$ et 0.

Les étapes de surréductions dans les travaux de Slage, Lankford, et Fay [SLA74, LB79, FAY79], s'accompagnent d'étapes de normalisation, nous appellerons ceci **surréduction normalisante** que l'on notera par le signe $\rightsquigarrow\!\!\!\rightsquigarrow$ et qui est définie de la façon suivante :

$$t \rightsquigarrow\!\!\!\rightsquigarrow_{[p,g\to d,\sigma]} t' \Longleftrightarrow t \rightsquigarrow_{[p,g\to d,\sigma]} t_1 \;\; et \;\; t' = t_1 \downarrow$$

Le théorème 1.10 précédent reste valable pour la surréduction normalisante.

La surréduction est donc, comme on vient de le voir, une méthode qui permet de trouver un ensemble complet de R-unificateurs de deux termes. Mais elle a comme défaut d'être inefficace sur le plan pratique, chaque solution étant calculée plusieurs fois (il existe plusieurs branches différentes qui aboutissent à la même solution). C'est ainsi qu'un grand nombre de stratégies sont apparues, ayant toutes pour but d'améliorer l'efficacité, tout en gardant la complétude et la correction de la méthode. Nous allons maintenant présenter les principales.

1.2.4 La surréduction basique

Cette méthode a été définie et prouvée complète par Hullot dans [HUL80b, HUL80c]. Elle est basée sur une stratégie de choix de l'occurrence à surréduire. De façon explicite, on interdit de surréduire à une position apportée par une substitution d'une étape précédente de surréduction. Cette stratégie est définie pour une chaîne de surréduction, à chaque étape, on définit l'ensemble des occurrences basiques, c'est-à-dire, l'ensemble des positions où la surréduction est possible. Une définition plus formelle peut être la suivante :

Définition 1.12 Soit la surdérivation suivante :

$$t_0 \rightsquigarrow^{*}_{[p_0,\, g_0 \to d_0, \sigma_0]} \cdots \rightsquigarrow^{*}_{[p_{n-1},\, g_{n-1} \to d_{n-1}, \sigma_{n-1}]} t_n$$

Soient $U_0, ..., U_{n-1}$ des ensembles d'occurrences non variables de $t_0, ..., t_{n-1}$ respectivement. On dit que la dérivation est **basique** ssi

- $U_0 = \overline{O}(t_0)$
- $\forall i = 0, ..., n-1, p_i \in U_i$
- $\forall i = 0, ..., n-2, U_{i+1} = (U_i - \{v \in U_i \ / \ p_i \leq v\}) \cup \{p_i.v \ / \ v \in \overline{O}(d_i)\}$

Pour chaque i, les occurrences de U_i sont les occurrences dites basiques, les autres, c'est-à-dire $\overline{O}(t_i) - U_i$, sont les occurrences dites protégées. $\diamond$

Exemple 1.13 Soit R le système de réécriture canonique suivant :

$$\begin{array}{lcrcl} (r_1) & : & 0 + x_1 & \to & x_1 \\ (r_2) & : & s(0) & \to & 1 \end{array}$$

et soit le terme $x + s(y)$ à surréduire. Deux branches sont possibles :
$x + s(y) \rightsquigarrow_{[\epsilon,(r_1),\{x \leftarrow 0, x_1 \leftarrow s(y)\}]} s(y) \rightsquigarrow_{[\epsilon,(r_2),\{y \leftarrow 0\}]} 1$
$x + s(y) \rightsquigarrow_{[1,(r_2),\{y \leftarrow 0\}]} x + 1 \ \rightsquigarrow_{[\epsilon,(r_1),\{x \leftarrow 0\}]} 1$.
Les deux branches arrivent à la même solution $\{x \leftarrow 0, y \leftarrow 0\}$, mais la première branche n'est pas basique (car après le premier pas, l'occurrence ϵ de $s(y)$ est protégée puisque $s(y)$ a été apporté par la substitution), tandis que la deuxième l'est. La surréduction basique interdit donc de poursuivre la première branche après la première étape.

C'est Hullot qui a prouvé que le théorème 1.10 de complétude restait valable pour la surréduction basique [HUL80b]. Il a donné en plus des conditions suffisantes de terminaison, à savoir :

Théorème 1.14 *Si toute surdérivation basique, issue d'un membre droit de règle d'un système de réécriture, termine alors toute surdérivation basique termine, dans ce système de réécriture.*

Comme nous l'avons vu précédemment, il y a une correspondance entre une chaîne de surréduction et une chaîne de réécriture. Une surdérivation basique correspond à une chaîne de réécriture de bas en haut (innermost), on doit donc toujours choisir l'occurrence le plus bas possible dans le terme. Pour choisir entre deux positions de même profondeur, il faut compléter la stratégie de sélection de l'occurrence à surréduire.

1.2.5 La surréduction basique de gauche à droite

Tout d'abord, définissons la notion “ à gauche de ” pour une occurrence.

Définition 1.15 Soient u et v, deux occurrences d'un terme t. On dit que u est à gauche de v ssi il existe des occurrences p, u', v' et deux entiers j, j' tels que

- $u = p.j.u'$,
- $v = p.j'.v'$ et
- $j < j'$.

On notera $u \lhd v$. $\diamond$

La stratégie de gauche à droite correspond donc à choisir l'occurrence la plus à gauche lorsqu'on a le choix entre deux occurrences comparables avec $\lhd$. Herold dans [HER86] a formalisé cette idée et l'a combinée avec la surréduction basique pour obtenir la **surréduction basique de gauche à droite**.

Définition 1.16 C'est la même que pour la surréduction basique (définition 1.12) mais avec un nouvel ensemble d'occurrences basiques défini par :

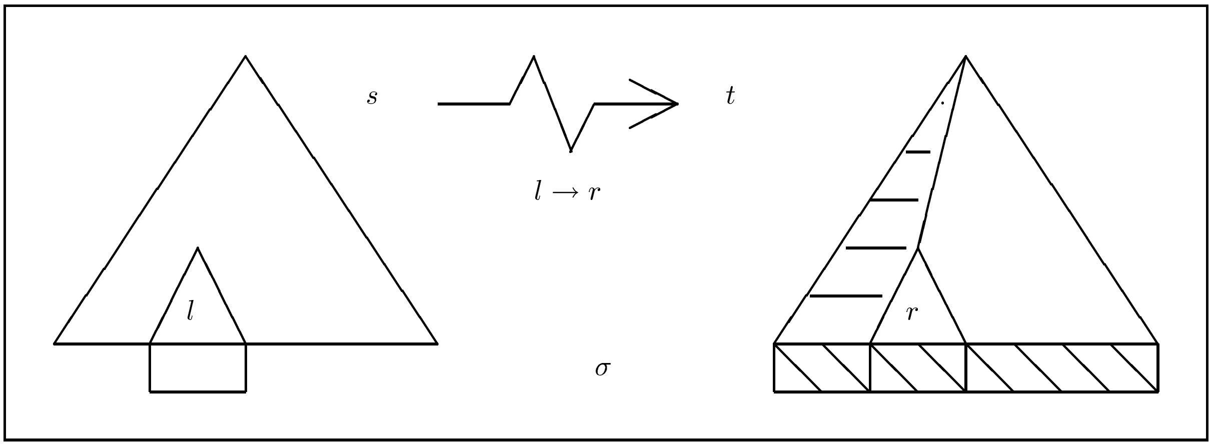
$$U_{i+1} = (U_i - \{v \in U_i \ / \ p_i \leq v \ ou \ v \lhd p_i\}) \cup \{p_i.v \ / \ v \in \overline{O}(d_i)\}$$

$\diamond$

Exemple 1.17 Soit R le système canonique constitué de l'unique règle (r_1) $g(0) \rightarrow 0$. Essayons de surréduire le terme $g(x) + g(y)$. Nous obtenons là encore deux branches :
$g(x) + g(y) \rightsquigarrow_{[1,(r_1),\{x \leftarrow 0\}]} 0 + g(y) \rightsquigarrow_{[2,(r_1),\{y \leftarrow 0\}]} 0 + 0$
$g(x) + g(y) \rightsquigarrow_{[2,(r_1),\{y \leftarrow 0\}]} g(x) + 0 \rightsquigarrow_{[1,(r_1),\{x \leftarrow 0\}]} 0 + 0$.
Les deux branches sont basiques, mais en revanche, seule la première est basique de gauche à droite, puisque dans la deuxième branche, après la première étape, l'occurrence 1 est protégée (elle est à gauche de 2) et donc interdite.

Le théorème 1.10 de complétude a été établi par Herold pour la surréduction basique de gauche à droite. A une chaîne de réécriture de gauche à droite (leftmost) et de bas en haut (innermost) correspond une surdérivation basique de gauche à droite.

Les deux stratégies précédentes peuvent être résumées sur le schéma suivant.



La partie hachurée en biais correspond aux occurrences protégées pour la surréduction basique. Si elle est de gauche à droite, on protège en plus les occurrences qui se trouvent dans la partie hachurée horizontalement.

Ces notions ont été combinées dans la thèse de Réty [RÉT87, RÉT88] pour obtenir une méthode toujours (correcte et) complète de R-unification, à savoir la surréduction normalisante basique de gauche à droite.

1.2.6 La surréduction normalisante, basique de gauche à droite

Il s'agit d'une combinaison non triviale des stratégies vues précédemment, que nous allons présenter ici. La combinaison simple ne fournit plus une méthode complète comme le montre l'exemple suivant.

Exemple 1.18 [RETY] Soit R le système de réécriture canonique suivant :

$$\begin{array}{lcrcl} (r_1) & : & g_1(g_2(x',y'),g_2(x',y')) & \rightarrow & f(h(x'),h(y')) \\ (r_2) & : & f(x',x') & \rightarrow & x' \\ (r_3) & : & h(h(x')) & \rightarrow & x' \end{array}$$

Essayons de résoudre $g_1(g_2(y,y),z) \stackrel{?}{=} 0$. On obtient, par surréduction basique où les occurrences protégées sont précédées du symbole %, l'arbre suivant.

$g_1(g_2(y,y),z) \stackrel{?}{=} 0 \rightsquigarrow_{[1,(r_1),\{z \leftarrow g_2(y,y)\}]} f(h(y),h(y)) \stackrel{?}{=} 0$

$\rightsquigarrow_{[1,(r_2),\{x' \leftarrow h(y)\}]} \%h(y) \stackrel{?}{=} 0$

$\rightsquigarrow_{[1.1,(r_3),\{y \leftarrow h(x')\}]} f(x',h(\%h(x'))) \stackrel{?}{=} 0 \rightsquigarrow_{[1.2,(r_3),\emptyset]} f(x',x') \stackrel{?}{=} 0 \rightsquigarrow_{[1,(r_2),\emptyset]} x' \stackrel{?}{=} 0$

On obtient donc deux branches, la première n'aboutit à aucune solution car on ne peut pas appliquer d'étape de surréduction ni sur $h(y)$ (car l'occurrence est protégée), ni sur 0. La deuxième donne la solution $\{y \leftarrow h(0), z \leftarrow g_2(h(0),h(0))\}$, qui est l'unique solution.

Si maintenant, on veut faire de la surréduction normalisante, l'équation $f(h(y),h(y)) \stackrel{?}{=} 0$ est normalisée et remplacée par $h(y) \stackrel{?}{=} 0$ et on perd ainsi la deuxième branche. Or si l'on protège $h(y)$ pour être basique, même pendant les étapes de réécriture, on voit sur cet exemple, que l'on perd la complétude de la méthode. De plus, il se peut que la normalisation ne respecte pas les occurrences basiques.

Pour contrer tout cela, Réty a défini un nouvel ensemble d'occurrences, appelées occurrences faiblement basiques, pour la surréduction normalisante basique. Pour avoir la définition formelle de ce nouvel ensemble, on peut se reporter à [RÉT88], p59–69. Intuitivement, on peut dire que l'on évite de protéger, par rapport à la réécriture basique, les occurrences utilisées dans la substitution qui n'étaient pas protégées d'avance. On peut décomposer en deux parties, d'un côté la surréduction qui est basique de gauche à droite, et de l'autre, la mise en forme normale du terme après chaque étape de surréduction, qui se fait par une dérivation faiblement basique, c'est-à-dire qui évite de protéger de nouvelles occurrences tout en respectant les occurrences protégées d'avance. Nous aboutissons au théorème de complétude suivant.

Théorème 1.19 (Réty) *Soient R un système de réécriture canonique, et t_0 et t'_0 deux termes. L'ensemble des substitutions σ telles que :*

- *il existe une surdérivation normalisante basique de gauche à droite issue de $t_0 \stackrel{?}{=} t'_0$:*
$$t_0 \stackrel{?}{=} t'_0 \rightsquigarrow^{*}_{[\sigma_0]} \ldots \rightsquigarrow^{*}_{[\sigma_{n-1}]} t_n \stackrel{?}{=} t'_n$$
et t_n et t'_n sont syntaxiquement unifiables par l'unificateur principal β et $\beta \circ \sigma_{n-1} \circ \ldots \circ \sigma_0$ est normalisée sur $V(t_0 \stackrel{?}{=} t'_0)$.
- *$\sigma = \beta \circ \sigma_{n-1} \circ \ldots \circ \sigma_0$.*

est un ensemble complet de R-unificateurs de t_0 et t'_0.

Nous avons étudié en détail ces différentes stratégies car elles vont réapparaître au cours de cette thèse. Il existe bien d'autres stratégies de surréduction dans les systèmes de réécriture canoniques, nous allons les présenter de façon plus rapide maintenant.

1.2.7 D'autres stratégies de surréduction dans les systèmes de réécriture canoniques

Comme précédemment, on peut définir toute autre stratégie de choix de l'occurrence dans le sens de la largeur (de droite à gauche par exemple). Dans [BGM87] une procédure indéterministe englobe toutes ces stratégies.

Martelli, Moiso et Rossi dans [MMR86, MMR87], ont défini une stratégie de haut en bas (outermost), mais par une formulation totalement différente, dans laquelle on ne retrouve plus une correspondance avec la réécriture de haut en bas. La surréduction ne se fait qu'en tête des membres des équations, on dispose d'une règle de décomposition, ceci nous impose donc de travailler sur un ensemble d'équations.

Exemple 1.20 Soit R le système de réécriture canonique suivant :

$$\begin{array}{rcrcl} (r_1) & : & f(g(x')) & \rightarrow & f'(x') \\ (r_2) & : & h(0) & \rightarrow & g(0) \end{array}$$

Si l'on veut résoudre l'équation $f(h(x)) \stackrel{?}{=} f'(y)$ par la méthode de Martelli, Moiso et Rossi, on obtiendra :

$f(h(x)) \stackrel{?}{=} f'(y) \Rightarrow_{(r_1)} g(x') \stackrel{?}{=} h(x) \ \wedge \ f'(x') \stackrel{?}{=} f'(y)$

$\Rightarrow_{(r_2)} h(x) \stackrel{?}{=} h(0) \ \wedge \ g(x') \stackrel{?}{=} g(0) \ \wedge \ f'(x') \stackrel{?}{=} f'(y)$

$\Rightarrow_{(dec)} x \stackrel{?}{=} 0 \ \wedge \ y \stackrel{?}{=} 0$

Sur cet exemple on voit très bien que l'on ne peut pas faire correspondre des étapes de réécriture. En effet, aucune instanciation de $f(h(x)) \stackrel{?}{=} f'(y)$ ne permettrait de faire une étape de réécriture avec la règle (r_1), il faut appliquer d'abord (r_2). Cette stratégie permet aussi de diminuer l'espace de recherche grâce à des études sur les opérateurs de têtes, mais cela ne s'applique que dans un nombre très limité de cas. Un inconvénient de la méthode est que l'on peut surréduire les sous-termes des membres gauches, ce qui est source d'inefficacité.

Dershowitz et Sivakumar dans [DS87, DS88] ont amélioré le critère de terminaison précédent en introduisant un graphe de portée des opérateurs, c'est-à-dire, en construisant un graphe avec les symboles de fonctions de R. Ceci permet de ne regarder que les branches de surréduction susceptibles de donner des solutions.

Exemple 1.21 Soit R constitué de :

$$(r_1) \quad : \quad f(g(x')) \quad \rightarrow \quad f(x')$$

Si l'on veut résoudre l'équation $f(x) \stackrel{?}{=} g(y)$, on voit qu'il n'y a pas de solution, car un terme ayant pour symbole de tête f (resp. g) ne se surréduit qu'en un terme ayant pour symbole de tête f (resp. g). Mais la surréduction classique va donner la branche infinie suivante :

$$f(x) \stackrel{?}{=} g(y) \rightsquigarrow_{[1,(r_1),\{x \leftarrow g(x_1)\}]} f(x_1) \stackrel{?}{=} g(y) \rightsquigarrow_{[1,(r_1),\{x_1 \leftarrow g(x_2)\}]} f(x_2) \stackrel{?}{=} g(y)...$$

Le graphe de portée des opérateurs nous permet de dire immédiatement qu'il n'y a pas de solution.

Cette notion accompagnée d'une nouvelle formulation de la procédure de surréduction ainsi que des améliorations de cette méthode seront développées au chapitre suivant.

Une raison de l'inefficacité de la surréduction est la redondance dans le calcul des solutions, plusieurs chemins différents aboutissant à la même solution. Bockmayr, Krischer et Werner dans [KB91, BKW92, BKW93], proposent de remédier à cela avec la stratégie LSE normale de surréduction. Cette stratégie, tout en étant complète, a la propriété suivante : deux chemins différents de surréduction LSE normale ne peuvent pas générer la même substitution. Pour cela ils ont numéroté les règles du système de réécriture et ont combiné la stratégie de bas en haut et de gauche à droite avec une stratégie de choix de la règle à utiliser pour surréduire. Ainsi ils évitent de calculer plusieurs fois la même solution même si le système de réécriture a des paires critiques. De plus, après chaque étape de LSE surréduction, une étape de normalisation est faite.

D'autres méthodes de surréduction ont été développées mais avec des conditions supplémentaires sur le système de réécriture pour que la complétude soit garantie.

Nous allons maintenant décrire en particulier deux méthodes d'E-unification lorsque E est quelconque.

1.2.8 La méthode de Gallier et Snyder

C'est une méthode qui opère par transformations de systèmes de termes comme pour l'unification standard, développée par Martelli Montanari [MM82], mais adaptée à toute théorie équationnelle E. Elle est détaillée dans [GS89]. Donnons d'abord quelques définitions propres à cette méthode. On appellera système un multi-ensemble de paires de termes que l'on notera S. Pour rester cohérent avec ce qui précède, nous noterons une paire de termes $s \stackrel{?}{=} t$. La substitution θ est un E-unificateur de S ssi $\forall s \stackrel{?}{=} t \in S$ on a $\theta s =_E \theta t$, (la notion de $=_E$ est la même que précédemment). L'ensemble des E-unificateurs de S sera noté $UE(S)$. La paire de termes $x \stackrel{?}{=} t$ est en forme résolue dans S et x est dite variable résolue ssi x est une variable qui apparaît nulle part ailleurs dans S, et en particulier $x \notin V(t)$. Le système S sera en forme résolue si toutes les paires de termes qui le composent sont en formes résolues dans S. Si un système $S = \{x_1 \stackrel{?}{=} t_1, ..., x_n \stackrel{?}{=} t_n\}$ est en forme résolue alors, l'ensemble restreint à la substitution σ_S, définie par $\sigma_S = \{x_1 \leftarrow t_1, ..., x_n \leftarrow t_n\}$ est un ensemble complet de E-unificateurs de S, que l'on notera $ECU_E(S)$.

Voici maintenant les règles de transformations d'un système S, utilisées pour trouver les E-unificateurs de S.

1) Trivialisation

$$\frac{S \cup \{s \stackrel{?}{=} s\}}{S}$$

2) Décomposition

$$\frac{S \cup \{f(s_1, ..., s_n) \stackrel{?}{=} f(t_1, ..., t_n)\}}{S \cup \{s_1 \stackrel{?}{=} t_1, ..., s_n \stackrel{?}{=} t_n\}}$$

3) Instanciation

$$\frac{S \cup \{x \stackrel{?}{=} t\}}{\sigma S \cup \{x \stackrel{?}{=} t\}}$$

si $x \stackrel{?}{=} t$ n'est pas en forme résolue, $x \notin V(t)$ et $\sigma = \{x \leftarrow t\}$.

4) Surréduction

$$\frac{S \cup \{s \stackrel{?}{=} t\}}{S \cup \{s \stackrel{?}{=} l, r \stackrel{?}{=} t\}}$$

si s ou t est non variable on supposera que s est non variable, $l \doteq r \in E \cup E^{-1}$, $V(l \doteq r) \cap V(S \cup \{s \stackrel{?}{=} t\}) = \emptyset$.
et si l n'est pas une variable tête(s) = tête(l).

5) Imitation

$$\frac{S \cup \{x \stackrel{?}{=} t\}}{S \cup \{x \stackrel{?}{=} f(y_1,...,y_n), x \stackrel{?}{=} t\}}$$ si x est une variable et $f \in \Sigma_n$ avec $n > 0$, $y_1,...,y_n$ sont de nouvelles variables, $f = \text{tête}(t)$ si t n'est pas une variable.

Les trois premières règles de transformation correspondent à la mise en forme résolue d'un système S, lorsque celle-ci existe. Si S a une solution syntaxique, elle sera calculée en utilisant uniquement ces trois règles.

Pour montrer que deux termes sont E-égaux, on utilise une chaîne de dérivation entre les deux termes. La quatrième règle va simuler les étapes de cette dérivation. Elle ressemble à une étape de surréduction appliquée en tête du terme, excepté que les deux termes s et l ne sont pas forcément unifiables et qu'il n'y a pas de notion d'ordre. Cette règle ne doit pas être appliquée deux fois de suite sur un même terme.

La dernière règle doit être suivie immédiatement par la règle d'instanciation, appliquée à la paire $x \stackrel{?}{=} f(y_1,...,y_n)$. Elle peut être vue comme une règle de surréduction en sous-terme, on instancie x pour pouvoir le décomposer. Le choix du symbole de tête f est non déterministe lorsque t est aussi une variable.

Exemple 1.22 [Gallier, Snyder] Soit E le système suivant constitué d'une unique équation :

$$(e_1) \quad : \quad f(g(z)) \quad \doteq \quad z$$

Essayons de trouver un E-unificateur pour le système $S = \{h(x) \stackrel{?}{=} h(g(f(x)))\}$. Pour cela appliquons les règles de transformations présentées auparavant.
$\{h(x) \stackrel{?}{=} h(g(f(x)))\} \Rightarrow_{dec} \{x \stackrel{?}{=} g(f(x))\}$
$\Rightarrow_{imi} \{x \stackrel{?}{=} g(x_1), x \stackrel{?}{=} g(f(x))\}$
$\Rightarrow_{ins} \{x \stackrel{?}{=} g(x_1), g(x_1) \stackrel{?}{=} g(f(g(x_1)))\}$
$\Rightarrow_{dec} \{x \stackrel{?}{=} g(x_1), x_1 \stackrel{?}{=} f(g(x_1))\}$
$\Rightarrow_{sur} \{x \stackrel{?}{=} g(x_1), x_1 \stackrel{?}{=} z_1, f(g(z_1)) \stackrel{?}{=} f(g(x_1))\}$
$\Rightarrow_{ins} \{x \stackrel{?}{=} g(x_1), x_1 \stackrel{?}{=} z_1, f(g(z_1)) \stackrel{?}{=} f(g(z_1))\}$
$\Rightarrow_{tri} \{x \stackrel{?}{=} g(x_1), x_1 \stackrel{?}{=} z_1\}$
$\{x \leftarrow g(x_1)\}$ est donc un E-unificateur de $S = \{h(x) \stackrel{?}{=} h(g(f(x)))\}$, (on a bien évidemment restreint la substitution aux variables apparaissant dans le système à résoudre).

L'ensemble des règles de transformation fournit une méthode correcte et complète d'E-unification. La preuve de complétude est basée sur une notion d'arbre de preuve. A chaque E-unificateur du système S, on fait correspondre un arbre de preuve, cet arbre est 'simplifié' et il y a une équivalence entre cette 'simplification' et l'application des règles de transformation sur le système S. La simplification de l'arbre de preuve termine, et la racine de l'arbre simplifié est le système en forme résolue.

Cette méthode, qui a l'avantage d'être 'simple ', est inefficace pratiquement. En effet, la règle de surréduction (4) peut être appliquée sur une variable, ce qui crée beaucoup de nouvelles paires dans le système car chaque membre d'une équation de E est unifiable avec une variable. De plus la règle d'imitation comporte un indéterminisme lorsque les deux termes de la paire sont des variables, et aucune notion d'ordre n'apparaît dans les règles.

Dans l'objectif de remédier à ces problèmes, Gallier et Snyder ont proposé une amélioration de leur système de transformation. Ils ont introduit une notion d'ordre en séparant les équations en deux ensembles : E est l'ensemble des équations inorientables par rapport à l'ordre et R l'ensemble des équations orientables (des règles). Le nouvel ensemble de transformation est composé des règles de trivialisation, décomposition et instanciation de l'ancien système, et d'une nouvelle règle, la lazy paramodulation qui remplace les deux autres et qui est définie de la façon suivante :

4) Lazy paramodulation

$$\frac{S \cup \{s \stackrel{?}{=} t\}}{S \cup \{s|_p \stackrel{?}{=} l, s[p \leftarrow r] \stackrel{?}{=} t\}}$$

si p est une position non variable de s,
$l \doteq r \in E \cup E^{-1} \cup R$ à un renommage près,
et tête$(s|_p)$ = tête(l) si l est non variable.

Dans le cas où l n'est pas une variable, une étape de lazy paramodulation devra être suivie d'une étape de décomposition appliquée sur $s|_p \stackrel{?}{=} l$. Cette règle va permettre d'essayer toutes les surréductions possibles à partir d'une paire de termes.

La preuve de complétude de ce nouveau système est décomposée en deux parties. On commence par montrer que cet ensemble de règles de transformation est complet si $E \cup R$ forme un système confluent (Church Rosser) sur les termes clos, en utilisant un ordre de réduction total sur les termes clos. Puis ensuite, on montre que la génération de paires critiques, ce qui nous permet d'aboutir à un système confluent sur les termes clos, peut être simulée par la règle de lazy paramodulation. Gallier et Snyder conjecturent que la lazy paramodulation peut être faite uniquement pour $p = \epsilon$ lorsque ni s ni t ne sont des variables, ce qui améliorerait encore l'efficacité.

Nous allons maintenant présenter une autre méthode d'E-unification, celle de Dougherty et Johann [DJ90, DJ92].

1.2.9 La méthode de Dougherty et Johann

Cette méthode peut être vue comme une amélioration de la méthode présentée auparavant. Elle est décrite par un ensemble de règles de transformation de système et comporte notamment les mêmes règles de trivialisation, décomposition et instanciation que la méthode précédente. La différence se situe donc au niveau de la définition de la règle de paramodulation. Pour définir la règle de relaxed paramodulation, qui va remplacer celle de lazy paramodulation, nous avons besoin de quelques notions supplémentaires.

Définition 1.23 On dit que deux termes s et t sont **top-unifiables** ssi s et t ont le même symbole à chaque occurrence u qui est une occurrence non variable de s et de t. $\diamond$

On peut savoir si deux termes sont top-unifiables, si en appliquant récursivement la règle de décomposition sur les deux termes, on obtient un ensemble de paires de termes, dont chaque élément a au moins un côté réduit à un variable. Si deux termes sont unifiables, ils sont bien évidemment top-unifiables.

La notation $dec(s \stackrel{?}{=} t)$ représentera le système obtenu après avoir appliqué autant de fois que possible, la règle de décomposition en partant de la paire $s \stackrel{?}{=} t$.

4) Relaxed paramodulation

$$\frac{S \cup \{s \stackrel{?}{=} t\}}{S \cup \{dec(s|_p \stackrel{?}{=} l), s[p \leftarrow r] \stackrel{?}{=} t\}}$$

si p est une position non variable de s,
$l \doteq r \in E \cup E^{-1}$ à un renommage près,
et $s|_p$ et l sont top-unifiables.

Cette règle est donc plus restrictive que la règle de lazy paramodulation puisque $s|_p$ et l doivent être top-unifiables au lieu d'avoir uniquement le même symbole de tête lorsque l n'est pas une variable.

Exemple 1.24 [Dougherty, Johann] Considérons l'ensemble E d'équations suivant :

$$\begin{array}{lcrcl} (e_1) & : & 0+z & \doteq & z \\ (e_2) & : & (-y)+y & \doteq & 0 \\ (e_3) & : & (u+v)+w & \doteq & u+(v+w) \end{array}$$

et essayons de résoudre $(-0)+x \stackrel{?}{=} x$ en utilisant le nouveau système.
$\{(-0)+x \stackrel{?}{=} x\} \Rightarrow_{rela,(e_3)^{-1}} \{(-0) \stackrel{?}{=} u,\ x \stackrel{?}{=} v+w,\ (u+v)+w \stackrel{?}{=} x\}$
$\Rightarrow_{rela,(e_1)} \{u \stackrel{?}{=} (-0),\ v \stackrel{?}{=} 0,\ w \stackrel{?}{=} z,\ z \stackrel{?}{=} x,\ (u+v)+w \stackrel{?}{=} x\}$
$\Rightarrow_{inst^*} \{u \stackrel{?}{=} (-0),\ v \stackrel{?}{=} 0,\ w \stackrel{?}{=} x,\ z \stackrel{?}{=} x,\ ((-0)+0)+x \stackrel{?}{=} x\}$
$\Rightarrow_{rela,(e_2)} \{u \stackrel{?}{=} (-0),\ v \stackrel{?}{=} 0,\ w \stackrel{?}{=} x,\ z \stackrel{?}{=} x,\ y \stackrel{?}{=} 0,\ y \stackrel{?}{=} 0,\ 0+x \stackrel{?}{=} x\}$
$\Rightarrow_{rela,(e_1)} \{u \stackrel{?}{=} (-0),\ v \stackrel{?}{=} 0,\ w \stackrel{?}{=} x,\ z \stackrel{?}{=} x,\ y \stackrel{?}{=} 0,\ y \stackrel{?}{=} 0,\ z' \stackrel{?}{=} x,\ z' \stackrel{?}{=} x\}$
$\Rightarrow_{inst^*,triv^*} \{u \stackrel{?}{=} (-0),\ v \stackrel{?}{=} 0,\ w \stackrel{?}{=} x,\ z \stackrel{?}{=} x,\ y \stackrel{?}{=} 0,\ z' \stackrel{?}{=} x\}$

Ce système est en forme résolue et donne comme solution l'identité. Mais ce n'est pas la seule dérivation possible puisqu'il y a toujours de l'indéterminisme au niveau du choix de l'équation à utiliser dans la règle de relaxed paramodulation. Ainsi nous obtenons :
$\{(-0)+x \stackrel{?}{=} x\} \Rightarrow_{rela,(e_2)} \{0 \stackrel{?}{=} y,\ x \stackrel{?}{=} y,\ 0 \stackrel{?}{=} x\}$
$\Rightarrow_{inst^*,triv} \{y \stackrel{?}{=} 0,\ x \stackrel{?}{=} 0\}$
Ce qui aboutit à la solution $\{x \leftarrow 0\}$. On peut remarquer qu'à partir du système $\{(-0)+x \stackrel{?}{=} x\}$, on ne peut pas appliquer la règle de relaxed paramodulation avec l'équation e_1 car $0+z$ et $(-0)+x$ ne sont pas top-unifiables. En revanche, on aurait pu appliquer la règle de lazy paramodulation puisque les deux termes ont le même symbole de tête, mais on serait arrivé à une impasse :
$\{(-0)+x \stackrel{?}{=} x\} \Rightarrow_{lazy,(e_1)} \{(-0)+x \stackrel{?}{=} 0+z,\ z \stackrel{?}{=} x\}$
$\Rightarrow_{dec} \{(-0) \stackrel{?}{=} 0,\ x \stackrel{?}{=} z\}$
et il n'y a pas de solution pour $(-0) \stackrel{?}{=} 0$.

Théorème 1.25 (Dougherty, Johann) *Les trois règles standards plus la règle de relaxed paramodulation, forment une méthode complète d'E-unification pour un ensemble quelconque d'équations E.*

Comme pour Gallier et Snyder, Dougherty et Johann ont commencé par prouver que leur méthode était complète pour un ensemble d'équations confluent, puis ils ont étendu leur preuve à un ensemble quelconque d'équations.

Chapitre 2

E-unification par surréduction contrainte

Dans ce chapitre nous nous intéressons au problème de l'unification modulo une théorie équationnelle E quelconque. La surréduction est une méthode de résolution d'équations dans les théories équationnelles qui peuvent être représentées par un système de réécriture *canonique*. Beaucoup de théories n'ont pas cette propriété. Cependant la complétion ordonnée, décrite dans le premier chapitre, permet d'obtenir un système final, peut-être infini, confluent sur les termes clos. La surréduction peut être combinée avec la complétion ordonnée pour obtenir une méthode d'E-unification quelle que soit la théorie équationnelle E [OS90].

Pour améliorer l'efficacité de la surréduction, Hullot dans [HUL80a], a introduit la surréduction basique qui nous permet de réduire l'espace de recherche tout en restant complet. Une notion similaire, mais pour la complétion, a été introduite plus récemment dans [SL91]. La notion de basicité a d'abord été exprimée grâce à des marquages pour repérer les positions apportées par les substitutions des étapes antérieures, et ainsi les protéger pour les étapes suivantes. Le formalisme de réécriture sous contraintes, défini dans [KKR90], semble idéal pour exprimer la notion de basicité. Et [NR92a, NR92b] ont donné le premier système d'inférence de complétion basique dans le cadre contraint.

Nous allons présenter ici une combinaison entre la complétion basique et la surréduction basique, les deux exprimées dans un cadre contraint, qui va nous fournir une méthode complète et générale d'E-unification. Dans une première partie, nous allons définir notre notion de contrainte ainsi que les nouvelles définitions de réécriture, surréduction, superposition, adaptées à ce cadre de travail. Nous détaillerons ensuite les étapes de simplification autorisées lors de la complétion, qui vont nous permettre d'obtenir une méthode efficace (dans des exemples tels que la complétion de la théorie des groupes, on a un système final similaire à celui obtenu après complétion ordinaire), et complète pour la surréduction (conservation du lemme de correspondance entre réécriture et surréduction, adapté au formalisme des contraintes). Nous présenterons ensuite notre système d'inférence, qui couvrira à la fois la complétion et la surréduction dans un cadre contraint, et qui autorise des équations initiales avec contraintes. Puis nous donnerons les preuves de correction et de complétude de notre méthode ainsi que quelques exemples. Ce système d'inférence est implanté dans le logiciel UREVEAL qui sera présenté au dernier chapitre.

2.1 Les nouvelles définitions dans le cadre contraint

Dans la suite nous allons utiliser plusieurs symboles d'égalité mais le symbole simple '$=$' sera exclusivement réservé à l'égalité syntaxique. Le symbole '$>$' représentera un ordre de simplification fixé total sur les termes clos (en abrégé : c.s.o.).

2.1.1 Les contraintes

Nous disposons de deux types de contraintes, les contraintes égalitaires de la forme $t = t'$ et les contraintes d'ordres de la forme $t > t'$ où t et t' sont deux termes quelconques. Nous autorisons les contraintes dans les équations initiales, mais nous devons pouvoir distinguer à tout moment de notre procédure ces contraintes de celles introduites par une étape de superposition ou de surréduction. Pour cela on introduit deux nouveaux symboles '$=_n$' et '$>_n$' ('n' désignant nouveau) pour représenter les contraintes égalitaires et les contraintes d'ordres créées, dans les étapes ultérieures, par notre système d'inférence.

Une contrainte générale sera une conjonction finie entre tous ces types de contraintes, et sera de manière générale, représentée entre des doubles crochets $[\![\ ]\!]$; elle sera souvent désignée par les lettres c, C, T. Pour une contrainte donnée $[\![C]\!]$, on désignera par $SOL(C)$ l'ensemble des substitutions τ telles que $\tau(C)$ s'évalue à 'vrai', c'est-à-dire τ est un unificateur *syntaxique* de toutes les contraintes égalitaires et pour toutes les contraintes d'ordres $t > t'$ ou $t >_n t'$ on a τt supérieur à $\tau t'$ avec l'ordre fixé au début de la procédure. Une contrainte $[\![C]\!]$ sera **insatisfiable** ssi $SOL(C)$ est vide. Une contrainte sera en **forme résolue** si sa partie égalitaire est uniquement composée de contraintes du type $x = t$ ou $x =_n t$. Ceci peut être obtenu en appliquant l'algorithme d'unification syntaxique décrit dans le premier chapitre; si cet algorithme échoue, la contrainte est insatisfiable. Pour toute contrainte $[\![C]\!]$, $V(C)$ sera l'ensemble des variables apparaissant dans la contrainte C et $V_a(C)$ l'ensemble des variables assignées par la contrainte C, c'est-à-dire $x \in V_a(C)$ si C contient une contrainte de la forme $x = t$ ou $x =_n t$ où t est un terme quelconque et C est en forme résolue.

Un **terme contraint** est une paire composée d'un terme usuel et d'une contrainte; il sera généralement représenté par $s[\![C]\!]$. Une **équation contrainte** est une paire de termes contraints ayant mêmes contraintes, que l'on notera $s \doteq t[\![C]\!]$. L'orientation d'une équation ne jouant (dans notre approche), aucun rôle théorique, on considérera les mots **règle** et équation comme synonymes. On dira qu'un terme non contraint t' est un **propagé** d'un terme contraint $t[\![T]\!]$ ssi il existe une substitution τ telle que $\tau \in SOL(T)$ et $t' = \tau t$, et on notera quelquefois $t' = \overline{t}$.

Lorsqu'on cherchera à résoudre une équation contrainte, on parlera plutôt d'unificande. Un **unificande** est une paire composée d'une équation et d'une contrainte que l'on notera sous la forme $s \stackrel{?}{\doteq} t[\![C]\!]$. Pour un ensemble de règles R quelconque, une solution modulo R d'un unificande sera par définition une substitution σ telle que $\sigma \in SOL(C)$ et $\sigma s =_R \sigma t$.

La notion de réécriture joue un rôle très important, à la fois dans les étapes de complétion pour avoir une bonne simplification sans laquelle aucune efficacité n'est envisageable, et dans les étapes de surréduction pour pouvoir normaliser les unificandes.

2.1.2 La réécriture contrainte

Nous allons ici définir la notion de réécriture contrainte, c'est-à-dire la réécriture d'un terme contraint $t[\![T]\!]$ par une équation contrainte $g \doteq d[\![C]\!]$. Pour cela nous avons besoin de la notion de filtrage entre deux termes contraints. Intuitivement nous pouvons dire qu'un c-filtre (un filtre entre deux termes contraints) est un filtre entre les propagés des deux termes. D'autre part il faut qu'une étape de réécriture conserve la partie protégée d'un terme (la contrainte), elle ne doit ni protéger de nouvelles parties ni déprotéger d'anciennes parties protégées. Cette idée est similaire à celle de la réécriture faiblement basique de [RÉT87].

Si on réécrit le terme $f(g(a),x)[\![x=g(b)]\!]$ avec la règle $f(y,z) \doteq i(y,z)[\![z=g(z')]\!]$ on doit obtenir $i(g(a),x)[\![x=g(b)]\!]$ modulo peut-être un renommage de la variable x. Intuitivement le filtre est $\{y \leftarrow g(a), z \leftarrow g(b)\}$. La première partie de la substitution doit-être appliquée sur le membre droit de la règle alors que la deuxième partie doit être mise dans la contrainte. Dans notre c-filtre nous devons donc avoir une partie propageable (la partie avec laquelle on doit instancier le membre droit de la règle utilisée) et une partie non propageable (la partie que l'on doit mettre dans la contrainte). Notre c-filtre sera donc représenté par $\{y \leftarrow_p g(a), z \leftarrow_{np} g(b)\}$. Pour tout c-filtre σ, on notera σ_p et σ_{np} respectivement, les parties propageables et non propageables de σ.

Nous avons besoin de définir un passage entre contrainte et substitution de façon à obtenir un c-filtre directement utilisable. De toute contrainte satisfiable C, on peut obtenir de sa partie égalitaire, une solution et donc une substitution unique; on résout la partie égalitaire de C en appliquant l'algorithme d'unification syntaxique, à toutes les contraintes qui la composent, on obtient une substitution qui est l'unificateur le plus général.

Soit $t[\![C]\!]$ un terme contraint, on notera :

- UC l'unificateur le plus général de la partie égalitaire de C,
- $UC_{np}(t) = UC|_{V(t)}$ on restreint UC aux variables apparaissant dans t, de plus les instanciations obtenues seront non propageables,
- $\overline{UC_p}(t)$ sera le complémentaire de $UC_{np}(t)$: c'est-à-dire on restreint UC aux variables n'apparaissant pas dans t, et les instanciations de cette substitution seront propageables.

Exemple 2.1 Soit le terme contraint $f(x,g(y))[\![x=h(a) \wedge y=b \wedge z=i(u) \wedge x>i(u)]\!]$ on a :

$UC = \{x \leftarrow h(a), y \leftarrow b, z \leftarrow i(u)\}$

$UC_{np}(f(x,g(y))) = \{x \leftarrow_{np} h(a), y \leftarrow_{np} b\}$

$\overline{UC_p}(f(x,g(y))) = \{z \leftarrow_p i(u)\}$

$UC_{np}(t)$ représente la restriction aux variables de t, de la substitution utilisée pour passer d'un terme contraint $t[\![C]\!]$ à son propagé $\overline{t}$ (en ne considérant que la partie égalitaire de la contrainte).

Une substitution peut être vue comme une contrainte, l'affectation '$x \leftarrow s$' est remplacée par la contrainte égalitaire '$x =_n t$'. Si dans une même substitution on a à la fois $x \leftarrow_p s$ et $x \leftarrow_{np} s$ alors on ne conserve que $x \leftarrow_{np} s$, cela correspond au fait que si dans un terme contraint, un même sous-terme apparaît à deux positions distinctes, l'une protégée et l'autre non, alors les deux positions doivent être considérées comme protégées. Donnons à présent une définition formelle d'un filtre contraint.

Définition 2.2 Soient $s[\![C]\!]$ et $t[\![T]\!]$ deux termes contraints. On dit qu'une substitution σ est un c-filtre (ou filtre contraint) de $s[\![C]\!]$ vers $t[\![T]\!]$ ssi :

(i) $\overline{\sigma(s[\![C]\!])} = \overline{t[\![T]\!]}$ (où la notation $\overline{}$ signifie le terme propagé),

(ii) σ admet une décomposition $\sigma = \sigma_p \cup \sigma_{np}$ sur $D(\sigma) = V(s[\![C]\!])$, telle que :

- $\forall x \in D(\sigma_p)$, $\sigma_p(x)$ soit un sous-terme de t,
- $\forall x \in D(\sigma_{np})$, $\sigma_{np}(x)$ soit un sous-terme de $[\![T]\!]$. $\diamond$

Nous présentons maintenant un système d'inférence qui permet de calculer un c-filtre entre deux termes contraints. Pour cela nous avons besoin de définir deux nouveaux symboles de fonctions binaires $\lhd$ et $\lhd_c$ partant de l'ensemble des termes contraints, et à valeur dans l'ensemble des substitutions. Le symbole $\lhd_?$ sera utilisé lorsque la règle d'inférence sera valable aussi bien pour $\lhd$ que pour $\lhd_c$. Soient deux termes contraints qui n'ont pas de variables en commun, un c-filtre du premier sur le second est calculé en appliquant le plus possible, l'ensemble des règles d'inférences suivantes :

1) Décomposition

$$\frac{S \wedge f(s_1, ..., s_n)[\![C]\!] \lhd_? f(s'_1, ..., s'_n)[\![C']\!]}{S \wedge s_1[\![C]\!] \lhd_? s'_1[\![C']\!] \wedge ... \wedge s_n[\![C]\!] \lhd_? s'_n[\![C']\!]}$$

2a) Collision

$$\frac{S \wedge f(s_1, ..., s_n)[\![C]\!] \lhd_? g(s'_1, ..., s'_k)[\![C']\!]}{Echec}$$

2b) Collision variable

$$\frac{S \wedge f(s_1, ..., s_n) \lhd x}{Echec}$$

3) Instanciation à gauche

$$\frac{S \wedge x[\![C \wedge x = f(s_1, ..., s_n)]\!] \lhd_c t'[\![C']\!]}{S \wedge f(x_1, ..., x_n)[\![C \wedge x_1 = s_1 \wedge x_n = s_n]\!] \lhd_c t'[\![C']\!] \wedge \{x \leftarrow_p f(x_1, ..., x_n)\}}$$

si t' n'est pas une variable et
$x_1, ..., x_n$ sont des nouvelles variables.

4) Instanciation à droite

$$\frac{S \wedge t[\![C]\!] \lhd_c x[\![C']\!]}{S \wedge \overline{t} \lhd \overline{x} \wedge UC_{np}(t)}$$ si t est non variable ou $t \in V_a(C)$.

5) Solution propageable

$$\frac{S \wedge x[\![C]\!] \lhd_c t'[\![C']\!]}{S \wedge \{x \leftarrow_p t'\}}$$ si $x \notin V_a(C)$.

6) Solution non propageable

$$\frac{S \wedge x \lhd t'}{S \wedge \{x \leftarrow_{np} t'\}}$$

Des exemples détaillés de calcul de filtre contraint seront donnés dans le paragraphe suivant. Les deux premières règles sont les règles classiques de décomposition lorsque les symboles de tête sont identiques et d'échec lorsqu'ils sont différents. La règle *3)* correspond à une propagation du terme à filtrer, cette propagation est faite pas à pas pour ne pas dépasser la contrainte du côté droit. La règle *4)* fait une propagation des deux termes et ajoute au résultat la substitution utilisée pour la propagation du terme de gauche, toutes les instanciations obtenues après cette étape seront non propageables puisqu'elles associeront à une variable du côté gauche une partie contrainte du côté droit. Dans la règle *4)*, comme dans la règle *5)*, la contrainte du côté droit (C') n'est pas conservée mais l'information qu'elle contient, qui pourrait être perdue, sera rajoutée au niveau de la règle de réécriture. Les deux dernières règles trouvent les instanciations, propageables lorsqu'elles correspondent à des parties non contraintes du côté droit et non propageables sinon.

Cet algorithme termine toujours et on obtient soit un échec, et alors il n'y a pas de c-filtre entre nos deux termes contraints, soit un ensemble de la forme $\{x_1 \leftarrow s_1 \wedge ... \wedge x_n \leftarrow s_n\}$ où les assignements sont pour certains propageables, et pour d'autres non propageables. Si après instanciation maximum, (c'est-à-dire si $\forall i, j = 1...n, x_j \notin V(s_i)$) on a $\forall i, j = 1...n, x_i = x_j \Rightarrow s_i = s_j$ alors, $\{x_1 \leftarrow s_1, ..., x_n \leftarrow s_n\}$ est un c-filtre entre les deux termes contraints.

Regardons maintenant comment on va utiliser ce c-filtre dans la définition de la réécriture contrainte.

Définition 2.3 Un terme contraint $t[\![T]\!]$ se réécrit en $t'[\![T']\!]$ à la position p et avec l'équation contrainte $g \doteq d[\![C]\!]$, ssi :

i) il existe un c-filtre τ de $g[\![C]\!]$ sur $t|_p[\![T]\!]$, c'est-à-dire τ est solution de $g[\![C]\!] \lhd_c t|_p[\![T]\!]$,

ii) $\sigma = \tau \circ \overline{UC_p}(g)$ est solution de C et $\sigma g > \sigma d$,

iii) $t' = t[p \leftarrow \sigma_p d]$ et $T' = T \wedge \sigma_{np}$.

On notera $t[\![T]\!] \longrightarrow_{[p, g \doteq d[\![C]\!], \sigma]} t'[\![T']\!]$ ou plus simplement $t[\![T]\!] \longrightarrow t'[\![T']\!]$. $\diamond$

Dans le deuxième point, $\overline{UC_p}(g)$ correspond à l'instanciation des variables contraintes de d qui ne sont pas dans g, (i.e. la partie de la contrainte que l'on aurait pu perdre en appliquant les règle *4)* ou *5)* du calcul du filtre contraint). Dans le troisième point, on transforme la substitution $\sigma_{np} = \{x_1 \leftarrow_{np} s_1, ..., x_k \leftarrow_{np} s_k\}$ en la contrainte $[\![x_1 =_? s_1 \wedge ... \wedge x_k =_? s_k]\!]$, avec '$=_?$' signifie '$=$' si s_i provient d'une contrainte initiale et '$=_n$' sinon.

2.1.3 Exemples de réécriture contrainte

Dans les exemples qui vont suivre, on supposera que l'on dispose d'un ordre de simplification total sur les termes clos, et qui oriente toutes nos équations de la gauche vers la droite.

Exemple 2.4 Essayons de réécrire le terme $f(g(a), x)[\![x = g(i(b))]\!]$ avec la règle contrainte $f(y, g(z)) \doteq h(y, z)[\![z = i(z')]\!]$. Détaillons le calcul du c-filtre.

$$
\begin{array}{l}
\quad f(y, g(z))[\![z = i(z')]\!] \lhd_c f(g(a), x)[\![x = g(i(b))]\!] \\
\longrightarrow_1 \; y[\![z = i(z')]\!] \lhd_c g(a)[\![x = g(i(b))]\!] \;\wedge\; g(z)[\![z = i(z')]\!] \lhd_c x[\![x = g(i(b))]\!] \\
\longrightarrow_5 \; y \leftarrow_p g(a) \;\wedge\; g(z)[\![z = i(z')]\!] \lhd_c x[\![x = g(i(b))]\!] \\
\longrightarrow_4 \; y \leftarrow_p g(a) \;\wedge\; g(i(z')) \lhd g(i(b)) \;\wedge\; z \leftarrow_{np} i(z') \\
\longrightarrow_{1-1} \; y \leftarrow_p g(a) \;\wedge\; z' \lhd b \;\wedge\; z \leftarrow_{np} i(z') \\
\longrightarrow_6 \; y \leftarrow_p g(a) \;\wedge\; z' \leftarrow_{np} b \;\wedge\; z \leftarrow_{np} i(z')
\end{array}
$$

On obtient donc le match contraint $\tau = \{y \leftarrow_p g(a), z' \leftarrow_{np} b, z \leftarrow_{np} i(b)\}$, d'autre part, $\overline{UC_p}(f(y, g(z))) = Id$ puisque seulement z est instancié dans la contrainte et z est une variable du membre gauche de l'équation. Ce qui donne l'étape de réécriture suivante :

$$f(g(a), x)[\![x = g(i(b))]\!] \longrightarrow h(g(a), z)[\![x = g(i(b)) \;\wedge\; z = i(b)]\!]$$

On peut supprimer la contrainte $x = g(i(b))$ puisqu'elle est satisfiable et que x n'apparaît plus dans le terme. Le résultat de la réécriture sera donc $h(g(a), z)[\![z = i(b)]\!]$. Dans cet exemple, comme dans les suivants, les contraintes sont vues comme des contraintes initiales, mais si on avait eu $f(g(a), x)[\![x =_n g(i(b))]\!]$ à réécrire, on aurait obtenu $h(g(a), z)[\![z =_n i(b)]\!]$ puisque le terme $i(b)$ provient d'une contrainte nouvelle.

Exemple 2.5 Soient le terme $f(h(z))[\![z = i(a)]\!]$ et la règle $f(x) \doteq g(x)[\![x = h(i(y))]\!]$. Calculons le filtre contraint.

$$
\begin{array}{ll}
 & f(x)[\![x = h(i(y))]\!] \lhd_c f(h(z))[\![z = i(a)]\!] \\
\longrightarrow_1 & x[\![x = h(i(y))]\!] \lhd_c h(z)[\![z = i(a)]\!] \\
\longrightarrow_3 & h(x_1)[\![x_1 = i(y)]\!] \lhd_c h(z)[\![z = i(a)]\!] \ \wedge\ x \leftarrow_p h(x_1) \\
\longrightarrow_1 & x_1[\![x_1 = i(y)]\!] \lhd_c z[\![z = i(a)]\!] \ \wedge\ x \leftarrow_p h(x_1) \\
\longrightarrow_4 & i(y) \lhd i(a) \ \wedge\ x \leftarrow_p h(x_1) \ \wedge\ x_1 \leftarrow_{np} i(y) \\
\longrightarrow_1 & y \lhd a \ \wedge\ x \leftarrow_p h(x_1) \ \wedge\ x_1 \leftarrow_{np} i(y) \\
\longrightarrow_6 & y \leftarrow_{np} a \ \wedge\ x \leftarrow_p h(x_1) \ \wedge\ x_1 \leftarrow_{np} i(y)
\end{array}
$$

Le c-filtre est donc $\{x \leftarrow_p h(x_1), y \leftarrow_{np} a, x_1 \leftarrow_{np} i(a)\}$ ce qui donne la réécriture :

$$f(h(z))[\![z = i(a)]\!] \longrightarrow g(h(x_1))[\![x_1 = i(a)]\!]$$

Dans le résultat, h n'est pas protégée car il provient d'une occurrence de $f(h(z))[\![z = i(a)]\!]$ qui n'était pas protégée. Essayons maintenant de réécrire un terme dont la contrainte est vide avec une règle qui possède une contrainte uniquement sur le membre droit.

Exemple 2.6 Soient le terme non contraint $f(a)[\![\ ]\!]$ et la règle $f(x) \doteq i(y)[\![y = g(x)]\!]$. Le filtre contraint calculé à partir de $f(x)[\![y = g(x)]\!] \lhd_c f(a)[\![\ ]\!]$ est $\{x \leftarrow_p a\}$, mais là $\overline{UC_p}(f(x)) = \{y \leftarrow_p g(x)\}$.La substitution utilisée pour la réécriture sera donc $\{y \leftarrow_p g(a), x \leftarrow_p a\}$ et on obtiendra :

$$f(a)[\![\ ]\!] \longrightarrow i(g(a))[\![\ ]\!]$$

L'exemple suivant examine le cas où la règle est non contrainte et où le terme est contraint.

Exemple 2.7 Soient la règle $f(y, g(z)) \doteq i(y, h(z))[\![\ ]\!]$ et le terme $f(g(a), x)[\![x = g(b)]\!]$. La procédure de calcul du filtre contraint se déroule de la façon suivante :

$$
\begin{array}{ll}
 & f(y, g(z))[\![\ ]\!] \lhd_c f(g(a), x)[\![x = g(b)]\!] \\
\longrightarrow_1 & y[\![\ ]\!] \lhd_c g(a)[\![x = g(b)]\!] \ \wedge\ g(z)[\![\ ]\!] \lhd_c x[\![x = g(b)]\!] \\
\longrightarrow_5 & y \leftarrow_p g(a) \ \wedge\ g(z)[\![\ ]\!] \lhd_c x[\![x = g(b)]\!] \\
\longrightarrow_4 & y \leftarrow_p g(a) \ \wedge\ g(z) \lhd g(b) \\
\longrightarrow_1 & y \leftarrow_p g(a) \ \wedge\ z \lhd b \\
\longrightarrow_6 & y \leftarrow_p g(a) \ \wedge\ z \leftarrow_{np} b
\end{array}
$$

Ce qui donne l'étape de réécriture suivante :

$$f(g(a), x)[\![x = g(b)]\!] \longrightarrow i(g(a), h(z))[\![z = b]\!]$$

Voici un exemple où l'on obtient une substitution dont une instanciation est à la fois propageable et non propageable.

Exemple 2.8 Soient le terme $f(g(a), y)[\![y = g(a)]\!]$ et l'équation $f(x, x) \doteq i(x, x)[\![\ ]\!]$. On obtient le c-filtre $\{x \leftarrow_p g(a), x \leftarrow_{np} g(a)\}$ que l'on utilise sous la forme $\{x \leftarrow_{np} g(a)\}$ dans l'étape de réécriture suivante :

$$f(g(a), y)[\![y = g(a)]\!] \longrightarrow i(x, x)[\![x = g(a)]\!]$$

Avec ces quelques exemples, on voit que la contrainte d'un terme que l'on réécrit ne fait que 'diminuer'. C'est ainsi qu'un terme sans contrainte ne peut se réécrire qu'en un terme sans contrainte et ceci quelque soit la règle utilisée. Cette propriété peut se voir directement avec le système d'inférence. En effet la partie nouvelle de la contrainte σ_{np} d'une étape de réécriture est la partie non propageable du c-filtre. Cette partie correspond à l'instanciation de variables par des termes ou sous-termes de la contrainte T du terme que l'on réécrit. Donc les 'termes' de σ_{np} sont des sous-termes de T, lorsque T est vide σ_{np} l'est aussi.

2.1.4 La surréduction contrainte

Cette notion est plus simple à définir que la réécriture contrainte puisqu'elle ne fait pas intervenir de mécanisme extérieur.

Définition 2.9 Un terme contraint $t[\![T]\!]$ se surréduit en $t'[\![T']\!]$ à la position p et avec l'équation contrainte $g \doteq d[\![C]\!]$ ssi :

i. p est une position non variable de t et $V(t[\![T]\!]) \cap V(g \doteq d[\![C]\!]) = \emptyset$,

ii. $t' = t[p \leftarrow d]$ et $T' = T \ \wedge \ C \ \wedge \ t|_p =_n g \ \wedge \ g >_n d$.

On notera $t[\![T]\!] \rightsquigarrow_{[p, g \doteq d[\![C]\!]]} t'[\![T']\!]$ ou plus simplement $t[\![T]\!] \rightsquigarrow t'[\![T']\!]$. $\diamond$

Dans cette définition, la substitution utilisée n'apparaît pas de façon explicite, elle est stockée dans la contrainte, c'est la solution la plus générale de la partie égalitaire de la contrainte. Bien évidemment, si la nouvelle contrainte est insatisfiable, en pratique on ne fera pas l'étape de surréduction. La notion de basicité pour cette surréduction est automatique : en effet puisque la partie apportée par la substitution est dans la contrainte et que l'on applique la surréduction uniquement à des positions non variables du terme, on ne fera jamais une étape à une occurrence apportée par une substitution.

Exemple 2.10 Soit la règle suivante :

$$(e_1) \quad : \quad g(x,y) \quad \doteq \quad f(x)[\![y = f(u) \ \wedge \ u > x]\!]$$

avec un ordre lpo où la précédence sur les opérateurs est $g > f > h$. Nous avons :

$$\begin{aligned}
g(f(z), f(h(z)))[\![C]\!] &\rightsquigarrow_{[\epsilon, e_1]} f(x)[\![C \ \wedge \ y = f(u) \ \wedge \ g(f(z), f(h(z))) =_n g(x,y) \ \wedge \ u > x \ \wedge \\
&\qquad g(x,y) >_n f(x)]\!] \\
g(f(z), h(f(z)))[\![C]\!] &\rightsquigarrow_{[\epsilon, e_1]} f(x)[\![C \ \wedge \ y = f(u) \ \wedge \ g(f(z), h(f(z))) =_n g(x,y) \ \wedge \ u > x \ \wedge \\
&\qquad g(x,y) >_n f(x)]\!] \\
g(h(z), f(z))[\![C]\!] &\rightsquigarrow_{[\epsilon, e_1]} f(x)[\![C \ \wedge \ y = f(u) \ \wedge \ g(h(z), f(z)) =_n g(x,y) \ \wedge \ u > x \ \wedge \\
&\qquad g(x,y) >_n f(x)]\!]
\end{aligned}$$

Les deux dernières surréductions seront éliminées car les contraintes sont insatisfiables : la deuxième n'a pas de solution pour sa partie égalitaire et la troisième a une contrainte d'ordre insatisfiable puisque $u > x$ 'devient' $z > h(z)$.

Nous utiliserons aussi de la surréduction normalisante, qui consiste à normaliser les termes après chaque étape de surréduction. On notera $t[\![T]\!] \rightsquigarrow\!\!\!\rightsquigarrow t''[\![T'']\!]$ ce qui est équivalent à $t[\![T]\!] \rightsquigarrow t'[\![T']\!] \longrightarrow^*_R t''[\![T'']\!]$ où $t''[\![T'']\!]$ est une forme normale de $t'[\![T']\!]$ par rapport à un ensemble d'équations R. Cette notion est similaire à celle de [RÉT87] mais exprimée dans un formalisme différent. La notion de surréduction contrainte, comme celle de réécriture est aussi définie sur les équations contraintes ou les unificandes, de la façon suivante : $s \doteq t[\![C]\!] \rightsquigarrow s' \doteq t'[\![C']\!]$ ssi $s[\![C]\!] \rightsquigarrow s'[\![C']\!]$ et $t' = t$, ou $t[\![C]\!] \rightsquigarrow t'[\![C']\!]$ et $s' = s$.

Le but de toutes ces définitions est de décrire une méthode d'E-unification pour toute théorie équationnelle E. Pour cela nous allons utiliser notre surréduction contrainte et en même temps nous allons compléter par complétion basique notre système d'équations E.

2.1.5 La superposition contrainte

Donnons ici la définition de ce qu'est la base du mécanisme de complétion basique à savoir, la génération de paires critiques entre deux équations contraintes.

Définition 2.11 Soient $g_1 \doteq d_1[\![C_1]\!]$ et $g_2 \doteq d_2[\![C_2]\!]$ deux équations contraintes. L'ensemble des paires critiques obtenues par superposition contrainte de la première équation dans la deuxième, est l'ensemble des équations :

$$g_2[u \leftarrow d_1] \doteq d_2[\![C_1 \ \wedge\ C_2 \ \wedge\ (g_2|_u =_n g_1) \ \wedge\ (g_1 >_n d_1) \ \wedge\ (g_2 >_n d_2)]\!]$$

où u est une position quelconque non variable de g_2. $\diamond$

Tout comme pour la surréduction contrainte, l'unificateur et les restrictions dues à l'ordre sont stockés dans la contrainte. Ces contraintes sont exprimées à l'aide des symboles $=_n$ et $>_n$ introduits précédemment car ce sont des contraintes créées par notre procédure. Là encore nous ne conserverons que les paires critiques qui auront une contrainte satisfiable.

Exemple 2.12 Soient les deux équations contraintes suivantes :

$$\begin{array}{lcrcl} (e_1) & : & f(x_1,y_1) & \doteq & h(y_1,x_1)[\![x_1 = a]\!] \\ (e_2) & : & g(f(x_2,i(y_2)),f(z_2,i(a))) & \doteq & b[\![x_2 = b]\!] \end{array}$$

avec un ordre lpo où la précédence sur les opérateurs est $f > g > h > a > b$. On peut créer une paire critique de (e_1) dans (e_2) en position 1, on obtient :

$$\begin{array}{l} g(h(y_1,x_1),f(z_2,i(a))) \doteq b[\![x_1 = a \ \wedge\ x_2 = b \ \wedge\ f(x_2,i(y_2)) =_n f(x_1,y_1) \ \wedge \\ \qquad\qquad f(x_1,y_1) >_n h(y_1,x_1) \ \wedge\ g(f(x_2,i(y_2)),f(z_2,i(a))) >_n b]\!] \end{array}$$

Cette paire critique ne sera pas conservée car la contrainte est insatisfiable, on a après résolution : $x_1 = a \ \wedge\ x_2 = b \ \wedge\ x_1 =_n x_2$, ce qui n'est pas possible. En revanche en position 2 on a :

$$\begin{array}{l} g(f(x_2,i(y_2)),h(x_1,y_1)) \doteq b[\![x_1 = a \ \wedge\ x_2 = b \ \wedge\ f(z_2,i(a)) =_n f(x_1,y_1) \ \wedge \\ \qquad\qquad f(x_1,y_1) >_n h(y_1,x_1) \ \wedge\ g(f(x_2,i(y_2)),f(z_2,i(a))) >_n b]\!] \end{array}$$

Cette paire critique sera conservée sous la forme :

$$g(f(x_2,i(y_2)),h(x_1,y_1)) \doteq b[\![x_1 = a \ \wedge\ x_2 = b \ \wedge\ y_1 =_n i(a)]\!]$$

les contraintes d'ordres sont vérifiées et la contrainte satisfiable $z_2 =_n a$ est supprimée puisque la variable z_2 n'apparaît plus dans l'équation.

Soient S un ensemble d'équations contraintes et $g \doteq d[\![C]\!]$ une équation contrainte, on notera par $CP(g \doteq d[\![C]\!], S)$ l'ensemble de toutes les paires critiques obtenues par superposition contrainte de l'équation $g \doteq d[\![C]\!]$ dans chaque équation de S, de chaque équation de S dans $g \doteq d[\![C]\!]$ et de $g \doteq d[\![C]\!]$ dans elle-même.

La génération de paires critiques permet de définir une procédure de complétion mais pour que celle-ci soit efficace, elle doit comporter une règle de simplification qui nous permettra d'éliminer les équations inutiles. Plusieurs problèmes se posent ici : d'une part nous sommes en complétion basique et d'autre part, nous voulons utiliser les règles contraintes obtenues pendant la complétion, dans des étapes de surréduction contrainte. Pour cela nous allons voir que la règle de simplification habituelle, c'est-à-dire la mise en forme normale des deux côtés d'une équation par réécriture avec des règles du système, ne permet pas d'obtenir un système final confluent sur les termes clos et donc une procédure d'E-unification complète. Dans le paragraphe suivant nous allons présenter les restrictions nécessaires à la réécriture contrainte pour avoir une simplification efficace tout en conservant la complétude de notre méthode d'E-unification.

2.2 Etapes de réécriture libre et quasi-normale

La première restriction (la réécriture libre), va nous permettre d'obtenir un système final d'équations confluent sur les termes clos, c'est-à-dire si deux termes clos sont égaux modulo l'ensemble des équations initiales, ils auront dans le système final la même forme normale. Elle traduit le fait explicite suivant : une équation possédant une partie protégée ne doit pas être utilisée pour simplifier un terme quand la partie protégée de l'équation correspond à une partie non protégée du terme. Ceci pour ne pas perdre de possibilité ultérieure de faire des paires critiques. En effet si une règle non contrainte est simplifiée par une règle contrainte, nous n'avons plus la possibilité de faire des paires critiques au niveau de la contrainte de la règle qui a simplifié ce qui aurait été possible au niveau de la règle simplifiée. Détaillons tout ceci par l'exemple suivant provenant de [NR92a].

Exemple 2.13 Soit le système E initial composé des trois équations suivantes :

$$\begin{array}{llrcl} (e_1) & : & a & \doteq & b[\![\ ]\!] \\ (e_2) & : & f(g(x)) & \doteq & g(x)[\![\ ]\!] \\ (e_3) & : & f(g(a)) & \doteq & b[\![\ ]\!] \end{array}$$

sous un ordre lpo où la précédence est $f > g > a > b$. Par superposition contrainte entre (e_2) et (e_3) on obtient :

$$(e_4) \quad : \quad g(x) \doteq b[\![f(g(x)) =_n f(g(a)) \ \wedge \ f(g(x)) >_n g(x) \ \wedge \ f(g(a)) >_n b]\!]$$

que l'on conservera sous la forme :

$$(e_4) \quad : \quad g(x) \doteq b[\![x =_n a]\!]$$

Si on n'impose aucune restriction pour la simplification, cette règle peut être utilisée pour simplifier la règle (e_3) et la remplacer par la règle :

$$(e_5) \quad : \quad f(b) \doteq b[\![\ ]\!]$$

La paire critique obtenue par superposition contrainte entre (e_2) et (e_4) :

$$(e_6) \quad : \quad f(b) \doteq g(x)[\![x =_n a]\!]$$

est simplifiée par (e_5) et trivialisée par (e_4). Ainsi le système final $\{(e_1), (e_2), (e_4), (e_5)\}$ est clos par superposition contrainte, toute nouvelle paire critique pouvant être trivialisée. Mais il n'existe pas de preuve par réécriture pour montrer que $g(b)$ et b sont E-égaux et pourtant : $g(b) =_{(e_2)} f(g(b)) =_{(e_1)} f(g(a)) =_{(e_3)} b$. Ceci provient du fait que nous n'avons pas fait la paire critique entre (e_1) et (e_3). La position 'libre' de a dans la règle (e_3) a été simplifiée par une règle où a était protégé, nous avons donc perdu la possibilité de faire une paire critique en position a. Nous devons donc interdire la simplification de la règle (e_3) par la règle (e_4).

2.2.1 La réécriture libre

Donnons maintenant la définition formelle d'une étape de réécriture libre (dans un cadre contraint).

Définition 2.14 Une étape de réécriture $t[\![T]\!] \longrightarrow_{[p, g \doteq d[\![C]\!], \sigma]} t'[\![T']\!]$ et dite libre ssi

- $\{V(d) \backslash V(g)\} \cap V_a(C) = \emptyset$

- pour chaque variable $x \in V_a(C)$ et position u de x dans g, $p.u$ est une position variable ou sous une position variable de t, (c'est-à-dire le sous-terme de t correspondant à x est lui aussi contraint). $\diamond$

La première condition traduit le fait que nous ne voulons pas simplifier avec des règles ayant de nouvelles variables contraintes du côté droit. Sinon nous perdrions la complétude de notre E-unification comme on le verra dans l'exemple du paragraphe suivant. Cette restriction est nouvelle par rapport à la complétion basique de [NR92a].

Les étapes de simplification dans la partie complétion de notre procédure consisteront donc à réécrire les deux côtés d'une équation mais uniquement par des étapes de réécriture libre. Voyons maintenant un exemple de simplification.

Exemple 2.15 Soient les trois équations contraintes suivantes :

$$\begin{array}{lcrcl} (e_1) & : & f(h(y)) & \doteq & g(y)[\![y = i(z)]\!] \\ (e_2) & : & f(x) & \doteq & c[\![x = h(i(a))]\!] \\ (e_3) & : & f(h(i(a))) & \doteq & c[\![\ ]\!] \end{array}$$

sous un ordre lpo avec la précédence $f > g > h > i > a > c$. La deuxième équation est simplifiée par la première, on a le filtre contraint suivant $\{z \leftarrow_{np} a, y \leftarrow_{np} i(a)\}$, après simplification on conservera donc l'équation $g(y) \doteq c[\![y = i(a)]\!]$. En revanche la troisième équation ne pourra pas être simplifiée par la première, car la partie contrainte de (e_1) se filtre sur une partie non contrainte de (e_3), donc l'étape de réécriture n'est pas libre.

Remarque: Grâce à l'implémentation *UREVEAL*, nous avons pu vérifier que ces restrictions pour la simplification ne nous empêchent pas d'obtenir la complétion des systèmes standards (e.g. théorie des groupes ...).

Pour prouver la complétude de notre méthode d'E-unification, nous allons naturellement utiliser le lemme de correspondance entre réécriture et surréduction, appelé dans la suite "lifting lemma", mais nous devons là encore restreindre notre réécriture sinon nous ne pouvons pas établir ce lemme dans le cadre contraint.

2.2.2 La réécriture quasi-normale

Donnons tout d'abord un exemple qui montre qu'appliqué sans restriction, le "lifting lemma" est faux dans notre contexte.

Exemple 2.16 Soit E le système d'équations composé des quatre règles suivantes :

$$\begin{array}{lcrcl} (1) & : & f(x, c) & \doteq & x[\![\ ]\!] \\ (2) & : & f(i(y), z) & \doteq & g(y)[\![\ ]\!] \\ (3) & : & i(a) & \doteq & a[\![\ ]\!] \\ (4) & : & g(y) & \doteq & x[\![x =_n i(y)]\!] \end{array}$$

sous l'ordre de simplification lpo avec la précédence $f > g > c > i > a$. La quatrième règle est le résultat d'une superposition contrainte entre les deux premières. Essayons maintenant d'unifier modulo E le terme $g(x)$ avec a, nous avons la solution $x = a$; en effet :

$$g(a) \longrightarrow_{[\epsilon,(4),\{y \leftarrow_p a, x \leftarrow_p i(a)\}]} i(a) \longrightarrow_{[\epsilon,(3),Id]} a$$

Mais dans ce cas nous n'avons pas de correspondance avec la surréduction contrainte pour la deuxième étape car on a $g(x) \rightsquigarrow_{[\epsilon,(4)]} y[\![y = i(x)]\!]$ et là nous ne pouvons plus appliquer de surréduction car nous sommes en position variable. Cette défaillance du "lifting lemma" provient ici du fait, que le filtre utilisé pour la première étape de réécriture n'était pas en forme normale.

Définition 2.17 Une étape de réécriture $t[\![T]\!] \longrightarrow_{[p, g \doteq d[\![C]\!], \sigma]} t'[\![T']\!]$ est dite quasi-normale par rapport à un ensemble d'équations S ssi le c-filtre σ est en S forme normale sur les variables de d qui ne sont pas dans g. $\diamond$

Pour que le c-filtre soit normalisé sur les autres variables, nous utiliserons la réécriture en profondeur d'abord. Revenons à l'exemple précédent.

Exemple 2.16 (suite) La première étape de notre dérivation était en profondeur d'abord mais pas quasi-normale. Pour normaliser $g(a)$ avec des étapes de réécriture quasi-normale nous devons générer de nouvelles paires critiques. On obtient (5) par superposition de (3) dans (2) :

$$(5) \quad : \quad f(a,z) \quad \doteq \quad g(y)[\![y =_n a]\!]$$

puis (6) par superposition entre (5) et (1) :

$$(6) \quad : \quad g(y) \quad \doteq \quad x[\![y =_n a \ \wedge \ x =_n a]\!]$$

Nous pouvons donc ainsi normaliser $g(a)$ avec l'étape quasi-normale suivante :

$$g(a) \longrightarrow_{[\epsilon,(6),\{y \leftarrow_p a, x \leftarrow_p a\}]} a$$

ce qui correspond à la surréduction contrainte suivante :

$$g(x) \rightsquigarrow_{[\epsilon,(6)]} y[\![x = a \ \wedge \ y = a]$$

Regardons sur cet exemple ce qui se produit si on omet la première condition pour qu'une étape de simplification soit libre. La règle (4) peut simplifier la règle (2) et la remplacer par $(2b) \quad : \quad f(i(y),z) \doteq i(y)[\![\quad]\!]$ et nous ne pouvons plus obtenir la règle (6), et donc nous ne trouvons plus la solution de notre problème d'unification par surréduction contrainte. La première condition de la définition de la réécriture libre est donc indispensable pour établir la complétude de notre procédure d'E-unification.

Nous pouvons dès à présent établir les principaux points qui vont permettre le succès de notre méthode.

- Au cours de notre complétion basique nous ne permettons la simplification qu'avec des étapes de réécriture libre.
- Nous montrerons que le système final ainsi obtenu est confluent sur les termes clos, et que chaque forme normale peut être calculée par une dérivation en profondeur d'abord et dont les étapes sont toutes quasi-normales.
- Dans ce cadre, le lemme de correspondance peut être démontré et ainsi on obtient la complétude de notre méthode.

Voyons maintenant le système d'inférence permettant de faire en même temps la complétion du système d'équations et de la surréduction dans le système d'équations, les deux dans un cadre contraint.

2.3 Les règles d'inférences pour la complétion équationnelle et l'unification dans un cadre contraint

Notre système d'inférence travaille sur un triplet (E, R, G) où :

- E est un ensemble d'équations contraintes, au début de la procédure, il contiendra l'ensemble des équations initiales. Rappelons que les équations initiales peuvent avoir des contraintes,
- R est aussi un ensemble d'équations contraintes, il est vide au début de la procédure et contiendra à la fin quand celle-ci terminera, un ensemble d'équations confluent sur les termes clos,

- G est un ensemble d'unificandes ou de buts, au départ il sera initialisé avec l'unificande comportant les deux termes que l'on veut unifier modulo E.

Nous acceptons les contraintes dans les équations initiales, elles seront représentées avec les symboles $=$ et $>$, mais nous devons propager ces équations et pour cela nous utilisons la notion de propagation minimale.

2.3.1 Propagation minimale par rapport aux contraintes initiales

Pour cela nous devons définir les solutions de la partie initiale d'une contrainte.

Définition 2.18 Soit C une contrainte quelconque, on notera par $SOL_0(C)$ l'ensemble des substitutions solutions de la partie initiale de C, c'est-à-dire solutions de la contrainte après avoir éliminé toutes les composantes en $=_n$ et $>_n$ de C. $\diamond$

Si une contrainte C n'a pas de composante initiale, $SOL_0(C)$ sera l'ensemble vide.

Contrairement aux autres procédures de complétion basique, les étapes de propagation nécessaires ne sont pas faites au début de la procédure mais au cours de celle-ci, au fur et à mesure des besoins. D'autre-part cette propagation est faite uniquement avec les solutions *minimales* de SOL_0 dans le sens suivant : soient σ et τ deux substitutions de $SOL_0(C)$ (avec C une contrainte quelconque), on dira que σ est plus générale que τ sur les variables de C ssi il existe une substitution α telle que $\tau = \alpha \circ \sigma$ sur $V(C)$. Une solution σ de $SOL_0(C)$ sera minimale s'il n'existe pas de solution de $SOL_0(C)$ strictement plus générale que σ. Cette notion de minimalité est facilement décidable par rapport aux contraintes représentant des égalités syntaxiques, mais il est naturellement plus difficile d'obtenir un ensemble de solutions minimales par rapport à une contrainte d'ordre. Pour cette raison, notre approche, ici, sera seulement opérationnelle (c.f. paragraphe 4.2 du chapitre 4). Voyons les propagations que nous autorisons sur un exemple.

Exemple 2.19 Soit E le système constitué de l'unique équation suivante :

$$(1) \quad f(x) \quad \doteq \quad g(x) \quad [\![x > b]\!]$$

sous un ordre lpo où $f > g > a > b$. L'ensemble des solutions de la contrainte est infini mais $\{\{x \leftarrow a\}, \{x \leftarrow f(x)\}, \{x \leftarrow g(x)\}\}$ est un ensemble minimal complet de solutions de $[\![x > b]\!]$. Ainsi notre propagation générera seulement les trois équations suivantes au cours de la complétion :

$$\begin{array}{c} f(a) \doteq g(a)[\![\ ]\!] \\ f(f(x)) \doteq g(f(x))[\![\ ]\!] \\ f(g(x)) \doteq g(g(x))[\![\ ]\!] \end{array}$$

Les détails pratiques sur la façon de générer des ensembles minimal complet de solutions d'une contrainte d'ordre seront donnés dans le chapitre implémentation. Nous avons maintenant toutes les définitions nécessaires pour établir notre système d'inférence.

2.3.2 Les règles d'inférence

1) Choix et génération de paires critiques

$$\frac{(E \cup \{s \doteq t \ [\![c]\!]\}, R, G)}{(E \cup CP\{s \doteq t \ [\![c]\!], R\}, R \cup \{s \doteq t \ [\![c]\!]\}, G)}$$

2) Propagation minimale

$$\frac{(E, R \cup \{s \doteq t \ [\![c]\!]\}, G)}{(E \cup \{\sigma(s \doteq t \ [\![c]\!])\}, R \cup \{s \doteq t \ [\![c]\!]\}, G)}$$

si c est satisfiable, $s \neq t$, et σ est **minimale** dans $SOL_0(c)$.

3) Suppression d'une équation

$$\frac{(E, R \cup \{s \doteq t \ [\![c]\!]\}, G)}{(E, R, G)}$$

si $(s = t)$ ou c est **insatisfiable**.

4) Surréduction contrainte

$$\frac{(E, R, G \cup \{s \stackrel{?}{=} t \ [\![c]\!]\})}{(E, R, G \cup \{s \stackrel{?}{=} t \ [\![c]\!] \ , \ s' \stackrel{?}{=} t' \ [\![c']\!]\})}$$

si $(s \stackrel{?}{=} t \ [\![c]\!]) \quad \rightsquigarrow_{(R)} \quad (s' \stackrel{?}{=} t' \ [\![c']\!])$.

5r) Simplification d'une équation

$$\frac{(E \cup \{s \doteq t[\![c]\!]\}, R, G) \quad OR \quad (E, R \cup \{s \doteq t[\![c]\!]\}, G)}{(E \cup \{s' \doteq t[\![c']\!]\}, R, G)}$$

si $s[\![c]\!] \rightarrow_{E \cup R} s'[\![c']\!]$ par une étape de **réécriture libre**.

5u) Simplification d'un unificande

$$\frac{(E, R, G \cup \{s \stackrel{?}{=} t \ [\![c]\!]\})}{(E, R, G \cup \{s' \stackrel{?}{=} t' \ [\![c']\!]\})}$$

si $(s' \stackrel{?}{=} t' \ [\![c']\!]) = (s \stackrel{?}{=} t \ [\![c]\!])\downarrow_R$.

6) Génération d'une solution

$$\frac{(E, R, G \cup \{s \stackrel{?}{=} t \ [\![c]\!]\})}{(E, R, G \cup \{s \stackrel{?}{=} t \ [\![c]\!] \ , \ \emptyset \ [\![c \wedge (s = t)]\!]\})}$$

si s est unifiable syntaxiquement avec t.

7) Suppression d'un unificande

$$\frac{(E, R, G \cup \{s \stackrel{?}{=} t \ [\![c]\!]\})}{(E, R, G)}$$

si c est insatisfiable.

8) Simplification d'une solution

$$\frac{(E, R, G \cup \{\emptyset \ \llbracket c \wedge (s >_n t) \rrbracket\})}{(E, R, G \cup \{\emptyset \ \llbracket c \rrbracket\})} \qquad \text{si } c \wedge (s >_n t) \text{ est satisfiable.}$$

Quelques remarques sont nécessaires pour la bonne compréhension de ces règles.

Commentaires sur les règles d'inférence

Dans ce cadre R contient les équations qui ont déjà servi pour la superposition et E contient toutes les autres équations. La partie complétion de la procédure sera terminée lorsque E sera vide.

La règle 8) joue un rôle essentiellement pratique, elle met en lumière le fait que les contraintes d'ordres créées par la procédure dans les unificandes de G sont utilisées uniquement pour la satisfiabilité de la contrainte et non pas pour la génération de substitutions. Elles doivent être satisfiables, en particulier pour éviter d'utiliser des règles inorientables de façon infinie et de tomber ainsi dans un cycle. D'un autre côté les solutions non closes obtenues vérifieront rarement les contraintes d'ordres nouvelles, alors que bien évidemment elles sont correctes même sans les contraintes d'ordres nouvelles, (cela correspondrait au fait d'utiliser les équations dans les deux sens). C'est pour cela que nous avons décidé de donner les solutions en supprimant les contraintes en $>_n$ (et uniquement celles-là).

Notre propagation minimale est moins ‘ coûteuse ’ que la schématisation de [KKR90] même si celle-ci est appliquée uniquement sur les contraintes initiales. De plus notre règle n'est pas appliquée uniquement au début de la procédure comme dans [NR92a] ce qui permet d'exécuter notre procédure d'unification, même si l'ensemble complet des solutions minimales d'une contrainte initiale est infini.

Nous considérons des chaînes de dérivation partant du triplet initial (E_0, R_0, G_0) et où pour tout $i > 0$, le triplet (E_i, R_i, G_i) est obtenu en appliquant une des règles d'inférence sur le triplet $(E_{i-1}, R_{i-1}, G_{i-1})$. Ces dérivations seront équitables dans le sens suivant :

- $E_\infty = \emptyset$ et pour chaque équation $g \doteq d\llbracket C \rrbracket$ dans R_∞, et pour toute substitution σ telle que σ est minimale dans $SOL_0(C)$ alors il existe un entier $k > 0$ tel que $\sigma(g \doteq d\llbracket C \rrbracket) \in E_k$.

- $\forall n \geq 0$, pour chaque équation $g \doteq d\llbracket C \rrbracket$ de R_∞ et chaque unificande dans G_n, le résultat de la surréduction contrainte, de l'unificande par l'équation, lorsqu'elle est possible, est contenu dans un des ensembles G_k.

Cette équité correspond au fait que l'on veut calculer le complété de notre système d'équations initiales et que lorsque celles-ci sont contraintes, toutes les propagations nécessaires sont effectuées. De plus toutes les chaînes de surréduction contrainte, obtenues à partir de notre but initial en utilisant les règles du complété, seront calculées.

Nous avons maintenant à établir la preuve de complétude de notre méthode d'E-unification. Pour cela nous allons montrer dans un premier temps, que le système final, R_∞ est confluent sur les termes clos dans un certain sens. Une reformulation du “lifting lemma” nous fournira notre théorème final : la complétude de la surréduction contrainte.

2.4 Confluence sur les termes clos par la réécriture quasi-normale

Par la suite nous noterons S le système final R_∞ obtenu à partir de E_0 par la partie complétion basique de notre procédure. Nous pouvons remarquer que la confluence de S sur les termes clos peut être obtenue suivant la même preuve que [NR92a] puisque nous autorisons moins de simplification : les simplifications autorisées dans [NR92a] doivent vérifier une seule parmi les deux conditions que nous exigeons dans notre approche présente (c.f. paragraphe 2.2.1). La complétion basique a aussi été prouvée complète dans [BGSL93] mais avec un formalisme utilisant le marquage pour repérer les positions basiques. Là encore la simplification est plus importante que dans notre procédure, elle est en fait similaire à celle de [NR92a].

Donnons quand même ici, les grandes lignes d'une preuve de complétude.

2.4.1 Lemme des paires critiques dans le cadre contraint

Considérons une S-preuve, entre deux termes clos, contenant le pic suivant :

$$s' \longleftarrow_{[p, g_1 \doteq d_1[\![C_1]\!], \sigma]} s \longrightarrow_{[q, g_2 \doteq d_2[\![C_2]\!], \theta]} s''$$

Supposons que la position p est au-dessus de la position q. Le seul cas que nous devons étudier est le cas qui ne peut pas se produire dans un cadre non contraint, à savoir, la seconde étape de réécriture est faite sous (ou dans) une position variable de g_1, que l'on notera par x, et que cette variable est contrainte, ce qui peut correspondre au fait que nous devions faire une paire critique au niveau de la contrainte C_1, ce que l'on interdit. Nous allons donc étudier les différents cas possibles en fonction de la variable x. Pour les autres cas où q n'est pas sous une position variable de g_1, la démonstration habituelle reste valable [BAC87].

Si toutes les contraintes concernant x sont des contraintes initiales, notre système final S contiendra une propagation de la règle $g_1 \doteq d_1[\![C_1]\!]$ et donc S contiendra aussi la paire critique entre cette règle et $g_2 \doteq d_2[\![C_2]\!]$ qui permettra de supprimer le pic. Si des règles ont été simplifiées, on suit la même démarche que dans le cadre non contraint, c'est-à-dire si la règle simplifiée peut toujours être utilisée, c'est celle-ci que l'on prend sinon on utilise la règle qui a servi à simplifier.

Si maintenant la contrainte concernant x a été créée par notre procédure. On a dans C_1 une contrainte du type $x =_n t$ avec t se filtrant sur un sous-terme de s.

- Soit la position q correspond à une variable de t ou est sous une position variable de t, on notera par y cette variable. La variable y est non contrainte et σ possède donc l'instanciation $y \leftarrow t_1$ avec t_1 un sous-terme de s. La position q correspond à un sous-terme de t_1, on peut donc réécrire[1] s'' en s_1 avec la règle $g_1 \doteq d_1[\![C_1]\!]$ en position p et la substitution σ où seulement l'instanciation de y a changé : $y \leftarrow t'_1$ où t'_1 est la réécriture de t_1 par la seconde règle. D'autre part on peut réécrire tous les sous-termes t_1 de s' avec $g_2 \doteq d_2[\![C_2]\!]$ et on obtient s_1. C'est la méthode habituelle pour réduire les pics en position variable. On peut suivre les différentes étapes sur la figure 2.1.

- Soit la position q correspond à une position non variable de t. C'est le cas totalement différent du cadre non contraint puisque nous ne pouvons pas faire directement la paire critique qui réduirait le pic car elle devrait être faite dans une position de la contrainte. Voir figure 2.2.

[1] On aura peut-être besoin auparavant de réécrire certains sous termes de s'' identiques à t_1 avec la règle $g_2 \doteq d_2[\![C_2]\!]$, dans le cas où g_1 possède plusieurs fois la même variable.

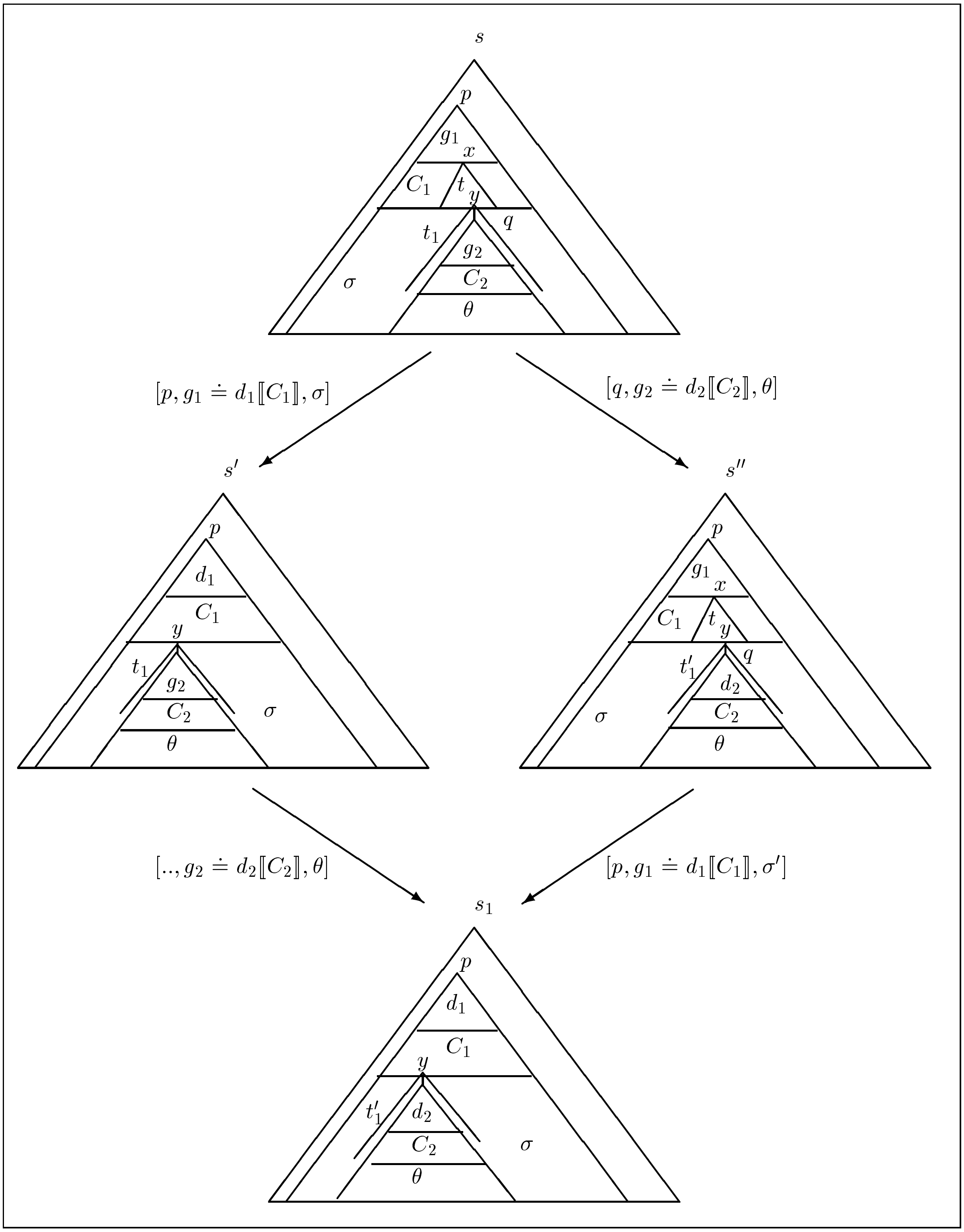


Figure 2.1: Confluence sous une position variable

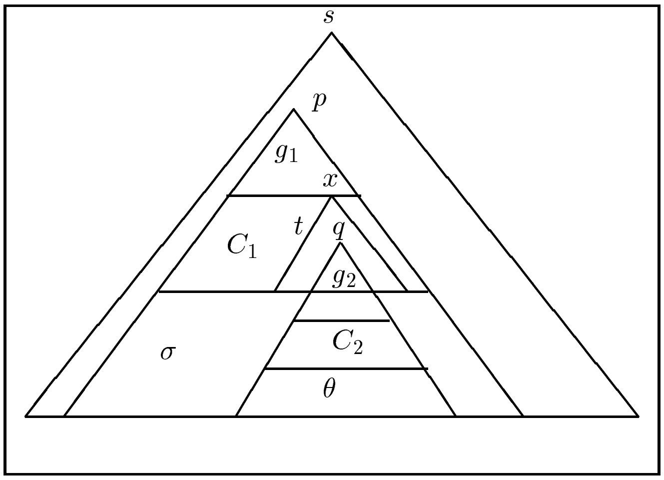


Figure 2.2: Superposition dans la contrainte

C_1 comporte une contrainte nouvelle, la règle $g_1 \doteq d_1[\![C_1]\!]$ a donc été obtenue par superposition contrainte entre deux autres équations disons de $g_3 \doteq d_3[\![C_3]\!]$ dans $g_4 \doteq d_4[\![C_4]\!]$ en position u (voir figure 2.3), ce qui donne :

$$g_4[u \leftarrow d_3] \doteq d_4[\![C_3 \ \wedge \ C_4 \ \wedge \ g_4|_u =_n g_3 \ \wedge \ g_3 >_n d_3 \ \wedge \ g_4 >_n d_4]\!]$$

On peut supposer que la contrainte $x =_n t$ provient de la résolution de $g_4|_u =_n g_3$ et non pas de C_3 ou C_4, et que t n'a pas de variable contrainte, (sinon il faut établir une récurrence sur le nombre de contraintes en $=_n$). t est donc un sous-terme de g_3 ou de g_4. (Pour simplifier les figures on supposera que C_3 et C_4 sont vides.)

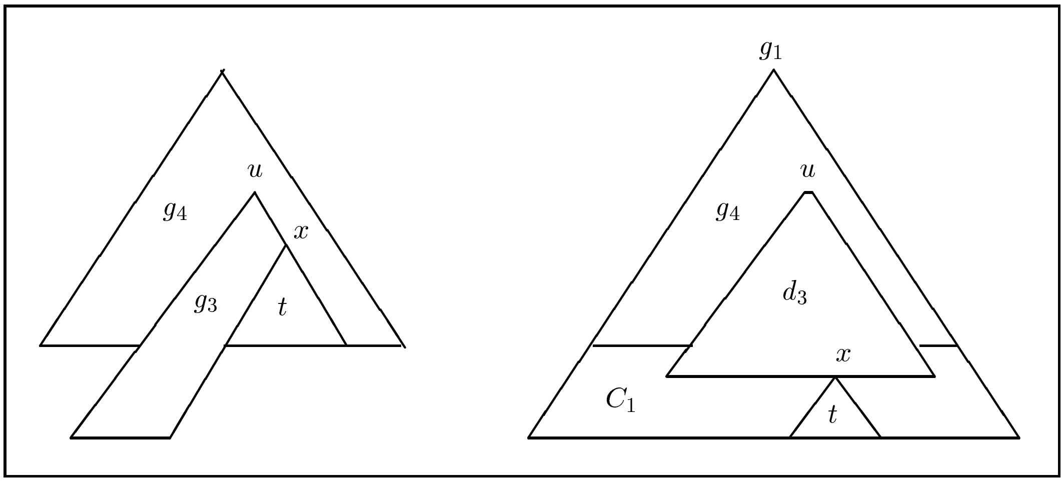


Figure 2.3: Superposition contrainte de g_3 dans g_4 aboutissant à $g_1[\![C_1]\!]$

- Si t est un sous-terme de g_4, $g_4|_{u.p_t} = t$ et $g_3|_{p_t} = x$. Il existe donc une paire critique de $g_2 \doteq d_2[\![C_2]\!]$ dans $g_4 \doteq d_4[\![C_4]\!]$ dans une position non variable du sous-terme t de g_4. On obtient :

  $$g_4[k \leftarrow d_2] \doteq d_4[\![C_4 \ \wedge \ C_2 \ \wedge \ g_4|_k =_n g_2 \ \wedge \ g_4 >_n d_4 \ \wedge \ g_2 >_n d_2]\!]$$

  k est une position qui est en dessous de la position $u.p_t$ de g_4 qui elle-même correspond à une position variable de g_3. Donc on peut refaire une paire critique (puisque l'on a transformé un terme qui correspondait à une variable de g_3), de $g_3 \doteq d_3[\![C_3]\!]$ dans la paire critique obtenue ci-dessus, en position u (voir figure 2.4), et on trouve :

$$(e_6)\ \ g_4[u \leftarrow d_3] \doteq d_4[\![C_4\ \wedge\ C_2\ \wedge\ g_4|_k =_n g_2\ \wedge\ g_4 >_n d_4\ \wedge\ g_2 >_n d_2\ \wedge\ C_3 \wedge \\ g_4[k \leftarrow d_2]|_u =_n g_3\ \wedge\ g_3 >_n d_3\ \wedge\ g_4[k \leftarrow d_2] >_n d_4]\!]$$

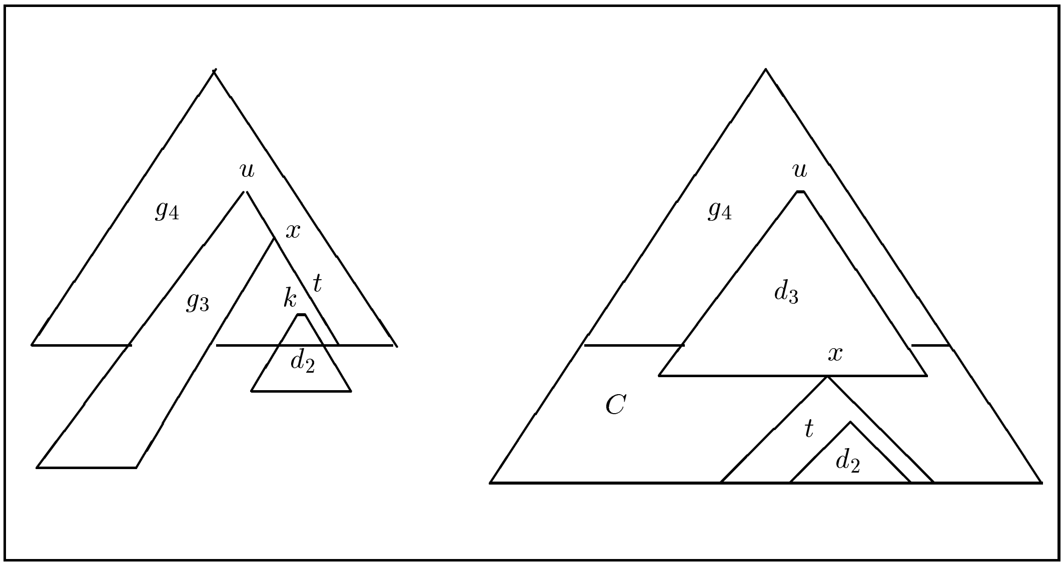


Figure 2.4: Superposition qui fait confluer le pic

C'est cette règle qui va nous permettre de réécrire s'' à la position p en s' et ainsi faire confluer le pic, voir figure 2.5. Cette règle correspond en fait à la paire critique que l'on aurait calculée directement si nous n'étions pas dans un cadre contraint.

– Si t est un sous-terme de g_3, $g_3|_{p_t} = t$ et $g_4|_{u.p_t} = x$. Les figures sont analogues au cas précédent. Il existe une paire critique de $g_2 \doteq d_2[\![C_2]\!]$ dans $g_3 \doteq d_3[\![C_3]\!]$ dans une position non variable du sous-terme t de g_3. On obtient :

$$g_3[k \leftarrow d_2] \doteq d_3[\![C_3\ \wedge\ C_2\ \wedge\ g_3|_k =_n g_2\ \wedge\ g_3 >_n d_3\ \wedge\ g_2 >_n d_2]\!]$$

k est une position qui est en dessous de la position p_t de g_3 qui elle-même correspond à une position variable de g_4. On peut donc toujours faire une paire critique de cette règle dans g_4 à la position u, (puisque l'on a transformé un terme qui correspondait à une variable de g_4). On obtient :

$$g_4[u \leftarrow d_3] \doteq d_4[\![C_3\ \wedge\ C_2\ \wedge\ g_3|_k =_n g_2\ \wedge\ g_3 >_n d_3\ \wedge\ g_2 >_n d_2\ \wedge\ C_4 \wedge \\ g_4|_u =_n g_3[k \leftarrow d_2]\ \wedge\ g_4 >_n d_4\ \wedge\ g_3[k \leftarrow d_2] >_n d_3]\!]$$

Comme pour le cas précédent, cette règle va nous permettre de réécrire s'' à la position p en s' et ainsi faire confluer le pic.

Nous avons donné ici la démarche générale à suivre pour éliminer un pic dans le cadre contraint. Mais au cours du processus de génération de paires critiques (décrit ci-dessus), une des règles nécessaires peut ne plus être dans le système car elle a été simplifiée, dans ce cas on suit le même processus que dans le cadre général. Pour générer une paire critique, nous avons besoin de deux règles, supposons que la première règle ait été simplifiée. Soit la règle simplifiée convient pour faire la paire critique et donc c'est la règle simplifiée que l'on utilise. Soit elle ne permet plus de faire la superposition attendue et dans ce cas on fait une superposition avec la règle qui a simplifié et la seconde règle, on obtient une paire critique que l'on peut superposer si nécessaire avec la règle simplifiée. De plus si l'une des contraintes d'ordres devient insatisfiable, cela signifie que l'on peut utiliser la règle d'où provient la contrainte d'ordre dans le sens inverse pour simplifier l'un des deux termes du pic ou une paire critique au cours du calcul. Détaillons quelques-uns de ces cas sur un exemple.

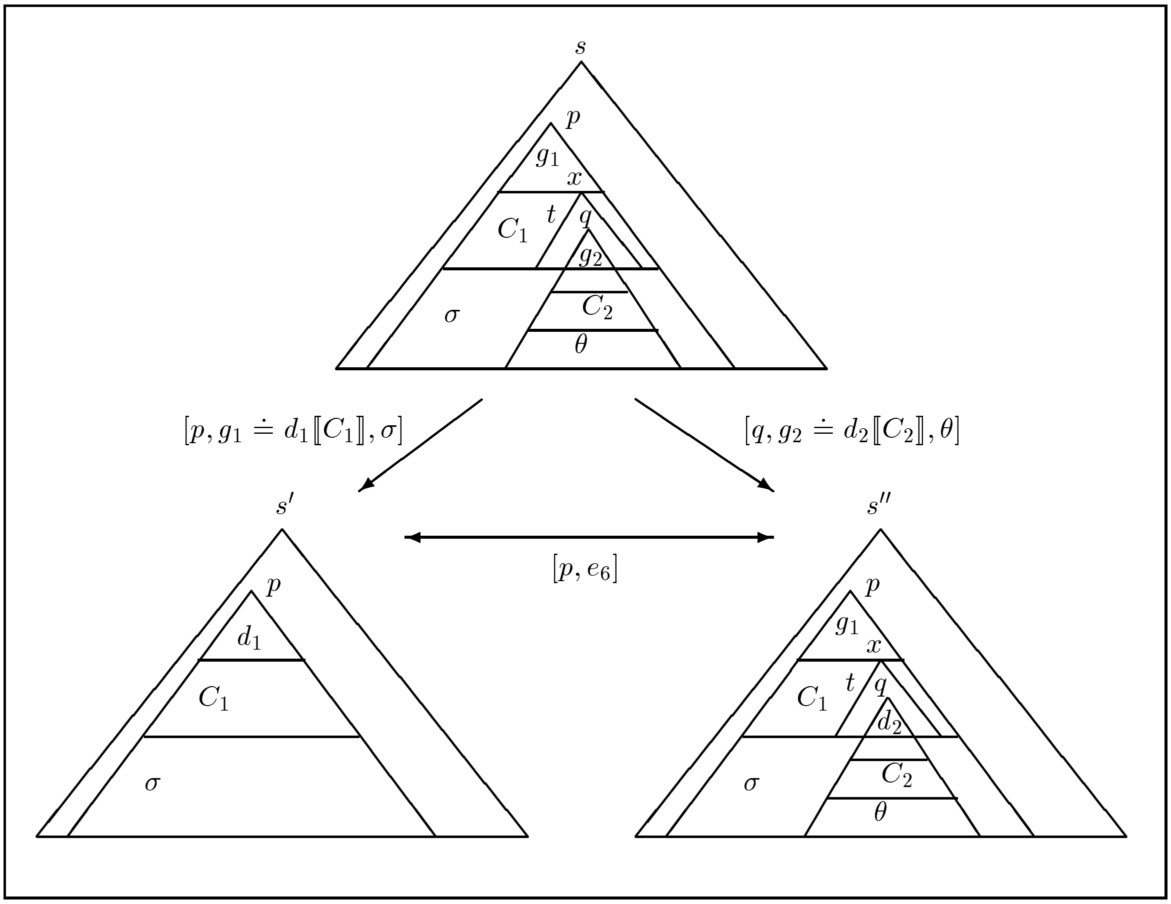


Figure 2.5: Confluence pour une superposition dans une contrainte

Exemple 2.20 Soit E le système d'équations suivant :

$$\begin{array}{lcrcl}(e_1) & : & i(a) & \doteq & b[\![\ ]\!] \\ (e_2) & : & f(x,i(b)) & \doteq & g(x)[\![\ ]\!] \\ (e_3) & : & f(i(z),y) & \doteq & h(y)[\![\ ]\!] \\ (e_4) & : & g(x) & \doteq & h(y)[\![x =_n i(z)\ \wedge\ y =_n i(b)]\!]\end{array}$$

On suppose que l'on dispose d'un ordre de simplification, total sur les termes clos et qui oriente toutes ces équations de la gauche vers la droite. (e_4) est le résultat d'une superposition contrainte entre (e_2) et (e_3). Avec ces quatre règles on peut créer le pic suivant :

$$h(i(b)) \longleftarrow_{(e_4)} g(i(a)) \longrightarrow_{(e_1)} g(b)$$

Pour réduire ce pic on doit faire une paire critique entre (e_4) et (e_1), mais cette superposition doit avoir lieu dans la contrainte de (e_4) au niveau de $i(z)$, ce qui est interdit. La règle (e_4) est le résultat de la superposition contrainte entre (e_2) et (e_3) et $i(z)$ est un sous-terme du membre gauche de (e_3). On peut donc calculer la paire critique de (e_1) dans (e_3), on obtient : $f(b,y) \doteq h(y)[\![z =_n a]\!]$ ce qui est équivalent à $(e_5) \quad : \quad f(b,y) \doteq h(y)[\![\ ]\!]$. Cette équation est superposable avec (e_2), ce qui donne $(e_6) \quad : \quad g(x) \doteq h(y)[\![x =_n b\ \wedge\ y =_n i(b)]\!]$. La règle (e_6) permet de faire confluer directement les deux termes $h(i(b))$ et $g(b)$. Ceci représente le cas où il n'y a pas de simplification au cours du calcul des paires critiques et où il n'y a pas de problème relatif à l'ordre.

Supposons maintenant que nous ayons dans notre système E l'équation suivante :

$$(e_7) \quad : \quad f(b,x') \quad \doteq \quad c[\![\ ]\!]$$

La paire critique (e_5) est simplifiée par (e_7) et donne $(e_8) \quad : \quad c \doteq h(y)[\![\ ]\!]$. Cette équation ne peut plus être superposée avec (e_2), mais dans ce cas on fait une paire critique entre l'équation qui a simplifié (c'est-à-dire (e_7)) et (e_2) et on obtient $(e_9) \quad : \quad c \doteq g(x)[\![x =_n b\ \wedge\ x' =_n i(b)]\!]$ (la dernière contrainte peut être supprimée). Maintenant faisons intervenir l'ordre et étudions différents cas possibles.

- Si c et $h(y)$ sont incomparables et $c > g(b)$. On peut faire une paire critique entre (e_8) et (e_9) on obtient : $(e_{10}) \quad : \quad g(x) \doteq h(y)[\![x =_n b\ \wedge\ c >_n h(y)]\!]$. Cette règle permet de faire confluer le pic si la contrainte d'ordre est satisfaite sinon on est dans le dernier cas.
- Si c et $h(y)$ sont incomparables et $g(b) > c$. Les deux extrémités du pic confluent soit vers c si $h(i(b)) > c$, soit vers $h(i(b))$ si $c > h(i(b))$.
- Si $h(y) > c$ ou si après instanciation on a $\sigma(h(y)) > c$, les deux termes confluent vers c si $g(b) > c$ ou vers $g(b)$ sinon.

Cet exemple montre les principaux mécanismes utilisés pour réduire un pic. Pour faire une preuve complète et totalement détaillée, on devra établir plusieurs niveaux de récurrence notamment au niveau du nombre de contraintes nouvelles ($=_n$) et au niveau du nombre de simplifications d'une règle (quand une règle qui a simplifié une autre règle est elle-même simplifiée à son tour etc...).

2.4.2 Normalisation par réduction S-quasi-normale

La confluence sur les termes clos de S prouve que deux termes clos égaux modulo E, (l'ensemble des équations initiales), ont la même forme normale dans S. Prouvons maintenant que l'on peut atteindre la forme normale de tout terme clos en utilisant uniquement une chaîne de réécriture contrainte S-quasi-normale, dont les étapes sont en profondeur d'abord. Pour cela nous avons besoin d'un premier lemme. Rappelons que la notion de réécriture quasi-normale est donnée dans la définition 2.17.

Lemme 2.21 Soit $s \longrightarrow_{[p, l \doteq r[\![C]\!], \tau]} t$ une étape de réécriture en profondeur d'abord et S-quasi-normale. Alors τ est en S forme normale sur $V(r)$.

Preuve: L'étape de réécriture est S-quasi-normale donc τ est en S forme normale sur $V(r) \backslash V(l)$. Reste à étudier le cas des variables qui sont à la fois dans l et dans r, c'est-à-dire pour $x \in V(l) \cap V(r)$.

Cas 1 : si l est non variable, x est un sous-terme strict de l et $\tau l = s|_p$. La réécriture est en profondeur d'abord ce qui implique que tout sous-terme strict de $s|_p$ est irréductible donc τx est en S forme normale.

Cas 2 : si l est une variable, $l = x$; ce cas est impossible car x est aussi dans $V(r)$ et $\tau l > \tau r$ avec un ordre de simplification. □

Remarque: On déduit de la preuve précédente que chaque étape de réécriture, en profondeur d'abord et S-quasi-normale, utilise un filtre en S-forme normale quand la règle utilisée n'a pas le membre gauche réduit à une variable, (par exemple : $x \doteq y[\![x = a \wedge y = b]\!]$). La combinaison en "profondeur d'abord et S-quasi-normale" peut ainsi être vue comme une généralisation de la notion classique de réécriture à filtre normalisé.

Passons maintenant au lemme clef de la démonstration.

Lemme 2.22 Soit $r_0 \equiv g \doteq d[\![T]\!]$ une règle obtenue par superposition contrainte entre les règles $r_1 \equiv g_1 \doteq d_1[\![T_1]\!]$ et $r_2 \equiv g_2 \doteq d_2[\![T_2]\!]$, deux autres règles de S. Soit τ une substitution close, définie sur les variables de r_0, telle que $\tau g > \tau d$ et $\tau \in SOL(T)$. Alors le terme τg peut être réécrit avec une règle de S par une étape de réécriture S-quasi-normale.

Avant de détailler la preuve, regardons ce qui se passe sur un exemple.

Exemple 2.23 Soit E le système d'équations suivant :

$$\begin{array}{lcrcl} (1) & : & f(h(x,y),x) & \doteq & f(a,y)[\![\ ]\!] \\ (2) & : & f(z,b) & \doteq & g(z)[\![\ ]\!] \\ (3a) & : & h(u,a) & \doteq & i(u,c)[\![\ ]\!] \\ (3b) & : & i(b,c) & \doteq & e[\![\ ]\!] \end{array}$$

sous un ordre lpo où la précédence est $f > g > h > i > e > c > b > a$. Par superposition contrainte entre (1) et (2) on obtient la paire critique suivante :

$$(4) \quad : \quad f(a,y) \quad \doteq \quad g(z)[\![z =_n h(b,y) \ \wedge \ x =_n b]\!]$$

Considérons le terme clos $t = f(a,a)$, t peut se réécrire par réécriture contrainte avec la règle (4) en $g(h(b,a))$. Le filtre utilisé est $\{y \leftarrow_p a, z \leftarrow_p h(b,a)\}$. Ce n'est pas une étape de réécriture quasi-normale puisque $h(b,a)$ est réductible. On peut normaliser $h(b,a)$ avec des étapes de réécriture en profondeur d'abord de la façon suivante :

$$h(b,a) \longrightarrow_{(3a)} i(b,c) \longrightarrow_{(3b)} e$$

Mais on a une paire critique entre (1) et $(3a)$:

$$(5a) \quad : \quad f(i(x,c),x) \quad \doteq \quad f(a,y)[\![y =_n a]\!]$$

$(5a)$ se superpose avec $(3b)$:

$$(5b) \quad : \quad f(e,x) \quad \doteq \quad f(a,y)[\![x =_n b \ \wedge \ y =_n a]\!]$$

finalement $(5b)$ se superpose avec (2) :

$$(6) \quad : \quad f(a,y) \quad \doteq \quad g(z)[\![y =_n a \ \wedge \ z =_n e]\!]$$

Avec cette règle on peut réécrire $t = f(a,a)$ en $g(e)$ en une étape de réécriture quasi-normale.

Nous allons montrer que dans le cadre général, cela se passe comme sur l'exemple, à savoir, quand le filtre n'est pas normalisé, on utilise les règles qui servent à le normaliser pour faire des paires critiques avec les règles de départ. On peut représenter cela avec la figure 2.6.

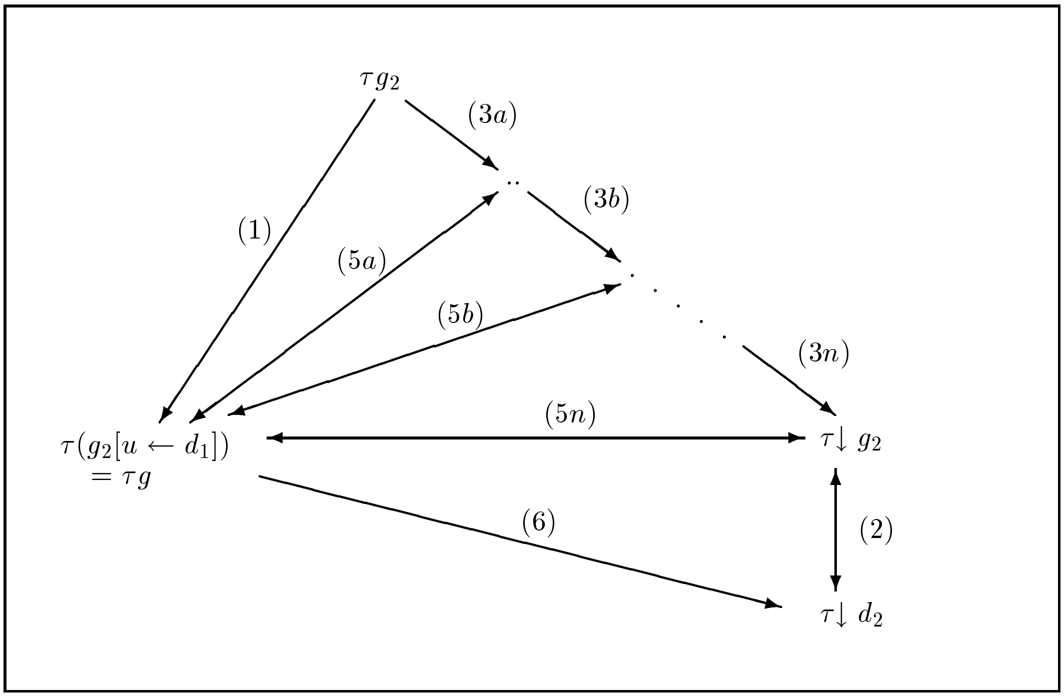


Figure 2.6: Suppression des étapes non quasi-normales

Les flèches étiquetées $(3a)$...$(3n)$ sont les règles utilisées pour la normalisation de τ. Les flèches $(5a)$, ...$(5n)$ correspondent aux paires critiques calculées au fur et à mesure avec les règles qui ont simplifié τ. Le triangle du bas correspond à la superposition donnant la règle finale, (sur l'exemple précédent $n = b$). Certaines flèches sont doubles car on ne sait pas à priori, dans quel sens elles vont s'orienter. Dans certains cas on ne développera pas le raisonnement jusqu'à la création de la règle (6) car une des règles $(5i)$ peut permettre de réécrire τg directement en une étape de réécriture quasi-normale. Sur l'exemple précédent, si la précédence avait été $a > b > c > e$, la règle $(5b)$ aurait été utilisée pour réécrire $f(a, a)$ en $f(e, b)$ en une étape quasi normale, le filtre utilisé étant $\{x \leftarrow_p a, y \leftarrow_p b\}$, (bien évidemment, $f(e, b)$ se réécrit en $g(e)$ avec (2) et une étape quasi-normale).

Etudions maintenant les différents cas possibles de façon précise.

Preuve: Considérons que la règle r_0 aie été obtenue par superposition contrainte de g_1 dans g_2 à la position u. On a

- soit (i) : $g = g_2[u \leftarrow d_1]$, $\;d = d_2$, $\;T = T_1 \;\wedge\; T_2 \;\wedge\; g_2|_u =_n g_1 \;\wedge\; g_1 >_n d_1 \;\wedge\; g_2 >_n d_2$,

- soit (ii) : $g = d_2$, $\;d = g_2[u \leftarrow d_1]$, $\;T = T_1 \;\wedge\; T_2 \;\wedge\; g_2|_u =_n g_1 \;\wedge\; g_1 >_n d_1 \;\wedge\; g_2 >_n d_2$.

τ est une substitution close définie sur $V(r_0)$ et telle que $\tau g > \tau d$ et $\tau \in SOL(T)$. Si τ est normalisé sur $V(d)\backslash V(g)$ par rapport aux équations de S alors on peut réécrire τg avec la règle r_0 en une étape S-quasi-normale.

Maintenant soit $x \in V(d)\backslash V(g)$ tel que τx ne soit pas en S forme normale. On peut supposer que ce x est unique sinon on devra faire le même raisonnement pour tous les éléments de $V(d)\backslash V(g)$ dont l'instanciation par τ n'est pas en S forme normale, (ils sont en nombre fini). Soit la substitution $\tau\!\downarrow$ définie de la façon suivante : $\forall y \in V(r_0)$ si $y \neq x$ $\tau\!\downarrow y = \tau y$ et $\tau\!\downarrow x = (\tau x)\!\downarrow$ (la S forme normale de τx).

$\boxed{A}$ Si $\tau\!\downarrow \in SOL(T)$ et $\tau\!\downarrow g > \tau\!\downarrow d$ alors $\tau g = \tau\!\downarrow g$ peut être réécrit par r_0 avec le filtre $\tau\!\downarrow$, par une étape S quasi-normale.

$\boxed{B}$ Si $\tau\!\downarrow \in SOL(T)$ et $\tau\!\downarrow g \not> \tau\!\downarrow d$, ce cas n'est pas possible puisque la seule différence entre τ et $\tau\!\downarrow$ porte sur une variable qui n'appartient pas à g donc $\tau\!\downarrow g = \tau g > \tau d > \tau\!\downarrow d$, $(\tau x > \tau\!\downarrow x)$.

$\boxed{C}$ Si $\tau\!\downarrow \notin SOL(T)$, on peut supposer que T_1 et T_2 sont vides. En effet, dans le cas contraire nous avons deux possibilités. Soit T_1 et T_2 contiennent des contraintes initiales et dans ce cas là, on travaille avec les équations propagées que S contiendra grâce à la règle de propagation minimale. Soit T_1 et T_2 contiennent des contraintes nouvelles, et là on établit une récurrence sur le nombre de contraintes nouvelles à l'état brut (avant d'avoir décomposé ou simplifié les contraintes), ce nombre est bien plus petit pour T_1 et T_2 que pour T. Supposons donc que T_1 et T_2 sont vides, et alors : $\tau\!\downarrow \notin SOL(g_2|_u =_n g_1 \ \wedge \ g_1 >_n d_1 \ \wedge \ g_2 >_n d_2)$.

$\boxed{C.A}$ Si $\tau\!\downarrow \in SOL(g_2|_u =_n g_1)$ alors $\tau\!\downarrow \notin SOL(g_1 >_n d_1 \ \wedge \ g_2 >_n d_2)$.

Cas (i) : $x \in V(d)$ c'est-à-dire $x \in V(d_2)$ donc $x \notin V(g_1 \doteq d_1)$ (car les variables sont disjointes entre les deux règles) d'où $\tau\!\downarrow g_1 > \tau\!\downarrow d_1$. La seule possibilité pour que $\tau\!\downarrow$ ne soit pas solution de T est donc que $\tau\!\downarrow g_2 \not> \tau\!\downarrow d_2$, c'est-à-dire $\tau\!\downarrow d_2 > \tau\!\downarrow g_2$ puisque $\tau\!\downarrow$ est une substitution close et que l'ordre est total sur les termes clos. Ce qui n'est pas possible car

$$\begin{array}{lcll}
\tau\!\downarrow (g_2|_u) & = & \tau\!\downarrow g_1 & car \quad \tau\!\downarrow \in SOL(g_2|_u =_n g_1) \\
\tau\!\downarrow g_1 & = & \tau g_1 & car \quad x \notin V(g_1 \doteq d_1) \ \ et \ \ \forall y \neq x, \tau\!\downarrow y = \tau y \\
\tau g_1 & = & \tau(g_2|_u) & car \quad \tau \in SOL(g_2|_u =_n g_1)
\end{array}$$

De l'égalité $\tau\!\downarrow (g_2|_u) = \tau(g_2|_u)$ on en déduit que $x \notin V(g_2|_u)$ et comme $x \notin V(g_2) \backslash V(g_2|_u)$ car $x \notin V(g)$ on a $x \notin V(g_2)$ et donc $\tau\!\downarrow g_2 = \tau g_2$. Comme de plus $\tau g_2 > \tau d_2 > \tau\!\downarrow d_2$ on a $\tau\!\downarrow g_2 > \tau\!\downarrow d_2$, ce qui fournit une contradiction.

Cas (ii) : $x \in V(g_2[u \leftarrow d_1])$, $x \notin V(d_2)$ et on a $\tau\!\downarrow (g_2|_u) = \tau\!\downarrow g_1$.

$\boxed{C.A.A}$ Si $x \in V(d_1)$, $x \notin V(g_2 \doteq d_2)$, $\tau\!\downarrow (g_2|_u) = \tau(g_2|_u) = \tau g_1$, on en déduit l'égalité $\tau g_1 = \tau\!\downarrow g_1$, ce qui montre que $x \notin V(g_1)$ et donc $\tau\!\downarrow g_1 = \tau g_1 > \tau d_1 > \tau\!\downarrow d_1$. De plus $x \notin V(g_2 \doteq d_2)$ donc $\tau\!\downarrow g_2 = \tau g_2 > \tau d_2 = \tau\!\downarrow d_2$. Ce qui contredit l'hypothèse.

$\boxed{C.A.B}$ Si $x \in V(g_2)$, $x \notin V(g_1 \doteq d_1)$ donc $\tau\!\downarrow g_1 > \tau\!\downarrow d_1$. Pour que $\tau\!\downarrow$ ne soit pas solution de la contrainte T il faut donc que $\tau\!\downarrow g_2 \not> \tau\!\downarrow d_2$ c'est-à-dire $\tau\!\downarrow d_2 > \tau\!\downarrow g_2$. Dans ce cas là, $\tau g = \tau\!\downarrow g = \tau\!\downarrow d_2$ peut se réécrire avec la règle $d_2 \doteq g_2$ et le filtre $\tau\!\downarrow$ par une étape de réécriture S-quasi-normale car si $\tau\!\downarrow$ n'était pas normalisé sur $V(g_2) \backslash V(d_2)$, il suffit de le normaliser vu que l'on a supposé que T_2 était vide.

Passons maintenant au cas le plus important.

$\boxed{C.B}$ Si $\tau\!\downarrow \notin SOL(g_2|_u =_n g_1)$. On a $x \in V(g_2|_u) \cup V(g_1)$, donc il existe un sous-terme strict t de $g_2|_u$ ou de g_1 tel que $\tau x = \tau t$ et τt ne soit pas normalisé par rapport à S. Supposons que τt se réécrive à la position p.

$\boxed{C.B.A}$ Si p est une position variable ou sous une position variable de t, $\exists y \in V(t)$ tel que $\tau y = t'$ et t' n'est pas normalisé.

$\boxed{C.B.A.A}$ Si en remplaçant $\tau y = t'$ par $\tau y = t'\!\downarrow$, τ est toujours solution de $[\![g_2|_u =_n g_1]\!]$, on reprend le raisonnement du début de cette preuve avec ce nouveau τ et on aboutit à la preuve de notre lemme.

$\boxed{C.B.A.B}$ Sinon $t' = \tau t''$ avec t'' sous-terme de $g_2|_u$ ou de g_1 et on recommence le raisonnement en $C.B.A$ avec $\tau t''$ à la place de τt. (Ceci ne pourra se reproduire qu'un nombre fini de fois puisque T est satisfiable ($\tau \in SOL(T)$), et que V(T) est fini).

$\boxed{C.B.B}$ Sinon p est une position non variable de t. Il existe donc un sous-terme s non variable de t, dont une instanciation peut se réécrire en tête. Soit

$$\tau t = t_0 \rightarrow_{[r_a]} t_1 \rightarrow_{[r_b]} \dots \rightarrow_{[r_n]} t_n = (\tau t)\downarrow$$

une chaîne de réécriture en profondeur d'abord qui permet de réécrire τt en sa forme normale. Cette dérivation existe et est finie puisque S est confluent sur les termes clos. Décomposons maintenant en fonction du terme qui contient t.

$\boxed{C.B.B.A}$ Si t est un sous-terme de $g_2|_u$, $t = g_2|_{u.p_t}$, $s = g_2|_{u.p_t.p}$ et $x = g_1|_{p_t}$. Ceci entraîne alors que nous nous trouvons dans le cas (ii) pour l'utilisation de la paire critique r_0 car on a $x \in V(d)$ donc comme $x \in V(g_1)$ on ne peut pas avoir $x \in V(d_2)$. On a donc $g = d_2$, $d = g_2[u \leftarrow d_1]$ et $T = T_1 \wedge T_2 \wedge g_2|_u =_n g_1 \wedge g_1 >_n d_1 \wedge g_2 >_n d_2$.

Une instanciation de s se réécrit en tête avec la règle r_a de S, donc il existe une paire critique de r_a dans $g_2 \doteq d_2$, faite à la position $u.p_t.p$ et on obtient :

$$(5) \quad : \quad g_2[u.p_t.p \leftarrow d_a] \doteq d_2[\![g_2|_{u.p_t.p} =_n g_a \wedge g_a >_n d_a \wedge g_2 >_n d_2]\!]$$

Supposons que la normalisation de τt se fasse en une seule étape de réécriture. Si ce n'est pas le cas on refera une paire critique entre la règle obtenue ci-dessus et r_b dans le sous-terme d_a et ainsi de suite jusqu'à r_n. La position $u.p_t.p$ de g_2 correspond à une variable de g_1, la paire critique (5) faite dans g_2 a modifié uniquement la position $u.p_t.p$ de g_2, on peut donc faire une paire critique de $g_1 \doteq d_1$ dans (5) à la position u, ce qui donne :

$$\begin{aligned}(6) \quad : \quad & g_2[u.p_t.p \leftarrow d_a][u \leftarrow d_1] \doteq d_2[\![g_2|_{u.p_t.p} =_n g_a \wedge g_a >_n d_a \wedge g_2 >_n d_2 \wedge \\ & g_2[u.p_t.p \leftarrow d_a]|_u =_n g_1 \wedge g_1 >_n d_1 \wedge g_2[u.p_t.p \leftarrow d_a] >_n d_2]\!](i.e.)\end{aligned}$$

$$\begin{aligned}(6) \ : \ g_2[u \leftarrow d_1] \doteq d_2[\![g_2|_{u.p_t.p} =_n g_a \wedge g_a >_n d_a \wedge g_2 >_n d_2 \wedge g_2[u.p_t.p \leftarrow d_a]|_u =_n g_1 \\ \wedge g_1 >_n d_1 \wedge g_2[u.p_t.p \leftarrow d_a] >_n d_2]\!]\end{aligned}$$

Soit τ' la substitution définie de la façon suivante :

- $\tau' = \sigma_a[V(g_a \doteq d_a)]$ où σ_a est la substitution utilisée pour la réécriture de τt avec la règle r_a.

- $\tau' = \tau[V(r_0)]$ excepté pour x où $\tau' x = (\tau t)\downarrow = \tau'(t[p \leftarrow d_a])$ (normalisation de τt en une étape avec la règle r_a).

Montrons que τ' est solution de la partie égalitaire de la contrainte de l'équation (6). Nous avons $\tau'(g_2|_{u.p_t.p}) = \tau'(t|_p) = \tau(t|_p)$ car $x \notin V(g_2)$ et donc $x \notin V(t|_p)$. On réécrit τt par l'étape suivante $\tau t \rightarrow_{[p, r_a, \sigma_a]} t_1$ donc $\tau(t|_p) = \sigma_a g_a = \tau' g_a$, ce qui montre que $\tau' \in SOL(g_2|_{u.p_t.p} =_n g_a)$. Il reste à montrer que $\tau' \in SOL(g_2[u.p_t.p \leftarrow d_a]|_u =_n g_1)$. Sur les variables de cette contrainte, la seule différence entre τ et τ' se situe au niveau de x c'est-à-dire au niveau de l'occurrence p_t de $g_2[u.p_t.p \leftarrow d_a]|_u$ et de g_1. Nous avons donc juste à vérifier que $\tau'(g_2[u.p_t.p \leftarrow d_a]|_{u.p_t}) = \tau' g_1|_{p_t}$. On a la suite d'égalité : $\tau' g_1|_{p_t} = \tau' x = (\tau t)\downarrow = \tau'(t[p \leftarrow d_a]) = \tau' t[p \leftarrow \tau' d_a] = \tau t[p \leftarrow \sigma_a d_a] = \tau(g_2|_{u.p_t})[p \leftarrow \sigma_a d_a] = \tau(g_2[u.p_t.p \leftarrow \sigma_a d_a]|_{u.p_t}) = \tau'(g_2[u.p_t.p \leftarrow d_a]|_{u.p_t})$ donc $\tau' \in SOL(g_2|_{u.p_t.p} =_n g_a)$.

Si maintenant τ' est aussi solution de la partie ordre de la contrainte de la règle (6), $\tau g = \tau d_2$ peut se réécrire avec la règle (6) (inversée) et le filtre τ' (qui correspond à la normalisation de τ) par une étape de réécriture S-quasi-normale, (on a bien $\tau' g = \tau' d_2 > \tau'(g_2[u \leftarrow d_1]) = \tau' d$ car $\tau g > \tau d$). Examinons à présent les différentes contraintes d'ordres de la règle (6).

∘ $g_a >_n d_a$, on a $\tau' g_a = \sigma_a g_a > \sigma_a d_a = \tau' d_a$ donc τ' est solution de cette contrainte.

∘ $g_2 >_n d_2$, τ' est défini de telle sorte que $\forall y \neq x$ on a $\tau y = \tau' y$, $x \notin V(g_2 \doteq d_2)$ et $\tau g_2 > \tau d_2$ donc $\tau' g_2 > \tau' d_2$ τ' est aussi solution de cette contrainte.

∘ $g_2[u.p_t.p \leftarrow d_a] >_n d_2$, si τ' n'est pas solution de cette contrainte on déduit que $\tau' d_2 > \tau'(g_2[u.p_t.p \leftarrow d_a])$ et donc la règle (5) et le filtre τ' peuvent être directement utilisés pour réécrire τg avec une étape de réécriture S-quasi-normale.

$\circ$ $g_1 >_n d_1$, si $\tau' d_1 > \tau' g_1$ on a

$$\begin{array}{rcll} \tau' d_2 & = & \tau d_2 & \\ & > & \tau g_2[u \leftarrow d_1] & car \;\; \tau g > \tau d \\ & = & \tau g_2[u \leftarrow \tau d_1] & \\ & = & \tau' g_2[u \leftarrow \tau d_1] & \\ & > & \tau' g_2[u \leftarrow \tau' d_1] & car \;\; \tau d_1 > \tau' d_1 \\ & > & \tau' g_2[u \leftarrow \tau' g_1] & car \;\; \tau' d_1 > \tau' g_1 \\ & = & \tau' g_2[u \leftarrow \tau' g_2[u.p_t.p \leftarrow d_a]|_u] & car \;\; \tau' \in SOL(g_2[u.p_t.p \leftarrow d_a]|_u =_n g_1 \\ & = & \tau'(g_2[u.p_t.p \leftarrow d_a]) & \end{array}$$

τ' ne vérifiera pas non plus la contrainte $g_2[u.p_t.p \leftarrow d_a] >_n d_2$ et donc on retombe dans le cas précédent ou l'on peut utiliser directement la règle (5).

$\boxed{C.B.B.B}$ Si t est un sous-terme de g_1, $t = g_1|_{p_t}$, $s = g_1|_{p_t.p}$ et $x = g_2|_{u.p_t}$. Ce cas est en tout point similaire au cas précédent. Donnons quand même les grandes étapes de la preuve. Cette situation peut aussi bien se produire dans le cas (i) que dans le cas (ii) mais cela ne change en rien la démarche à suivre. Le terme s est un sous-terme de g_1 et une instanciation de s se réécrit avec la règle r_a, on peut donc créer une paire critique de g_a dans g_1 à la position $p_t.p$ on obtient :

$$(5) \quad : \quad g_1[p_t.p \leftarrow d_a] \doteq d_1[\![g_1|_{p_t.p} =_n g_a \;\wedge\; g_a >_n d_a \;\wedge\; g_1 >_n d_1]\!]$$

L'occurrence qui a été modifiée dans g_1 correspond à une variable de $g_2|_u$, on peut donc superposer le membre gauche de (5) dans g_2 à la position u ce qui donne :

$$\begin{array}{c} (6) \;:\; g_2[u \leftarrow d_1] \doteq d_2[\![g_1|_{p_t.p} =_n g_a \;\wedge\; g_a >_n d_a \;\wedge\; g_1 >_n d_1 \;\wedge\; g_2|_u =_n g_1[p_t.p \leftarrow d_a] \\ \wedge\; g_2 >_n d_2 \;\wedge\; g_1[p_t.p \leftarrow d_a] >_n d_1]\!] \end{array}$$

Nous définissons la substitution τ' de la même façon que précédemment, τ' est toujours solution de la partie égalitaire de la contrainte de la règle (6). Si τ' est aussi solution de la partie ordre de la contrainte, la règle (6) et τ' nous permettent d'arriver à la conclusion de notre lemme. Sinon passons les différentes contraintes d'ordres en revue.

$\circ$ $g_a >_n d_a$, τ' est toujours solution de cette contrainte.

$\circ$ $g_1 >_n d_1$, là encore τ' est solution de cette contrainte car $x \notin V(g_1 \doteq d_1)$, et $\tau g_1 > \tau d_1$.

$\circ$ $g_2 >_n d_2$, si τ' n'est pas solution de cette contrainte, alors on a $\tau' d_2 > \tau' g_2$.

- cas (i) : on a $\tau' g_2[u \leftarrow d_1] = \tau' g = \tau g > \tau d = \tau d_2 > \tau' d_2 > \tau' g_2$ donc $\tau' g_2[u \leftarrow d_1] > \tau' g_2$, on en déduit que $\tau' d_1 > \tau' g_2|_u$ et comme τ' est solution de la partie égalitaire de la contrainte, on a $\tau' g_2|_u = \tau' g_1[p_t.p \leftarrow d_a]$ finalement $\tau' d_1 > \tau' g_1[p_t.p \leftarrow d_a]$ et dans ce cas on peut directement utiliser la règle (5) et τ' pour réécrire τg avec une étape S-quasi normale.

- cas (ii) : τg peut directement être réécrit avec la règle $d_2 \doteq g_2$ et le filtre τ'.

$\circ$ $g_1[p_t.p \leftarrow d_a] >_n d_1$, si τ' n'est pas solution de cette contrainte, on a : $\tau' d_1 > \tau' g_1[p_t.p \leftarrow d_a]$.

- cas (i) : on peut directement utiliser la règle (5) et τ' pour réécrire τg avec une étape S-quasi-normale.

- cas (ii) : on a $\tau' d_1 > \tau' g_1[p_t.p \leftarrow d_a] = \tau' g_2|_u$ donc $\tau' g_2[u \leftarrow d_1] > \tau' g_2$. D'autre part $\tau' d_2 = \tau' g = \tau g > \tau d = \tau g_2[u \leftarrow d_1] > \tau' g_2[u \leftarrow d_1]$, ce qui prouve que $\tau' d_2 > \tau' g_2$ et on retombe dans le cas où τg peut directement être réécrit avec la règle $d_2 \doteq g_2$ et le filtre τ'.

Nous avons passé tous les différents cas en revue et ainsi nous avons prouvé notre lemme. □

De ce dernier lemme, nous déduisons la proposition suivante :

Proposition 2.24 *Soit t un terme clos quelconque, il existe une chaîne de réécriture contrainte en profondeur d'abord et S-quasi-normale, allant de t à sa S forme normale : $t\downarrow$.*

Preuve: S est confluent sur les termes clos, donc il existe une chaîne finie de réécriture, que l'on peut choisir en profondeur d'abord, de t à sa forme normale.

$$t \longrightarrow_{[p_1, g_1 \doteq d_1 [\![C_1]\!], \sigma_1]} t_1 \longrightarrow \dots \longrightarrow_{[p_n, g_n \doteq d_n [\![C_n]\!], \sigma_n]} t_n = t\downarrow$$

Si toutes les étapes sont S-quasi-normales, nous n'avons rien à démontrer, sinon supposons que l'étape $t_{i-1} \longrightarrow_{[p_i, g_i \doteq d_i [\![C_i]\!], \sigma_i]} t_i$ soit la première étape qui n'est pas S-quasi-normale. Le c-filtre σ_i n'est donc pas en S forme normale sur $V(d_i) \backslash V(g_i)$.

Si $\sigma_i\downarrow$, la S forme normale de σ_i, est toujours solution de la contrainte C_i, l'étape de réécriture est faite avec la même règle et le filtre $\sigma_i\downarrow$ et elle devient ainsi S-quasi normale.

Si $\sigma_i\downarrow$ n'est plus solution de C_i à cause d'une contrainte initiale de C_i, alors S contiendra la propagée de $g_i \doteq d_i [\![C_i]\!]$ par rapport à cette contrainte initiale et c'est cette règle qui nous permettra de réécrire t_{i-1} en une étape de réécriture S-quasi normale, (tout en restant en profondeur d'abord).

Si $\sigma_i\downarrow$ n'est plus solution de C_i à cause d'une contrainte nouvelle, cela signifie que $g_i \doteq d_i [\![C_i]\!]$ a été obtenue par superposition contrainte de deux autres règles de S. Mais grâce au lemme 2.22, on sait que $\sigma_i g_i (= t_{i-1}|_{p_i})$ peut se réécrire avec une étape S-quasi-normale. Cette réécriture est faite à l'occurrence p_i car la chaîne de dérivation est en profondeur d'abord. On obtient la nouvelle chaîne de dérivation, en profondeur d'abord, suivante :

$$t \longrightarrow_{[p_1, g_1 \doteq d_1 [\![C_1]\!], \sigma_1]} t_1 \longrightarrow \dots t_{i-1} \longrightarrow_{[p_i, g'_i \doteq d'_i [\![C'_i]\!], \sigma'_i]} t'_i \longrightarrow \dots \longrightarrow_{[p_k, g'_k \doteq d'_k [\![C'_k]\!], \sigma'_k]} t'_k = t\downarrow$$

avec les i premières étapes S-quasi-normales. Il ne peut pas exister de chaîne infinie entre t et $t\downarrow$, car nous avons un ordre de simplification, donc en répétant un nombre fini de fois cette démarche, on construira ainsi une chaîne de dérivation en profondeur d'abord et S-quasi normale pour normaliser t. □

Nous allons utiliser cette propriété de notre système S pour montrer à présent que notre procédure d'E-unification calcule bien toutes les solutions d'un problème donné.

2.5 Complétude de la méthode générale d'unification et exemples

Dans ce paragraphe nous supposerons que S est le système contraint final c'est-à-dire confluent et terminant sur les termes clos, par rapport à un ordre de simplification. Nous commencerons par prouver une nouvelle version du "lifting lemma" à la Hullot appropriée au cadre contraint.

2.5.1 Correspondance entre étape de réécriture et surréduction

Nous allons établir la correspondance entre une étape de réécriture contrainte en profondeur d'abord et S-quasi-normale et une étape de surréduction contrainte. Nous rappelons que la réécriture d'un terme sans contrainte génère un terme sans contrainte.

Lemme 2.25 Soient $t_0 [\![C_0]\!]$ un terme contraint, $g \doteq d [\![C]\!]$ une équation de S et ρ_0 une substitution telle que $\rho_0 \in SOL(C_0)$ et ρ_0 est en S forme normale sur $V(t_0)$. Si $\rho_0 t_0 \longrightarrow_{[p, g \doteq d [\![C]\!], \sigma]} t'_1$ est une étape de réécriture en profondeur d'abord et S-quasi-normale alors $t_0 [\![C_0]\!] \rightsquigarrow_{[p, g \doteq d [\![C]\!]\ ]} t_1 [\![C_1]\!]$ et il existe une substitution ρ_1 telle que :

- $\rho_1 \in SOL(C_1)$,
- $t'_1 = \rho_1 t_1$ et ρ_1 est en S forme normale sur $V(t_1)$,

- $\rho_1 = \rho_0[D(\rho_0)]$ et $D(\rho_0) \subseteq D(\rho_1)$.

Ce que l'on peut schématiser par :

$$\begin{array}{ccc}
\rho_0 t_0 & \longrightarrow_{[p, g \doteq d[\![C]\!], \sigma]} & t'_1 \\
\uparrow \rho_0 & & \uparrow \rho_1 \\
t_0\ [\![C_0]\!] & \leadsto_{[p, g \doteq d[\![C]\!]\,]} & t_1\ [\![C_1]\!]
\end{array}$$

Preuve: On impose bien sûr qu'il n'y ait pas de conflit de variables et pour cela on renomme si nécessaire les variables de la règle $g \doteq d[\![C]\!]$ de façon à ce que l'on ait $V(g \doteq d[\![C]\!]) \cap (V(t_0[\![C_0]\!]) \cup D(\rho_0)) = \emptyset$. La substitution ρ_0 est en S forme normale sur $V(t_0)$ et $\rho_0 t_0$ est réécrit en position p, p est donc une position non variable de t_0. Nous avons donc l'étape de surréduction contrainte suivante : $t_0[\![C_0]\!] \leadsto_{[p, g \doteq d[\![C]\!]\,]} t_1[\![C_1]\!]$ avec $t_1 = t_0[p \leftarrow d]$, et $C_1 = C_0 \ \wedge\ t_0|_p =_n g \ \wedge\ C \ \wedge\ g >_n d$. Soit ρ_1 la substitution définie par $\rho_1 = \sigma \cup \rho_0$ ce qui est possible puisque $D(\sigma) \cap D(\rho_0) = \emptyset$. On a :

- $\rho_1 \in SOL(C_0)$ car $\rho_0 \in SOL(C_0)$,
- $\rho_1(t_0|_p) = \rho_0(t_0|_p) = (\rho_0 t_0)|_p = \sigma g = \rho_1 g$ donc $\rho_1 \in SOL(t_0|_p =_n g)$,
- $\rho_1 \in SOL(C)$ car $\sigma \in SOL(C)$,
- $\rho_1 g = \sigma g > \sigma d = \rho_1 d$ d'où $\rho_1 \in SOL(g >_n d)$.

Ce qui montre que $\rho_1 \in SOL(C_1)$. D'autre part on a :

$$\begin{aligned}
\rho_1(t_1[\![C_1]\!]) &= \rho_1 t_1 \text{ car } \rho_1 \in SOL(C_1) \\
&= \rho_1(t_0[p \leftarrow d]) = (\rho_1 t_0)[p \leftarrow \rho_1 d] \text{ car } p, \\
&= (\rho_0 t_0)[p \leftarrow \sigma d] \text{ car } \rho_1 = \rho_0[V(t_0)] \text{ et } \rho_1 = \sigma[V(d)], \\
&= t'_1, \text{ ce qui montre que } \rho_1 t_1 = t'_1.
\end{aligned}$$

Montrons maintenant que ρ_1 est en S forme normale sur $V(t_1)$. Soit $x \in V(t_1)$, c'est-à-dire $x \in V(t_0[p \leftarrow d])$.

Si $x \in V(t_0)$ alors $\rho_1 x = \rho_0 x$ mais comme ρ_0 est en S forme normale sur $V(t_0)$ on a $\rho_1 x$ normalisé.

Sinon $x \in V(d)$ donc $\rho_1 x = \sigma x$, d'après le lemme 2.21, l'étape de réécriture étant faite en profondeur d'abord et S-quasi-normale on a que σ est normalisé sur $V(d)$ et donc $\rho_1 x$ est normalisé.

La substitution ρ_1 est donc bien en S forme normale sur $V(t_1)$. Par la définition même de ρ_1, on déduit directement que $\rho_1 = \rho_0[D(\rho_0)]$ et $D(\rho_0) \subseteq D(\rho_1)$. Ceci conclut la démonstration de notre lemme. □

A partir de maintenant la preuve de complétude de la surréduction redevient classique.

2.5.2 Complétude de la surréduction contrainte

Soit $u \stackrel{?}{=} v$ un unificande initial. Si cet unificande a une solution close (après avoir skolémisé les variables si nécessaire), alors il existe une substitution τ telle que $\tau u \longleftrightarrow^*_S \tau v$. Mais comme S est confluent sur les termes clos, il existe une S vallée entre les termes τu et τv. D'après la proposition 2.24, on peut supposer que cette S vallée est constituée uniquement d'étapes de réécriture en profondeur d'abord et S-quasi-normales. Et ainsi d'après notre lemme précédent, les étapes de réécriture peuvent être remplacées par des étapes de surréduction contrainte ce qui permet d'obtenir une substitution θ plus générale que τ et telle que $\theta u =_S \theta v$. Notre théorème ci-dessous est juste la description formelle de ceci.

Théorème 2.26 *Complétude de la surréduction contrainte. Soit $s_0 \stackrel{?}{=} t_0[\![C_0]\!]$ une équation contrainte et supposons qu'il existe une substitution ρ_0 telle que :*

- $\rho_0 \in SOL(s_0 \stackrel{?}{=} t_0[\![C_0]\!])$ *c'est-à-dire* $\rho_0 s_0 =_S \rho_0 t_0$ *et* $\rho_0 \in SOL(C_0)$,
- ρ_0 *est en S forme normale sur* $V(s_0 \stackrel{?}{=} t_0)$,
- $\rho_0 s_0$ *et* $\rho_0 t_0$ *sont clos.*

Alors ils existent une dérivation $(s_0 \stackrel{?}{=} t_0[\![C_0]\!]) \rightsquigarrow^*_S (s_n \stackrel{?}{=} t_n[\![C_n]\!])$ *et une substitution ρ_n telles que* $\rho_n \in SOL(C_n)$, $\rho_n s_n = \rho_n t_n$ *et* $\rho_n = \rho_0[D(\rho_0)]$.

Preuve: D'après la proposition 2.24, puisque $\rho_0 s_0$ et $\rho_0 t_0$ sont clos et qu'ils ont la même forme normale, on déduit qu'il existe une vallée entre $\rho_0 s_0$ et $\rho_0 t_0$ dont les étapes de réécriture sont en profondeur d'abord et S-quasi-normales.

Nous allons faire une preuve par induction noethérienne sur la relation de réécriture et à partir de la paire $\{\rho_0 s_0, \rho_0 t_0\}$ vue comme un multi-ensemble.

Cas 1 : Si $\rho_0 s_0 = \rho_0 t_0$, en ne faisant aucune étape de surréduction et avec la substitution ρ_0 nous avons prouvé notre théorème.

Cas 2 : Soit $(\rho_0 s_0 \stackrel{?}{=} \rho_0 t_0) \longrightarrow_S (s'_1 \stackrel{?}{=} t'_1)$ la première étape de réécriture en profondeur d'abord et S-quasi-normale de notre vallée. A partir du lemme 2.25 nous déduisons que $(s_0 \stackrel{?}{=} t_0[\![C_0]\!]) \rightsquigarrow_S (s_1 \stackrel{?}{=} t_1[\![C_1]\!])$ et il existe une substitution $\rho_1 \in SOL(C_1)$, S normalisée sur $V(s_1 \stackrel{?}{=} t_1)$ et telle que $(s'_1 \stackrel{?}{=} t'_1) = \rho_1(s_1 \stackrel{?}{=} t_1)$, $\rho_1 = \rho_0[D(\rho_0)]$ et $D(\rho_0) \subseteq D(\rho_1)$. Nous avons donc le schéma suivant :

$$\begin{array}{ccc}
\rho_0(s_0 \stackrel{?}{=} t_0) & \longrightarrow_S & s'_1 \stackrel{?}{=} t'_1 \\
\uparrow \rho_0 & & \uparrow \rho_1 \\
s_0 \stackrel{?}{=} t_0\ [\![C_0]\!] & \rightsquigarrow_S & s_1 \stackrel{?}{=} t_1\ [\![C_1]\!]
\end{array}$$

Montrons maintenant que $s_1 \stackrel{?}{=} t_1[\![C_1]\!]$ avec la substitution ρ_1 vérifient encore les hypothèses de notre théorème. Nous avons vu que $\rho_1 \in SOL(C_1)$, ρ_1 est normalisée sur $V(s_1 \stackrel{?}{=} t_1)$ et $\rho_1(s_1 \stackrel{?}{=} t_1) = (s'_1 \stackrel{?}{=} t'_1)$. D'autre part, s'_1 et t'_1 sont clos car $\rho_0(s_0 \stackrel{?}{=} t_0) \longrightarrow_S (s'_1 \stackrel{?}{=} t'_1)$ et $\rho_0 s_0$, $\rho_0 t_0$ sont clos. Puisque $\rho_0(s_0 \stackrel{?}{=} t_0) \longrightarrow_S \rho_1(s_1 \stackrel{?}{=} t_1) = (s'_1 \stackrel{?}{=} t'_1)$, on a $\rho_1 s_1 =_S \rho_1 t_1$.

Nous avons bien $\{\rho_0 s_0, \rho_0 t_0\} << \{\rho_1 s_1, \rho_1 t_1\}$ puisque $\rho_0 s_0 \longrightarrow_S \rho_1 s_1$ ou $\rho_0 t_0 \longrightarrow_S \rho_1 t_1$, nous pouvons donc appliquer notre hypothèse de récurrence sur $s'_1 \stackrel{?}{=} t'_1$. Il existe donc une chaîne de surréduction $(s_1 \stackrel{?}{=} t_1[\![C_1]\!]) \rightsquigarrow^*_S (s_n \stackrel{?}{=} t_n[\![C_n]\!])$ et une substitution $\rho_n \in SOL(C_n)$ telle que $\rho_n s_n = \rho_n t_n$ et $\rho_n = \rho_1[D(\rho_1)]$ mais comme $\rho_1 = \rho_0[D(\rho_0)]$ et $D(\rho_0) \subseteq D(\rho_1)$ on a $\rho_n = \rho_0[D(\rho_0)]$, ce qui prouve notre théorème. □

Remarque: La même démonstration peut être faite en remplaçant les étapes de surréduction contrainte par des étapes de surréduction contrainte normalisante. On obtient ainsi la situation suivante :

$$\begin{array}{ccccc}
\rho_0(s_0 \stackrel{?}{=} t_0) & \longrightarrow_S & s'_1 \stackrel{?}{=} t'_1 & \longrightarrow^*_S & \rho_1(s_1{\downarrow} \stackrel{?}{=} t_1{\downarrow}) \\
\uparrow \rho_0 & & \uparrow \rho_1 & & \uparrow \rho_1 \\
s_0 \stackrel{?}{=} t_0[\![C_0]\!] & \rightsquigarrow_S & s_1 \stackrel{?}{=} t_1[\![C_1]\!] & \longrightarrow^*_S & s_1{\downarrow} \stackrel{?}{=} t_1{\downarrow}\ [\![C_1]\!] \\
 & & & & \\
s_0 \stackrel{?}{=} t_0[\![C_0]\!] & & \rightsquigarrow & & s_1{\downarrow} \stackrel{?}{=} t_1{\downarrow}\ [\![C_1]\!]
\end{array}$$

La normalisation ayant lieu uniquement sur les termes s_1 et t_1.

2.5.3 Exemples et commentaires

Le premier exemple montre comment on peut trouver des solutions avant d'avoir un système confluent sur les termes clos.

Exemple 2.27 Soit le système E d'équations suivant :

$$\begin{array}{rcrcl} (1) & : & i(g(h(y)),z) & \doteq & b[\![\ ]\!] \\ (2) & : & i(x,a) & \doteq & f(x)[\![\ ]\!] \\ (3) & : & h(a) & \doteq & b[\![\ ]\!] \\ (4) & : & g(a) & \doteq & b[\![\ ]\!] \end{array}$$

sous un ordre lpo où la précédence est $i > f > g > h > a > b$. Pour une meilleure lisibilité, nous avons supprimé les contraintes d'ordres qui sont triviales. Nous cherchons à résoudre l'unificande $f(g(x)) \stackrel{?}{=} b$ modulo E. Une étape de surréduction est possible avec la règle (4) :

$$f(g(x)) \stackrel{?}{=} b[\![\ ]\!] \rightsquigarrow_{(4)} f(b) \stackrel{?}{=} b[\![x =_n a]\!]$$

Ce nouvel unificande n'a pas de solution pour l'instant et il n'y a plus de surréduction contrainte possible. On crée donc une paire critique entre (1) et (2) ce qui donne:

$$(5) \quad : \quad f(x) \doteq b[\![x =_n g(h(y))]\!]$$

Avec cette nouvelle règle on peut surréduire les deux unificandes :

$$\begin{array}{c} f(b) \stackrel{?}{=} b[\![x =_n a]\!] \rightsquigarrow_{(5)} b \stackrel{?}{=} b[\![x =_n a \ \wedge\ x' =_n b \ \wedge\ x' =_n g(h(y))]\!] \\ f(g(x)) \stackrel{?}{=} b[\![\ ]\!] \rightsquigarrow_{(5)} b \stackrel{?}{=} b[\![x' =_n g(x) \ \wedge\ x' =_n g(h(y))]\!] \end{array}$$

La première surréduction aboutit à un unificande dont la contrainte est insatisfiable, il est donc éliminé. En revanche la deuxième donne la solution $\{x \leftarrow h(y)\}$ qui est la solution la plus générale de notre problème. La complétion continue et donne (6) entre (3) et (1) et finalement (7) entre (6) et (2) :

$$\begin{array}{c} (6) \quad : \quad i(g(b),x) \doteq b[\![\ ]\!] \\ (7) \quad : \quad f(x) \doteq b[\![x =_n g(b)]\!] \end{array}$$

En appliquant une étape de surréduction avec la règle (7) sur le premier unificande on obtient une deuxième solution :

$$f(g(x)) \stackrel{?}{=} b[\![\ ]\!] \rightsquigarrow_{(7)} b \stackrel{?}{=} b[\![g(x) =_n x' \ \wedge\ x' =_n g(b)]\!]$$

Ce qui donne comme solution $\{x \leftarrow b\}$, elle est moins générale que la première solution puisque c'est la normalisation de $x \leftarrow h(y)$ pour l'instanciation close $\{y \leftarrow a\}$. La procédure s'arrête puisque le système est complété et que toute nouvelle surréduction aboutirait à un unificande dont la contrainte serait insatisfiable. Sur cet exemple nous avons obtenu la solution la plus générale avant d'avoir un système complet.

Voyons maintenant un exemple qui nous montre l'utilité des contraintes d'ordres dans les étapes de surréduction.

Exemple 2.28 Soit le système composé uniquement de l'équation de commutativité :

$$(1) \quad : \quad x' * y' \quad \doteq \quad y' * x'[\![\ ]\!]$$

et essayons de résoudre le but $x*y \stackrel{?}{=} a*b$ sous un ordre lpo où la précédence est $* > b > a$. Nous avons une première solution triviale $\{x \leftarrow a,\ y \leftarrow b\}$. Nous appliquons une étape de surréduction contrainte sur le côté gauche de notre unificande (en appliquant sur le côté droit on obtient une contrainte d'ordre insatisfiable car $b * a > a * b$).

$$x * y = a * b[\![\ ]\!] \rightsquigarrow_{(1)} y' * x' \stackrel{?}{=} a * b[\![x =_n x' \ \wedge\ y =_n y' \ \wedge\ x' * y' >_n y' * x']\!]$$

On obtient une seconde solution $\{x \leftarrow b,\ y \leftarrow a\}$. En appliquant de nouveau une étape de surréduction contrainte on a :

$$y' * x' \stackrel{?}{=} a * b[\![x =_n x' \ \wedge\ y =_n y' \ \wedge\ x' * y' >_n y' * x']\!] \rightsquigarrow_{(1)}$$
$$y'' \stackrel{?}{=} x''[\![x =_n x' \ \wedge\ y =_n y' \ \wedge\ x' * y' >_n y' * x' \ \wedge\ y' =_n x'' \ \wedge\ x' =_n y'' \ \wedge\ x'' * y'' >_n y'' * x'']\!]$$

mais cet unificande est supprimé car en résolvant la partie ordre on obtient $x' * y' > y' * x' > x' * y'$ ce qui est impossible puisque l'ordre est terminant. Notre procédure s'arrêtera donc après avoir calculé les deux solutions de notre but initial ce qui n'aurait pas été le cas si on n'avait pas ajouté des contraintes d'ordres dans les étapes de surréduction. On voit ici que même lorsque les règles ne sont pas orientables, la surréduction peut terminer.

L'exemple suivant montre qu'en pratique, si les contraintes d'ordres créées par des étapes de surréduction persistent dans la solution (possible lorsque celle-ci est non close), on peut les supprimer pour exprimer la solution. C'est le rôle joué par la règle (8) de notre système d'inférence.

Exemple 2.29 Soit le système E composé des équations de commutativité et d'associativité :

$$\begin{array}{lcrcl} (1) & : & x * y & \doteq & y * x[\![\ ]\!] \\ (2) & : & (x * y) * z & \doteq & x * (y * z)[\![\ ]\!] \end{array}$$

sous un ordre lpo où $* > a > b$, a et b étant deux constantes, et $*$ a un status de gauche à droite, (pour permettre d'orienter la règle d'associativité de gauche à droite).

Essayons de trouver la solution non close $\{x \leftarrow a * u,\ y \leftarrow b * z\}$ pour l'unificande $(x * b) * z \stackrel{?}{=} (a * y) * u[\![\ ]\!]$ par rapport au système E. On obtient la chaîne de surréduction suivante. Pour la lisibilité nous n'avons pas accumulé les contraintes et les contraintes d'ordres triviales sont automatiquement supprimées :

$$\begin{array}{l}
\quad ((x * b) * z \stackrel{?}{=} (a * y) * u) \\
\rightsquigarrow_{(2)} (x_1 * (y_1 * z_1)) * z \stackrel{?}{=} (a * y) * u \ [\![(x =_n x_1 * y_1) \wedge (z_1 =_n b)]\!] \\
\rightsquigarrow_{(1)} (x_1 * (y_2 * x_2)) * z \stackrel{?}{=} (a * y) * u \ [\![(x_2 =_n y_1) \wedge (y_2 =_n z_1) \wedge (x_2 * y_2 >_n y_2 * x_2)]\!] \\
\rightsquigarrow_{(2)} x_3 * (y_3 * z_3) \stackrel{?}{=} (a * y) * u \ [\![(x_3 =_n x_1) \wedge (y_3 =_n y_2 * x_2) \wedge (z_3 =_n z)]\!] \\
\rightsquigarrow_{(2)} x_3 * (x_4 * (y_4 * z_4)) \stackrel{?}{=} (a * y) * u \ [\![(y_3 =_n x_4 * y_4) \wedge (z_4 =_n z_3)]\!] \\
\rightsquigarrow_{(1)} x_3 * (x_4 * (y_5 * x_5)) \stackrel{?}{=} (a * y) * u \ [\![(x_5 =_n y_4) \wedge (y_5 =_n z_4) \wedge (x_5 * y_5 >_n y_5 * x_5)]\!] \\
\rightsquigarrow_{(2)} x_3 * (x_4 * (y_5 * x_5)) \stackrel{?}{=} x_6 * (y_6 * z_6) \ [\![(x_6 =_n a) \wedge (y_6 =_n y) \wedge (z_6 =_n u)]\!] \\
\rightsquigarrow_{(2)} x_3 * (x_4 * (y_5 * x_5)) \stackrel{?}{=} x_6 * (x_7 * (y_7 * z_7)) \ [\![(y_6 =_n x_7 * y_7) \wedge (z_7 =_n z_6)]\!]
\end{array}$$

Si on applique la règle d'inférence '*Génération d'une solution*' on obtient la contrainte : $[\![(x_3 =_n x_6) \wedge (x_4 =_n x_7) \wedge (y_5 =_n y_7) \wedge (x_5 =_n z_7) \wedge (x =_n x_1 * y_1) \wedge (z_1 =_n b) \wedge (x_2 =_n y_1) \wedge (y_2 =_n z_1) \wedge (x_3 =_n x_1) \wedge (y_3 =_n y_2 * x_2) \wedge (z_3 =_n z) \wedge (y_3 =_n x_4 * y_4) \wedge (z_4 =_n z_3) \wedge (x_5 =_n y_4) \wedge (y_5 =_n z_4) \wedge (x_6 =_n a) \wedge (y_6 =_n y) \wedge (z_6 =_n u) \wedge (y_6 =_n x_7 * y_7) \wedge (z_7 =_n z_6) \wedge (x_2 * y_2 >_n y_2 * x_2) \wedge (x_5 * y_5 >_n y_5 * x_5)]\!]$.

Si on applique la règle 'simplification d'une solution', les contraintes d'ordres disparaissent et cela donne $x \leftarrow x_1 * y_1 = x_3 * x_2 = x_6 * y_4 = a * x_5 = a * z_7 = a * z_6 = a * u$, et $y \leftarrow y_6 = x_7 * y_7 = x_4 * y_5 = y_2 * z_4 = z_1 * z_3 = b * z$; c'est-à-dire la solution $\{x \leftarrow a * u, y \leftarrow b * z\}$ Pour satisfaire la contrainte d'ordre, il aurait fallu considérer la substitution $\{x \leftarrow a * a, y \leftarrow b * b\}$. Cette suppression permet donc d'obtenir des substitutions non closes. Cette suppression n'est effectuée que dans les unificandes de la forme $\emptyset[\![C]\!]$, afin de laisser suffisamment de temps pour que les contraintes d'ordres puissent jouer leur rôle : couper des branches infinies (voir l'exemple précédent).

Nous ne sommes intéressés que par les solutions normalisées de l'unificande que l'on a à résoudre (d'après le théorème 2.26). Pour cela nous pouvons soit normaliser les solutions obtenues soit supprimer tous les unificandes qui possèdent une contrainte du type $[\![x =_n t]\!]$ avec t un terme qui n'est pas en forme normale par rapport à notre système d'équations.

Exemple 2.30 Soit E le système d'équations suivant :

$$\begin{array}{rcrcl} (1) & : & h(0,y) & \doteq & y[\![\ ]\!] \\ (2) & : & f(0) & \doteq & 0[\![\ ]\!] \end{array}$$

sous un ordre lpo où $f > h > 0$. Essayons de résoudre $h(x, f(x)) \stackrel{?}{=} 0$ par rapport à E. L'unificande peut être réduit par les deux règles, on a :

$$h(x, f(x)) \stackrel{?}{=} 0[\![\ ]\!] \rightsquigarrow_{(1)} y \stackrel{?}{=} 0[\![h(x, f(x)) =_n h(0, y) \ \wedge\ h(0, y) >_n y]\!]$$

Ce qui donne l'unificande $y \stackrel{?}{=} 0[\![x =_n 0 \ \wedge\ y =_n f(0)]\!]$. Si on normalise la contrainte, on trouve la solution $\{x \leftarrow 0\}$.

Mais cette solution est trouvée par l'autre surréduction possible :

$$\begin{array}{l} h(x, f(x)) \stackrel{?}{=} 0[\![\ ]\!] \rightsquigarrow_{(2)} h(x, 0) \stackrel{?}{=} 0[\![f(x) =_n f(0)]\!] \\ \rightsquigarrow_{(1)} y \stackrel{?}{=} 0[\![f(x) =_n f(0) \ \wedge\ h(x, 0) =_n h(0, y)]\!] \end{array}$$

Ce qui donne directement la solution $\{x \leftarrow 0\}$.

D'autres exemples traités par notre logiciel UREVEAL seront donnés dans le chapitre sur l'implémentation.

2.6 Conclusion

Nous avons défini une procédure générale d'unification qui devrait être plus efficace que les procédures de Gallier Snyder [GS89] et de Dougherty Johann [DJ92]. Entre autres, la notion d'ordre, qui a pour but de réduire l'espace de recherche, joue un rôle très limité dans leurs méthodes, alors qu'elle est très utilisée dans la notre. Des comparaisons en temps seraient intéressantes; mais nous ne pouvons pas les effectuer car nous ne disposons pas d'implémentations de leurs algorithmes, équivalentes à la notre (en C et sur SUN 4).

Nous allons voir dans le chapitre suivant, un moyen pour améliorer l'efficacité de la surréduction en introduisant un graphe d'opérateurs. D'autres optimisations pourront être envisagées, comme par exemple l'introduction de stratégies équivalentes à la LSE surréduction [KB91, BKW92], ce qui permettrait de calculer une fois et une seule chaque solution. Des stratégies au niveau de la complétion de notre système, en fonction du problème à résoudre, sont également à l'étude.

La complétion et la surréduction sont faites dans un cadre contraint, ce qui permet d'être basique. Les contraintes peuvent aussi nous permettre de différer le moment où l'on calcule l'unificateur pour créer de nouvelles paires critiques ou pour faire des étapes de surréduction. Jusqu'à présent ce fait n'est pas utilisé car l'unification syntaxique est simple et peu coûteuse. En revanche, si on décidait de travailler dans des théories axiomatiques, en permettant par exemple à certains opérateurs d'être associatifs et commutatifs, il faudrait résoudre les contraintes non plus syntaxiquement mais modulo les axiomes. Ceci est beaucoup plus coûteux et peut donner un grand nombre de solutions, (même en ne conservant que les solutions 'minimales'). L'intérêt des contraintes est donc de cumuler le plus possible de contraintes avant de résoudre, ce qui permet de réduire l'ensemble des solutions, (car $SOL(C_1 \wedge C_2) = SOL(C_1) \cap SOL(C_2)$). Avec par exemple un opérateur $*$ associatif et commutatif, la résolution de $[\![x * y = a * a * b * b]\!]$

donne cinq solutions, mais si l'on ajoute la nouvelle contrainte $[\![x = a]\!]$ on n'obtient plus qu'une solution. Cet aspect sera abordé dans nos travaux à venir.

Chapitre 3

La surréduction dirigée par un graphe

Dans ce chapitre nous allons présenter des travaux de recherche débutés durant mon stage de DEA et qui se sont poursuivis durant la première année de ma thèse. La surréduction est une méthode complète d'unification modulo un système de réécriture confluent et terminant, mais hélas, cette procédure termine rarement. Nous avons vu dans le chapitre précédent plusieurs moyens pour améliorer la terminaison de la surréduction. Nous allons ici reformuler la méthode de Dershowitz et Sivakumar [DS87, DS88] basée sur un graphe d'opérateurs afin de la rendre équitable, et apporter une nouvelle idée qui est l'introduction d'un graphe de termes. Dans les deux cas, le graphe est construit à partir du système de réécriture et de l'équation à résoudre. L'ensemble complet de solutions est obtenu, par surréduction, en étudiant toutes les surdérivations provenant de l'équation à résoudre. Cependant seulement les surdérivations qui aboutissent à une équation syntaxiquement unifiable donnent des solutions, les autres sont inutiles. L'utilisation d'un graphe va nous permettre de ne pas calculer certaines de ces branches et ainsi de réduire l'arbre de recherche.

Dans la première partie de ce chapitre nous allons présenter le graphe d'opérateurs et le système d'inférence permettant d'obtenir une méthode de R-unification équitable [1]. Dans une deuxième partie, nous allons transformer le graphe d'opérateurs en un graphe de termes. Ces travaux ont été réalisés avant d'avoir défini notre nouvelle méthode d'unification basée sur la surréduction contrainte, et décrite dans le chapitre précédent. C'est pour cela que nous présentons cette stratégie dans le cadre de la surréduction usuelle. Dans la dernière partie de ce chapitre, nous élaborons une méthode pour optimiser la surréduction contrainte, en introduisant un graphe d'opérateurs. Dans un but d'homogénéisation, on pourrait formuler cette méthode en considérant les restrictions imposées par le graphe comme un nouveau type de contrainte. Ceci entre parfaitement dans le rôle général joué par les contraintes, mais impose des règles lourdes de manipulation sur celles-ci. Pour cette raison, nous n'avons pas adopté ce point de vue dans ce travail (tout en espérant l'aborder dans un avenir pas trop lointain). La présentation faite dans cette thèse n'aura ainsi qu'un seul objectif : voir le graphe des opérateurs (ou termes) comme un moyen quelque peu "externe", utilisé pour atténuer le coût de la surréduction, avec des contraintes de type simple.

Les deux premières parties relatent des travaux réalisés en collaboration avec P. Réty, qui ont fait l'objet d'un article publié à la conférence RTA91 [CR91a] et de plusieurs rapports internes [CHA90, CR91b, LIM92].

[1] Une méthode est équitable si toute solution est calculée au bout d'un temps fini.

3.1 Utilisation d'un graphe d'opérateurs

Nous allons présenter ici la notion de graphe d'opérateurs, ainsi que le formalisme et le système d'inférence utilisés pour obtenir une procédure de surréduction terminant plus souvent que la surréduction ordinaire.

3.1.1 Construction du graphe d'opérateurs

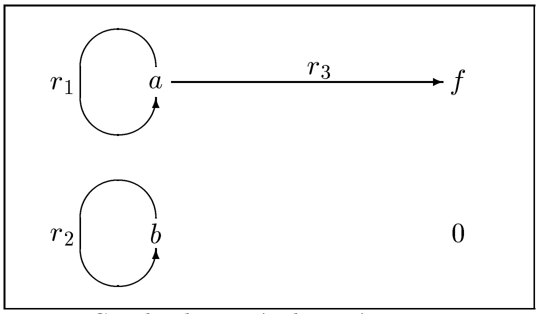
Exemple 3.1 Soit R le système de réécriture suivant :

$$\begin{array}{lcrcl} (r1) & : & a(f(x)) & \rightarrow & a(x) \\ (r_2) & : & b(f(x)) & \rightarrow & b(x) \\ (r_3) & : & a(0) & \rightarrow & f(0) \end{array}$$

Si on demande de calculer l'ensemble complet des unificateurs de $a(x) \stackrel{?}{=} b(x)$ par surréduction ordinaire, le programme va boucler sans fin car :

$$a(x) \stackrel{?}{=} b(y) \rightsquigarrow_{[r1,\{x \leftarrow f(x_1)\}]} a(x_1) \stackrel{?}{=} b(y) \rightsquigarrow_{[r1,\{x_1 \leftarrow f(x_2)\}]} \cdots$$

est une branche infinie. Or en regardant le système de réécriture, on peut s'apercevoir tout de suite que notre équation n'a pas de solution. En effet un terme commençant par a ne peut se surréduire qu'en un terme commençant par a en utilisant $r1$ ou commençant par f en utilisant r_3 et un terme commençant par b ne peut se surréduire qu'en un terme commençant par b en utilisant r_2. Donc les symboles de tête a et b ne peuvent pas être rendus égaux par surréduction. Cela se traduit dans le graphe de portée des opérateurs ci-dessous, par le fait qu'il n'y a pas de chemin faisant confluer a et b vers un même opérateur.



Graphe de portée des opérateurs.

Comme a et b ne sont pas joignables dans le graphe, on en déduira directement que l'équation $a(x) \stackrel{?}{=} b(y)$ n'a pas de solution dans R.

L'ensemble des nœuds du graphe est l'ensemble des symboles de fonctions apparaissant dans le système de réécriture et dans l'équation à résoudre. Soient a et b deux nœuds du graphe, il y a un arc de a vers b ssi il existe dans le système de réécriture R, une règle $s \rightarrow t$ avec $t\hat{e}te(s) = a$ et $t\hat{e}te(t) = b$ ou $t\hat{e}te(s) = a$ et t est une variable. Nous obtenons ainsi un graphe ordonné fini. Pour ne pas confondre les arcs du graphe avec la relation de réécriture, on notera par $\xrightarrow{gr}$ les arcs du graphe.

Définition 3.2 Soient a et b deux nœuds d'un graphe d'opérateurs G, on dit qu'il existe un chemin de a à b dans G ssi il existe un nombre fini de nœuds $a_0, a_1, ..., a_n \in G$ avec $a_0 = a$ et $a_n = b$ et $\forall i = 1...n-1$, l'arc $a_i \xrightarrow{gr} a_{i+1}$ est dans G. On dit que a et b sont joignables dans G s'il existe un nœud c de G tel qu'il y ait un chemin de a à c et un chemin de b à c. $\diamond$

Nous avons la propriété suivante : une équation $s \stackrel{?}{=} t$ n'aura pas de solution dans le système de réécriture R si $tête(s)$ et $tête(t)$ ne sont pas joignables dans le graphe d'opérateurs G associé à R.

3.1.2 Le formalisme utilisé

Nous allons présenter une formulation équationnelle de la surréduction différente de celle de [MMR87] puisque nous introduisons de nouveaux types d'équations pour limiter les redondances. La procédure de surréduction, décrite par un système d'inférence au paragraphe suivant, utilise des systèmes d'équations, appelés **unificandes**, comme dans [KIR84] sauf que nous n'utiliserons pas de multiéquations, c'est-à-dire d'équations à plus de deux arguments.

Soit R un système de réécriture. On définit cinq types d'équations et pour chacun d'eux un ensemble de solutions normalisées.

1. On appelle équation modulo R ou équation en $\doteq_R$ tout objet de la forme $t \doteq_R t'$ où $SOL(t \doteq_R t') = \{\theta | \theta \text{ est normalisé et } \theta t =_R \theta t'\}$.

2. On appelle équation dirigée modulo R ou équation en $\stackrel{\rightarrow}{=}_R$ tout objet de la forme $t \stackrel{\rightarrow}{=}_R t'$ où $SOL(t \stackrel{\rightarrow}{=}_R t') = \{\theta | \theta \text{ est normalisé et } \theta t \rightarrow_R^* \theta t' \text{ par une dérivation basique }\}$.

3. On appelle équation syntaxique ou équation en $\doteq$ tout objet de la forme $t \doteq t'$ où $SOL(t \doteq t') = \{\theta | \theta \text{ est normalisé et } \theta t = \theta t'\}$.

4. On appelle équation triviale l'équation notée T où $SOL(T) = \{\theta | \theta \text{ est normalisé }\}$.

5. On appelle équation fausse l'équation notée F où $SOL(F) = \emptyset$.

Soient deux nouveaux symboles associatifs et commutatifs $\wedge$ et $\vee$ d'arité variable. Un unificande peut être défini de la façon récurrente suivante :

- toute équation $t \doteq_R t'$ ou $t \stackrel{\rightarrow}{=}_R t'$ ou $t \doteq t'$ ou T ou F est un unificande,

- si $S_1, ..., S_n$ sont des unificandes, alors $\wedge(S_1, ..., S_n)$ et $\vee(S_1, ..., S_n)$ sont des unificandes.

Un unificande de la forme $\wedge S_1 ... S_n$ est appelé un facteur conjonctif. La définition de l'ensemble des solutions est étendue aux unificandes de la façon naturelle suivante :

$$SOL(\wedge S_1 ... S_n) = \cap_{i=1}^n SOL(S_i)$$
$$SOL(\vee S_1 ... S_n) = \cup_{i=1}^n SOL(S_i)$$

Les deux symboles $\wedge$ et $\vee$ seront aussi bien utilisés en notation préfixe sous forme aplatie $(\wedge(S_1, \wedge(S_2, S_3)) = \wedge S_1, S_2, S_3)$ qu'en notation infixe $(S_1 \wedge S_2 \wedge S_3)$.

Définition 3.3 Un unificande S est dit en forme disjonctive s'il est de la forme $S = \vee C_1 ... C_n$ où C_1 ... C_n sont des facteurs conjonctifs. $\diamond$

On peut toujours mettre un unificande en forme disjonctive en effectuant la distributivité de $\wedge$ sur le $\vee$. La forme disjonctive d'un unificande sera notée S_{dc}; et bien évidemment, on a $SOL(S_{dc}) = SOL(S)$.

Définition 3.4 L'unificande $S = x_1 \doteq t_1 \wedge ... \wedge x_n \doteq t_n$ contient un cycle de variables s'il existe des variables $x_{i_1}, ..., x_{i_k}$ parmi $x_1, ..., x_n$ telles que $x_{i_2} \in V(t_{i_1})$, ..., $x_{i_k} \in V(t_{i_{k-1}})$, $x_{i_1} \in V(t_{i_k})$ et au moins un des termes $t_{i_1}, ..., t_{i_k}$ est non variable. $\diamond$

Définition 3.5 Un facteur conjonctif C est dit en forme résolue si :

- $C = F$ ou $C = T$ ou

- $C = x_1 \doteq t_1 \wedge ... \wedge x_n \doteq t_n$ avec $\forall ij, \quad i \neq j \quad \Rightarrow \quad x_i \neq x_j$ et C ne contient pas de cycle de variables. $\diamond$

Définition 3.6 Un unificande S est dit trivial si toute substitution est solution, formellement si $SOL(S)$ est l'ensemble de toutes les substitutions normalisées. $\diamond$

L'unificande trivial a pour ensemble complet d'unificateurs $\{Id\}$ (l'identité). Pour tout unificande S, on a l'équivalence suivante : $\theta \in SOL(S) \equiv \theta S$ est trivial

Nous allons maintenant décrire la procédure de surréduction utilisant un graphe d'opérateurs.

3.1.3 La procédure de surréduction

Dans les règles d'inférence qui vont suivre, s et t représenteront des termes non variables, $s_1, ..., s_n, t_1, ..., t_n$ seront les sous termes au rang 1 de s et t, x et y seront des variables et $g \to d$ sera une règle de réécriture, ($g_1, ..., g_n$ seront les sous-termes au rang 1 de g). La relation induite par les règles d'inférence est notée $\longrightarrow$ et le symbole $\bigvee$ voudra dire que l'on fait une disjonction entre toutes les solutions de la condition.

La première règle prend en compte une équation en $\doteq_R$ et effectue en un seul passage les opérations suivantes :

- l'unification syntaxique lorsque celle-ci est évidente (l'un des termes est une variable qui n'est pas dans l'autre terme),

- l'unification modulo R sur les sous-termes lorsqu'aucune surréduction n'est appliquée en tête,

- l'ensemble des surréductions issues des deux membres de l'équation en appliquant au moins une règle en tête et pouvant aboutir à une solution (utilisation du graphe).

(Surréduction1)

$x \doteq_R t \longrightarrow$ $t \doteq_R x$	$(x \doteq t)$ $\bigvee t\|_u \doteq g \wedge t[u \leftarrow d] \doteq_R x$	 si $u \in \overline{O}(t)$ et $t\hat{e}te(t\|_u) = t\hat{e}te(g)$
$x \doteq_R y \longrightarrow$	$x \doteq y$	
$s \doteq_R t \longrightarrow$	$s_1 \doteq_R t_1 \wedge ... \wedge s_n \doteq_R t_n$	si $t\hat{e}te(s) = t\hat{e}te(t)$
	$\bigvee s_1 \stackrel{\rightarrow}{=}_R g_1 \wedge ... \wedge s_n \stackrel{\rightarrow}{=}_R g_n \wedge d \doteq_R t$	si $t\hat{e}te(g) = t\hat{e}te(s)$ et $t\hat{e}te(d), t\hat{e}te(t)$ sont joignables dans le graphe
	$\bigvee t_1 \stackrel{\rightarrow}{=}_R g_1 \wedge ... \wedge t_n \stackrel{\rightarrow}{=}_R g_n \wedge s \doteq_R d$	si $t\hat{e}te(g) = t\hat{e}te(t)$ et s'il y a un chemin de $t\hat{e}te(t)$ à $t\hat{e}te(s)$ dans le graphe

Dans la dernière partie de cette règle d'inférence, la condition pour la surréduction en tête du côté gauche (s) impose que $t\hat{e}te(s)$ et $t\hat{e}te(t)$ soient joignables dans le graphe, (en passant par l'intermédiaire de $t\hat{e}te(g)$ et $t\hat{e}te(d)$), alors que pour la surréduction en tête du côté droit (t) on impose qu'il y ait un chemin de $t\hat{e}te(t)$ à $t\hat{e}te(s)$. Les deux conditions sont différentes pour éviter d'appliquer en même temps des surréductions en tête de s et en tête de t et ainsi éviter des redondances. Avec les conditions ainsi définies, on commencera par surréduire s, puis on surréduira t lorsque plus aucune surréduction ne sera possible en tête de s.

Notre deuxième règle d'inférence prend en compte une équation en $\stackrel{\rightarrow}{=}_R$ et effectue les mêmes opérations que la première, sauf que les surréductions ne peuvent être appliquées que sur le membre gauche de l'équation. Le côté droit de l'équation ne peut être que décomposé.

(Surréduction2)

$x \stackrel{\rightarrow}{=}_R t \longrightarrow \quad x \doteq t$

$x \stackrel{\rightarrow}{=}_R y \longrightarrow \quad x \doteq y$

$s \stackrel{\rightarrow}{=}_R y \longrightarrow \quad y \doteq s$
$\quad \bigvee s|_u \doteq g \wedge s[u \leftarrow d] \stackrel{\rightarrow}{=}_R y \qquad$ si $u \in \overline{O}(s)$ et $tête(s|_u) = tête(g)$

$s \stackrel{\rightarrow}{=}_R t \longrightarrow \quad s_1 \stackrel{\rightarrow}{=}_R t_1 \wedge ... \wedge s_n \stackrel{\rightarrow}{=}_R t_n \qquad$ si $tête(s) = tête(t)$
$\quad \bigvee s_1 \stackrel{\rightarrow}{=}_R g_1 \wedge ... \wedge s_n \stackrel{\rightarrow}{=}_R g_n \wedge d \stackrel{\rightarrow}{=}_R t \qquad$ si $tête(g) = tête(s)$ et s'il y a un chemin de $tête(d)$ à $tête(t)$ dans le graphe

Lorsqu'une de ces deux règles d'inférence génère un unificande vide celui-ci est remplacé par F. Les variables de la règle $g \to d$ sont bien évidemment renommées pour éviter tout conflit.

Ces règles qui font des étapes de surréduction sont accompagnées par des règles de simplification qui vont nous permettre d'aboutir à des unificandes dont les facteurs conjonctifs sont en forme résolue.

Pour mettre en forme résolue les facteurs conjonctifs :

(Résoudre-$\doteq$) $\quad \dfrac{t_1 \doteq t'_1 \wedge ... \wedge t_n \doteq t'_n}{S'}$ où $\begin{cases} S' \text{ est un facteur conjonctif en forme résolue} \\ \text{et } SOL(S') = SOL(t_1 \doteq t'_1 \wedge ... \wedge t_n \doteq t'_n) \\ \text{(On résout par l'algorithme d'unification} \\ \text{syntaxique)} \end{cases}$

Pour supprimer les équations qui feraient aboutir à des solutions non normalisées :

(Supp-$\doteq$) $\quad \dfrac{x \doteq t}{F}$ si t n'est pas en forme normale

Pour supprimer le plus possible d'équations :

(Supp-$\doteq_R$) $\quad \dfrac{t \doteq_R t}{T}$

(Supp- $\stackrel{\rightarrow}{=}_R$) $\quad \dfrac{t \stackrel{\rightarrow}{=}_R t}{T}$

(Supp-$\vee$-T) $\quad \dfrac{S_1 \vee ... \vee S_{i-1} \vee T \vee S_{i+1} \vee ... \vee S_n}{T}$

(Supp-$\vee$-F) $\quad \dfrac{S_1 \vee ... \vee S_{i-1} \vee F \vee S_{i+1} \vee ... \vee S_n}{S_1 \vee ... \vee S_n}$

(Supp-$\wedge$-T) $\quad \dfrac{S_1 \wedge ... \wedge S_{i-1} \wedge T \wedge S_{i+1} \wedge ... \wedge S_n}{S_1 \wedge ... \wedge S_n}$

(Supp-$\wedge$-F) $\quad \dfrac{S_1 \wedge ... \wedge S_{i-1} \wedge F \wedge S_{i+1} \wedge ... \wedge S_n}{F}$

Pour mettre sous forme disjonctive un unificande :

(Distributivité) $\quad \dfrac{S_1 \wedge ... \wedge S_n \wedge (Q_1 \vee ... \vee Q_p)}{(S_1 \wedge ... \wedge S_n \wedge Q_1) \vee ... \vee (S_1 \wedge ... \wedge S_n \wedge Q_p)}$

Au départ nous partons d'un système de réécriture R canonique et d'un unificande qui est composé de l'équation que nous voulons résoudre modulo R. Soit V l'ensemble des variables de l'unificande à résoudre. Pour prouver que les règles d'inférence réalisent de la R-unification, il faut montrer que chaque règle est correcte et complète sur les variables de V, c'est-à-dire que pour toute transformation de l'unificande S en un unificande S' avec une ou plusieurs règles d'inférence, on a $SOL(S) = SOL(S')[V]$. Mais ce n'est pas suffisant car la règle qui transforme tout unificande en lui-même est correcte et complète, et pourtant elle ne permet pas de résoudre des équations modulo R. Il faut montrer en plus, que pour toute solution θ, on pourra toujours atteindre sous une hypothèse d'équité à définir, un unificande S_θ dont θ est une solution syntaxique.

Théorème 3.7 *Toutes les règles d'inférence présentées précédemment sont correctes et complètes.*

La preuve de ce théorème est totalement similaire à celle faite pour le système d'inférence utilisant un graphe de termes et qui se trouve en annexe de ce chapitre. Le système d'inférence avec graphe de termes va être décrit dans la partie suivante de ce chapitre.

Il nous faut définir maintenant une bonne stratégie d'application des règles d'inférence, c'est-à-dire une stratégie qui permettra d'obtenir un ensemble complet de solutions.

Définition 3.8 Hypothèse et procédure d'équité
Une dérivation $S_0 \longrightarrow ... \longrightarrow S_n \longrightarrow ...$ est dite équitable si elle est réalisée par la procédure d'équité suivante :
Tant que l'unificande courant contient des équations en $\doteq_R$ ou $\vec{=}_R$ faire :

- Appliquer la règle **(Surréduction1)** sur toutes les équations en $\doteq_R$ et **(Surréduction2)** sur toutes les équations en $\vec{=}_R$.
- Appliquer les règles de simplification au maximum.
- Appliquer la règle de distributivité pour mettre l'unificande en forme disjonctive et réappliquer les règles de simplification. (Du point de vue pratique cette étape peut être faite qu'une fois de temps en temps pour éviter une trop grande duplication des équations.)

$\diamond$

La procédure ci-dessus est à fortiori correcte. En particulier, aucune équation en $\doteq_R$ et $\vec{=}_R$ n'est délaissée indéfiniment.

Théorème 3.9 *Soient S_0 un unificande et $S_0 \longrightarrow ... \longrightarrow S_n \longrightarrow ...$ une dérivation équitable issue de S_0 et soit $\theta_n \in SOL(S_n)$ alors $\theta_n \in SOL(S_0)$*

De plus elle est complète, ce qui signifie que l'on a bien défini une méthode de R-unification qui trouve toute solution au bout d'un temps fini.

Théorème 3.10 *Soient S_0 un unificande, $\theta_0 \in SOL(S_0)$ et $S_0 \xrightarrow{*} ... \xrightarrow{*} S_n \xrightarrow{*} ...$ une dérivation équitable issue de S_0 où $S_1, ..., S_n, ...$ sont des unificandes en forme disjonctive obtenus après un certain nombre d'étapes de la procédure d'équité. Alors il existe un unificande S_i dans cette dérivation et une substitution θ_i tels que : θ_i est solution d'un facteur conjonctif de S_i qui ne contient pas d'équation en $\doteq_R$ ou en $\vec{=}_R$, et $\theta_i = \theta_0[V(S_0)]$, ($\theta_i$ peut être calculée par unification syntaxique).*

Comme pour le théorème de correction et complétude des règles d'inférence, les preuves sont similaires à celles présentées en annexe, pour le graphe de termes.

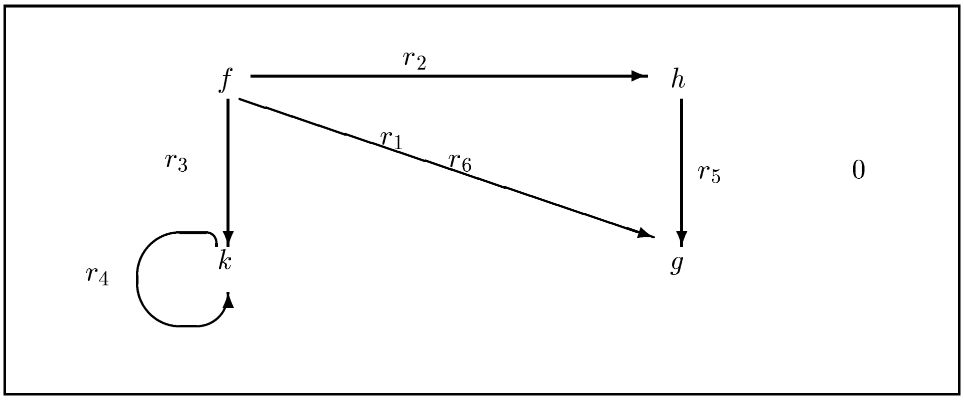
Nous allons maintenant exécuter cette procédure sur deux exemples.

3.1.4 Exemples de surréduction avec graphe d'opérateurs

Soit le système de réécriture canonique R suivant :

$$\begin{array}{lcrcl}
(r_1) & : & f(h(x)) & \rightarrow & g(x) \\
(r_2) & : & f(g(x)) & \rightarrow & h(x) \\
(r_3) & : & f(k(x)) & \rightarrow & k(x) \\
(r_4) & : & k(k(x)) & \rightarrow & k(x) \\
(r_5) & : & h(0) & \rightarrow & g(0) \\
(r_6) & : & f(f(0)) & \rightarrow & g(0)
\end{array}$$

On lance notre procédure de surréduction avec l'unificande suivant : $S = f(f(x)) \doteq_R g(y)$. On commence par construire le graphe des opérateurs et on obtient :



Graphe de portée des opérateurs.

On voit que pour unifier un terme qui commence par f avec un terme qui commence par g on a plusieurs chemins possibles, mais il ne faut pas transformer le terme qui commence par f en un terme qui commence par k, car ensuite k et g ne sont pas joignables.

En exécutant la procédure décrite auparavant, on obtient :

$S = f(f(x)) \doteq_R g(y)$

$$\begin{array}{ll}
\longrightarrow_{Sur} & (f(x) \overset{\rightarrow}{=}_R h(x_1) \wedge g(x_1) \doteq_R g(y)) \\
& \vee(f(x) \overset{\rightarrow}{=}_R g(x_2) \wedge h(x_2) \doteq_R g(y)) \\
& \vee(f(x) \overset{\rightarrow}{=}_R f(0) \wedge g(0) \doteq_R g(y))
\end{array}$$

$$\begin{array}{ll}
\longrightarrow_{Sur} & (x \overset{\rightarrow}{=}_R g(x_3) \wedge h(x_3) \overset{\rightarrow}{=}_R h(x_1) \wedge x_1 \doteq_R y) \\
& \vee((x \overset{\rightarrow}{=}_R h(x_4) \wedge g(x_4) \overset{\rightarrow}{=}_R g(x_2)) \vee (x \overset{\rightarrow}{=}_R g(x_5) \wedge h(x_5) \overset{\rightarrow}{=}_R g(x_2)) \\
& \quad \vee(x \overset{\rightarrow}{=}_R f(0) \wedge g(0) \overset{\rightarrow}{=}_R g(x_2))) \wedge x_2 \overset{\rightarrow}{=}_R 0 \wedge g(0) \doteq_R g(y)) \\
& \vee(x \overset{\rightarrow}{=}_R 0 \wedge 0 \doteq_R y)
\end{array}$$

$$\begin{array}{ll}
\longrightarrow_{Dis} & (x \overset{\rightarrow}{=}_R g(x_3) \wedge h(x_3) \overset{\rightarrow}{=}_R h(x_1) \wedge x_1 \doteq_R y) \\
& \vee(x \overset{\rightarrow}{=}_R h(x_4) \wedge g(x_4) \overset{\rightarrow}{=}_R g(x_2) \wedge x_2 \overset{\rightarrow}{=}_R 0 \wedge g(0) \doteq_R g(y)) \\
& \vee(x \overset{\rightarrow}{=}_R g(x_5) \wedge h(x_5) \overset{\rightarrow}{=}_R g(x_2) \wedge x_2 \overset{\rightarrow}{=}_R 0 \wedge g(0) \doteq_R g(y)) \\
& \vee(x \overset{\rightarrow}{=}_R f(0) \wedge g(0) \overset{\rightarrow}{=}_R g(x_2) \wedge x_2 \overset{\rightarrow}{=}_R 0 \wedge g(0) \doteq_R g(y)) \\
& \vee(x \overset{\rightarrow}{=}_R 0 \wedge 0 \doteq_R y)
\end{array}$$

$$\begin{array}{ll}
\longrightarrow_{Sur} & (x \doteq g(x_3) \wedge x_3 \overset{\rightarrow}{=}_R x_1 \wedge x_1 \doteq y) \\
& \vee(x \doteq h(x_4) \wedge x_4 \overset{\rightarrow}{=}_R x_2 \wedge x_2 \doteq 0 \wedge 0 \doteq_R y) \\
& \vee(x \doteq g(x_5) \wedge x_5 \overset{\rightarrow}{=}_R 0 \wedge g(x_0) \overset{\rightarrow}{=}_R g(x_2) \wedge x_2 \doteq 0 \wedge 0 \doteq_R y) \\
& \vee(x \doteq f(0) \wedge 0 \overset{\rightarrow}{=}_R x_2 \wedge x_2 \doteq 0 \wedge 0 \doteq_R y) \\
& \vee(x \doteq 0 \wedge y \doteq 0)
\end{array}$$

$$\begin{array}{ll}
\longrightarrow_{Sur} & (x \doteq g(x_3) \wedge x_3 \doteq x_1 \wedge x_1 \doteq y) \\
& \vee(x \doteq h(x_4) \wedge x_4 \doteq x_2 \wedge x_2 \doteq 0 \wedge y \doteq 0)
\end{array}$$

$\vee(x \doteq g(x_5) \wedge x_5 \doteq 0 \wedge 0 \overset{?}{=}_R x_2 \wedge x_2 \doteq 0 \wedge y \doteq 0)$
$\vee(x \doteq f(0) \wedge x_2 \doteq 0 \wedge x_2 \doteq 0 \wedge y \doteq 0)$
$\vee(x \doteq 0 \wedge y \doteq 0)$
$\longrightarrow_{Sur-Simp} (x \doteq g(y) \wedge x_3 \doteq x_1 \wedge x_1 \doteq y)$
$\vee(F)$
$\vee(x \doteq g(0) \wedge y \doteq 0 \wedge x_5 \doteq 0 \wedge x_2 \doteq 0)$
$\vee(x \doteq f(0) \wedge y \doteq 0 \wedge x_2 \doteq 0)$
$\vee(x \doteq 0 \wedge y \doteq 0)$

Le deuxième facteur conjonctif est transformé en F car la résolution donne $x \doteq h(0)$, mais comme $h(0)$ n'est pas en forme normale, cette équation est transformée en F. On obtient bien toutes les solutions et notre procédure termine, ce qui n'est pas le cas de la surréduction ordinaire même avec les optimisations basique de gauche à droite, car on aura toujours la branche infinie suivante :

$$f(f(x)) \doteq_R g(y) \rightsquigarrow_{[r_3,\{x \leftarrow k(x_1)\}]} f(k(x_1)) \doteq_R g(y) \rightarrow_{[r_3]} k(x_1) \doteq_R g(y)$$
$$\rightsquigarrow_{[r_4,\{x_1 \leftarrow k(x_2)\}]} k(x_2) \doteq_R g(y) \rightsquigarrow_{[r_4,\{x_2 \leftarrow k(x_3)\}]} \cdots$$

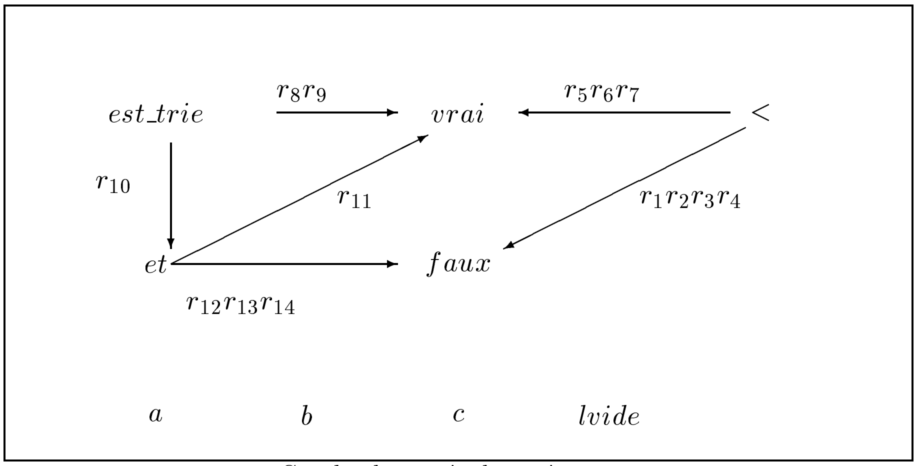
Regardons maintenant un second exemple où l'équation à résoudre n'a pas de solution malgré la présence d'un chemin dans le graphe et où la surréduction ordinaire ne termine pas alors que notre procédure de surréduction termine. Soit R le système de réécriture canonique suivant :

$$\begin{array}{lcrcl}
(r_1) & : & x < x & \rightarrow & faux \\
(r_2) & : & b < a & \rightarrow & faux \\
(r_3) & : & c < a & \rightarrow & faux \\
(r_4) & : & c < b & \rightarrow & faux \\
(r_5) & : & a < b & \rightarrow & vrai \\
(r_6) & : & a < c & \rightarrow & vrai \\
(r_7) & : & b < c & \rightarrow & vrai \\
(r_8) & : & est_trie(lvide) & \rightarrow & vrai \\
(r_9) & : & est_trie(adjt(x, lvide)) & \rightarrow & vrai \\
(r_{10}) & : & est_trie(adjt(x, adjt(y, z))) & \rightarrow & (x < y) \ et \ est_trie(adjt(y, z)) \\
(r_{11}) & : & vrai \ et \ vrai & \rightarrow & vrai \\
(r_{12}) & : & vrai \ et \ faux & \rightarrow & faux \\
(r_{13}) & : & faux \ et \ vrai & \rightarrow & faux \\
(r_{14}) & : & faux \ et \ faux & \rightarrow & faux
\end{array}$$

On exécute notre procédure de surréduction avec l'unificande $S : est\_trie(adjt(x, adjt(a, z))) \doteq_R vrai$. On veut connaître les listes triées qui ont comme second élément un a, il n'en existe pas car a est le plus petit élément. Par surréduction ordinaire on obtient la branche infinie suivante :

$$est\_trie(adjt(x, adjt(a, z)) \doteq_R vrai \rightarrow_{[r_{10}]} (x < a) \ et \ est\_trie(adjt(a, z)) \doteq_R vrai$$
$$\rightsquigarrow_{[r_{10},\{z \leftarrow adjt(y_1, z_1)\}]} (x < a) \ et \ ((a < y_1) \ et \ est\_trie(adjt(y_1, z_1)) \doteq_R vrai$$
$$\rightsquigarrow_{[r_{10},\{z_1 \leftarrow adjt(y_2, z_2)\}]} \cdots$$

La construction du graphe des opérateurs donne la figure suivante :



Graphe de portée des opérateurs.

Décrivons maintenant l'exécution pas à pas de notre procédure :

$S = est\_trie(adjt(x, adjt(a, z))) \doteq_R vrai$

$\longrightarrow_{Sur}$ $(adjt(x, adjt(a, z)) \stackrel{\to}{=}_R lvide \wedge true \doteq_R true)$
$\quad \vee(adjt(x, adjt(a, z)) \stackrel{\to}{=}_R adjt(x_1, lvide) \wedge true \doteq_R true)$
$\quad \vee(adjt(x, adjt(a, z)) \stackrel{\to}{=}_R adjt(x_2, adjt(y_2, z_2)) \wedge (x_2 < y_2)\ et\ est\_trie(adjt(y_2, z_2)) \doteq_R true)$

$\longrightarrow_{Simp}$ $(adjt(x, adjt(a, z)) \stackrel{\to}{=}_R lvide)$
$\quad \vee(adjt(x, adjt(a, z)) \stackrel{\to}{=}_R adjt(x_1, lvide))$
$\quad \vee(adjt(x, adjt(a, z)) \stackrel{\to}{=}_R adjt(x_2, adjt(y_2, z_2)) \wedge (x_2 < y_2)\ et\ est\_trie(adjt(y_2, z_2)) \doteq_R true)$

$\longrightarrow_{Sur}$ (F)
$\quad \vee(x \stackrel{\to}{=}_R x_1 \wedge adjt(a, z) \stackrel{\to}{=}_R lvide)$
$\quad \vee(x \stackrel{\to}{=}_R x_2 \wedge adjt(a, z) \stackrel{\to}{=}_R adjt(y_2, z_2) \wedge (x_2 < y_2) \stackrel{\to}{=}_R vrai$
$\quad\quad \wedge est\_trie(adjt(y_2, z_2)) \stackrel{\to}{=}_R true \wedge true \doteq_R true)$

$\longrightarrow_{Simp}$ $(x \stackrel{\to}{=}_R x_1 \wedge adjt(a, z) \stackrel{\to}{=}_R lvide)$
$\quad \vee(x \stackrel{\to}{=}_R x_2 \wedge adjt(a, z) \stackrel{\to}{=}_R adjt(y_2, z_2) \wedge (x_2 < y_2) \stackrel{\to}{=}_R vrai \wedge est\_trie(adjt(y_2, z_2)) \stackrel{\to}{=}_R true)$

$\longrightarrow_{Sur}$ $(x \doteq x_1 \wedge F)$
$\quad \vee(x \doteq x_2 \wedge a \stackrel{\to}{=}_R y_2 \wedge z \stackrel{\to}{=}_R z_2$
$\quad\quad \wedge((x_2 \stackrel{\to}{=}_R a \wedge y_2 \stackrel{\to}{=}_R b \wedge vrai \stackrel{\to}{=}_R vrai)$
$\quad\quad\quad \vee(x_2 \stackrel{\to}{=}_R a \wedge y_2 \stackrel{\to}{=}_R c \wedge vrai \stackrel{\to}{=}_R vrai)$
$\quad\quad\quad \vee(x_2 \stackrel{\to}{=}_R b \wedge y_2 \stackrel{\to}{=}_R c \wedge vrai \stackrel{\to}{=}_R vrai)) \wedge ((adjt(y_2, z_2)) \stackrel{\to}{=}_R lvide \wedge vrai \stackrel{\to}{=}_R vrai)$
$\quad\quad\quad \vee(adjt(y_2, z_2)) \stackrel{\to}{=}_R adjt(x_3, lvide) \wedge vrai \stackrel{\to}{=}_R vrai)$
$\quad\quad\quad \vee(adjt(y_2, z_2)) \stackrel{\to}{=}_R adjt(x_4, adjt(y_4, z_4)) \wedge (x_4 < y_4)\ et\ est\_trie(adjt(y_4, z_4)) \stackrel{\to}{=}_R vrai))$

Une étape de simplification supprime les équations $vrai \stackrel{\to}{=}_R vrai$ et le premier facteur conjonctif.

$\longrightarrow_{Sur}$ $(x \doteq x_2 \wedge y_2 \doteq a \wedge z \doteq z_2 \wedge ((x_2 \doteq a \wedge y_2 \doteq b) \vee (x_2 \doteq a \wedge y_2 \doteq c) \vee (x_2 \doteq b \wedge y_2 \doteq c)) \wedge B$

où B est le résultat de l'application de surréduction sur le dernier facteur conjonctif.

$\longrightarrow_{Dis}$ $(x \doteq x_2 \wedge y_2 \doteq a \wedge z \doteq z_2 \wedge x_2 \doteq a \wedge y_2 \doteq b \wedge B)$
$\quad \vee(x \doteq x_2 \wedge y_2 \doteq a \wedge z \doteq z_2 \wedge x_2 \doteq a \wedge y_2 \doteq c \wedge B)$
$\quad \vee(x \doteq x_2 \wedge y_2 \doteq a \wedge z \doteq z_2 \wedge x_2 \doteq b \wedge y_2 \doteq c \wedge B)$

$\longrightarrow_{Simp}$ $F \wedge F \wedge F$

Notre procédure termine. Sur cet exemple, le graphe d'opérateurs permet d'exploiter le fait que pour résoudre $t_1\ et\ t_2 \doteq_R vrai$, il faut appliquer r_{11}, c'est-à-dire surréduire t_1 et t_2 en $vrai$.

Le graphe d'opérateurs est donc un moyen pour faire terminer plus souvent la procédure d'unification dans les systèmes de réécriture canoniques. Nous allons voir maintenant un moyen plus performant pour la terminaison, puisque nous n'allons plus nous contenter d'observer les chemins reliant les symboles de tête des termes, mais nous allons utiliser les chemins reliant les termes entre eux, et ceci grâce à un graphe de termes.

3.2 Utilisation d'un graphe de termes

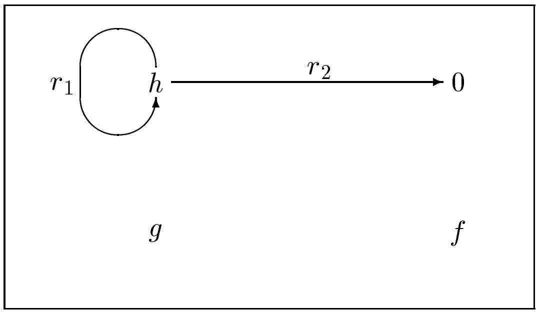
Nous allons commencer en présentant un petit exemple qui montre les améliorations que l'on obtient avec un graphe de termes. Soit R le système de réécriture canonique suivant :

$$\begin{array}{lcrcl} (r_1) & : & h(f(g(y))) & \rightarrow & h(f(y)) \\ (r_2) & : & h(0) & \rightarrow & 0 \end{array}$$

et essayons de résoudre l'équation $h(x) \doteq_R 0$. En utilisant la surréduction basique, on obtient deux branches :

$$h(x) \doteq_R 0 \rightsquigarrow_{[r_1,\{x \leftarrow f(g(x_1))\}]} h(f(x_1)) \doteq_R 0 \rightsquigarrow_{[r_1,\{x_1 \leftarrow g(x_2)\}]} h(f(x_2)) \doteq_R 0 \rightsquigarrow \ldots$$
$$\rightsquigarrow_{[r_2,\{x \leftarrow 0\}]} 0 \doteq_R 0$$

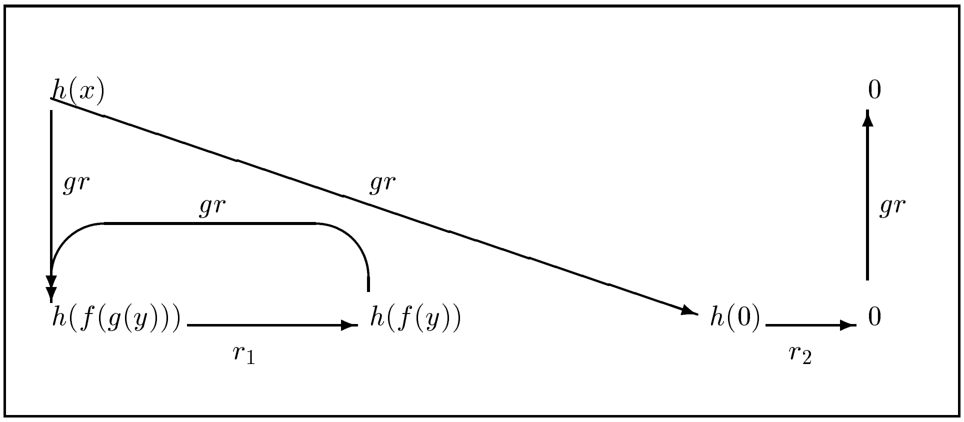
La première branche ne terminera pas et ne donnera pas de solution alors que la deuxième donne l'unique solution modulo R de notre équation, à savoir $\{x \leftarrow 0\}$. La première branche est donc inutile, mais le graphe des opérateurs présenté dans la première partie de ce chapitre ne nous permet pas de supprimer cette branche. Le graphe des opérateurs est le suivant :



Graphe de portée des opérateurs.

Ce graphe nous permet d'appliquer r_1 autant de fois que l'on veut puisqu'il existera toujours un chemin pour aller de h à 0.

Notre idée a donc été de ne plus se contenter des symboles de tête des termes, mais de regarder les termes dans leur ensemble. Pour cela nous construisons un graphe de termes où les nœuds sont les côtés gauches et les côtés droits des règles se trouvant dans le système de réécriture ainsi que les deux côtés de l'équation à résoudre. Les arcs sont les flèches de réécriture $\rightarrow$ et des flèches $\xrightarrow{gr}$ entre deux termes unifiables syntaxiquement, les variables étant implicitement renommées. Ainsi sur notre exemple, on obtient le graphe de termes suivant :



Graphe de termes.

Maintenant, comme pour le graphe des opérateurs, on n'applique que les règles qui apparaissent sur les chemins allant de $h(x)$ à 0. Ici nous n'en avons qu'un, et il utilise r_2, nous allons donc obtenir la solution $\{x \leftarrow 0\}$ et la procédure terminera. En général, le graphe de termes est plus compliqué car des arcs peuvent être créés entre des sous-termes, et les deux côtés de l'équation à résoudre peuvent être surréduits.

Après avoir présenté, de façon plus complète, le graphe de termes avec un algorithme de construction, nous donnerons le système d'inférence qui nous permettra d'obtenir une procédure d'unification dans les systèmes de réécriture canoniques. Nous terminerons en détaillant un exemple.

3.2.1 Construction d'un graphe de termes

Dans ce paragraphe, plusieurs graphes de termes seront dessinés mais certains arcs manqueront pour que les figures ne soient pas trop chargées.

Soit une chaîne de surréductions basiques issue d'un terme t_0. Dans chaque étape de surréduction $t_i \rightsquigarrow_{[u_i, l_i \rightarrow r_i, \sigma_i]} t_{i+1}$, l'occurrence u_i ne doit pas appartenir à la partie apportée par une substitution dans une étape précédente. De plus $t_i|_{u_i}$ et le côté gauche de la règle l_i sont unifiables. Donc, nous pouvons schématiser cette chaîne de surréduction basique en ne considérant que t_0 et les deux côtés des règles de réécriture. Ceux-ci sont vus comme les nœuds du graphe, les arcs sont les flèches de réécriture et les unifications syntaxiques possibles. La partie apportée par les substitutions n'apparaît pas dans le graphe. Ce graphe est bien évidemment fini puisque le système de réécriture est fini.

Par exemple, avec la règle de réécriture $(r_1)\ :\ h(x, g(y)) \rightarrow i(x, y)$, l'étape de surréduction $h(f(x_1), y_1) \rightsquigarrow_{[\epsilon, r_1, \{x \leftarrow f(x_1), y_1 \leftarrow g(y)\}]} i(f(x_1), y)$ est résumée par le chemin suivant dans le graphe de termes $h(f(x_1), y_1) \xrightarrow{gr} h(x, g(y)) \rightarrow_{r_1} i(x, y)$. La flèche $\xrightarrow{gr}$ nous dit que deux termes sont unifiables. Puisque la substitution n'est pas appliquée sur les termes du graphe, le dernier terme de l'étape de surréduction $i(f(x_1), y)$ est une instance du terme $i(x, y)$ qui est dans le graphe.

Inversement si le graphe contient $\xrightarrow{gr}$ chaque fois que deux termes sont unifiables, alors il y a un chemin dans le graphe pour chaque chaîne de surréduction basique issue de t_0. C'est ainsi qu'en considérant tous les chemins qui aboutissent à un terme s du graphe, nous connaissons toutes les chaînes de surréduction basique issues de t_0 et pouvant aboutir à une instance de s.

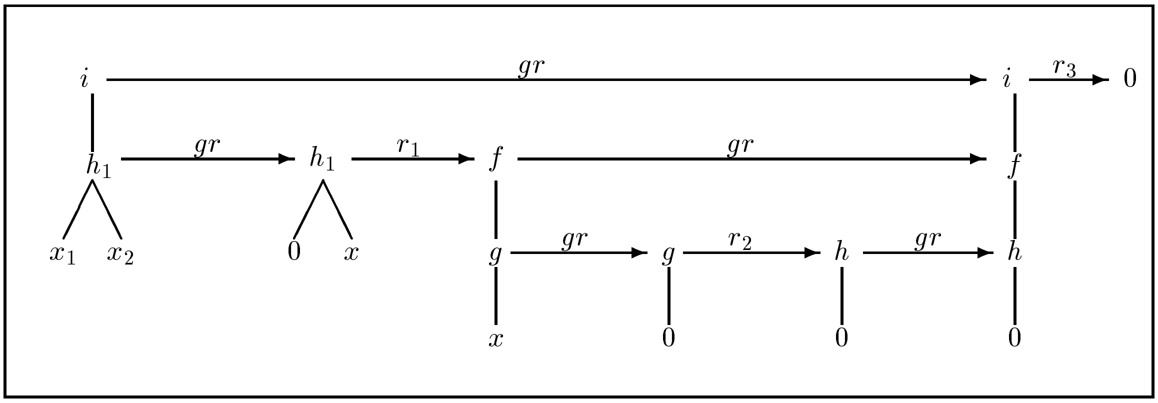
Cependant certaines étapes de surréduction peuvent être appliquées en sous-terme. Par exemple avec le système de réécriture suivant :

$$\begin{array}{rcrcl}(r_1) & : & h_1(0,x) & \rightarrow & f(g(x)) \\ (r_2) & : & g(0) & \rightarrow & h(0) \\ (r_3) & : & i(f(h(0))) & \rightarrow & 0\end{array}$$

Nous avons la chaîne de surréduction basique :

$$i(h_1(x_1,x_2)) \rightsquigarrow_{[1,r_1,\{x_1\leftarrow 0,x_2\leftarrow x\}]} i(f(g(x))) \rightsquigarrow_{[2,r_2,\{x\leftarrow 0\}]} i(f(h(0))) \rightsquigarrow_{[\epsilon,r_3,Id]} 0$$

Les deux premières étapes sont appliquées en sous-termes. Pour obtenir un chemin en tête de $i(h_1(x_1,x_2))$ à 0, on commence par ajouter les flèches $\xrightarrow{gr}$ correspondantes aux unifications syntaxiques possibles, puis on ajoute la flèche $\xrightarrow{gr}$ entre $f(g(x))$ et $f(h(0))$, car il y a un chemin en sous-terme, et qui correspond à la seconde étape de surréduction, puis on fait de même de $i(h_1(x_1,x_2))$ à $i(f(h(0)))$. Cela ne veut pas dire que $i(h_1(x_1,x_2))$ et $i(f(h(0)))$ sont syntaxiquement unifiables, mais simplement que $i(h_1(x_1,x_2))$ peut être surréduit uniquement en sous-terme en une instance de $i(f(h(0)))$; la flèche $\xrightarrow{gr}$ a donc deux significations. On obtient finalement le graphe de termes suivant :



Graphe de termes.

Voyons maintenant un nouveau type d'arc $\xleftrightarrow{gr}$ qui nous permettra de compléter notre graphe. Les termes dans le graphe sont séparés en deux catégories, nous avons les termes représentant les côtés gauches des règles de réécriture, et les termes représentants les côtés droits. Les deux termes que nous voulons unifier font aussi partie du graphe, et sont considérés comme des côtés droits. L'arc $\xleftrightarrow{gr}$ sera utilisé pour relier deux termes, faisant partie de l'ensemble des côtés droits, qui pourront être surréduits, (peut être en 0 étape), uniquement en sous-terme, en deux termes syntaxiquement unifiables. Pour résoudre une équation $t \doteq_R t'$, on surréduit les deux côtés de l'équation jusqu'à ce que l'on obtienne l'équation $t_2 \doteq_R t'_2$, avec t_2 et t'_2 deux termes syntaxiquement unifiables. Les étapes de surréduction peuvent se schématiser de la façon suivante :

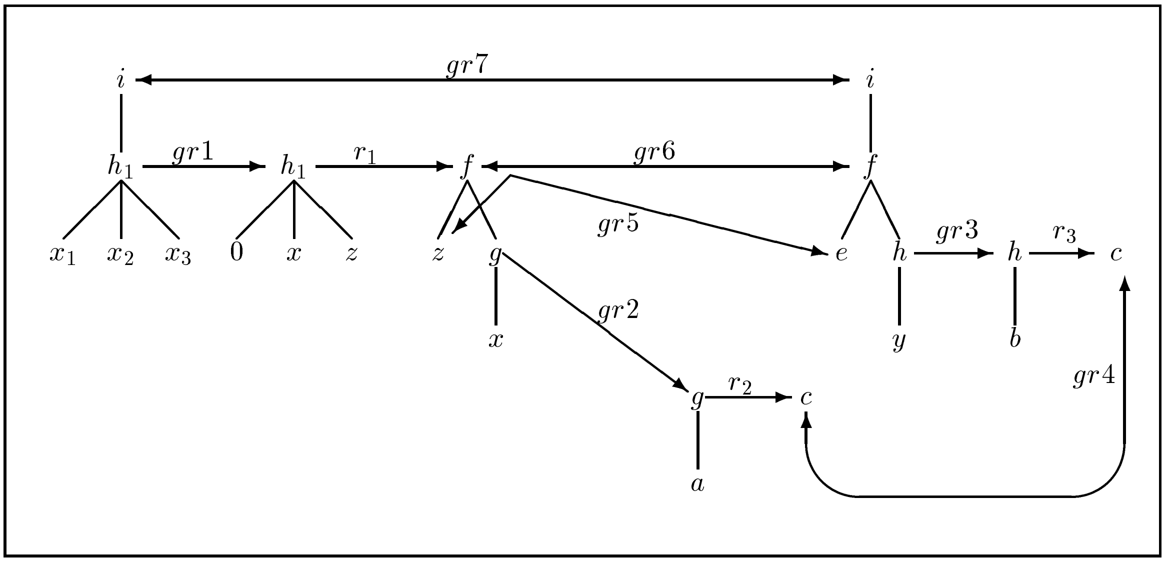
$$\begin{array}{c} t \rightsquigarrow^* \rightsquigarrow_{[\epsilon]} t_1 \rightsquigarrow^*_{[\neq\epsilon]} t_2 \\ t' \rightsquigarrow^* \rightsquigarrow_{[\epsilon]} t'_1 \rightsquigarrow^*_{[\neq\epsilon]} t'_2 \end{array}$$

Le terme t est transformé en t_1 par une chaîne de surréduction dont la dernière étape est en tête, et t_1 est surréduit, uniquement en sous-terme, en t_2; de même pour le terme t'. Les termes t_1 et t'_1 sont des instances de termes, côtés droits du graphe; nous ajoutons dans le graphe une flèche $\xleftrightarrow{gr}$ entre ces deux termes. Nous pouvons remarquer que les flèches $\xrightarrow{gr}$ vont d'un côté droit vers un côté gauche, comme les flèches de réécriture, alors que les flèches $\xleftrightarrow{gr}$ relient ensemble deux côtés droits. Présentons maintenant un nouvel exemple sur lequel on voit que l'addition de flèches $\xleftrightarrow{gr}$ peut engendrer la création de nouvelles flèches.

Exemple 3.11 Soit R le système de réécriture canonique suivant :

$$\begin{array}{lcrcl} (r_1) & : & h_1(0,x,z) & \rightarrow & f(z,g(x)) \\ (r_2) & : & g(a) & \rightarrow & c \\ (r_3) & : & h(b) & \rightarrow & c \end{array}$$

Voyons en détail la construction du graphe de termes lorsque nous voulons résoudre dans R l'équation $i(h_1(x_1,x_2,x_3)) \stackrel{?}{=}_R i(f(e,h(y)))$.



Graphe de termes

Les flèches $\xrightarrow{gr}$ sont ici numérotées dans l'ordre d'apparition. Nous mettons en premier 1, 2, 3, 4, et 5 qui correspondent à des unifications syntaxiques, puis nous en déduisons 6 et ensuite 7.

Voyons maintenant la procédure de construction d'un graphe de termes. Elle est décrite par un système d'inférence.

3.2.2 L'algorithme de construction d'un graphe de termes

Cet algorithme est dans sa forme la plus simple, il pourrait être optimisé en évitant d'introduire des arcs inutiles. G représentera le graphe courant, l est un terme ou sous-terme d'un membre gauche d'une règle, r et r' sont des termes ou sous-termes de membres droits de règles. l, r et r' sont dans le graphe et on suppose de plus que r et r' ne font pas partie d'un même terme. On notera $\xrightarrow{*}$ pour un chemin dans le graphe qui peut contenir des flèches $\xrightarrow{gr}$ et des flèches de réécriture toutes dans le même sens, $\xleftrightarrow{gr}^?$ représente zéro ou une étape de $\xleftrightarrow{gr}$. Voici maintenant les deux règles d'inférence :

(Add $\xrightarrow{gr}$)

$$\frac{G}{G \cup \{r \xrightarrow{gr} l\}}$$

si r est une variable ou l est une variable
ou $r = f(r_1, ..., r_n)$, $l = f(l_1, ..., l_n)$ et $\forall i = 1..n$,
$r_i \xrightarrow{*} l_i \in G$.

(Add $\overset{gr}{\longleftrightarrow}$)

$$\frac{G}{G \cup \{r \overset{gr}{\longleftrightarrow} r'\}} \quad \begin{array}{l} \text{si } r \text{ est une variable ou } r' \text{ est une variable} \\ \text{ou } r = f(r_1, ..., r_n),\ r' = f(r'_1, ..., r'_n) \text{ et } \forall i = 1..n, \\ \exists t_i, t'_i \text{ tel que } r_i \overset{*}{\longrightarrow} t_i \overset{gr}{\longleftrightarrow}^? t'_i \overset{*}{\longleftarrow} r'_i \in G. \end{array}$$

Voici l'algorithme de construction utilisé. Soit t et t' les deux termes à unifier modulo un système de réécriture canonique R. La construction se fait en deux étapes :

1. **Initialisation** $G \leftarrow R \cup \{t, t'$ considérés par convention comme des côtés droits $\}$. On met dans le graphe tous les termes qui apparaissent dans le système de réécriture ainsi que les flèches de réécriture et on ajoute les deux termes à unifier.

2. **Addition des flèches** on applique les deux règles d'inférence tant que l'on ajoute de nouvelles flèches.

Cet algorithme termine puisque le graphe contient un nombre fini de termes (et de sous-termes) et donc il ne peut contenir qu'un nombre fini de flèches (R est fini).

Ici le graphe dépend des termes à unifier. Si on veut unifier plusieurs paires de termes modulo un même système de réécriture, on peut éviter de reconstruire l'ensemble du graphe à chaque unification en procédant de la façon suivante :

1- Initialiser le graphe avec R et ajouter toutes les flèches possibles.

2- Ajouter t et t', deux termes que l'on veut unifier, dans le graphe ainsi que toutes les nouvelles flèches entre t et t' et les autres termes ou sous-termes du graphe.

3- Quand l'unification est finie, supprimer t et t' du graphe ainsi que toutes les flèches qui partaient ou arrivaient à l'un de ces deux termes et repartir au point 2 avec deux nouveaux termes à unifier.

Nous disposons donc maintenant d'un graphe de termes, il ne nous reste plus qu'à définir la procédure d'unification qui intègre un tel graphe.

3.2.3 La procédure de surréduction utilisant un graphe de termes

Le formalisme utilisé pour décrire cette procédure est identique à celui utilisé dans la première partie de ce chapitre. Nous disposons des 5 mêmes types d'équations et nous travaillons sur des unificandes. Nous avons de plus la propriété suivante :

- dans une équation $t \doteq_R t'$, les termes t et t' sont des côtés droits ou des sous-termes de côté droit dans le graphe,

- dans une équation $t \stackrel{\rightarrow}{=}_R t'$, le terme t est un côté droit ou un sous-terme d'un côté droit et le terme t' est un côté gauche ou un sous-terme d'un côté gauche dans le graphe.

Rappelons que dans les règles d'inférence qui vont suivre, x et y représentent des variables, s et t sont des termes non variables, $s_1, .., s_n, t_1, .., t_n$ sont les sous-termes de niveau un de s et t, et $g \rightarrow d$ est une règle du système de réécriture R. La notation $\overset{gr}{\longleftrightarrow}^?$ signifie zéro ou une étape $\overset{gr}{\longleftrightarrow}$ dans le graphe et $\overset{*}{\longrightarrow}$ représente un chemin pouvant contenir des flèches $\overset{gr}{\longrightarrow}$ et des flèches de réécriture. Le symbole $\bigvee$ signifie qu'il y a plusieurs facteurs conjonctifs s'il existe plusieurs chemins dans le graphe satisfaisant la condition. Si la règle d'inférence crée un facteur conjonctif vide, il est remplacé par T, si en revanche aucune condition n'est satisfiable, elle crée donc un facteur disjonctif vide, celui-ci est remplacé par F.

Les règles d'inférence diffèrent des règles utilisées pour le graphe d'opérateurs uniquement au niveau des conditions.

(Surréduction$\doteq_R$)

$$
\begin{array}{lll}
x \doteq_R t \longrightarrow & (x \doteq t) & \\
t \doteq_R x & \bigvee t|_u \doteq g \wedge t[u \leftarrow d] \doteq_R x & \text{si } u \in \overline{O}(t) \text{ et } t\hat{e}te(t|_u) = t\hat{e}te(g) \\
 & & \\
x \doteq_R y \longrightarrow & x \doteq y & \\
 & & \\
s \doteq_R t \longrightarrow & s_1 \doteq_R t_1 \wedge ... \wedge s_n \doteq_R t_n & \text{si } s \overset{gr}{\longleftrightarrow} t \\
 & \bigvee s_1 \stackrel{\rightarrow}{=}_R g_1 \wedge ... \wedge s_n \stackrel{\rightarrow}{=}_R g_n \wedge d \doteq_R t & \text{si } s \overset{gr}{\longrightarrow} g \rightarrow d \overset{*}{\longrightarrow} \overset{gr}{\longleftrightarrow}{}^{?} \overset{*}{\longleftarrow} t \\
 & \bigvee t_1 \stackrel{\rightarrow}{=}_R g_1 \wedge ... \wedge t_n \stackrel{\rightarrow}{=}_R g_n \wedge s \doteq_R d & \text{si } s \overset{gr}{\longleftrightarrow}{}^{?} \overset{*}{\longleftarrow} d \leftarrow g \overset{gr}{\longleftarrow} t
\end{array}
$$

(Surréduction $\stackrel{\rightarrow}{=}_R$)

$$
\begin{array}{lll}
x \stackrel{\rightarrow}{=}_R t \longrightarrow & x \doteq t & \\
 & & \\
x \stackrel{\rightarrow}{=}_R y \longrightarrow & x \doteq y & \\
 & & \\
s \stackrel{\rightarrow}{=}_R y \longrightarrow & y \doteq s & \\
 & \bigvee s|_u \doteq g \wedge s[u \leftarrow d] \stackrel{\rightarrow}{=}_R y & \text{si } u \in \overline{O}(s) \text{ et } t\hat{e}te(s|_u) = t\hat{e}te(g) \\
 & & \\
s \stackrel{\rightarrow}{=}_R t \longrightarrow & s_1 \stackrel{\rightarrow}{=}_R t_1 \wedge ... \wedge s_n \stackrel{\rightarrow}{=}_R t_n & \text{si } s \overset{gr}{\longrightarrow} t \\
 & \bigvee s_1 \stackrel{\rightarrow}{=}_R g_1 \wedge ... \wedge s_n \stackrel{\rightarrow}{=}_R g_n \wedge d \stackrel{\rightarrow}{=}_R t & \text{si } s \overset{gr}{\longrightarrow} g \rightarrow d \overset{*}{\longrightarrow} t
\end{array}
$$

Les conditions signifient bien-sûr que le chemin décrit doit être dans le graphe. Ces deux règles sont accompagnées des mêmes règles de simplification que pour la procédure utilisant un graphe d'opérateurs, (le graphe n'intervenant pas pour ces règles). Nous avons la même procédure d'équité (définition3.8) et les mêmes théorèmes de correction et de complétude.

Théorème 3.12 (Correction) *Soient S_0 un unificande, $S_0 \overset{*}{\longrightarrow} \ldots \overset{*}{\longrightarrow} S_n \overset{*}{\longrightarrow} \ldots$ une dérivation équitable issue de S_0 et $\theta_n \in SOL(S_n)$ alors $\theta_n \in SOL(S_0)$.*

Théorème 3.13 (Complétude) *Soient S_0 un unificande, $S_0 \overset{*}{\longrightarrow} \ldots \overset{*}{\longrightarrow} S_n \overset{*}{\longrightarrow} \ldots$ une dérivation équitable issue de S_0 où $S_1, .., S_n, ..$ sont des unificandes en forme disjonctive obtenus après un certain nombre d'étapes de la procédure d'équité, et $\theta_0 \in SOL(S_0)$. Alors il existe un unificande S_i dans cette dérivation et une substitution θ_i tels que*

- *θ_i est solution d'un facteur conjonctif de S_i qui ne contient pas d'équation en $\doteq_R$ ou $\stackrel{\rightarrow}{=}_R$,*
- *$\theta_i = \theta_0[(S_0)]$*

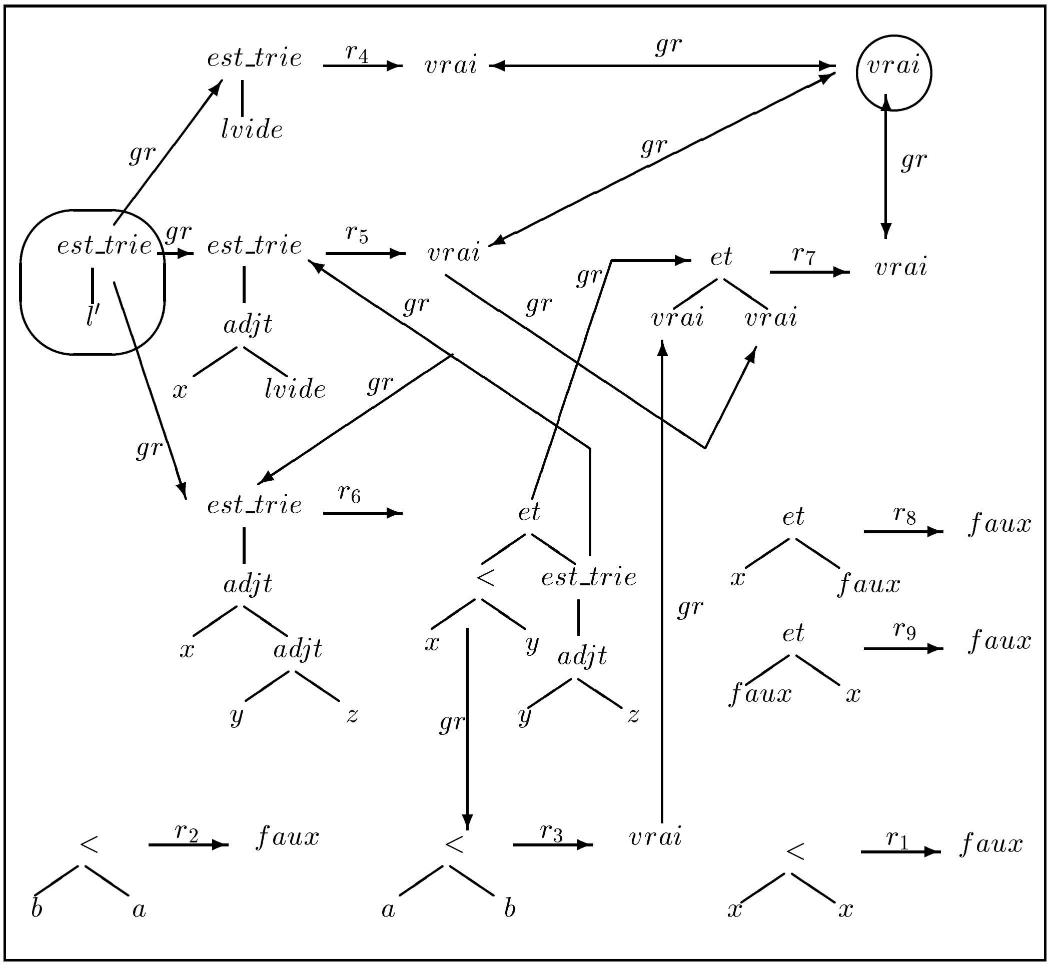
Les preuves de ces deux théorèmes, ainsi que les preuves de correction et de complétude des règles d'inférence, sont présentées en annexe (qui de fait donne les preuves validant notre approche, présentée au congrès RTA'93). Dans cette annexe nous avons aussi toute une partie prouvant les propriétés de notre graphe de termes. Détaillons maintenant cette procédure sur un exemple.

3.2.4 Exemple d'unification par surréduction dirigée par un graphe de termes

Nous allons reprendre ici l'exemple des listes triées mais nous allons nous limiter à deux constantes pour pouvoir détailler une exécution complète de notre procédure. Soit le système de réécriture suivant :

$$
\begin{array}{lcrcl}
(r_1) & : & x < x & \rightarrow & faux \\
(r_2) & : & b < a & \rightarrow & faux \\
(r_3) & : & a < b & \rightarrow & vrai \\
(r_4) & : & est\_trie(lvide) & \rightarrow & vrai \\
(r_5) & : & est\_trie(adjt(x, lvide)) & \rightarrow & vrai \\
(r_6) & : & est\_trie(adjt(x, adjt(y, z))) & \rightarrow & (x < y) \ et \ est\_trie(adjt(y, z)) \\
(r_7) & : & vrai \ et \ vrai & \rightarrow & vrai \\
(r_8) & : & x \ et \ faux & \rightarrow & faux \\
(r_9) & : & faux \ et \ x & \rightarrow & faux
\end{array}
$$

Et nous voulons connaître toutes les listes triées, c'est-à-dire trouver les solutions de l'équation $est\_trie(l') \doteq_R vrai$ (où l' est une variable). On obtient le graphe suivant où les deux termes à unifier sont encerclés. Nous avons indiqué uniquement les flèches utiles.



Graphe de termes

L'exécution de notre procédure donne après chaque étape de surréduction ou de simplification les unificandes suivants :

1. Après surréduction

$$
(((l' \stackrel{\rightarrow}{=}_R lvide) \wedge (vrai \doteq_R vrai)) \vee ((l' \stackrel{\rightarrow}{=}_R adjt(x_1, lvide)) \wedge (vrai \doteq_R vrai))
$$
$$
\vee((l' \stackrel{\rightarrow}{=}_R adjt(x_2', adjt(x_2, l))) \wedge ((x_2' < x_2) \ et \ est\_trie(adjt(x_2, l))) \doteq_R vrai)))
$$

2. Après simplification

$$
((l' \stackrel{\rightarrow}{=}_R lvide) \vee (l' \stackrel{\rightarrow}{=}_R adjt(x_1, lvide))
$$
$$
\vee((l' \stackrel{\rightarrow}{=}_R adjt(x_2', adjt(x_2, l_2))) \wedge ((x_2' < x_2) \ et \ est\_trie(adjt(x_2, l_2))) \doteq_R vrai)))
$$

3. Après surréduction

$$((l' \doteq lvide) \vee (l' \doteq adjt(x_1, lvide)) \vee ((l' \doteq adjt(x'_2, adjt(x_2, l_2))) \wedge ((x'_2 < x_2) \overset{\rightarrow}{=}_R vrai) \wedge$$
$$(est\_trie(adjt(x_2, l_2))) \overset{\rightarrow}{=}_R vrai) \wedge (vrai \doteq_R vrai)))$$

4. Après simplification

$$((l' \doteq lvide) \vee (l' \doteq adjt(x_1, lvide)) \vee ((l' \doteq adjt(x'_2, adjt(x_2, l_2))) \wedge ((x'_2 < x_2) \overset{\rightarrow}{=}_R vrai) \wedge$$
$$(est\_trie(adjt(x_2, l_2))) \overset{\rightarrow}{=}_R vrai)))$$

5. Après surréduction

$$((l' \doteq lvide) \vee (l' \doteq adjt(x_1, lvide))$$
$$\vee((l' \doteq adjt(x'_2, adjt(x_2, l_2))) \wedge (x'_2 \overset{\rightarrow}{=}_R a) \wedge (x_2 \overset{\rightarrow}{=}_R b) \wedge (vrai \overset{\rightarrow}{=}_R vrai)$$
$$\wedge(((adjt(x_2, l_2) \overset{\rightarrow}{=}_R adjt(x_3, lvide)) \wedge (vrai \overset{\rightarrow}{=}_R vrai))$$
$$\vee((adjt(x_2, l_2) \overset{\rightarrow}{=}_R adjt(x'_4, adjt(x_4, l_4))) \wedge ((x'_4 < x_4) \; et \; est\_trie(adjt(x_4, l_4))) \overset{\rightarrow}{=}_R vrai))))$$

6. Après simplification et distribution

$$((l' \doteq lvide) \vee (l' \doteq adjt(x_1, lvide))$$
$$\vee((l' \doteq adjt(x'_2, adjt(x_2, l_2))) \wedge (x'_2 \overset{\rightarrow}{=}_R a) \wedge (x_2 \overset{\rightarrow}{=}_R b) \wedge (adjt(x_2, l_2) \overset{\rightarrow}{=}_R adjt(x_3, lvide)))$$
$$\vee((l' \doteq adjt(x'_2, adjt(x_2, l_2))) \wedge (x'_2 \overset{\rightarrow}{=}_R a) \wedge (x_2 \overset{\rightarrow}{=}_R b) \wedge (adjt(x_2, l_2) \overset{\rightarrow}{=}_R adjt(x'_4, adjt(x_4, l_4)))$$
$$\wedge((x'_4 < x_4) \; et \; est\_trie(adjt(x_4, l_4))) \overset{\rightarrow}{=}_R vrai))))$$

7. Après surréduction

$$((l' \doteq lvide) \vee (l' \doteq adjt(x_1, lvide))$$
$$\vee((l' \doteq adjt(x'_2, adjt(x_2, l_2))) \wedge (x'_2 \doteq a) \wedge (x_2 \doteq b) \wedge (x_2 \overset{\rightarrow}{=}_R x_3) \wedge (l_2 \overset{\rightarrow}{=}_R lvide))$$
$$\vee((l' \doteq adjt(x'_2, adjt(x_2, l_2))) \wedge (x'_2 \doteq a) \wedge (x_2 \doteq b) \wedge (x_2 \overset{\rightarrow}{=}_R x'_4) \wedge (l_2 \overset{\rightarrow}{=}_R adjt(x_4, l_4))$$
$$\wedge((x'_4 < x_4) \overset{\rightarrow}{=}_R vrai) \wedge (est\_trie(adjt(x_4, l_4))) \overset{\rightarrow}{=}_R vrai) \wedge (vrai \overset{\rightarrow}{=}_R vrai)))$$

8. Après simplification

$$((l' \doteq lvide) \vee (l' \doteq adjt(x_1, lvide))$$
$$\vee((l' \doteq adjt(x'_2, adjt(x_2, l_2))) \wedge (x'_2 \doteq a) \wedge (x_2 \doteq b) \wedge (x_2 \overset{\rightarrow}{=}_R x_3) \wedge (l_2 \overset{\rightarrow}{=}_R lvide))$$
$$\vee((l' \doteq adjt(x'_2, adjt(x_2, l_2))) \wedge (x'_2 \doteq a) \wedge (x_2 \doteq b) \wedge (x_2 \overset{\rightarrow}{=}_R x'_4) \wedge (l_2 \overset{\rightarrow}{=}_R adjt(x_4, l_4))$$
$$\wedge((x'_4 < x_4) \overset{\rightarrow}{=}_R vrai) \wedge (est\_trie(adjt(x_4, l_4))) \overset{\rightarrow}{=}_R vrai)))$$

9. Après surréduction

$$((l' \doteq lvide) \vee (l' \doteq adjt(x_1, lvide))$$
$$\vee((l' \doteq adjt(x'_2, adjt(x_2, l_2))) \wedge (x'_2 \doteq a) \wedge (x_2 \doteq b) \wedge (x_2 \doteq x_3) \wedge (l_2 \doteq lvide))$$
$$\vee((l' \doteq adjt(x'_2, adjt(x_2, l_2))) \wedge (x'_2 \doteq a) \wedge (x_2 \doteq b) \wedge (x_2 \doteq x'_4)$$
$$\wedge(l_2 \doteq adjt(x_4, l_4)) \wedge (x'_4 \overset{\rightarrow}{=}_R a) \wedge (x_4 \overset{\rightarrow}{=}_R b) \wedge (vrai \overset{\rightarrow}{=}_R vrai)$$
$$\wedge(((adjt(x_4, l_4) \overset{\rightarrow}{=}_R adjt(x_5, lvide)) \wedge (vrai \overset{\rightarrow}{=}_R vrai))$$
$$\vee((adjt(x_4, l_4) \overset{\rightarrow}{=}_R adjt(x'_6, adjt(x_6, l_6))) \wedge ((x'_6 < x_6) \; et \; est\_trie(adjt(x_6, l_6)) \overset{\rightarrow}{=}_R vrai)))$$

10. Après simplification

$$((l' \doteq lvide) \vee (l' \doteq adjt(x_1, lvide))$$
$$\vee((l' \doteq adjt(x'_2, adjt(x_2, l_2))) \wedge (x'_2 \doteq a) \wedge (x_2 \doteq b) \wedge (x_3 \doteq b) \wedge (l_2 \doteq lvide))$$
$$\vee((l' \doteq adjt(x'_2, adjt(x_2, l_2))) \wedge (x'_2 \doteq a) \wedge (x_2 \doteq b) \wedge (x'_4 \doteq b) \wedge (l_2 \doteq adjt(x_4, l_4))$$
$$\wedge(x'_4 \overset{\rightarrow}{=}_R a) \wedge (x_4 \overset{\rightarrow}{=}_R b)$$
$$\wedge((adjt(x_4, l_4) \overset{\rightarrow}{=}_R adjt(x_5, lvide))$$
$$\vee((adjt(x_4, l_4) \overset{\rightarrow}{=}_R adjt(x'_6, adjt(x_6, l_6))) \wedge ((x'_6 < x_6) \; et \; est\_trie(adjt(x_6, l_6)) \overset{\rightarrow}{=}_R vrai)))$$

11. Après surréduction

$$((l' \doteq lvide) \vee (l' \doteq adjt(x_1, lvide))$$
$$\vee((l' \doteq adjt(x'_2, adjt(x_2, l_2))) \wedge (x'_2 \doteq a) \wedge (x_2 \doteq b) \wedge (x_3 \doteq b) \wedge (l_2 \doteq lvide))$$
$$\vee((l' \doteq adjt(x'_2, adjt(x_2, l_2))) \wedge (x'_2 \doteq a) \wedge (x_2 \doteq b) \wedge (x'_4 \doteq b) \wedge (l_2 \doteq adjt(x_4, l_4))$$
$$\wedge(x'_4 \doteq a) \wedge (x_4 \doteq b) \wedge (((x_4 \overset{\rightarrow}{=}_R x_5) \wedge (l_4 \overset{\rightarrow}{=}_R lvide))$$
$$\vee((x_4 \overset{\rightarrow}{=}_R x'_6) \wedge (l_4 \overset{\rightarrow}{=}_R adjt(x_6, l_6)) \wedge ((x'_6 < x_6) \overset{\rightarrow}{=}_R vrai) \wedge (est\_trie(adjt(x_6, l_6)) \overset{\rightarrow}{=}_R vrai)$$

$\wedge(vrai \stackrel{\rightarrow}{=}_R vrai)))$

12. Après simplification

$((l' \doteq lvide) \vee (l' \doteq adjt(x_1, lvide))$
$\vee((l' \doteq adjt(x'_2, adjt(x_2, l_2))) \wedge (x'_2 \doteq a) \wedge (x_2 \doteq b) \wedge (x_3 \doteq b) \wedge (l_2 \doteq lvide)))$

La procédure termine et on obtient trois solutions desquelles on peut déduire les quatre solutions closes de notre équation. Par surréduction ordinaire on obtient la branche infinie suivante :

$est\_trie(l') \rightsquigarrow_{[r6]} (x' < x)\ et\ est\_trie(adjt(x, l))$
$\rightsquigarrow_{[r6]} (x' < x)\ et\ ((x < x_1)\ et\ est\_trie(adjt(x_1, l_1)) \rightsquigarrow_{[r6]} \ldots$

Cette branche correspond à une surréduction basique de gauche à droite et chaque terme est en forme normale, ce qui montre bien que la terminaison est apportée par l'introduction d'un graphe de termes.

3.2.5 Les règles de fusion

Nous allons présenter ici une idée pouvant améliorer la méthode précédente au niveau des règles de simplifications [CR92, LIM92]; cette idée a été approfondie par S. Limet.

L'exemple précédent se déroule bien parce que la plupart des règles de réécriture ont des termes clos du côté droit. Si l'on remplace la règle (r_7) par $vrai\ et\ y \rightarrow y$, on obtient à la troisième étape le facteur conjonctif suivant :

$$(l' \doteq adjt(x'_2, adjt(x_2, l_2))) \wedge ((x'_2 < x_2) \stackrel{\rightarrow}{=}_R\ ) \wedge (est\_trie(adjt(x_2, l_2)) \stackrel{\rightarrow}{=}_R y) \wedge (y \doteq_R vrai)$$

Alors $y \doteq_R vrai$ est transformé en $y \doteq vrai$ et $(est\_trie(adjt(x_2, l_2)) \stackrel{\rightarrow}{=}_R y)$ peut être surréduit de façon infinie. Donc notre procédure ne termine pas. Pour pallier à ce défaut nous devons prendre en compte le fait que y est égal à $vrai$ et ainsi résoudre l'équation $(est\_trie(adjt(x_2, l_2)) \stackrel{\rightarrow}{=}_R vrai)$ qui termine.

Pour cela nous avons pensé à introduire des règles de fusion du type :

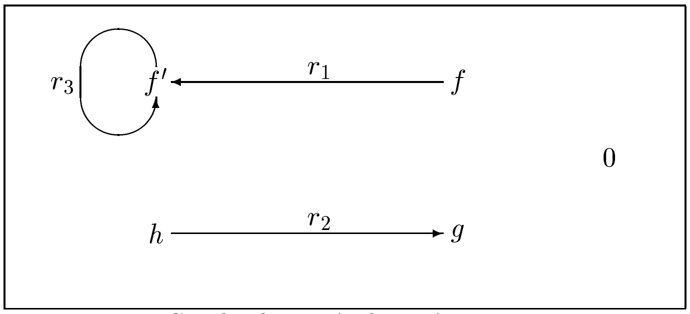
$$\begin{array}{rcl} s \stackrel{\rightarrow}{=}_R x \wedge x \doteq t & \longrightarrow & s \stackrel{\rightarrow}{=}_R t \wedge x \doteq t \\ s \doteq_R x \wedge x \doteq t & \longrightarrow & s \doteq_R t \wedge x \doteq t \\ s \stackrel{\rightarrow}{=}_R x \wedge x \doteq_R t & \longrightarrow & s \doteq_R t \wedge x \doteq_R t \\ & \ldots & \end{array}$$

Nous espérons ainsi que la méthode reste correcte, complète et termine encore plus souvent, aussi bien pour le graphe de termes que pour le graphe d'opérateurs.

Exemple 3.14 Soit le système de réécriture canonique R suivant :

$$\begin{array}{lcrcl} (r_1) & : & f(g(x)) & \rightarrow & f'(x) \\ (r_2) & : & h(0) & \rightarrow & g(0) \\ (r_3) & : & f'(g(x)) & \rightarrow & f'(f'(x)) \end{array}$$

Et essayons de trouver les solutions de l'équation $f(h(x)) \doteq_R f'(y)$. On obtient le graphe des opérateurs suivant :



Graphe de portée des opérateurs.

On voit que pour unifier un terme commençant par f et un terme commençant par f' il faut appliquer r_1. Pour cela il faut unifier $f(h(x))$ à $f(g(x))$, ceci a pour unique solution la substitution $\{x \leftarrow 0\}$ en utilisant r_2. Maintenant on peut appliquer r_1 et on obtient $f'(x) \doteq_R f'(y) \wedge x \doteq 0$. Une solution évidente est $y \doteq x$ et $x \doteq 0$ mais on peut encore appliquer la règle r_3 sur le membre droit de notre équation avec la substitution $\{y \leftarrow g(y_1)\}$, on obtient $f'(x) \doteq_R f'(f'(y_1)) \wedge x \doteq 0$. On ne peut plus appliquer de règle en tête. Les têtes étant identiques on va donc essayer d'unifier les fils et on obtient $x \doteq_R f'(y_1) \wedge x \doteq 0$ qui n'a pas de solution car 0 et f' ne sont pas joignables dans le graphe des opérateurs. Ceci peut être détecté uniquement si on applique une règle de fusion pour créer $0 \doteq_R f'(y)$, dans le cas contraire on boucle sans fin sur la règle r_3. Avec les règles de fusion cet exemple termine même avec une infinité de chemins reliant les symboles de tête.

Nous allons maintenant examiner un moyen d'introduire cette notion de graphe, dans la procédure d'unification générale décrite dans le chapitre précédent, pour améliorer l'efficacité des étapes de surréduction contrainte.

3.3 Utilisation d'un graphe dans des étapes de surréduction contrainte

Nous avons vu dans les sections précédentes, que l'on pouvait améliorer la terminaison et l'efficacité de la méthode d'unification par surréduction, en utilisant un graphe d'opérateurs ou un graphe de termes, si la surréduction est faite par rapport à un système de réécriture canonique. Ce n'était pas le cas au chapitre précédent. Dans notre procédure d'unification générale, la surréduction contrainte est faite en même temps que la complétion de notre système d'équations. Examinons comment des améliorations peuvent être obtenues en introduisant un graphe d'opérateurs dans ce nouveau cadre de travail. Nous n'utiliserons ici que des graphes d'opérateurs car ils sont plus simples à représenter, mais toutes les méthodes développées ci-dessous devront pouvoir être adaptées au cadre du graphe de termes.

3.3.1 Graphe d'opérateurs pour un système d'équations non canonique

Notre procédure d'E-unification autorise en entrée un ensemble d'équations quelconques (éventuellement contraintes) qui évoluera au cours de la procédure. Mais l'ensemble des symboles de fonctions est fixé dès le départ et restera inchangé tout au long des calculs. Les noeuds de notre graphe d'opérateurs seront donc les symboles de fonctions apparaissant dans le système d'équations initiales et dans l'unificande à résoudre.

La notion d'orientation n'intervient pas au niveau théorique de notre approche, mais pour optimiser l'utilisation du graphe, une équation orientable correspondra à un arc orienté et une équation inorientable correspondra à deux arcs symétriques dans le graphe. De plus les arcs seront étiquetés avec le numéro des équations qui les ont produits. Etudions l'ajout d'arcs dans le graphe en fonction des équations.

- Pour une équation sans contrainte : $s \doteq t[\![\ ]\!]$.

  ∘ Soit s et t sont non variables et on ajoute dans le graphe un arc entre $t\hat{e}te(s)$ et $t\hat{e}te(t)$, de $t\hat{e}te(s)$ vers $t\hat{e}te(t)$ si $s > t$, de $t\hat{e}te(t)$ vers $t\hat{e}te(s)$ si $t > s$ et dans les deux sens si s et t sont incomparables.

  ∘ Soit s est une variable et t est non variable, on ajoute des arcs de $t\hat{e}te(t)$ vers tous les noeuds du graphe.

  ∘ Soit t est une variable et s est non variable, on ajoute des arcs de $t\hat{e}te(s)$ vers tous les noeuds du graphe.

  ∘ Soit finalement les deux sont des variables disjointes et dans ce cas on a un graphe complet, c'est-à-dire il y a un arc entre tous les noeuds du graphes pris deux à deux.

- Pour une équation avec une contrainte : $s \doteq t[\![C]\!]$. On crée l'ensemble des équations propagées de $s \doteq t[\![C]\!]$ tel que pour deux équations disjointes de cet ensemble, $s_i \doteq t_i$ et $s_j \doteq t_j$ on ait $t\hat{e}te(s_i) \neq t\hat{e}te(s_j)$ ou $t\hat{e}te(t_i) \neq t\hat{e}te(t_j)$. Cet ensemble est fini puisque l'ensemble des symboles de fonction est fini. (Dans la majorité des cas cet ensemble sera réduit à un seul élément comme par exemple dans le cas où s et t sont tous deux non variables.) Pour chaque équation de cet ensemble on ajoute des arcs dans le graphe suivant le même procédé décrit pour une équation sans contrainte.

Voyons maintenant un exemple d'initialisation d'un graphe d'opérateurs à partir d'un ensemble d'équations contraintes.

Exemple 3.15 Soit E le système suivant :

$$\begin{array}{rcrcl} (1) & : & f(a,g(x)) & \doteq & g(x)[\![\ ]\!] \\ (2) & : & h(x,i(x)) & \doteq & x[\![\ ]\!] \\ (3) & : & g(x) & \doteq & x[\![x = a]\!] \\ (4) & : & f(b,x) & \doteq & x[\![x > b]\!] \end{array}$$

sous un ordre lpo avec une précédence $f > g > h > i > a > b$. La première équation met un arc de f vers g, avec la deuxième on ajoute un graphe de h vers tous les noeuds du graphe. En propageant la troisième équation on obtient $g(a) \doteq a[\![\ ]\!]$ et donc on a un arc de g vers a. Pour la quatrième on construit l'ensemble des propagées suivant : [2]

$$\begin{array}{lr} \{ & f(b,a) \doteq a[\![\ ]\!], \\ & f(b,f(x,y)) \doteq f(x,y)[\![\ ]\!], \\ & f(b,g(x)) \doteq g(x)[\![\ ]\!], \\ & f(b,h(x,y)) \doteq h(x,y)[\![\ ]\!], \\ & f(b,i(x)) \doteq i(x)[\![\ ]\!]\ \} \end{array}$$

On ajoutera donc un arc de f vers tous les noeuds du graphe excepté vers b. Finalement on obtient le graphe des opérateurs figure 3.15.

Le système d'équations initial va être complété au cours de la procédure. Pendant la complétion il y a génération de paires critiques et donc de nouvelles équations apparaissent dans

[2]Ici l'ensemble correspond à l'ensemble des propagations minimales de la règle (4), ce ne sera pas toujours le cas, en général on aura seulement une inclusion, de notre ensemble dans l'ensemble des propagations minimales.

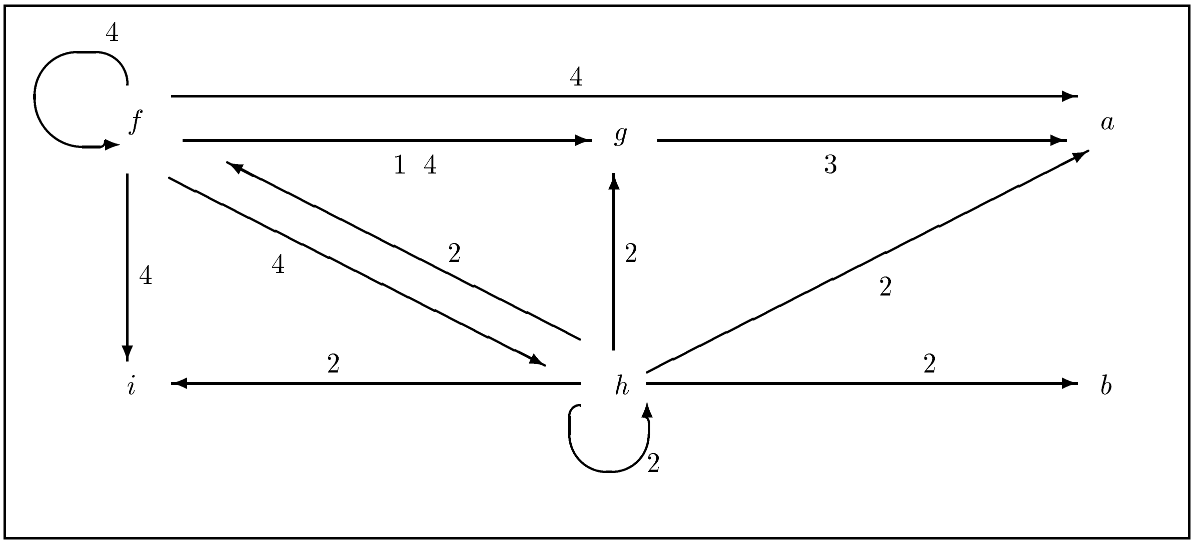


Figure 3.1: Graphe des opérateurs initial

notre système. A chaque création d'une nouvelle paire critique on ajoute au graphe le ou les arcs correspondants. Mais au cours de la complétion on peut aussi simplifier des équations. Dans ce cas on supprime du graphe tous les arcs étiquetés par le numéro de la règle simplifiée et on ajoute les arcs correspondant à la règle obtenue après simplification, si celle-ci n'a pas été trivialisée.

Exemple 3.15 (suite) La complétion crée une paire critique entre (3) et (1) qui est simplifiée par (3) et on obtient (5) : $f(a,a) \doteq a[\![\ ]\!]$. Puis il y a propagation minimale de (3) et (4) ce qui donne le système final suivant :

$$
\begin{array}{lcrcl}
(1) & : & f(a,g(x)) & \doteq & g(x)[\![\ ]\!] \\
(2) & : & h(x,i(x)) & \doteq & x[\![\ ]\!] \\
(4) & : & f(b,x) & \doteq & x[\![x > b]\!] \\
(5) & : & f(a,a) & \doteq & a[\![\ ]\!] \\
(6) & : & g(a) & \doteq & a[\![\ ]\!] \\
(7) & : & f(b,f(x,y)) & \doteq & f(x,y)[\![\ ]\!] \\
(8) & : & f(b,a) & \doteq & a[\![\ ]\!] \\
(9) & : & f(b,g(x)) & \doteq & g(x)[\![\ ]\!] \\
(10) & : & f(b,i(x)) & \doteq & i(x)[\![\ ]\!] \\
(11) & : & f(b,h(x,y)) & \doteq & h(x,y)[\![\ ]\!]
\end{array}
$$

La règle (3) a été simplifiée par la règle (6), on a le graphe final figure 3.15.

On peut remarquer que les deux graphes ont les mêmes arcs. Cela provient du fait que les nouvelles règles n'ont fait que surcharger les arcs existant au départ.

3.3.2 Utilisation du graphe des opérateurs

Le graphe des opérateurs ainsi construit, nous permettra de savoir quand deux termes pourront être unifiables ou non. Ainsi sur l'exemple précédent, l'unificande $g(x) \stackrel{?}{\doteq} i(a)$ n'aura pas de solution car g et i ne sont pas joignables dans le graphe, c'est-à-dire qu'il n'y a pas une suite d'arcs, tous dans le même sens, de g vers un sommet du graphe et de i vers ce même sommet. C'est ainsi qu'une étape de surréduction contrainte sera appliquée sur un unificande $u \stackrel{?}{\doteq} v[\![T]\!]$

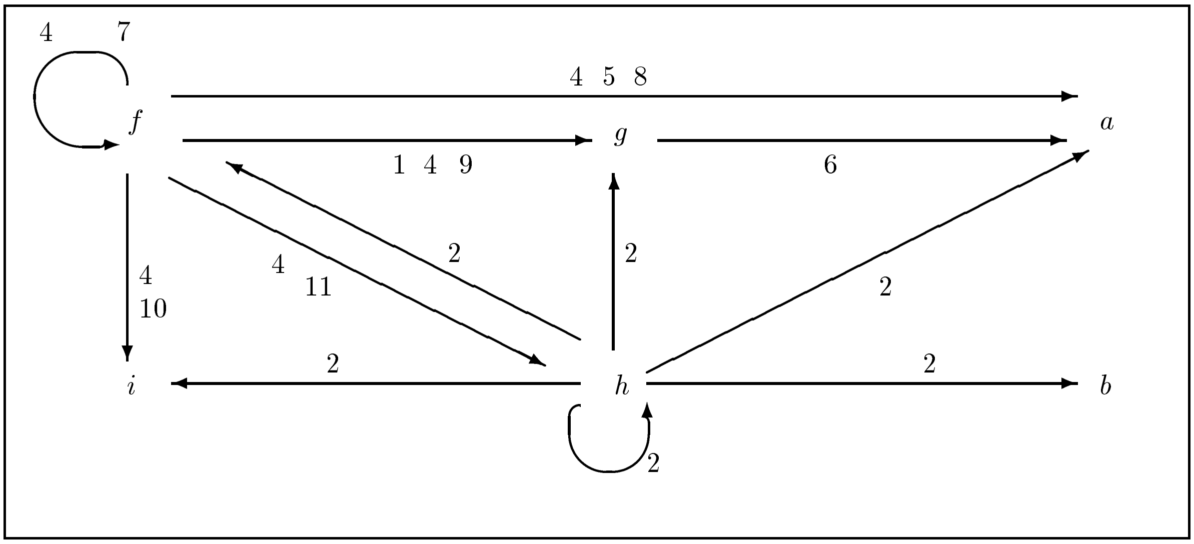


Figure 3.2: Graphe des opérateurs final

uniquement lorsque $tête(u)$ et $tête(v)$ seront joignables dans le graphe. On peut ainsi remplacer la règle d'inférence 4) par la règle suivante :

4') Surréduction contrainte dirigée

$$\frac{(E, R, G \cup \{s \stackrel{?}{=} t \ [\![c]\!]\})}{(E, R, G \cup \{s \stackrel{?}{=} t \ [\![c]\!] \ , \ s' \stackrel{?}{=} t' \ [\![c']\!]\})}$$

si $(s \stackrel{?}{=} t \ [\![c]\!]) \ \rightsquigarrow_{(R \cup E)} \ (s' \stackrel{?}{=} t' \ [\![c']\!])$,
et $tête(s)$ est joignable à $tête(t)$ dans le graphe.

Pour résoudre un unificande $u \stackrel{?}{=} v[\![T]\!]$ dans un ensemble d'équations, on commencera par compléter le système jusqu'à ce que $tête(u)$ et $tête(v)$ soient joignables dans le graphe. Si la complétion termine et si $tête(u)$ et $tête(v)$ ne sont toujours pas joignables dans le graphe, l'unificande n'aura pas de solution.

Si après initialisation du graphe des opérateurs celui-ci peut être partagé en plusieurs sous-graphes disjoints, c'est-à-dire qu'il n'y a aucun arc reliant un sous-graphe à un autre, et de plus si $tête(u)$ et $tête(v)$ sont dans deux sous-graphes différents, alors notre unificande n'aura pas de solutions. On peut arrêter la procédure avant même d'avoir fait la complétion de notre système.

Les étapes de surréduction contrainte ne sont pas uniquement faites en tête d'un des deux termes composant l'unificande à résoudre, contrairement à la surréduction définie dans le chapitre précédent. On pourra donc diriger uniquement les étapes en tête dans la chaîne de surréduction contrainte de la façon suivante : si une étape est faite en tête sur un unificande $u \stackrel{?}{=} v[\![T]\!]$, le numéro de la règle utilisée pour cette étape doit étiqueter un arc du graphe se trouvant sur un chemin qui permet de joindre $tête(u)$ et $tête(v)$. Détaillons ces différents cas sur un exemple.

3.3.3 Exemple de surréduction contrainte dirigée

Dans cet exemple nous allons voir comment le graphe peut diriger et faire terminer notre procédure.

Exemple 3.16 Soit E le système d'équations suivant :

$$\begin{array}{rclcl}
(1) & : & k(k(x)) & \doteq & k(x)[\![\ ]\!] \\
(2) & : & j(0,x) & \doteq & f(x)[\![\ ]\!] \\
(3) & : & j(x,h(y)) & \doteq & g(y)[\![\ ]\!] \\
(4) & : & j(x,g(y)) & \doteq & h(y)[\![\ ]\!] \\
(5) & : & f(k(x)) & \doteq & k(x)[\![\ ]\!]
\end{array}$$

sous un ordre lpo avec la précédence $j > f > g > h > k > 0$. Nous avons le graphe d'opérateurs figure 3.16.

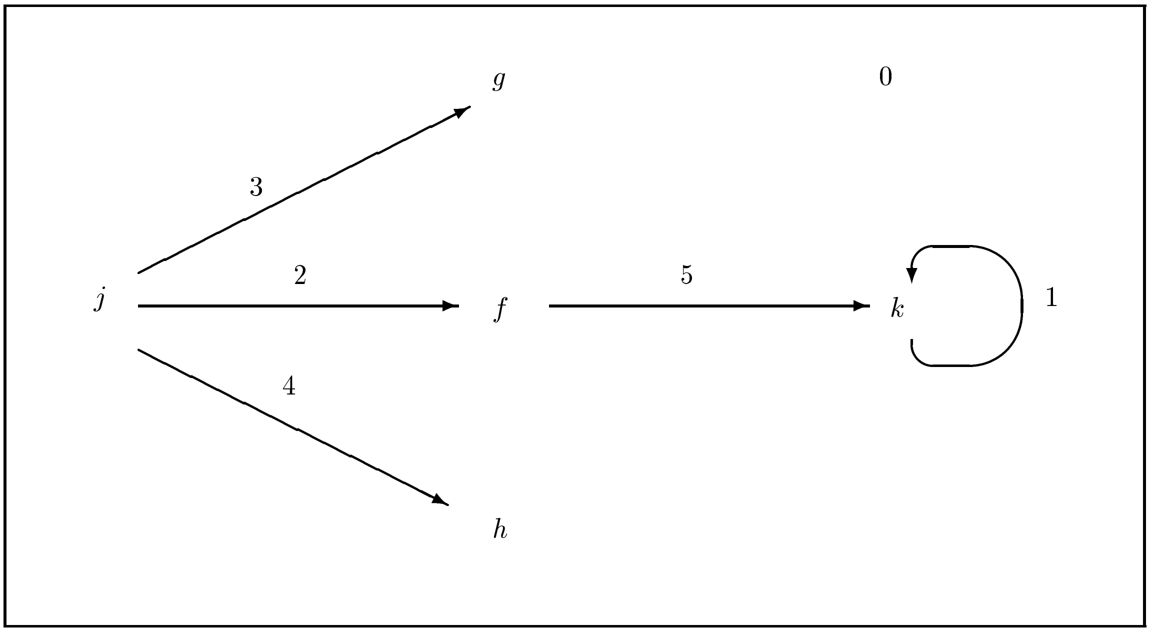


Figure 3.3: Graphe des opérateurs initial

Si on essaie de résoudre l'unificande $f(x) \stackrel{?}{=} 0$ la procédure s'arrêtera immédiatement en retournant qu'il n'y a pas de solution puisque 0 est isolé de f dans le graphe. Essayons de résoudre l'unificande $f(x) \stackrel{?}{=} g(y)$. Dans le graphe f et g ne sont pas joignables (mais ils ne sont pas isolés), donc il faut commencer par par une étape de complétion du système même si on peut faire une étape de surréduction contrainte sur l'unificande avec la règle (5) mais on est sûr que pour l'instant cette surréduction n'aboutirait pas à une solution. Ainsi on ajoute au système la paire critique obtenue entre (2) et (3) : (6) $\;:\;$ $f(x) \doteq g(y)[\![x =_n h(y)]\!]$. Avec cette nouvelle équation le graphe est complété (figure 3.16).

A présent f et g sont joignables dans le graphe grâce à la règle (6), on peut donc appliquer sur notre unificande des étapes de surréduction contrainte. Les étapes sont faites en tête, on est donc obligé de surréduire avec la règle (6), on obtient :

$$f(x) \stackrel{?}{=} g(y)[\![\ ]\!] \rightsquigarrow_{(6)} g(y_1) \stackrel{?}{=} g(y)[\![x =_n x_1 \ \wedge\ x_1 =_n h(y_1)]\!]$$

Ce qui donne comme solution $\{x \leftarrow h(y_1),\ y \leftarrow y_1\}$. On ne peut plus appliquer de surréduction, on complète donc de nouveau notre système et faisant une nouvelle paire critique entre (2) et (4) : (7) $\;:\;$ $f(x) \doteq h(y)[\![x =_n g(y)]\!]$. Cette paire critique finit de compléter notre système et le graphe d'opérateurs final est en figure 3.16.

Il n'y pas de nouveau chemin entre f et g notre procédure s'arrêtera donc alors qu'une surréduction sans graphe bouclerait avec la règle (1) après avoir fait une étape de surréduction avec

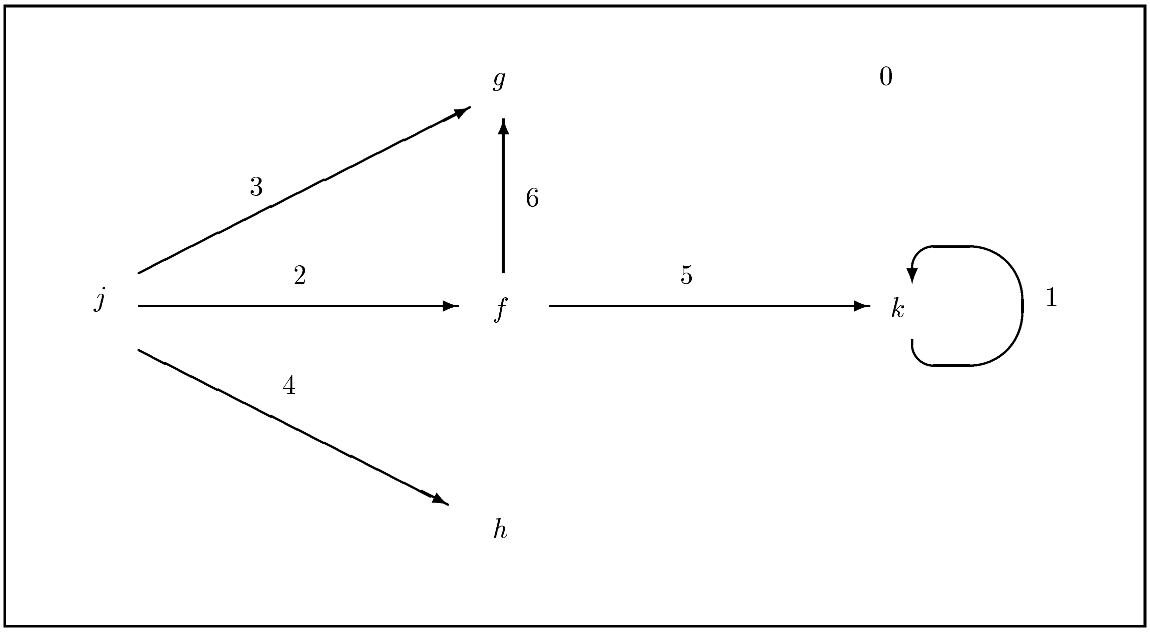


Figure 3.4: Evolution du graphe des opérateurs pendant la complétion

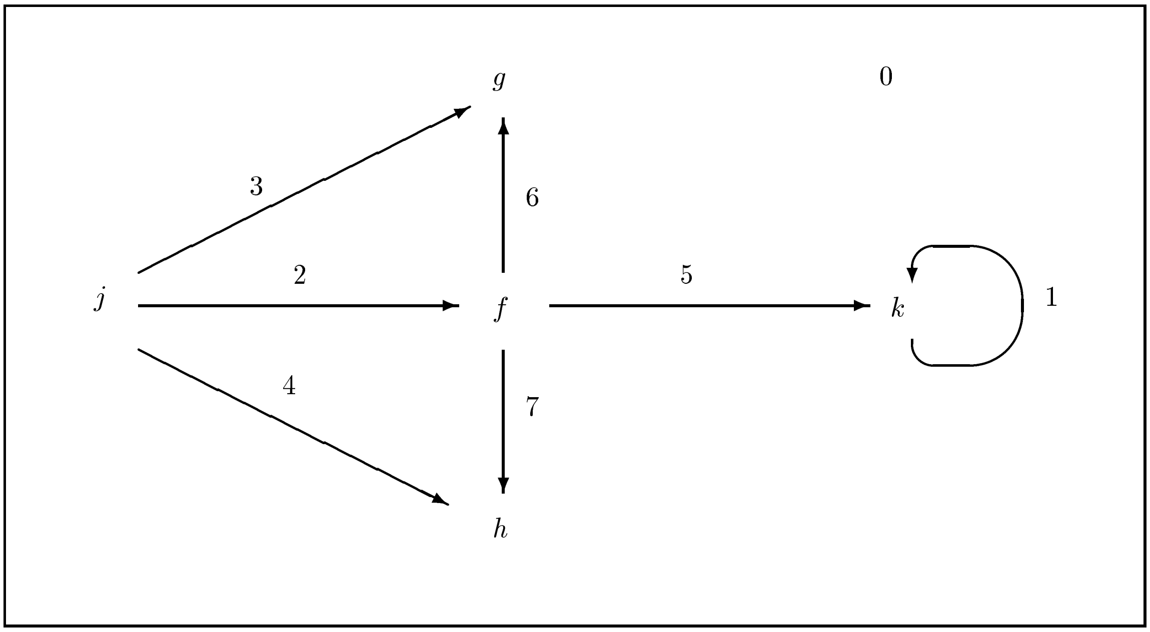


Figure 3.5: Graphe des opérateurs final

la règle (5).

$$f(x) \stackrel{?}{=} g(y)[\![\ ]\!] \rightsquigarrow_{(5)} k(x_1) \stackrel{?}{=} g(y)[\![x =_n k(x_1)]\!]$$
$$\rightsquigarrow_{(1)} k(x_2) \stackrel{?}{=} g(y)[\![x =_n k(x_1) \ \wedge \ x_1 =_n k(x_2)]\!] \rightsquigarrow_{(1)} \ldots.$$

Remarque: Ce n'est pas parce que le graphe des opérateurs peut se décomposer en plusieurs sous-graphes distincts que la complétion peut se restreindre aux équations composant le sous-graphe contenant $tête(u)$ et $tête(v)$ lorsque $u \stackrel{?}{=} v$ est l'unificande à résoudre.

3.4 Conclusion

Une implémentation de la surréduction dirigée par cette approche de graphe d'opérateurs a été réalisée en 1991, basée sur le logiciel *REVE* (en CLU); il reste à l'incorporer à notre logiciel actuel *UREVEAL*, qui est en cours de développement (cf. chapitre 4). Cette première implémentation nous a permis de tester notre méthode d'unification dans les systèmes de réécriture canoniques. Le graphe d'opérateurs permet de supprimer des chemins inutiles et donc fait terminer plus souvent la surréduction, mais nous pensons qu'une efficacité bien meilleure serait obtenue en combinant le graphe d'opérateurs avec des stratégies de surréduction, pour éviter les redondances dans la recherche des solutions. Notre perspective est donc d'intégrer un graphe d'opérateurs dans notre logiciel actuel. Nous désirons aussi poursuivre les recherches pour trouver une implication du graphe au niveau de la complétion de notre système d'équations initial, et ainsi, diriger la complétion en fonction de l'équation que nous avons à résoudre.

Annexe du Chapitre 3

Proofs of the narrowing procedure

We want to prove that the procedure allows to solve equations modulo R. For that, we have to show the correctness and the completeness of every inference rule, that is, for any inference step $S \xrightarrow{gr} S'$ to prove $SOL(S) = SOL(S')$. For completeness, we need some graph properties. But it is not enough because the rule which would transform each unificand into itself is correct and complete, and can't solve equations. We have to prove moreover that for any solution θ, a unificand S_θ is got, a syntactical solution of which is θ. In order to consider the length between S and S_θ we introduce a complexity measure c. Thus, we just have to show that along a fair derivation, the complexity decreases and gets equal to 0.

Notations: $\xrightarrow{gr}{}^*$ denotes a path in the graph that may contain $\xrightarrow{gr}$ arrows and rewriting arrows, and $\xrightarrow{gr}{}^+$ is the particular case where the path contains at least one step. $\xleftrightarrow{gr}{}^?$ denotes zero or one step of $\xleftrightarrow{gr}$.

A.1 Correction

Definition A.1 A unificand S is said **trivial** if $SOL(S)$ is the set of all normalized substitutions. $\diamond$

Remark : For a normalized substitution θ

$$\theta \in SOL(S) \Longleftrightarrow \theta(S) \text{ is trivial}$$

Proposition A.2 *(correction of* (Narrow-$\doteq_R$)*)*
If $S \longrightarrow S'$ by (Narrow-$\doteq_R$) *then $SOL(S') \subseteq SOL(S)$.*

Proof: Since (Narrow-$\doteq_R$) is applied only on $\doteq_R$ equations, suppose that $S = (s \doteq_R t)$. Let θ be in $SOL(S')$, there exists at least one trivial conjunctive factor $\theta(C)$ of $\theta(S')$. Consider all the possible cases according to the nature of s and t.

1. $C = s_1 \doteq_R t_1 \wedge \ldots \wedge sn \doteq_R t_n$ and $s \xleftrightarrow{gr} t$ is in the graph. By construction of the arrows in the graph we have $top(s) = top(t)$ because t is not a variable. $\forall i = 1 \ldots n$, $\theta(s_i \doteq_R t_i)$ is trivial, then $\forall i = 1 \ldots n$, $\exists u_i$, such that $\theta s_i \xrightarrow{*R} u_i \xleftarrow{*R} \theta t_i$ therefore $\forall f$ of arity n we have : $\theta(f(s_1, \ldots, sn)) \xrightarrow{*R} f(u_1, \ldots, u_n) \xleftarrow{*R} \theta(f(t_1, \ldots, t_n))$. Then for $f = top(s) = top(t)$ and $u = f(u_1, \ldots, u_n)$ we have $\theta s \xrightarrow{*R} u \xleftarrow{*R} \theta t$.
Then $\theta \in SOL(S)$.
2. $C = (s_1 \stackrel{\rightarrow}{=}_R l_1 \wedge \ldots \wedge sn \stackrel{\rightarrow}{=}_R l_n \wedge r \doteq_R t)$, and there are in the graph the following path : $s \xrightarrow{gr} l \rightarrow r \xrightarrow{gr}{}^* \xleftrightarrow{gr}{}^? {}^* \xleftarrow{gr} t$. $\theta(C)$ is trivial therefore $\theta s \xrightarrow{*R} \theta l$ and $\exists u$ such

that $\theta r \xrightarrow{*R} u \xleftarrow{*R} \theta t$ we obtain : $\theta s \xrightarrow{*R} \theta l \longrightarrow \theta r \xrightarrow{*R} u \xleftarrow{*R} \theta t$.
Then $\theta \in SOL(S)$.

3. $C = (t_1 \stackrel{\rightarrow}{=}_R l_1 \wedge \ldots \wedge tn \stackrel{\rightarrow}{=}_R l_n \wedge s \doteq_R r)$, and there are $s \overset{gr}{\longleftrightarrow}{}^{?} * \xleftarrow{gr} r \leftarrow l \xleftarrow{gr} t$ in the graph. $\theta(C)$ is trivial therefore $\theta t \xrightarrow{*R} \theta l$ and $\exists u$ such that $\theta s \xrightarrow{*R} u \xleftarrow{*R} \theta r$ we obtain : $\theta s \xrightarrow{*R} u \xleftarrow{*R} \theta r \longleftarrow \theta l \xleftarrow{*R} \theta t$.
Then $\theta \in SOL(S)$.
4. $C = (s|_v \doteq l \wedge s[v \leftarrow r] \doteq_R y)$. θC is trivial therefore $\theta(s|_v) = \theta(l)$ and $\exists u$ such that $\theta(s[v \leftarrow r]) \xrightarrow{*R} u \xleftarrow{*R} \theta y$. We have $\theta s = \theta(s[v \leftarrow s|_v]) = \theta(s)[v \leftarrow \theta(s|_v)] = \theta(s)[v \leftarrow \theta(l)] \rightarrow \theta(s)[v \leftarrow \theta(r)] = \theta(s[v \leftarrow r])$. Thus we obtain $\theta s \xrightarrow{*R} u \xleftarrow{*R} \theta y$.
Then $\theta \in SOL(S)$.
5. $C = (y \doteq s)$ θC is trivial therefore $\theta y = \theta s$, then $\theta \in SOL(S)$
6. $C = (x \doteq y)$. θC is trivial therefore $\theta x = \theta y$, then $\theta \in SOL(S)$

□

Proposition A.3 *(correction of* (Narrow-$\stackrel{\rightarrow}{=}_R$)*)*
If $S \longrightarrow S'$ by (Narrow-$\stackrel{\rightarrow}{=}_R$) *then $SOL(S') \subseteq SOL(S)$.*

Proof: Since (Narrow-$\stackrel{\rightarrow}{=}_R$) is applied only on $\stackrel{\rightarrow}{=}_R$ equations, suppose that $S = (s \stackrel{\rightarrow}{=}_R t)$. Let θ be in $SOL(S')$, there exists at least one trivial conjunctive factor $\theta(C)$ of $\theta(S')$. Consider all the possible cases according to the nature of s and t.

1. $C = s_1 \stackrel{\rightarrow}{=}_R t_1 \wedge \ldots \wedge sn \stackrel{\rightarrow}{=}_R t_n$ and $s \xrightarrow{gr} t$ is in the graph. By construction of the arrows of the graph we have $top(s) = top(t)$ because t is not a variable. θC is trivial therefore $\forall i = 1 \ldots n$ $\theta s_i \xrightarrow{*R} \theta t_i$ by a basic derivation, then $\theta s = \theta(top(s)(s_1, \ldots, sn)) \xrightarrow{*R} \theta(top(t)(t_1, \ldots, t_n)) = \theta t$ by a basic derivation.
Then $\theta \in SOL(S)$
2. $C = (s_1 \stackrel{\rightarrow}{=}_R l_1 \wedge \ldots \wedge sn \stackrel{\rightarrow}{=}_R l_n \wedge r \stackrel{\rightarrow}{=}_R t)$, and there is the path $s \xrightarrow{gr} l \rightarrow r \xrightarrow{gr}{}^{*} t$ in the graph. $\theta(C)$ is trivial therefore $\forall i = 1 \ldots n, \theta s_i \xrightarrow{*R} \theta l_i$ by a basic derivation then $\theta s \xrightarrow{*R} \theta l$ by a basic derivation and $\theta r \xrightarrow{*R} \theta t$ we obtain : $\theta s \xrightarrow{*R} \theta l \longrightarrow \theta r \xrightarrow{*R} \theta t$.
Then $\theta \in SOL(S)$.
3. $C = (y \doteq t)$ θC is trivial therefore $\theta y = \theta t$ then $\theta \in SOL(S)$.
4. $C = (y \doteq s)$ θC is trivial therefore $\theta y = \theta s$, then $\theta \in SOL(S)$
5. $C = (s|_v \doteq l \wedge s[v \leftarrow r] \stackrel{\rightarrow}{=}_R y)$. θC is trivial therefore $\theta(s|_v) = \theta(l)$ and $\theta(s[v \leftarrow r]) \xrightarrow{*R} \theta y$ by a basic derivation. We have $\theta s = \theta(s[v \leftarrow s|_v]) = \theta(s)[v \leftarrow \theta(s|_v)] = \theta(s)[v \leftarrow \theta(l)] \rightarrow \theta(s)[v \leftarrow \theta(r)] = \theta(s[v \leftarrow r])$. Thus we obtain $\theta s \xrightarrow{*R} \theta y$ by a basic derivation.
Then $\theta \in SOL(S)$.
6. $C = (x \doteq y)$. θC is trivial therefore $\theta x = \theta y$, then $\theta \in SOL(S)$

□

Proposition A.4 *(correction of the simplication rules)*
If $S \longrightarrow S'$ by a simplication rule then $SOL(S') \subseteq SOL(S)$

Proof: • For the rule (Solve-$\doteq$) :
Since it's applied only on conjunctive factor of $\doteq$ equations suppose that $S = (\wedge t_1 \doteq$

$t'_1 \dots tn \doteq t'n$). By definition we have $SOL(S') = SOL(S)$, then $SOL(S') \subseteq SOL(S)$.

• For the rule (Del-$\doteq$) :
It is applied only on $x \doteq t$ equation with t a term which is not in normal form. Suppose that $S = (x \doteq t)$ with t not in normal form, therefore $SOL(S) = \emptyset$ and $SOL(S') = SOL(F) = \emptyset$ then $SOL(S') \subseteq SOL(S)$.

• For the rule (Del-$\doteq_R$) :
It is applied only on $a \doteq_R a$ equation with a a constant. Suppose that $S = (a \doteq_R a)$ with a a constant, we have $S' = T$, all normalized substitution is solution of the equation $a \doteq_R a$, therefore $SOL(S) = \{\theta \text{ normalized}\} = SOL(T) = SOL(S')$ then $SOL(S') \subseteq SOL(S)$.

• For the rule (Del- $\overset{\rightarrow}{=}_R$) :
The proof is similar to the previous one.

• For the rule (Del-$\vee$-T) :
Suppose that $S = (\vee S_1 \dots S_{i-1} S_i S_{i+1} \dots S_n)$ with $S_i = T$ since it's the unique case where the rule (Del-$\vee$-T) is applied. We have $SOL(S) = \cup_{j=1}^{n} SOL(S_j) = \cup_{j=1}^{i-1} SOL(S_j) \cup SOL(T) \cup_{j=i+1}^{n} SOL(S_j) = SOL(T) = SOL(S')$. Then $SOL(S') \subseteq SOL(S)$.

• For the rules (Del-$\vee$-F), (Del-$\wedge$-T) and (Del-$\wedge$-F), the proofs are similar to the previous one.

• For the rule (Distributivity) :
Suppose that $S = \wedge S_1 \dots S_n (\vee Q_1 \dots Q_p)$, we obtain $S' = \vee(\wedge S_1 \dots S_n Q_1) \dots (\wedge S_1 \dots S_n Q_p)$. We have $SOL(S) = \cap_{i=1}^{n} SOL(S_i)(\cup_{j=1}^{p} SOL(Q_j))$. By distributivity of the joining, we obtain $SOL(S) = \cup_{j=1}^{p}(\cap_{i=1}^{n} SOL(S_i) SOL(Q_j)) = SOL(S')$. Then $SOL(S') \subseteq SOL(S)$. □

Theorem A.5 *The inference rules are correct.*

A.2 Graph properties

Notation: If t is a term appearing inside a basic narrowing derivation, $B(t)$ denotes the set of basic occurrences of t.

A.2.1 Relation between weak narrowing and the graph

Intuitively, the graph of terms makes narrowing steps, in which the instanciation would not be applied. We call this notion "weak narrowing".

Definition A.6 The step $t \overset{w}{\rightsquigarrow}_{[u,g\to d]} t'$ is a step of **weak narrowing** if

- $t|_u$ is not a variable
- $t|_u$ and g are unifiable and $t' = t[u \leftarrow d]$.

The notation $t \overset{w}{\rightsquigarrow}_{[st]} t'$ means there is a step $t \overset{w}{\rightsquigarrow}_{[u,g\to d]} t'$ where $u \neq \epsilon$ (st means subterm). ◇

Lemma A.7 If $f(t_1, \dots, t_n) \overset{w}{\rightsquigarrow}^*_{[st]} f(s_1, \dots, s_n)$ then for all i we have $t_i \overset{w}{\rightsquigarrow}^* s_i$.

Proof: By induction on the length p of the derivation.

- $p = 1$. The result holds.
- $p > 1$. Suppose $f(t_1, \dots, t_n) \stackrel{w}{\rightsquigarrow}^* f(t'_1, \dots, t'_n) \stackrel{w}{\rightsquigarrow} f(s_1, \dots, s_n)$.

From induction hypothesis, for all i, $t_i \stackrel{w}{\rightsquigarrow}^* t'_i$. In the other hand, there exists j such that $t'_j \stackrel{w}{\rightsquigarrow} s_j$ and for all $k \neq j$, $s_k = t'_k$; which proves the result. □

Basic narrowing implies weak narrowing, because in basic narrowing the part brought by the substitutions is not narrowed.

Proposition A.8 *If the derivation*

$$t_1 \rightsquigarrow_{[u_1, g_1 \to d_1, \sigma_1]} t_2 \rightsquigarrow \dots \rightsquigarrow t_n$$

is basic, then there exists

$$t'_1 \stackrel{w}{\rightsquigarrow}_{[u_1, g_1 \to d_1]} t'_2 \stackrel{w}{\rightsquigarrow} \dots \stackrel{w}{\rightsquigarrow} t'_n$$

such that $t_n = (\sigma_{n-1} \dots \sigma_1) t'_n$ *and* $B(t_n) = O(t'_n)$.

Proof: By induction on the length n.

- If $n = 1$ the property is obvious.
- Let $n > 1$ and suppose the property holds for $n - 1$.

$t_1 \rightsquigarrow^* t_n$ is basic, then so is $t_1 \rightsquigarrow^* t_{n-1}$. From induction hypothesis we get

$$t_1 \stackrel{w}{\rightsquigarrow}_{[u_1, g_1 \to d_1]} \dots \stackrel{w}{\rightsquigarrow} t'_{n-1}$$

with $t_{n-1} = (\sigma_{n-2} \dots \sigma_1) t'_{n-1}$ and $B(t_{n-1}) = O(t'_{n-1})$.

Consider the basic step

$$t_{n-1} = (\sigma_{n-2} \dots \sigma_1) t'_{n-1} \rightsquigarrow_{[u_{n-1}, g_{n-1} \to d_{n-1}, \sigma_{n-1}]} t_n$$

Since this step is basic we have $u_{n-1} \in B(t_{n-1})$, thus $u_{n-1} \in O(t'_{n-1})$.

Therefore

$$(\sigma_{n-2} \dots \sigma_1)(t'_{n-1}|_{u_{n-1}}) = ((\sigma_{n-2} \dots \sigma_1) t'_{n-1})|_{u_{n-1}} = t_{n-1}|_{u_{n-1}}$$

which is unifiable with g_{n-1}. It follows

$$t'_{n-1} \stackrel{w}{\rightsquigarrow}_{[u_{n-1}, g_{n-1} \to d_{n-1}]} t'_n = t'_{n-1}[u_{n-1} \leftarrow d_{n-1}]$$

We have

$$t_n = \sigma_{n-1}(t_{n-1}[u_{n-1} \leftarrow d_{n-1}]) = \sigma_{n-1}(((\sigma_{n-2} \dots \sigma_1) t'_{n-1})[u_{n-1} \leftarrow d_{n-1}])$$

where the basic part is the part brought by t'_{n-1} and d_{n-1}. By assuming that $V(d_{n-1}) \cap D(\sigma_{n-2} \dots \sigma_1) = \emptyset$ we have

$$t_n = \sigma_{n-1}((\sigma_{n-2} \dots \sigma_1)(t'_{n-1}[u_{n-1} \leftarrow d_{n-1}])) = (\sigma_{n-1} \dots \sigma_1) t'_n$$

Moreover $B(t_n) = O(t'_{n-1}[u_{n-1} \leftarrow d_{n-1}]) = O(t'_n)$. □

Lemma A.9 Let r be a right (sub)term of G, l be a left (sub)term of G. If r and l are unifiable, then there exists $r \xrightarrow{gr} l \in G$.

Proof: Obvious. □

Lemma A.10 Let r, r' be right (sub)terms of G. If r and r' are unifiable, then there exists $r \xleftrightarrow{gr} r' \in G$.

Proof: Obvious. □

Lemma A.11 Let r, t two right (sub)terms of G.
If $r \overset{w}{\rightsquigarrow}{}^{*}_{[\epsilon]} t$ by n steps, then there exists a path $r \xrightarrow{gr}{}^{*} t$ in G containing n rewrite arrows.

Proof: By induction on the length n of the derivation.
- $n = 0$. We have $r = t$.
- $n > 0$. Split the derivation into

$$r \overset{w}{\rightsquigarrow}_{[\epsilon, g \to d]} d \overset{w}{\rightsquigarrow}{}^{*}_{[\epsilon]} t$$

d is a right term of G, and from induction hypothesis there exists $d \xrightarrow{gr}{}^{*} t$ in G containing $n-1$ rewrite arrows.
In the other hand, r and g are unifiable, and from lemma A.9 we get $r \xrightarrow{gr} g \in G$.
It results $r \xrightarrow{gr}{}^{*} t \in G$ containing n rewrite arrows. □

For the next proofs, we need a non classical ordering on occurrences.

Definition A.12 For occurrences u, v we write $u \succ v$ iff $u = w.v$ where $w \neq \epsilon$.
◇

Note that $\succ$ is noetherian.

In the two following propositions, it is necessary to consider (sub)terms instead of terms to be able to apply the induction hypothesis in the proofs. Actually, only the results for terms are usefull.

Proposition A.13 *Let r be a right (sub)term of G, l be a left (sub)term of G. If*
- $r \overset{w}{\rightsquigarrow}{}^{+}_{[st]} t$,
- *and t and l are unifiable,*

then there exists $r \xrightarrow{gr} l \in G$.

Proof: If l is a variable the result is obvious. Otherwise, by noetherian induction on the multiset of narrowing occurrences occuring in the derivation, by using the multiset extension of $\succ$.

r is neither a variable nor a constant, and we have $\mathrm{top}(r) = \mathrm{top}(t) = \mathrm{top}(l)$. So, let's write $r = f(r_1, \ldots, r_n)$, $t = f(t_1, \ldots, t_n)$, $l = f(l_1, \ldots, l_n)$.

For each $i \in \{1, \ldots, n\}$:
from lemma A.7 we have $r_i \overset{w}{\rightsquigarrow}{}^{*} t_i$, which is decomposed into

$$r_i \overset{w}{\rightsquigarrow}{}^{*}_{[\epsilon, g_0 \to \ldots \to d_0]} \underset{k=1,\ldots,p}{(} s_k \overset{w}{\rightsquigarrow}{}^{+}_{[st:u_{k,1},\ldots,u_{k,q_k}]} s'_k \overset{w}{\rightsquigarrow}{}^{+}_{[\epsilon, g_k \to \ldots \to d_k]})^{*} s_{p+1} \overset{w}{\rightsquigarrow}{}^{*}_{[st]} t_i$$

We have $s_1 = r_i$ or $s_1 = d_0$, then s_1 appears in G; from lemma A.11 we get $r_i \xrightarrow{gr}{}^{*} s_1$ is in G.

For each $k \in \{1, \ldots, p\}$, let's try to apply the induction hypothesis on $s_k \overset{w}{\rightsquigarrow}{}^{+}_{[st]} s'_k$ and g_k.
- s'_k and g_k are unifiable.
- $s_k = d_{k-1}$ (unless $k = 1$ and $s_1 = r_i$), then s_k is a right (sub)term of G.
- For each $j \in \{1, \ldots, q_k\}$, there obviously exists an occurrence v of the derivation $r \overset{w}{\rightsquigarrow}{}^{+} t$ such that $v = i.u_{k,j}$, hence $v \succ u_{k,j}$. Therefore the multiset of narrowing occurrences of $r_i \overset{w}{\rightsquigarrow}{}^{*} t_i$ is smaller than that of $r \overset{w}{\rightsquigarrow}{}^{+} t$.

From induction hypothesis, we get $s_k \xrightarrow{gr} g_k \in G$. Thus, from lemma A.11 we get

$$s_1 (d_{k-1} \xrightarrow{gr} g_k \xrightarrow{gr}{}^{+} d_k)^{*} s_{p+1} \in G$$

- If $s_{p+1} \overset{w}{\rightsquigarrow}^* t_i$ contains 0 step, then $s_{p+1} = d_p$ (or r_i), hence it is a right (sub)term of G. Moreover $s_{p+1} = t_i$, which is unifiable with l_i. Thus from lemma A.9, $s_{p+1} \xrightarrow{gr} l_i \in G$.
- Else $s_{p+1} \overset{w}{\rightsquigarrow}^+_{[st]} t_i$. We have :
  - l_i is a left subterm.
  - $s_{p+1} = d_p$ (or r_i), hence is a right (sub)term of G.
  - the multiset of narrowing occurrences is smaller than that of $r \overset{w}{\rightsquigarrow}^+ t$.

  From induction hypothesis, we get $s_{p+1} \xrightarrow{gr} l_i \in G$.

  Finally

$$r_i \xrightarrow{gr}{}^* l_i \in G$$

Consider here the above results for every $i \in \{1, \ldots, n\}$. It obviously follows $r \xrightarrow{gr} l \in G$. □

Proposition A.14 *Let r_1, r_2 be right (sub)terms of G. It is assumed that they don't point in the same term at comparable occurrences for the subsumption ordering. If*

$$r_1 \overset{w}{\rightsquigarrow}^{n_1}_{[st]} t_1$$
$$r_2 \overset{w}{\rightsquigarrow}^{n_2}_{[st]} t_2$$

where t_1 and t_2 are unifiable and $n_1 + n_2 > 0$ (n_1, n_2 are the lengths of the derivations), then there exists $r_1 \xleftrightarrow{gr} r_2 \in G$.

Proof: By noetherian induction on the union of the multisets of narrowing occurrences occurring in the two derivations, by using the multiset extension of $\succ$.

Up to a renaming, we can suppose $r_1 \overset{w}{\rightsquigarrow}^+ t_1$, which implies r_1 is neither a variable nor a constant.

If t_2 is a variable, then $n_2 = 0$ and r_2 is a variable. Therefore we obviously have $r_1 \xleftrightarrow{gr} r_2 \in G$.

Otherwise, we have $\text{top}(r_1) = \text{top}(t_1) = \text{top}(t_2) = \text{top}(r_2)$. Let's write

$$r_1 = f(r_{1,1}, \ldots, r_{1,n})$$
$$t_1 = f(t_{1,1}, \ldots, t_{1,n})$$
$$t_2 = f(t_{2,1}, \ldots, t_{2,n})$$
$$r_2 = f(r_{2,1}, \ldots, r_{2,n})$$

For each $i \in \{1, \ldots, n\}$:
From lemma A.7 we have

$$r_{1,i} \overset{w}{\rightsquigarrow}^*_{[\epsilon]} (s_k \overset{w}{\rightsquigarrow}^+_{[st]} \overset{w}{\rightsquigarrow}^+_{[\epsilon]})^* s_{p+1} \rightsquigarrow^*_{[st]} t_{1,i}$$

By applying lemma A.11 and proposition A.13 as often as necessary (we don't detail), we get $r_{1,i} \xrightarrow{gr}{}^* s_{p+1} \in G$.

From lemma A.7 we have

$$r_{2,i} \overset{w}{\rightsquigarrow}^*_{[\epsilon]} (s'_k \overset{w}{\rightsquigarrow}^+_{[st]} \overset{w}{\rightsquigarrow}^+_{[\epsilon]})^* s'_{p+1} \rightsquigarrow^*_{[st]} t_{2,i}$$

By applying lemma A.11 and proposition A.13 as often as necessary (we don't detail), we get $r_{2,i} \xrightarrow{gr}{}^* s'_{p+1} \in G$.

We have

$$s_{p+1} \overset{w}{\rightsquigarrow}^*_{[st]} t_{1,i}$$
$$s'_{p+1} \overset{w}{\rightsquigarrow}^*_{[st]} t_{2,i}$$

and $t_{1,i}, t_{2,i}$ are unifiable. s_{p+1} and s'_{p+1} are right (sub)terms of G.

- If the two above derivations don't contain any step, then s_{p+1} and s'_{p+1} are unifiable and from lemma A.10, $s_{p+1} \xleftrightarrow{gr} s'_{p+1} \in G$.
- Otherwise the multiset of occurrences is smaller. From induction hypothesis it comes $s_{p+1} \xleftrightarrow{gr} s'_{p+1} \in G$. Finally we have

$$r_{1,i} \xrightarrow{gr}^* s_{p+1} \xleftrightarrow{gr} s'_{p+1} {}^* \xleftarrow{gr} r_{2,i} \in G_n^B$$

The above conclusions hold for every $i \in \{1, \ldots, n\}$. Therefore $r_1 \xleftrightarrow{gr} r_2$ is in G. □

We extend the two previous propositions to any weak narrowing derivation.

Proposition A.15 *Let r be a right term of G, and l be a left term of G. If $r \overset{w}{\leadsto}^* t$ and t and l are unifiable, then $r \xrightarrow{gr}^* l \in G$.*

Proof: We decompose $r \xrightarrow{gr}^* l$ into

$$r \overset{w}{\leadsto}^*_{[\epsilon]} d_0 (\overset{w}{\leadsto}^+_{[st]} \overset{w}{\leadsto}^+_{[\epsilon]} d_k)^* \overset{w}{\leadsto}^*_{[st]} t$$

By applying lemma A.11 and proposition A.13 as often as necessary (we don't detail), we get

$$r \xrightarrow{gr}^* d_0 (\xrightarrow{gr} \xrightarrow{gr}^*)^* \xrightarrow{gr} l \in G$$

□

Proposition A.16 *Let r_1, r_2 be two right terms of G. If*

$$r_1 \overset{w}{\leadsto}^*_{[st]} t_1$$
$$r_2 \overset{w}{\leadsto}^* t_2$$

and t_1, t_2 are unifiable, then $r_1 \xleftrightarrow{gr}^? {}^* \xleftarrow{gr} r_2 \in G$.

Proof: We decompose $r_2 \overset{w}{\leadsto}^* t_2$ into

$$r_2 \overset{w}{\leadsto}^? d'_0 (\overset{w}{\leadsto}^+_{[st]} \overset{w}{\leadsto}^+_{[\epsilon]} d'_k)^* d'_{p'} \overset{w}{\leadsto}^*_{[st]} t_2$$

By applying lemma A.11 and proposition A.13 as often as necessary (we don't detail), we get

$$r_2 \xrightarrow{gr}^* d'_{p'} \in G$$

r_1 and $d'_{p'}$ are (sub)-right-hand-sides. Either $r_1 = d'_{p'}$, or from proposition A.14, $r_1 \xleftrightarrow{gr} d'_{p'}$.
□

Proposition A.17 *Let r_1, r_2 be two right terms of G. If*

$$r_1 \overset{w}{\leadsto}^* t_1$$
$$r_2 \overset{w}{\leadsto}^* t_2$$

and t_1, t_2 are unifiable, then $r_1 \xrightarrow{gr}^* \xleftrightarrow{gr}^? {}^* \xleftarrow{gr} r_2 \in G$.

Proof: 1) If the two derivations {terminates by a step at top or is empty}. Thus

$$r_1 \overset{w}{\leadsto}^? d_0 (\overset{w}{\leadsto}^+_{[st]} \overset{w}{\leadsto}^+_{[\epsilon]} d_k)^* t_1$$

By applying lemma A.11 and proposition A.13 as often as necessary (we don't detail), we get $r_1 \xrightarrow{gr}^* t_1 \in G$. Similary, we get $r_2 \xrightarrow{gr}^* t_2 \in G$.
Since t_1, t_2 are unifiable right terms of G, from lemma /reflemme02 we obtain $t_1 \xleftrightarrow{gr} t_2 \in G$, which prove the property.

2) Otherwise there is at least one step $\overset{w}{\rightsquigarrow}_{[st]}$ that ends one of the derivations, say the first :

$$r_1 \overset{w}{\rightsquigarrow}^{?} d_0(\overset{w}{\rightsquigarrow}^{+}_{[st]}\overset{w}{\rightsquigarrow}^{+}_{[\epsilon]} d_k)^* d_p \overset{w}{\rightsquigarrow}^{+}_{[st]} t_1$$
$$r_2 \overset{w}{\rightsquigarrow}^{?} d'_0(\overset{w}{\rightsquigarrow}^{+}_{[st]}\overset{w}{\rightsquigarrow}^{+}_{[\epsilon]} d'_k)^* d'_{p'} \overset{w}{\rightsquigarrow}^{*}_{[st]} t_2$$

By applying lemma A.11 and proposition A.13 as often as necessary (we don't detail), we get

$$r_1 \xrightarrow{gr}{}^* d_p \in G$$
$$r_2 \xrightarrow{gr}{}^* d'_{p'} \in G$$

d_p and $d'_{p'}$ are (sub)-right-hand-sides. Either $d_p = d'_{p'}$, or from proposition A.14, $d_p \xleftrightarrow{gr} d'_{p'}$.
□

A.2.2 Relation between rewriting and the graph

From a basic rewriting derivation, it follows a basic narrowing derivation ([Hullot-80]), then a weak narrowing derivation (proposition A.8). Thus, the previous properties are transformed into the following ones, needed to prove the completeness.

Theorem A.18 *Let r be a right term of G, l a left term of G, and θ a normalized substitution. If $\theta(r) \rightarrow^{+}_{[st]} t$ by a basic rewriting derivation, and t, l are unifiable then $r \xrightarrow{gr} l \in G$.*

Proof: From the lemma that relies basic rewriting and basic narrowing [Hullot-80], there is a basic narrowing derivation $r \rightsquigarrow^{+}_{[st]} s$ such that $t = \sigma_0(s)$.
From proposition A.8 we get $r \overset{w}{\rightsquigarrow}^{+}_{[st]} s_1$ and $s = \sigma_1(s_1)$. Thus $t = \sigma_0.\sigma_1(s_1)$.
Let γ be a unifier of t and l. We have $\gamma(l) = \gamma(t) = \gamma.\sigma_0.\sigma_1(s_1)$ and we can suppose $V(s_1) \cap V(l) = \emptyset$, hence s_1 and l are unifiable. From proposition A.13 there exists $r \xrightarrow{gr} l \in G$. □

Theorem A.19 *Let r be a right term of G, l a left term of G, and θ a normalized substitution. If $\theta(r) \rightarrow^{*} t$ by a basic rewriting derivation, and t, l are unifiable then there exists a path $r \xrightarrow{gr}{}^* l \in G$.*

Proof: From the lemma that relies basic rewriting and basic narrowing [Hullot-80], there is a basic narrowing derivation $r \rightsquigarrow^{*} s$ such that $t = \sigma_0(s)$.
From proposition A.8 we get $r \overset{w}{\rightsquigarrow}^{*} s_1$ and $s = \sigma_1(s_1)$. Thus $t = \sigma_0.\sigma_1(s_1)$.
Let γ be a unifier of t and l. We have $\gamma(l) = \gamma(t) = \gamma.\sigma_0.\sigma_1(s_1)$ and we can suppose $V(s_1) \cap V(l) = \emptyset$, hence s_1 and l are unifiable. From proposition A.15 there exists $r \xrightarrow{gr}{}^* l \in G$. □

Theorem A.20 *Let r_1, r_2 be right terms of G. If there exists a normalized substitution θ such that $\theta(r_1) \rightarrow^{n_1}_{[st]} u \;{}^{n_2}_{[st]}\!\leftarrow \theta(r_2)$ where u is in normal form and $n_1 + n_2 > 0$, then there is a path $r_1 \xleftrightarrow{gr} r_2$ in G.*

Proof: We can choose an innermost strategy in the above rewriting derivations, which is basic. From the lemma that relies basic rewriting and basic narrowing [Hullot-80], there are some basic narrowing derivations

$$r_1 \rightsquigarrow^{n_1}_{[st]} u_1$$
$$r_2 \rightsquigarrow^{n_2}_{[st]} u_2$$

with $u = \sigma_1(u_1)$ and $u = \sigma_2(u_2)$.
From proposition A.8 we get $r_1 \overset{w\, n_1}{\leadsto}_{[st]} t_1$, $r_2 \overset{w\, n_2}{\leadsto}_{[st]} t_2$ with $u_1 = \gamma_1(t_1)$, $u_2 = \gamma_2(t_2)$.
Therefore $u = \sigma_1.\gamma_1(t_1) = \sigma_2.\gamma_2(t_2)$.
We can suppose that $V(t_1) \cap V(t_2) = \emptyset$, then t_1 and t_2 are unifiable.
From proposition A.14 there is a path $r_1 \overset{gr}{\longleftrightarrow} r_2$ in G. $\square$

Theorem A.21 *Let r_1, r_2 be right terms of G. If there exists a normalized substitution θ such that $\theta(r_1) \rightarrow^*_{[st]} u \; {}^*\!\!\leftarrow \theta(r_2)$ where u is in normal form, then there is a path $r_1 \overset{gr}{\longleftrightarrow}{}^? \; {}^*\!\overset{gr}{\longleftarrow} r_2$ in G.*

Proof: We can choose an innermost strategy in the above rewriting derivations, which is basic. From the lemma that relies basic rewriting and basic narrowing [Hullot-80], there are some basic narrowing derivations

$$r_1 \leadsto^*_{[st]} u_1$$
$$r_2 \leadsto^* u_2$$

with $u = \sigma_1(u_1)$ and $u = \sigma_2(u_2)$.
From proposition A.8 we get $r_1 \overset{w\, *}{\leadsto}_{[st]} t_1$, $r_2 \overset{w\, *}{\leadsto} t_2$ with $u_1 = \gamma_1(t_1)$, $u_2 = \gamma_2(t_2)$.
Therefore $u = \sigma_1.\gamma_1(t_1) = \sigma_2.\gamma_2(t_2)$.
We can suppose that $V(t_1) \cap V(t_2) = \emptyset$, then t_1 and t_2 are unifiable.
From proposition A.16 there is a path $r_1 \overset{gr}{\longrightarrow}{}^* \; \overset{gr}{\longleftrightarrow}{}^? \; {}^*\!\overset{gr}{\longleftarrow} r_2$ in G. $\square$

Theorem A.22 *Let r_1, r_2 be right terms of G. If there exists a normalized substitution θ such that $\theta(r_1) =_R \theta(r_2)$, then there is a confluent path $r_1 \overset{gr}{\longrightarrow}{}^* \; \overset{gr}{\longleftrightarrow}{}^? \; {}^*\!\overset{gr}{\longleftarrow} r_2$ in G.*

Proof: Since the rewriting system is confluent (Church-Rosser property)

$$\theta(r_1) \rightarrow^* u \; {}^*\!\!\leftarrow \theta(r_2)$$

We can suppose that u is in normal form, then we can choose an innermost strategy in the above rewriting derivations, which is basic. From the lemma that relies basic rewriting and basic narrowing [Hullot-80], there are some basic narrowing derivations

$$r_1 \leadsto^* u_1$$
$$r_2 \leadsto^* u_2$$

with $u = \sigma_1(u_1)$ and $u = \sigma_2(u_2)$.
From proposition A.8 we get $r_1 \overset{w\, *}{\leadsto} t_1$, $r_2 \overset{w\, *}{\leadsto} t_2$ with $u_1 = \gamma_1(t_1)$, $u_2 = \gamma_2(t_2)$.
Therefore $u = \sigma_1.\gamma_1(t_1) = \sigma_2.\gamma_2(t_2)$.
We can suppose that $V(t_1) \cap V(t_2) = \emptyset$, then t_1 and t_2 are unifiable.
From proposition A.17 there is a path $r_1 \overset{gr}{\longrightarrow}{}^* \; \overset{gr}{\longleftrightarrow}{}^? \; {}^*\!\overset{gr}{\longleftarrow} r_2$ in G. $\square$

A.3 Completeness and decreasing of the complexity

We prove here the completeness and the decreasing of the complexity. We introduce a set of protected variables W, which will permit subsequently to plung the equation into the unificand without variables conflict. (W will be the set of unificand variables).

Definition A.23 1. For a conjunctive factor C we define $\|C\|$ by :

- if C is trivial, $||C||$ is recursively defined by :
- $||t \doteq t'|| = 0$
- $||t \doteq_R t'||$ is the maximal length of rewriting derivations issued from $t \doteq_R t'$

- $||t \stackrel{\rightarrow}{=}_R t'||$ is the maximal length of rewriting derivations issued from t and leading to t'
- $||C_1 \wedge C_2|| = ||C_1|| + ||C_2||$.
- if C is not trivial, $||C|| = 0$.

2. We define $|C|$ by :
- $|t|$ is the number of elements of $O(t)$
- $|t \doteq t'| = 0$
- $|t \doteq_R t'| = |t| + |t'|$
- $|t \stackrel{\rightarrow}{=}_R t'| = |t| + |t'|$
- $|C_1 \wedge C_2| = |C_1| + |C_2|$.

3. $\#C$ is the number of the $\doteq_R$ equations of C.
4. $\vec{\#}C$ is the number of the $\stackrel{\rightarrow}{=}_R$ equations of C.

Let θ be a substitution and C a conjunctive factor, we define the measure of complexity c by the quadruplets of integer $c(\theta, C) = (||\theta C||, |C|, \#C, \vec{\#}C)$. We obtain a nœtherian order about this measure by lexicographique extension left toward right of integer order. $\diamond$

A.3.1 About one equation

Lemma A.24 Let $s \doteq_R t$ be an equation and W a set of variables. Let $\theta \in SOL(s \doteq_R t)$ and consider the step of narrowing $s \doteq_R t \longrightarrow S'$ by (Narrow-$\doteq_R$).
Then there exists $\theta' \in SOL(S')$ such that

$$\theta' = \theta \ \ [V \cup W]$$

and $\theta' S'$ contains a trivial conjunctive factor $\theta' C'$ such that

$$c(\theta', C') < c(\theta, s \doteq_R t)$$

Proof: By hypothesis : $\theta \in SOL(s \doteq_R t)$ then $\theta s =_R \theta t$, θ is normalized, and by confluence $\theta s \rightarrow^* u \, ^*\!\leftarrow \theta t$ where u is in normal form. We can suppose the strategy is innermost, then basic.
Let us study different cases according to the nature of s and t.

- If s and t aren't variables.

1. If no rewrite rule is applied at top :
We must have $top(\theta s) = top(\theta t)$, and as s and t are not variables, we have $top(s) = top(t)$ and $\forall i = 1 \ldots n$, $\theta s_i \doteq_R \theta t_i$. Let's write $\theta' = \theta$ and $C' = (s_1 \doteq_R t_1 \wedge \ldots \wedge s_n \doteq_R t_n)$. We have $\theta' \in SOL(C')$. Either s and t are syntacticaly unifiable by θ and there is in the graph an $(\stackrel{gr}{\longleftrightarrow})$ arrow between s and t (because s and t are right-hand-sides and from lemma A.10). Or s and t are not syntacticaly unifiable by θ and no rewrite rule is applied at top, therefore at least one rewrite rule is applied on a subterm; s and t being right-hand-sides, from theorem A.20 we deduce that the path $s \stackrel{gr}{\longleftrightarrow} t$ is in the graph. Then C' is a conjunctive factor of S' and $\theta' \in SOL(S')$. $\theta' C'$ is a trivial conjunctive factor of S'.
Let's prove now that $c(\theta', C') < c(\theta, s \doteq_R t)$.
We have : $\|\theta(s \doteq_R t)\| \geq \sum_{i=1}^n \|\theta'(s_i \doteq_R t_i)\|$.

$$\begin{aligned} \text{We have } |s \doteq_R t| &= |s| + |t| \\ &= |s_1| + |t_1| + \cdots + |s_n| + |t_n| + 2 \qquad \text{because } s \text{ and } t \text{ are not variables.} \\ &= \textstyle\sum_{i=1}^n |s_i \doteq_R t_i| \ + 2 \\ &> \textstyle\sum_{i=1}^n |s_i \doteq_R t_i| \end{aligned}$$

Then we have $c(\theta', C') < c(\theta, s \doteq_R t)$.

2. If some rewrite rules are applied at the top of θs :
Let $l \to r$ be the first one. We have :

$$\theta s \xrightarrow{*R} \sigma l \to \sigma r \xrightarrow{*R} u \xleftarrow{*R} \theta t$$

Let's write $\theta' = \theta\ \ [V \cup W]$ and $\theta' = \sigma\ \ [V(l)]$ (because $(V(l)) \cap (V \cup W) = \emptyset$). Since the above derivation is innermost, σ is normalized, then θ' is normalized. The derivation $\theta' s \xrightarrow{*R} \theta' l$ is basic and there isn't any reduction at the top of $\theta' s$, then $\theta' s_1 \xrightarrow{*R} \theta' l_1 \ldots \theta' s_n \xrightarrow{*R} \theta' l_n$ by basic derivations, we deduce that $\theta' \in SOL(s_1 \stackrel{\rightarrow}{=}_R l_1 \wedge \ldots \wedge s_n \stackrel{\rightarrow}{=}_R l_n)$. We have also $\theta' r \xrightarrow{*R} u \xleftarrow{*R} \theta' t$ then $\theta' \in SOL(r \doteq_R t)$.
Let's write $C' = (s_1 \stackrel{\rightarrow}{=}_R l_1 \wedge \ldots \wedge s_n \stackrel{\rightarrow}{=}_R l_n \wedge r \doteq_R t)$. We have $\theta' \in SOL(C')$. Let's prove that C' is a conjunctive factor of S'. We have $\theta' r =_R \theta' t$, r and t are two right-hand-sides of the graph, therefore, by theorem A.22, there is a confluent path in the graph : $r \xrightarrow{gr} {}^* \xleftrightarrow{gr} {}^? {}^* \xleftarrow{gr} t$. Since $l \to r$ is a rewrite rule then the path $l \to r$ is in the graph. The rule $l \to r$ is applied at the top of θs then, either θs and l are directly unifiable and the path $s \xrightarrow{gr} l$ is in the graph (s is a right-hand-side, l is a left-hand-side, and from lemma A.9), or θs and l are unifiable after applying some rewrite rules on the subterms of θs, then according to theorem A.18, the path $s \xrightarrow{gr} l$ is in the graph. Therefore the path $s \xrightarrow{gr} l \to r \xrightarrow{gr} {}^* \xleftrightarrow{gr} {}^? {}^* \xleftarrow{gr} t$ is in the graph. Then C' is a conjunctive factor of S', thus $\theta' \in SOL(S')$ and $\theta' C'$ is a trivial conjunctive factor of $\theta' S'$.
Let's prove now that $c(\theta', C') < c(\theta, s \doteq_R t)$.
Suppose a derivation of maximal length for :

$$\theta s \xrightarrow{*R}_{max} \sigma l$$
$$\sigma r \xrightarrow{*R}_{max} (\sigma r)\downarrow$$
and $$\theta t \xrightarrow{*R}_{max} (\theta t)\downarrow$$

Let :

$$D = (\theta s \xrightarrow{*R} \sigma l \longrightarrow \sigma r \xrightarrow{*R} u \xleftarrow{*R} \theta t)$$
$$D_{max} = (\theta s \xrightarrow{*R}_{max} \sigma l \longrightarrow \sigma r \xrightarrow{*R}_{max} (\sigma r)\downarrow\ et\ (\theta t)\downarrow \xleftarrow{*R}_{max} \theta t)$$

We have $length(D) \leq length(D_{max}) \leq \|\theta(s \doteq_R t)\|$

Let's write $A = \sum_{i=1}^{n} length(\theta s_i \xrightarrow{*R}_{max} \sigma l_i)$
$A \leq length(\theta s \xrightarrow{*R}_{max} \sigma l)$
and $B = \|\theta'(r \doteq_R t)\|$
$B = length(\sigma r \xrightarrow{*R}_{max} (\sigma r)\downarrow) + length(\theta t \xrightarrow{*R}_{max} (\theta t)\downarrow)$

We have :

$$\begin{aligned} length(D_{max}) = \quad & length(\theta s \xrightarrow{*R}_{max} \sigma l) + 1 + length(\sigma r \xrightarrow{*R}_{max} (\sigma r)\downarrow) \\ & + length(\theta t \xrightarrow{*R}_{max} (\theta t)\downarrow). \end{aligned}$$

Then $\|\theta' C'\| = A + B < length(D_{max}) \leq \|\theta(s \doteq_R t)\|$.
We have : $c(\theta', C') < c(\theta, s \doteq_R t)$.

3. If no rewrite rule is applied at the top of θs, but some are applied at the top of θt.
Consider the first rule $l \to r$ applied at top. We have :

$$\theta s \xrightarrow{*R} u \xleftarrow{*R} \sigma r \leftarrow \sigma l \xleftarrow{*R} \theta t$$

Let's write $\theta' = \theta\ \ [V \cup W]$ and $\theta' = \sigma\ \ [V(l)]$ (because $(V(l)) \cap (V \cup W) = \emptyset$). Since the above derivation is innermost, σ is normalized, then θ' is normalized. We have $\theta' t \xrightarrow{*R} \theta' l$ by basic derivation and there isn't any reduction at the top of $\theta' t$, then $\theta' t_1 \xrightarrow{*R} \theta' l_1 \ldots \theta' t_n \xrightarrow{*R} \theta' l_n$ by basic derivations, we deduce that $\theta' \in SOL(t_1 \stackrel{\rightarrow}{=}_R l_1 \wedge \ldots \wedge t_n \stackrel{\rightarrow}{=}_R l_n)$. We have also $\theta' s \xrightarrow{*R} u \xleftarrow{*R} \theta' r$ then $\theta' \in SOL(s \doteq_R r)$.
Let's write $C' = (t_1 \stackrel{\rightarrow}{=}_R l_1 \wedge \ldots \wedge t_n \stackrel{\rightarrow}{=}_R l_n \wedge s \doteq_R r)$. We have $\theta' \in SOL(C')$. Let's prove that C' is a conjunctive factor of S'. The rule $l \to r$ is applied at the top of θt then, either θt and l are unifiable directly and the path $t \xrightarrow{gr} l$ is in the graph, (because t is a right-hand-side, l is a left-hand-side, and from lemma A.9), or θt and l are unifiable after applying some rewrite rules on some subterms of θt, then according to theorem A.18, the path $t \xrightarrow{gr} l$ is in the graph. Since $l \to r$ is a rewrite rule then the path $l \to r$ is in the graph. We have $\theta' s \xrightarrow{*R} u \xleftarrow{*R} \theta' r$, and there isn't any rewrite rule applied at the top of $\theta' s$ then according to the theorem A.21, the path $s \overset{gr}{\longleftrightarrow}{}^{?} \ {}^{*}\!\xleftarrow{gr} r$ is in the graph. Therefore the path $s \overset{gr}{\longleftrightarrow}{}^{?} \ {}^{*}\!\xleftarrow{gr} r \leftarrow l \xleftarrow{gr} t$ is in the graph. Then C' is a conjunctive factor of S', thus $\theta' \in SOL(S')$ and $\theta' C'$ is a trivial conjunctive factor of $\theta' S'$.
The proof of $c(\theta', C') < c(\theta, s \doteq_R t)$ is similar to the previous one.

• If t is a variable y and s is not a variable.
θ is normalized then θy is the normal form of θs.

1. If θs is in normal form, we have $\theta y = \theta s$. Let's write $\theta' = \theta$ and $C' = (y \doteq s)$, $\theta' C'$ is a trivial conjunctive factor of $\theta' S'$ Let's prove now that $c(\theta', C') < c(\theta, s \doteq_R y)$.
We have $\|\theta' C'\| = 0 \leq \|\theta(s \doteq_R y)\|$, $|C'| = 0$ and $|s \doteq_R y| \geq 1$ because s is not a variable, therefore $|C'| < |s \doteq_R y|$. Then we have $c(\theta', C') < c(\theta, s \doteq_R y)$.

2. Else θs is not in normal form, therefore, at least one rewrite rule is applied on θs or on a subterm of θs. let's consider the first $l \to r$, on an indifferent basic strategie, applied at the occurrence v in θs. Then we have $\theta s|_v = \sigma l$ and $\theta s[v \leftarrow \sigma r] \xrightarrow{*R} \theta y$. Let's write $\theta' = \theta\ \ [V \cup W]$ and $\theta' = \sigma\ \ [V(l)]$ (because $(V(l)) \cap (V \cup W) = \emptyset$). Since the above derivation is innermost, σ is normalized, then θ' is normalized. We have $\theta' s|_v = \theta' l$ and $\theta' s[v \leftarrow \theta' r] \xrightarrow{*R} \theta' y$ then $\theta'(s[v \leftarrow r]) \xrightarrow{*R} \theta' y$. Let's write $C' = (s|_v \doteq l \wedge s[v \leftarrow r] \doteq_R y)$. We have $\theta' \in SOL(C')$ and C' is a conjunctive factor of S' thus $\theta' \in SOL(S')$ and $\theta' C'$ is a trivial conjunctive factor of $\theta' S'$.
Let's prove now that $c(\theta', C') < c(\theta, s \doteq_R y)$.
We have $\|\theta(s \doteq_R y)\| = length(\theta s \xrightarrow{*R}_{max} (\theta s) \downarrow) \geq length(\theta(s) \longrightarrow \theta(s)[v \leftarrow \sigma r] \xrightarrow{*R}_{max} (\theta s) \downarrow) < length(\theta(s)[v \leftarrow \sigma r] \xrightarrow{*R}_{max} (\theta s) \downarrow) = \|\theta' C'\|$ because $\|\theta'(s|_v \doteq l)\| = 0$ and $length(\theta' y \xrightarrow{*R}_{max} (\theta' y) \downarrow) = 0$ (y is a variable and θ' is normalised). Then we have $c(\theta', C') < c(\theta, s \doteq_R y)$.

• If s is a variable and t is not a variable :
It's the same reasoning as the precedent one.

• If s and t are variables (x and y) :
θ is solution of $x \doteq_R y$ then $\theta x \doteq_R \theta y$ and θ is normalized, therefore $\theta x = \theta y$. Let's write $\theta' = \theta$ and $C' = (x \doteq y)$. C' is a conjunctive factor of S' and $\theta' \in SOL(C')$ then $\theta' C'$ is a trivial conjunctive factor of $\theta' S'$.
Let's prove now that : $c(\theta', C') < c(\theta, x \doteq_R y)$.
We have : $0 = \|\theta'(x \doteq y)\| \leq \|\theta'(x \doteq_R y)\| = \|\theta(x \doteq_R y)\|$

$0 = |x \doteq y| = |x \doteq_R y| = 0$
$\#(x \doteq y) = 0$
$\#(x \doteq_R y) = 1$
Then we obtain $c(\theta', C') < c(\theta, x \doteq_R y)$. $\square$

Lemma A.25 Let $s \overset{\rightarrow}{=}_R t$ be an equation and W a set of variables. Let $\theta \in SOL(s \overset{\rightarrow}{=}_R t)$ and consider the step of narrowing $s \overset{\rightarrow}{=}_R t \longrightarrow S'$ by (Narrow- $\overset{\rightarrow}{=}_R$).
Then there exists $\theta' \in SOL(S')$ such that

$$\theta' = \theta \ \ [V \cup W]$$

and $\theta' S'$ contains a trivial conjunctive factor $\theta' C'$ such that

$$c(\theta', C') < c(\theta, s \overset{\rightarrow}{=}_R t)$$

Proof: By hypothesis : $\theta \in SOL(s \overset{\rightarrow}{=}_R t)$ then $\theta s \xrightarrow{*R} \theta t$ by a basic derivation and θ is normalized.
We study different cases according to the nature of s and t.

- If s and t aren't variables.

1. If no rewrite rule is applied at the top of θs :
We must have $top(\theta s) = top(\theta t)$, and as s and t are not variables, we have $top(s) = top(t)$ and $\forall i = 1 \dots n, \theta s_i \xrightarrow{*R} \theta t_i$. The derivation $\theta s \xrightarrow{*R} \theta t$ is basic and there isn't any derivation at the top of θs, then $\forall i = 1 \dots n,\ \theta s_i \xrightarrow{*R} \theta t_i$ is a basic derivation. Let's write $\theta' = \theta$ and $C' = (s_1 \overset{\rightarrow}{=}_R t_1 \wedge \dots \wedge s_n \overset{\rightarrow}{=}_R t_n)$. We have $\theta' \in SOL(C')$. Either s and t are syntacticaly unifiable by θ and there is in the graph an $(\xrightarrow{gr})$ arrow between s and t (because s is a left-hand-side, t is a right-hand-side, and from lemma A.9). Or s and t are not syntacticaly unifiable by θ and no rewrite rule is applied at the top of θs, therefore some rewrite rules are applied on some subterms of θs, then according to the theorem A.18 (s is a left-hand-sides and t is a right-hand-sides of the graph) we deduce that the path $s \xrightarrow{gr} t$ is in the graph. Then C' is a conjunctive factor of S' and $\theta' \in SOL(S')$. $\theta' C'$ is a trivial conjunctive factor of S'.
Let's prove now that $c(\theta', C') < c(\theta, s \overset{\rightarrow}{=}_R t)$.
We have : $\|\theta(s \overset{\rightarrow}{=}_R t)\| \geq \sum_{i=1}^{n} \|\theta'(s_i \overset{\rightarrow}{=}_R t_i)\|$.

We have
$$\begin{aligned} |s \overset{\rightarrow}{=}_R t| &= |s| + |t| \\ &= |s_1| + |t_1| + \dots + |s_n| + |t_n| + 2 \qquad \text{because } s \text{ and } t \text{ are not variables.} \\ &= \textstyle\sum_{i=1}^{n} |s_i \overset{\rightarrow}{=}_R t_i| \ + 2 \\ &> \textstyle\sum_{i=1}^{n} |s_i \overset{\rightarrow}{=}_R t_i| \end{aligned}$$

Then we have $c(\theta', C') < c(\theta, s \overset{\rightarrow}{=}_R t)$.

2. If at least one rewrite rule is applied at the top of θs :
Let $l \rightarrow r$ be the first one. We have :

$$\theta s \xrightarrow{*R} \sigma l \longrightarrow \sigma r \xrightarrow{*R} \theta t$$

Let's write $\theta' = \theta \ \ [V \cup W]$ and $\theta' = \sigma \ \ [V(l)]$ (because $(V(l)) \cap (V \cup W) = \emptyset$). Since the above derivation is innermost, σ is normalized, then θ' is normalized. We have $\theta' s \xrightarrow{*R} \theta' l$ by a basic derivation and there isn't any reduction at the top of $\theta' s$, then $\theta' s_1 \xrightarrow{*R} \theta' l_1 \dots \theta' s_n \xrightarrow{*R} \theta' l_n$ by basic derivations, we deduce that $\theta' \in SOL(s_1 \overset{\rightarrow}{=}_R l_1 \wedge \dots \wedge$

$s_n \overrightarrow{=}_R l_n$). We have also $\theta' r \xrightarrow{*R} \theta' t$ by a basic derivation then $\theta' \in SOL(r \overrightarrow{=}_R t)$.
Let's write $C' = (s_1 \overrightarrow{=}_R l_1 \wedge \ldots \wedge s_n \overrightarrow{=}_R l_n \wedge r \overrightarrow{=}_R t)$. We have $\theta' \in SOL(C')$. Let's prove that C' is a conjunctive factor of S'. We have $\theta' r \xrightarrow{*R} \theta' t$, by a basic derivation, r is a right-hand-sides and t is a left-hand-sides of the graph, therefore, by the theorem A.19, there is a confluent path in the graph : $r \xrightarrow{gr}{}^* t$. Since $l \to r$ is a rewrite rule then the path $l \to r$ is in the graph. We apply the rule $l \to r$ on top of θs then, either θs and l are directly unifiable and the path $s \xrightarrow{gr} l$ is in the graph, (s is a right-hand-sides, l is a left-hand-sides, and from lemma A.9), or θs and l are unifiable after applying some rewrite rules on the subterms of θs, then according to theorem A.18, the path $s \xrightarrow{gr} l$ is in the graph. Therefore the path $s \xrightarrow{gr} l \to r \xrightarrow{gr}{}^* t$ is in the graph. Then C' is a conjunctive factor of S', thus $\theta' \in SOL(S')$ and $\theta' C'$ is a trivial conjunctive factor of $\theta' S'$.
Let's prove now that $c(\theta', C') < c(\theta, s \overrightarrow{=}_R t)$.
Suppose a derivation of maximal length for :

$$\theta s \xrightarrow{*R}_{max} \sigma l$$

and $$\sigma r \xrightarrow{*R}_{max} \theta t$$

Let :

$$D = (\theta s \xrightarrow{*R} \sigma l \longrightarrow \sigma r \xrightarrow{*R} \theta t)$$
$$D_{max} = (\theta s \xrightarrow{*R}_{max} \sigma l \longrightarrow \sigma r \xrightarrow{*R}_{max} \theta t)$$

We have $length(D) \leq length(D_{max}) \leq \|\theta(s \overrightarrow{=}_R t)\|$

Let's write $A = \sum_{i=1}^{n} length(\theta s_i \xrightarrow{*R}_{max} \sigma l_i)$
$A \leq length(\theta s \xrightarrow{*R}_{max} \sigma l)$
$A = \sum_{i=1}^{n} length(\theta' s_i \xrightarrow{*R}_{max} \theta' l_i) = \sum_{i=1}^{n} \|\theta'(s_i \overrightarrow{=}_R l_i)\|$
and $B = \|\theta'(r \overrightarrow{=}_R t)\|$
$B = length(\sigma r \xrightarrow{*R}_{max} (\theta t))$

$\|\theta' C'\| = A + B < length(D_{max}) \leq \|\theta(s \overrightarrow{=}_R t)\|$
then we have : $c(\theta', C') < c(\theta, s \overrightarrow{=}_R t)$.

• If s is a variable y and t is not a variable.
We have $\theta y \xrightarrow{*R} \theta t$ but θ is normalized then $\theta y = \theta t$. Let's write $\theta' = \theta$ and $C' = (y \doteq t)$, $\theta' C'$ is a trivial conjunctive factor of $\theta' S'$.
Let's prove now that $c(\theta', C') < c(\theta, y \overrightarrow{=}_R t)$. We have $0 = \|\theta'(y \doteq t)\| \leq \|\theta(y \overrightarrow{=}_R t)\|$, $0 = |y \doteq t| < |y \overrightarrow{=}_R t|$ because t is not a variable. Then $c(\theta', C') < c(\theta, y \overrightarrow{=}_R t)$.

• If t is a variable y and s is not a variable.
We have $\theta s \xrightarrow{*R} \theta y$ by a basic derivation, θ is normalized then θy is a normal form of θs.

1. If θs is in normal form, we have $\theta y = \theta s$. Let's write $\theta' = \theta$ and $C' = (y \doteq s)$, $\theta' C'$ is a trivial conjunctive factor of $\theta' S'$ Let's prove now that $c(\theta', C') < c(\theta, s \overrightarrow{=}_R y)$.
We have $\|\theta' C'\| = 0 \leq \|\theta(s \overrightarrow{=}_R y)\|$, $|C'| = 0$ and $|s \overrightarrow{=}_R y| \geq 1$ because s is not a variable, therefore $|C'| < |s \overrightarrow{=}_R y|$. Then we have $c(\theta', C') < c(\theta, s \overrightarrow{=}_R y)$.

2. Else θs is not in normal form, therefore, at least one rewrite rule is applied on θs or on a subterm of θs. let's consider the first $l \to r$, on an indifferent basic strategie, applied at the occurrence v in θs. Then we have $\theta s|_v = \sigma l$ and $\theta s[v \leftarrow \sigma r] \xrightarrow{*R} \theta y$. Let's write

$\theta' = \theta \quad [V \cup W]$ and $\theta' = \sigma \quad [V(l)]$. $((V(l)) \cap (V \cup W) = \emptyset)$ We have $\theta' s|_v = \theta' l$ and $\theta' s[v \leftarrow \theta' r] \xrightarrow{*R} \theta' y$ then $\theta'(s[v \leftarrow \theta' r]) \xrightarrow{*R} \theta' y$. Let's write $C' = (s|_v \doteq l \wedge s[v \leftarrow r] \stackrel{\rightarrow}{=}_R y)$. We have $\theta' \in SOL(C')$ and C' is a conjunctive factor of S' thus $\theta' \in SOL(S')$ and $\theta' C'$ is a trivial conjunctive factor of $\theta' S'$.
Let's prove now that $c(\theta', C') < c(\theta, s \stackrel{\rightarrow}{=}_R y)$.
We have $\|\theta(s \stackrel{\rightarrow}{=}_R y)\| = length(\theta s \xrightarrow{*R}_{max} (\theta s) \downarrow) \geq length(\theta(s) \longrightarrow \theta(s)[v \leftarrow \sigma r] \xrightarrow{*R}_{max} (\theta s) \downarrow) < length(\theta(s)[v \leftarrow \sigma r] \xrightarrow{*R}_{max} (\theta s) \downarrow) = \|\theta' C'\|$ because $\|\theta'(s|_v \doteq l)\| = 0$ Then we have $c(\theta', C') < c(\theta, s \stackrel{\rightarrow}{=}_R y)$.

- If s and t are variables (x and y) :

θ is solution of $x \stackrel{\rightarrow}{=}_R y$ then $\theta x \xrightarrow{*R} \theta y$ and θ is normalized, therefore $\theta x = \theta y$. Let's write $\theta' = \theta$ and $C' = (x \doteq y)$. C' is a conjunctive factor of S' and $\theta' \in SOL(C')$ then $\theta' C'$ is a trivial conjunctive factor of $\theta' S'$.
Let's prove now that : $c(\theta', C') < c(\theta, x \stackrel{\rightarrow}{=}_R y)$.
We have : $0 = \|\theta'(x \doteq y)\| \leq \|\theta'(x \stackrel{\rightarrow}{=}_R y)\| = \|\theta(x \stackrel{\rightarrow}{=}_R y)\|$
$0 = |x \doteq y| = |x \stackrel{\rightarrow}{=}_R y| = 0$
$\#(x \doteq y) = 0 = \#(x \stackrel{\rightarrow}{=}_R y)$
$\vec{\#}(x \doteq y) = 0$
$\vec{\#}(x \stackrel{\rightarrow}{=}_R y) = 1$
Then we obtain $c(\theta', C') < c(\theta, x \stackrel{\rightarrow}{=}_R y)$. □

A.3.2 About unificands

Proposition A.26 *(Completeness)*
Let $S \longrightarrow S'$ be a step of narrowing and $\theta \in SOL(S)$. Then $SOL(S) \subseteq SOL(S')[V \cup V(S)]$.

Proof: According to the precedent lemmas with $W = V(S)$. □

Proposition A.27 *Let $S \longrightarrow S'$ be a step of simplication, then $SOL(S) = SOL(S')$.*

Proof: It is similar to the correctness proof of simplication rules. □

Notation: S_{dc} denotes the disjunctive form of the unificand S. We have $SOL(S_{dc}) = SOL(S)$. The notation $Q[e]$ where Q is a conjunctive factor and e is an equation means that e appears inside Q. $S[p \leftarrow S']$ is the unificand S in which the sub-unificand at occurrence p is replaced by S'.

For the proofs, we define the notions of antecedent and residu, as a restriction of the definition of [Huet-Levy-79].

Definition A.28 Consider one step of (Distributivity)

$$\wedge S_1 \dots S_i[e] \dots S_n(\vee Q_1 \dots Q_j[e"] \dots Q_p) \longrightarrow$$
$$\vee(\wedge S_1 \dots S_i[e] \dots S_n Q_1) \dots (\wedge S_1 \dots S_i[e] \dots S_n Q_j[e"]) \dots (\wedge S_1 \dots S_i[e] \dots S_n Q_p)$$

and note p, p' the occurrences of appearance of $e, e"$ in the left unificand (be careful, they are occurrences of equations in the unificand); and let $q_1, \dots, q_n$ and q' be the occurrences of appearance of e and $e"$ in the result unificand. We say that $q_1, \dots, q_n, q'$ are **residus** of p, p', and p, p' are **antecedents** of $q_1, \dots, q_n$. We'll also say that the equations $e, e"$ in the resulting unificand are residus of equations $e, e"$ in the left unificand.
We extend this to the notions of antecedent and residu through several steps of (Distributivity).
◇

Remark : It's easy to show that two different residus of e in S_{dc} can't belong to the same conjunctive factor of S_{dc}.

Lemma A.29 Let S be an unificand, p the occurrence of an equation in S and $q_1, \ldots, q_n$ the residus of p in S_{dc}. Then

1. $S \longrightarrow_{[p]} S'$ implies $S_{dc} \xrightarrow{*}_{[q_1,\ldots,q_n]} S''$ and $S''_{dc} = S'_{dc}$.
2. $S_{dc} \xrightarrow{*}_{[q_1,\ldots,q_n]} S''$ implies $S \longrightarrow_{[p]} S'$ and $S'_{dc} = S''_{dc}$.

Remark : Recall that the unificands are always supposed to be in flattened form. Thus an unificand in disjunctive-conjunctive form is a tree whose depth is at most two; and its occurrences are integer lists of length two. The first integer represents the position of a conjunctive factor in the unificand, the second gives the position of an equation in this conjunctive factor. For example, in $S = \vee e_1(\wedge e_2 e_3 e_4)e_5$ the equation e_2 appears to occurrence 2.1 .
The proof of the below proposition takes one's inspiration from [Rety-88].

Proposition A.30 *(decreasing of the complexity by narrowing about the disjunctive form of an unificand)*
Let S be an unificand, $\theta \in SOL(S)$, and e be an $\doteq_R$ or $\vec{=}_R$ equation of S. Let $S \longrightarrow S'$ be a step of narrowing issued from e, and according to proposition A.26 consider $\theta' \in SOL(S')$ such that $\theta' = \theta \ \ [V]$.
Then for each trivial conjunctive factor $\theta S_{dc}|i$ which have a residu q of e (that is $q = i.j$ and θ solution of e), the unificand $\theta' S'_{dc}$ has a trivial conjunctive factor $\theta' C'$ such that

$$c(\theta', C') < c(\theta, S_{dc}|i)$$

Proof: By hypothesis $\theta S_{dc}|q = \theta e$ is trivial, therefore $\theta \in SOL(e)$.

Let S'_0 be an unificand obtained by narrowing issued from e, that is $e \longrightarrow S'_0$.
According to lemma A.29, consider the step of narrowing issued from S_{dc} at the occurrence q:

$$S_{dc} \longrightarrow_{[q]} S_0 = S_{dc}[q \leftarrow S'_0]$$

S_{dc} and S_0 are written:

$$S_{dc} = \vee \ldots \vee (\wedge_m e_m \wedge e)$$
$$S_0 = \vee \ldots \vee (\wedge_m e_m \wedge S'_0)$$

where $S_{dc}|i = (\wedge_m e_m \wedge e)$. Therefore by hypothesis $\theta(\wedge_m e_m \wedge e)$ is trivial, then $\theta \in SOL(\wedge_m e_m \wedge e)$.

From lemma A.24 and lemma A.25 there exists $\theta'_0 \in SOL(S'_0)$ such that $\theta'_0 = \theta \ \ [V \cup V(S_{dc})]$ and $\theta'_0 S'_0$ have a trivial conjontive factor $\theta'_0 C'_0$ such that $c(\theta'_0, C'_0) < c(\theta, e)$. Therefore $\theta'_0 \in SOL(S'_0)$, and since $\theta'_0 = \theta \ \ [V(S_{dc})]$ we have $\theta'_0 \in SOL(\wedge_m e_m \wedge S'_0)$. It results $\theta'_0 \in SOL(S_0)$.

Since S'_0 is obtained by the narrowing rule, it is in disjunctive-conjunctive form and is written $S'_0 = \vee_k C_k \vee C'_0$. Then $S_0|i$ is written:

$$S_0|i = \wedge_m e_m \wedge S'_0 = \wedge_m e_m \wedge (\vee_k C_k \vee C'_0)$$

By applying the distributivity we see that $(S_0|i)_{dc}$ contains the conjunctive factor:

$$C_0 = \wedge_m e_m \wedge C'_0$$

Let's show that $\theta'_0 C_0$ is trivial and $c(\theta'_0, C_0) < c(\theta, S_{dc}|i)$. Recall that $S_{dc}|i = \wedge_m e_m \wedge e$. By hypothesis $\theta S_{dc}|i$ is trivial, and as it is equal to $\theta'_0 S_{dc}|i$ it results that $\theta'_0(\wedge_m e_m)$ is trivial.

We know moreover that $\theta'_0 C'_0$ is trivial, therefore $\theta'_0 C_0$ is trivial.
Compare now $c(\theta, S_{dc}|i)$ with $c(\theta'_0, C_0)$.

$$c(\theta, S_{dc}|i) = c(\theta, \wedge_m e_m \wedge e) = \sum_m c(\theta, e_m) + c(\theta, e) = \sum_m c(\theta'_0, e_m) + c(\theta, e)$$

$$c(\theta'_0, C_0) = c(\theta'_0, \wedge_m e_m \wedge C'_0) = \sum_m c(\theta'_0, e_m) + c(\theta'_0, C'_0)$$

We have seen that $c(\theta'_0, C'_0) < c(\theta, e)$, therefore $c(\theta'_0, C_0) < c(\theta, S_{dc}|i)$.

Let's show that S'_{dc} contains the conjunctive factor C_0. Let $q_1 = i_1.j_1, \ldots, q_n = i_n.j_n$ be the other residus of e in S_{dc}. From lemma A.29

$$S_{dc} \longrightarrow^*_{[q,q_1,\ldots,q_n]} S_n = S_{dc}[q \leftarrow S'_0][q_1 \leftarrow S'_0] \ldots [q_n \leftarrow S'_0]$$

with $(S_n)_{dc} = S'_{dc}$. As the occurrences $q, q_1, \ldots, q_n$ are in different conjunctive factors of S_{dc} , the unificand S_n is written

$$S_n = S_n|i \vee S_n|i_1 \vee \ldots \vee S_n|i_n \vee \ldots$$

Therefore

$$S'_{dc} = (S_n)_{dc} = (S_n|i)_{dc} \vee (S_n|i_1)_{dc} \vee \ldots \vee (S_n|i_n)_{dc} \vee \ldots$$

By definition, we have $S_0 = S_{dc}[q \leftarrow S'_0]$. Consequently

$$S_n|i = (S_{dc}[i.j \leftarrow S'_0][i_1.j_1 \leftarrow S'_0] \ldots [i_n.j_n \leftarrow S'_0])|i = (S_{dc}[i.j \leftarrow S'_0])|i = S_0|i$$

And as $(S_0|i)_{dc}$ contains the conjunctive factor C_0, the unificand S'_{dc} contains also C_0. Moreover $\theta'_0 C_0$ is trivial. Let's write $\theta' = \theta'_0$ and $C' = C_0$ to obtain the result. □

Proposition A.31 *(decreasing of the complexity by simplification about the disjunctive form of an unificand)*
Let S be an unificand, $\theta \in SOL(S)$. Let $S \longrightarrow S'$ be a step of simplification, and from proposition A.27 consider $\theta' \in SOL(S')$ such that $\theta' = \theta\ \ [V]$.
Then for each trivial conjunctive factor θC of θS_{dc}, the unificand $\theta' S'_{dc}$ has a trivial conjunctive factor $\theta' C'$ such that

$$c(\theta', C') \leq c(\theta, C)$$

Proof: Remark that if Q is a conjunctive factor of S, it admits some residus through (Distributivity), therefore in S_{dc}. Examine the case of each simplification rule.

1. Rule (Solve-$\doteq$). Let Q be the conjunctive factor on which is applied the rule: $Q \longrightarrow Q'$.
   - If C is a residu of Q then $C = Q \wedge \ldots$. Let's write $C' = Q' \wedge \ldots$. The factor C' is a conjunctive factor of S'_{dc}. By completeness of (Solve), there exists $\theta' \in SOL(C')$ such that $\theta' = \theta\ \ @[V@]$. And since the $\doteq$ equations have a null complexity we have $c(\theta', C') = c(\theta, C)$.
   - Else C is a conjunctive factor of S'_{dc} and we get $C' = C$, $\theta' = \theta$.

2. Rule (Del-$\doteq_R$). If the equation where is applyed the rule is in a conjunctive factor C such that θC is trivial in θS_{dc}. It is obvious that the same conjunctive factor in which the equation $a \doteq_R a$ was replaced by T is such that $\theta' C'$ is trivial in $\theta' S'_{dc}$. Else we get $C' = C$.

3. Rule (Del-$\stackrel{\rightarrow}{=}_R$). Same reasoning.

4. Rule (Del-$\doteq$). Let $x \doteq t$ be the equation on which is applied the rule. C is not a residu of $x \doteq t$ because θC is trivial and $SOL(x \doteq t) = \emptyset$. Therefore C is a conjuctive factor of S'_{dc} and we get $C' = C$, $\theta' = \theta$.

5. Rule (Del-$\wedge$-F). Same reasoning.

6. Rule (Del-$\wedge$-T). Let Q be the conjunctive factor on which is applied the rule: $Q \longrightarrow Q'$.

- If C is a residu of Q then $C = Q \wedge \ldots$. We write $C' = Q' \wedge \ldots$. The factor C' is a conjunctive factor of S'_{dc}. By completeness of (Solve), there exists $\theta' \in SOL(C')$ such that $\theta' = \theta \;\; [V]$. And since the equation T has a null complexity we have $c(\theta', C') = c(\theta, C)$.
- Else C is a conjunctive factor of S'_{dc} and we get $C' = C$, $\theta' = \theta$.

7. Rule (Distributivity). Then $S'_{dc} = S_{dc}$ and the result is trivial.

8. Rule (Del-$\vee$-F). Let $Q = \vee Q_1 \ldots Q_{i-1} F Q_{i+1} \ldots Q_p$ be the disjunctive factor on which is applied the rule.

- If the rule is applied on head in S, that is $S = Q$, then C is a factor of one $(Q_j)_{dc}$, therefore is in S'_{dc}.
- Else S and S' are written:

$$S = K[\wedge S_1 \ldots S_n(\vee Q_1 \ldots Q_{i-1} F Q_{i+1} \ldots Q_p)]$$
$$S' = K[\wedge S_1 \ldots S_n(\vee Q_1 \ldots Q_{i-1} Q_{i+1} \ldots Q_p)]$$

By applying (Distributivity) on S and S' we obtain:

$$S \longrightarrow S_{d_1} = K[\vee(\wedge S_1 \ldots S_n Q_1) \ldots (\wedge S_1 \ldots S_n F) \ldots (\wedge S_1 \ldots S_n Q_p)]$$
$$S' \longrightarrow S'_{d_1} = K[\vee(\wedge S_1 \ldots S_n Q_1) \ldots \ldots (\wedge S_1 \ldots S_n Q_p)]$$

The conjunctive factor C of S_{dc}, so of S_{d_1}, don't contain residu of $\wedge S_1 \ldots S_n F$ because by hypothesis θC is trivial. Therefore the suppression of $\wedge S_1 \ldots S_n F$ in S_{d_1}, that is the passage of S_{d_1} to S'_{d_1} don't modify C. It results that C is a conjunctive factor of S'_{dc}. Then we get $C' = C$ and $\theta' = \theta$.

9. Rule (Del-$\vee$-T). Let $Q = \vee Q_1 \ldots Q_{i-1} T Q_{i+1} \ldots Q_p$ be the disjunctive factor on which is applied the rule.

- If the rule is applied on head in S, that is $S = Q$, then $S' = T$, $\theta S'$ is trivial and with $C' = T$ we have

$$0 = c(\theta, C') \leq c(\theta, C')$$

- Else S and S' are written:

$$S = K[\wedge S_1 \ldots S_n(\vee Q_1 \ldots Q_{i-1} T Q_{i+1} \ldots Q_p)]$$
$$S' = K[\wedge S_1 \ldots S_n T]$$

By applying (Distributivity) on S we obtain:

$$S \longrightarrow S_{d_1} = K[\vee(\wedge S_1 \ldots S_n Q_1) \ldots (\wedge S_1 \ldots S_n T) \ldots (\wedge S_1 \ldots S_n Q_p)]$$

1. If C contains a residu of $\wedge S_1 \ldots S_n T$ in S_{d_1} so in S_{dc}, then C is a factor of S'_{dc}.
2. If C contains a residu of $\wedge S_1 \ldots S_n Q_j$ in S_{d_1} then in S_{dc}, the factor C is written $C = \wedge P S_1 \ldots S_n Q_j$. By hypothesis $\theta(C)$ is trivial, therefore $\theta(\wedge P S_1 \ldots S_n T)$ too. Write $C' = \wedge P S_1 \ldots S_n T$. Then C' is a factor of S'_{dc}, and we have:

$$c(\theta, C') = c(\theta, P) + \textstyle\sum_{j=1}^{n} c(\theta, S_j) \leq c(\theta, C)$$

3. Else C is a factor of S'_{dc} and we get $C' = C$.

□

Lemma A.32 Let $S_0 \longrightarrow \ldots \longrightarrow S_n \longrightarrow \ldots$ be a fair derivation and $\theta_0 \in SOL(S_0)$. Then there exists a unificand S_n in the derivation, $\theta_n \in SOL(S_n)$ such that $\theta_n = \theta_0 \;\; [V]$ and $\theta_n(S_n)_{dc}$ contains a trivial conjunctive factor $\theta_n C_n$ such that $c(\theta_n, C_n) = (0, 0, 0, 0)$.

Proof: $\theta_0(S_0)_{dc}$ has at least a trivial conjunctive factor, say $\theta_0 C_0$. At each unificand S_j in the derivation, we associate a substitution θ_j and a conjunctive factor C_j of $(S_j)_{dc}$ such that $\theta_j C_j$ is trivial, by the inductive definition :

(θ_0, C_0) is associated at S_0, and if (θ_i, C_i) is associated at S_i, from the two previous propositions, there exists a substitution $\theta_{i+1} \in SOL(S_{i+1})$ such that $\theta_{i+1} = \theta_i \ \ [V]$ and $\theta_{i+1}(S_{i+1})_{dc}$ contains a trivial conjunctive factor $\theta_{i+1}C_{i+1}$ such that:

1. If $S_i \longrightarrow_{[p_i]} S_{i+1}$ by a narrowing rule, then from proposition A.30
   - if the factor C_i of $(S_i)_{dc}$ contains a residu of p_i, we have $c(\theta_{i+1}, C_{i+1}) < c(\theta_i, C_i)$.
   - if it doesn't contain any residu, we take $C_{i+1} = C_i$, then $c(\theta_{i+1}, C_{i+1}) = c(\theta_i, C_i)$.
2. Else from the previous proposition we have

$$c(\theta_{i+1}, C_{i+1}) \leq c(\theta_i, C_i)$$

The serie of $(c(\theta_i, C_i))$ is weakly decreasing. Let's show it leads to $(0, 0, 0, 0)$.

Let j be such that $c(\theta_j, C_j) \neq (0, 0, 0, 0)$. The factor C_j contains at least an equation $\doteq_R$ ou $\vec{=}_R$. Along a loop that define the fair derivations, All the equations in S_j of the form $\doteq_R$ or $\vec{=}_R$ are narrowed, then a antecedent of C_j inside S_j. Let k be the number of this step. From proposition A.30 we have

$$c(\theta_k, C_k) < c(\theta_j, C_j)$$

□

Finaly, the completeness result can be deduced.

Theorem A.33 *Let S_0 be a unificand, $\theta_0 \in SOL(S_0)$, and $S_0 \longrightarrow^* \ldots \longrightarrow^* S_n \longrightarrow \ldots$ a fair derivation issued from S_0, where $S_1, \ldots, S_n, \ldots$ are the unificands in disjunctive form obtained at the end of some loop steps in the procedure.*

Then there exists a unificand S_i of this derivation and a substitution θ_i such that

- *θ_i is solution of a conjunctive factor of S_i that contains no $\doteq_R$ and $\vec{=}_R$ equation*
- *$\theta_i = \theta_0 \ [V(S_0)]$*

Proof: From previous lemma (with $V = V(S_0)$) there exists S_i in the derivation and $\theta_i \in SOL(S_i)$ such that $\theta_i = \theta_0 \ \ [V(S_0)]$ and $\theta_i S_i$ which is in disjunctive form contains a trivial conjunctive factor $\theta_i C_i$ such that $c(\theta_i, C_i) = (0, 0, 0, 0)$. Then C_i doesn't contain any equation of the form $\doteq_R$ and $\vec{=}_R$, and $\theta_i \in SOL(C_i)$. □

Chapitre 4

Implémentation de l'E-unification par surréduction contrainte : UREVEAL

L'équipe de réécriture de l'université d'Orléans s'intéresse beaucoup à l'aspect pratique du domaine. Depuis sa création elle développe des logiciels de démonstration automatique. A l'origine de ces travaux figure le logiciel *Reve* écrit par P. Lescanne [LES83] et Guttag [FG84]. Celui-ci a été développé par la suite en une famille *Reve2.n* puis *Reve3* par notamment H. et C. Kirchner à l'université de Nancy [KK86]. Tous ces logiciels étaient basés sur l'algorithme de complétion KB. J. Hsiang et J. Mzali ont implanté en premier l'approche UKB à l'université de Nancy et à Stony-Brook en 1988 [HM88]. Pour en accroître l'efficacité, S. Anantharaman, J. Hsiang et J. Mzali ont rajouté l'AC-complétion ainsi que de nouveaux mécanismes puissants comme la *cancellation*, ce qui a donné *SBREVE2* [AHM89]. Celui-ci complété par des stratégies est devenu *SBR3* et a permis d'obtenir des preuves automatiques célèbres [AH90, AABH90].

SBR3, tout comme ses ancêtres, était écrit en *CLU*. Ce langage développé au MIT est très peu répandu et son avenir est plutôt sombre. De ce fait nous avons décidé de réécrire *SBR3* en *C* pour pouvoir bénéficier de la rapidité et de la portabilité de ce langage. Après un an de travail, nous avons obtenu *REVEAL*, qui est un démonstrateur automatique comportant notamment les stratégies puissantes de *SBR3* et la complétion AC grâce à l'unification AC, de A. Boudet [BOU89].

Après avoir participé abondamment à la réalisation de ce projet, j'ai écrit un nouveau logiciel qui utilise les mêmes bases que *REVEAL* mais qui travaille dans un cadre contraint : *UREVEAL*. *UREVEAL* est une implantation de la méthode générale d'unification décrite au chapitre 2. Il ne comporte pas encore toutes les stratégies de *REVEAL* ni la possibilité de déclarer des opérateurs associatifs et commutatifs, mais il peut faire de la complétion et de la surréduction basiques, ce qui nous a permis de juger de l'efficacité de la méthode d'E-unification présentée au chapitre 2.

Ce chapitre va être consacré à la description de *UREVEAL*. Nous allons commencer par voir les structures de base de notre logiciel. La deuxième partie décrira l'aspect pratique de la propagation minimale. Nous détaillerons la liste des commandes actuellement disponibles dans *UREVEAL* au paragraphe 3. Le paragraphe 4 contiendra des exemples traités avec *UREVEAL* et nous terminerons par les améliorations que nous pensons apporter très prochaiment à notre logiciel.

4.1 Les structures de bases

L'objet qui est à la base de *UREVEAL* est le terme, toutes les autres structures (équation, contraintes,...) sont construites à partir des termes. La plupart des logiciels de démonstration automatique ont adopté une structure d'arbre pour représenter leurs termes, nous avons décidé d'utiliser une structure linéaire.

4.1.1 Les termes aplatis

Cette structure s'inspire de la définition de *flatterms* de J. Christian [CHR89] que nous avons légèrement complétée. Cette structure représente les termes sous forme aplatie "linéaire" avec les opérateurs en notation préfixe, elle a comme avantage de faciliter le parcours des sous-termes d'un terme sans utiliser la notion de récursivité.

Exemple 4.1 Le terme $f(a, g(x), h(b, y))$ sera stocké en mémoire de la façon suivante :

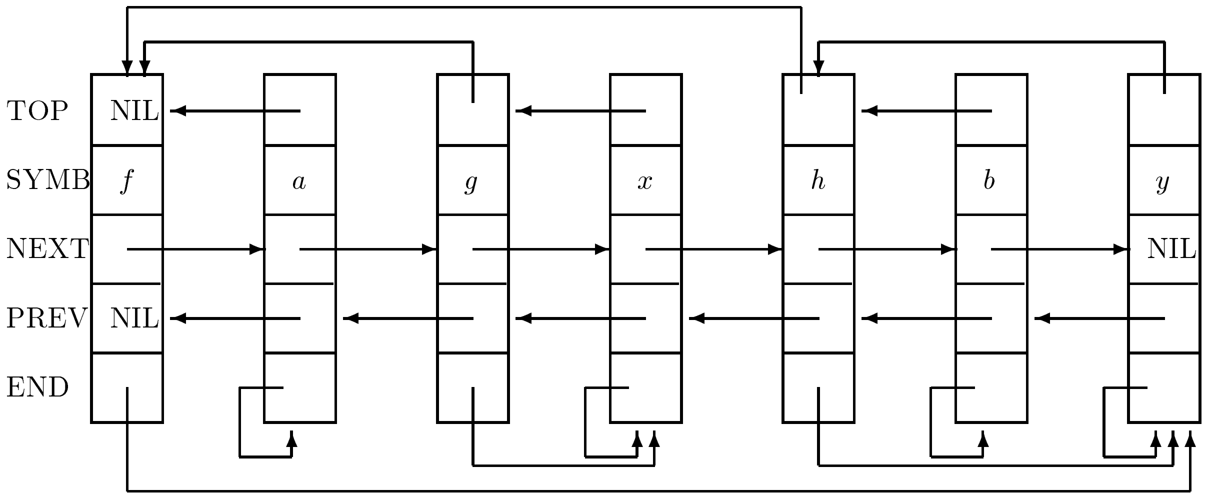


Figure 4.1: Représentation interne du terme $f(a, g(x), h(b, y))$.

Un *flatterm* est donc une structure doublement chaînée, plus précisément il est composé de cinq champs. Le champ *symbol* est un entier représentant le symbole associé à la structure. Si cet entier est strictement positif, le symbole représentera une fonction (incluant les constantes) et correspondra à une entrée dans la table des symboles. La table des symboles contiendra le nom de la fonction, son arité et diverses autres propriétés associées à la fonction. Si l'entier est strictement négatif, le symbole sera interprété comme une variable. Le nom des variables n'est pas significatif, ainsi, la première variable d'un terme sera représentée par -1, la deuxième, différente de la première, par -2 etc.

Nos flatterms comportent aussi des champs appelés *next* et *prev*, pointant respectivement sur les voisins immédiats de droite et de gauche, qui sont eux aussi des *flatterms*; et un champ *end* pointe sur le dernier *flatterm* qui fait partie du terme dont le symbole de tête est le *flatterm* courant. Pour une variable ou une constante, le champ *end* pointe sur le *flatterm* lui-même, ceci peut se voir sur la figure 4.1 pour les symboles a, b, x et y.

Ces quatre champs définissaient le *flatterm* à la J. Christian; nous y avons ajouté un cinquième champ, *top*, qui pointe sur la tête du *flatterm* qui a pour argument le *flatterm* courant. Nous avons aussi ajouté quelques autres champs pour économiser le temps de calcul. Par exemple le champ *size* est initialisé quand on a besoin pour la première fois de la taille du *flatterm* et il est réutilisé toutes les fois suivantes, si nécessaire, ce qui évite de parcourir de nouveau complètement le *flatterm*.

4.1.2 Les équations usuelles ou contraintes

Une équation est représentée par deux champs qui pointent sur des *flatterms*, le côté gauche et le côté droit de l'équation, et un champ entier *egal*. Ce dernier champ nous permet de distinguer les différents types d'équations, *egal* vaut 0 pour une inégalité (non encore utilisé dans *UREVEAL* mais seulement dans *REVEAL*), *egal* vaut 1 pour une égalité et *egal* vaut 2 lorsque le côté gauche est supérieur au côté droit par rapport à l'ordre utilisé. Ces trois champs sont accompagnés d'un champ entier *id* qui identifie l'équation, c'est le numéro de l'équation, et d'un autre champ qui nous facilite la gestion de la mémoire.

Ce champ *egal* qui peut prendre trois valeurs pour une équation usuelle, peut prendre d'autres valeurs lorsqu'une équation est vue comme une contrainte.

Une contrainte est une liste doublement chaînée par les champs *nextc* et *prevc* qui pointent respectivement sur la contrainte suivante et la contrainte précédente. Elle a aussi un champ *elt* qui pointe sur une équation particulière puisque dans ce cas-là, le champ *egal* d'une équation prend l'une des valeurs suivantes : 4 pour une contrainte égalitaire initiale, 5 pour une contrainte d'ordre initiale, 6 pour une contrainte égalitaire nouvelle créée par la procédure, 7 pour une contrainte d'ordre nouvelle créée par la procédure, et 8 pour une contrainte inégalitaire ($\neq$) utilisée uniquement pour avoir une contrainte typiquement insatisfiable : $[\![x \neq x]\!]$.

Les contraintes égalitaires seront placées au début de la contrainte globale tandis que les contraintes d'ordres seront à la fin. Pour l'instant les contraintes égalitaires sont résolues syntaxiquement par l'algorithme d'unification syntaxique (décomposition, instanciation et recherche de cycle). La partie égalitaire sera donc conservée sous forme résolue, c'est-à-dire sous la forme $[\![x_1 = t_1 \ \wedge \ ... \ \wedge \ x_n = t_n]\!]$ sans cycle dans les variables. Si la partie égalitaire est insatisfiable, on remplace la contrainte par $[\![x \neq x]\!]$. Les contraintes d'ordres sont instanciées au maximum, c'est-à-dire que plus aucune variable se trouvant en partie gauche d'une contrainte égalitaire (après résolution), n'apparaît dans les contraintes d'ordres. Trois cas peuvent alors se présenter pour une contrainte d'ordre $[\![t > s]\!]$. Soit l'ordre utilisé évalue à vrai cette contrainte et dans ce cas on supprime $[\![t > s]\!]$ de la contrainte globale. Soit l'ordre évalue à vrai la contrainte inverse $[\![s > t]\!]$, la contrainte globale est alors insatisfiable et est remplacée par $[\![x \neq x]\!]$. Soit on ne se trouve pas dans l'un des deux cas précédents, on conserve la contrainte en attendant qu'elle soit instanciée de nouveau.

Les substitutions que nous utiliserons seront stockées dans des contraintes. La substitution $\{x_1 \leftarrow t_1, ..., x_n \leftarrow t_n\}$ sera représentée par la contrainte $[\![x_1 = t_1 \ \wedge \ ... \ \wedge \ x_n = t_n]\!]$. Pour trouver l'unificateur syntaxique de deux termes s et t il suffit donc de mettre sous forme résolue la contrainte $[\![s = t]\!]$, de même pour un filtre de s vers t mais en interdisant d'instancier les variables de t.

Les équations et les contraintes nous servent à définir les objets que l'on manipule dans nos règles d'inférence, à savoir les équations contraintes.

4.1.3 Les équations contraintes et les unificandes

La structure d'équation contrainte est comme son nom l'indique composée d'un champ qui pointe sur une équation et d'un champ qui pointe sur une contrainte. Cette structure est complétée d'autres champs qui nous facilitent la manipulation des équations contraintes. Nous avons le champ *id* qui stocke le numéro de l'équation contrainte et le champ *size* qui contient sa taille.

Le champ *parents* pointe sur une chaîne d'entiers, il est utilisé pour conserver l'historique de l'équation, c'est-à-dire à partir de quelles équations elle a été obtenue si c'est le résultat d'une superposition et quelles sont les règles qui l'ont simplifiée. Nous disposons de deux derniers champs, un qui est utilisé pour gérer la mémoire et un autre qui nous sert dans la mise en pratique de notre propagation minimale des contraintes initiales.

Les équations contraintes sont aussi utilisées pour représenter les unificandes. Dans ce cas, le champ *egal* de la partie équation aura pour valeur -1, ce qui signifie que nous sommes en présence d'une équation à résoudre. Notre structure *unificand* sera une chaîne d'équations contraintes ce qui nous permettra de gérer à partir d'un seul *unificand* tout l'espace de recherche, chaque équation contrainte correspondra à une branche de surréduction. Les substitutions seront stockées au fur et à mesure du calcul dans la partie contrainte et lorsque la partie équation d'une équation contrainte d'un unificande sera vide, la contrainte, si elle est satisfiable représentera une solution de l'unificande initial.

Nous venons de décrire les principales structures qui sont définies et utilisées dans *UREVEAL*. Examinons maintenant comment, du point de vue pratique, nous réalisons une propagation minimale décrite au paragraphe 2.3.1.

4.2 La propagation minimale des contraintes initiales

Nous acceptons dans les équations initiales, deux types de contrainte : des contraintes égalitaires et des contraintes d'ordres. Pour propager de façon minimale par rapport à une contrainte égalitaire, il suffit de prendre l'unificateur le plus général, solution de la partie égalitaire de la contrainte, et de l'appliquer à l'équation. Cet unificateur existe et est unique puisqu'il correspond à une unification syntaxique. On obtient une équation qui n'a plus de contrainte égalitaire et qui remplace l'ancienne équation.

Pour une contrainte d'ordre, la propagation est beaucoup plus coûteuse car le nombre de solutions, même minimales, peut être infini. Nous allons donc procéder par étapes pour éviter de tomber dans un processus infini sans possibilité d'obtenir de nouvelles solutions. Du point de vue pratique, *UREVEAL* ne peut, pour l'instant, traiter correctement que les contraintes d'ordres du type $[\![x > t]\!]$ avec x une variable et t un terme clos.

4.2.1 Génération de termes par étape

Pour trouver les solutions d'une contrainte d'ordre $[\![x > t]\!]$, nous instancions x et nous vérifions si la contrainte est satisfaite (nous sommes en présence d'une solution possible) ou non (l'instanciation n'est pas une solution). Mais il y a une infinité d'instances possibles, c'est pour cela que nous décomposons en étapes, la génération des instanciations possibles de la variable x.

Définition 4.2 Nous appellerons terme de profondeur 1 tout terme ayant pour tête un symbole de fonction d'arité supérieur ou égal à 1 et dont les arguments sont exclusivement des variables, par exemple $f(x), h(x, y)$ si f est d'arité 1 et h d'arité 2. $\diamond$

Nous commençons par instancier x, par les constantes de notre système et par tous les termes possibles de profondeur 1. On obtient ainsi une liste finie d'instanciations possibles, correspondant à la première étape. A la deuxième étape, nous reprenons tous les termes de profondeur 1, et pour chacun d'entre eux, nous créons tous les termes possibles en remplaçant chaque variable par l'un des termes de la liste obtenue à la première étape. A chaque étape, nous faisons de même. Pour la i-ième étape, nous prenons les termes de profondeur 1, et nous remplaçons les variables par les termes obtenus à la (i-1)-ième étape. S'il y a plusieurs variables dans un même terme, on calcule toutes les permutations possibles.

Exemple 4.3 Soient f et h deux symboles de fonctions d'arité respective 1 et 2, et a, b deux constantes. Les termes de profondeur 1 sont $f(x)$ et $h(x, y)$. Les instanciations que l'on tentera à la première étape font partie de l'ensemble $\{a, b, f(x), h(x, y)\}$. Pour la deuxième étape on obtient l'ensemble suivant :

$$\begin{array}{llll}
\{f(a), & f(b), & f(f(x)), & f(h(x,y)) \\
h(a,y), & h(b,y), & h(f(x),y), & h(h(x,y),z) \\
h(a,a), & h(b,a), & h(f(x),a), & h(h(x,y),a) \\
h(a,b), & h(b,b), & h(f(x),b), & h(h(x,y),b) \\
h(a,f(x)), & h(b,f(x)), & h(f(x),f(y)), & h(h(x,y),f(z)) \\
h(a,h(x,y)), & h(b,h(x,y)), & h(f(x),h(y,z)), & h(h(x,y),h(z,u)) \\
h(x,a), & h(x,b), & h(x,f(y)), & h(x,h(y,z))\}
\end{array}$$

L'ensemble des termes grossit très vite, mais dans la pratique, en choisissant correctement la précédence des opérateurs pour un ordre lpo, on limitera le nombre d'étapes nécessaires à une propagation minimale complète. Voyons comment ces ensembles sont utilisés.

4.2.2 Propagation pas à pas des contraintes initiales

La structure d'équation contrainte comporte un champ entier *exp* qui va nous permettre de décomposer notre propagation. Une équation sans contrainte initiale a dans le champ *exp* la valeur -1, aucune propagation n'est nécessaire sur cette équation. Lorsque le champ *exp* a pour valeur 0, on doit propager par rapport aux contraintes initiales. Pour cela, nous commençons à propager par rapport aux contraintes égalitaires initiales en prenant l'unificateur le plus général que l'on applique à la fois sur l'équation et sur la contrainte restante. On obtient une nouvelle équation contrainte qui remplace l'ancienne et qui a la valeur 1 dans le champ *exp*. On a réalisé ainsi un pas de propagation.

Si le champ *exp* a une valeur i supérieure ou égale à 1, on doit propager par rapport à des contraintes d'ordres initiales (s'il y en a). Pour cela nous prenons l'ensemble des termes obtenus à la i-ième étape de notre processus de construction de terme décrit auparavant. Nous instancions la variable contrainte (côté gauche de la première contrainte d'ordre initiale rencontrée), par chacun des termes de notre ensemble de termes. Pour chaque instanciation, si la contrainte est satisfaite, on applique cette instanciation sur l'équation et sur la contrainte et on obtient une nouvelle équation contrainte dont le champ *exp* a pour valeur 1 (il y a peut-être d'autres contraintes d'ordres initiales à propager). Lorsque l'on a fini toutes les possibilités d'instanciation, deux cas sont possibles :

- soit toutes les instanciations que l'on a faites avec des termes contenant des variables, ont satisfait la contrainte. Alors la propagation par rapport à la contrainte d'ordre initiale choisie est finie. Toutes les autres propagations possibles par rapport à cette contrainte ne seraient que des instances de propagation déjà faites, car si σ est solution de la contrainte, $\forall\theta$, $\theta \circ \sigma$ est solution de la contrainte. L'ancienne équation contrainte est supprimée.
- soit une instanciation avec un terme contenant des variables ne satisfait pas la contrainte. Alors la propagation doit continuer, le i-ième pas est fini et le champ *exp* de l'équation contrainte de départ prend la valeur $i + 1$.

Appliquons cette méthode assez complexe à définir, sur deux exemples.

Exemple 4.4 Soient a et b deux constantes, f et h deux symboles de fonctions d'arité respective 1 et 2, $\{a, b, f, h\}$ constituant l'ensemble des symboles de fonctions de notre système. Soit l'équation initiale suivante :

$$(e_1) \quad : \quad h(x,y) \quad \doteq \quad f(z)[\![z = f(u) \ \wedge \ x > a \ \wedge \ y > b]\!]$$

sous un ordre lpo où la précédence sur les opérateurs est $h > f > a > b$. Le champ *exp* de la structure contenant cette équation contrainte, a pour valeur 0. Pour le premier pas de propagation, on remplace l'équation initiale par :

$$(e_2) \quad : \quad h(x,y) \quad \doteq \quad f(f(u))[\![x > a \ \wedge \ y > b]\!]$$

avec un champ *exp* qui a pour valeur 1. Pour le second pas de propagation, on propage par rapport à la contrainte $[\![x > a]\!]$. L'ensemble des termes de la première étape ($exp = 1$), est : $\{a, b, f(x), h(x,y)\}$. L'instanciation de x par a ne satisfait pas la contrainte, de même avec b. En revanche avec $f(x)$ on crée la nouvelle équation contrainte :

$$(e_3) \quad : \quad h(f(x),y) \quad \doteq \quad f(f(u))[\![y > b]\!]$$

et avec $h(x,y)$ on crée :

$$(e_4) \quad : \quad h(h(x,y),z) \quad \doteq \quad f(f(u))[\![z > b]\!]$$

Les variables sont renommées automatiquement. Toutes les instanciations avec des termes non clos $(f(x), h(x,y))$ ont satisfait la contrainte $[\![x > a]\!]$, la propagation par rapport à cette contrainte est donc finie et l'équation (e_2) est supprimée. On doit maintenant propager par rapport à la deuxième contrainte $[\![y > b]\!]$, cette propagation est faite au niveau des équations (e_3) et (e_4). La propagation de (e_3) se fait en une étape et donne les trois équations suivantes :

$$\begin{array}{lcrcl} (e_5) & : & h(f(x),a) & \doteq & f(f(y))[\![\ ]\!] \\ (e_6) & : & h(f(x),f(y)) & \doteq & f(f(z))[\![\ ]\!] \\ (e_7) & : & h(h(x),h(y,z)) & \doteq & f(f(u))[\![\ ]\!] \end{array}$$

La propagation de (e_3) est finie et (e_3) est supprimée. La propagation de (e_4) est analogue, on obtient :

$$\begin{array}{lcrcl} (e_8) & : & h(h(x,y),a) & \doteq & f(f(z))[\![\ ]\!] \\ (e_9) & : & h(h(x,y),f(z)) & \doteq & f(f(u))[\![\ ]\!] \\ (e_{10}) & : & h(h(x,y),h(z,u)) & \doteq & f(f(v))[\![\ ]\!] \end{array}$$

La propagation de (e_4) est finie et (e_4) est supprimée de notre ensemble d'équations. Les six équations $\{(e_5),(e_6),(e_7),(e_8),(e_9),(e_{10})\}$ constituent donc une propagation minimale complète de l'équation contrainte initiale (e_1). Cette propagation sera effectuée au cours de notre procédure, de la même manière que sur cet exemple.

Examinons un second exemple où nous avons besoin de générer deux ensembles de termes.

Exemple 4.5 Soient a, b, f, g deux constantes et deux symboles de fonctions d'arité 1. Soit l'équation initiale suivante :

$$(e_1) \quad : \quad f(x) \quad \doteq \quad g(y)[\![y = a \ \wedge \ x > g(a)]\!]$$

avec un ordre lpo où la précédence sur les opérateurs est $f > g > a > b$. Notre propagation minimale se fait en trois étapes. On commence par remplacer l'équation initiale par:

$$(e_2) \quad : \quad f(x) \quad \doteq \quad g(a)[\![x > g(a)]\!]$$

en propageant par rapport à la contrainte égalitaire initiale. Le champ *exp* de (e_2) vaut 1. Ensuite on propage la contrainte d'ordre. Le premier ensemble de termes est : $\{a, b, f(x), g(x)\}$. Seule l'instanciation avec $f(x)$ convient et donne la nouvelle règle :

$$(e_3) \quad : \quad f(f(x)) \quad \doteq \quad g(a)[\![\ ]\!]$$

On doit continuer la propagation puisque l'instanciation avec $g(x)$ ne satisfait pas la contrainte. La règle (e_2) est conservée mais son champ *exp* vaut maintenant 2. Le nouvel ensemble de termes (instanciations possibles pour x) est $\{f(a), f(b), f(f(x)), f(g(x)), g(a), g(b), g(f(x)), g(g(x))\}$. Seules les instanciations par $g(a)$ et $g(b)$ ne satisfont pas la contrainte, on obtient donc les six règles suivantes :

$$\begin{array}{rcrcl}
(e_4) & : & f(f(a)) & \doteq & g(a)[\![\ ]\!] \\
(e_5) & : & f(f(b)) & \doteq & g(a)[\![\ ]\!] \\
(e_6) & : & f(f(f(x))) & \doteq & g(a)[\![\ ]\!] \\
(e_7) & : & f(f(g(x))) & \doteq & g(a)[\![\ ]\!] \\
(e_8) & : & f(g(f(x))) & \doteq & g(a)[\![\ ]\!] \\
(e_9) & : & f(g(g(x))) & \doteq & g(a)[\![\ ]\!]
\end{array}$$

Les équations $(e_4), (e_5), (e_6), (e_7)$ sont subsumées par (e_3), la propagation de (e_2) est terminée et (e_2) est supprimée. Finalement, la propagation minimale de (e_1) donne donc uniquement trois équations $(e_3), (e_8), (e_9)$. En fait il est inutile d'essayer les instanciations de x, par un terme commençant par f, à la deuxième étape, puisque l'instanciation à la première étape par $f(x)$, a été acceptée.

Dans les étapes de propagation comme dans les étapes de simplification, l'ordre joue un rôle très important.

4.2.3 L'ordre utilisé dans *UREVEAL*

Pour l'instant un seul ordre est disponible dans notre implémentation, c'est l'ordre lexicographique avec status [DER82]. Cet ordre impose à l'utilisateur de choisir une précédence entre les symboles de fonctions, c'est-à-dire d'ordonner les symboles de fonctions avant de débuter la procédure d'unification. Si la précédence est totale, l'ordre sera total sur les termes clos.

Passons maintenant en revue la liste des commandes disponibles dans *UREVEAL*.

4.3 Description des différentes commandes disponibles dans *UREVEAL*

Nous pouvons séparer ces commandes en deux parties. D'un côté les commandes qui permettent la saisie du problème et de l'autre celles qui permettent de résoudre le problème. *UREVEAL* attend l'entrée d'une commande lorsque le prompt '->' est affiché.

4.3.1 La saisie des données

Nous allons énumérer dans ce paragraphe les différentes commandes faites pour faciliter la saisie d'un problème.

- *terminal* : permet la lecture au clavier d'une liste d'équations contraintes. Les différents symboles sont représentés par une chaîne d'au plus 50 caractères. Les symboles de fonctions commencent par une lettre comprise entre a et t où entre A et T, les symboles '+','*','.' sont aussi acceptés. Les variables commencent par $u, v, w, x, y, z, U, V, W, X, Y, Z$. Les variables sont automatiquement renommées, leur nom initial n'est pas conservé. Les opérateurs sont aussi bien acceptés en notation préfixe qu'en notation infixe. Les deux côtés d'une équation sont séparés par le symbole '='. Si l'équation est contrainte, juste après l'équation on tape le caractère '[' suivi des contraintes séparées par '&' et on termine par ']'. Une contrainte est deux termes séparés soit par '=' pour une contrainte égalitaire, soit par '>' pour une contrainte d'ordre. Les équations contraintes sont séparées les unes des autres par un ';'. Si l'équation n'est pas contrainte, le ';' est mis juste après l'équation. La dernière équation contrainte doit être terminée par ';;' et alors la liste des équations contraintes reconnues par l'analyseur, et donc stockées en mémoire, est affichée et le prompt '−>' réapparaît. Si pendant la lecture des équations une erreur lexicale ou syntaxique est découverte, un message d'erreur s'affiche et l'analyse est stoppée jusqu'au ';' suivant (c'est le ';' qui synchronise l'analyseur).

- *display* : affiche à l'écran la liste des équations contraintes déjà enregistrées en mémoire.

- *precedence* : permet de lire la précédence des opérateurs. On entre les symboles du plus petit au plus grand, séparés par un caractère blanc ' '. On termine par ' ;' séparé du dernier opérateur par un blanc. Attention tous les opérateurs entrés dans la précédence doivent déjà figurer dans la table des symboles, la précédence doit être entrée après les équations.

- *status* : permet de lire le *statut* des opérateurs. On donne le nom de l'opérateur suivi de *ltr* pour un *statut* de gauche à droite, *rtl* pour un *statut* de droite à gauche et de *multi* pour un *statut* multi-ensemble. Multi-ensemble est le *statut* de chaque opérateur par défaut. On termine par ';'.

- *symbole* : affiche la liste des symboles de fonction contenus dans la table des symboles, donne en plus pour chaque fonction son arité, son *statut*, sa théorie (vide pour l'instant C ou AC dans une version ultérieure).

- *help* : affiche la liste des commandes disponibles.

Après avoir initialisé un problème nous avons plusieurs choix possibles pour exécuter notre procédure.

4.3.2 Exécution de la procédure

- *completion* : fait la complétion basique du système d'équations initiales. Pour cela on utilise les règles de notre système d'inférence qui correspondent à la complétion. Pour l'instant la propagation minimale est faite avec la stratégie suivante : lorsque E est vide, on regarde si dans R il y a une équation contrainte à propager, si c'est le cas, on applique sur cette équation un pas de notre procédure de propagation minimale décrite au paragraphe précédent, on met les équations que l'on obtient dans E et on reprend la complétion. Lorsque la complétion est terminée, c'est-à-dire lorsque E est vide et qu'il n'y a plus d'équation à propager dans R, on affiche le système complété.

- *unification* : lit une équation que l'on veut résoudre dans la théorie décrite par l'ensemble des équations initiales, résout cette équation par surréduction contrainte avec uniquement les équations initiales. Cette méthode est complète lorsque l'ensemble des équations initiales est complet. Les solutions sont affichées au fur et à mesure qu'elles sont calculées. Lorsque la procédure termine, on affiche l'ensemble de toutes les solutions que l'on a trouvées.

- *narrow* : lit une équation que l'on veut résoudre, complète le système d'équation initiale et résout l'équation avec l'ensemble complet d'équations (si la complétion termine).

- *p_narrow* : lit une équation que l'on veut résoudre puis fait la complétion du système et la résolution de l'unificande de façon alternée. On applique quelques pas de surréduction contrainte sur l'unificande, puis on ajoute des équations à notre système d'équations initiales en faisant des pas de complétion basique, et on recommence la résolution de l'unificande et ainsi de suite.

- *quit* : pour terminer une session de travail, sort de *UREVEAL*.

Toutes ces commandes sont activées en tapant, après le prompt '−>', le nom complet de la commande souhaitée ou juste les premiers caractères de la commande, s'il y a ambiguïté, la première dans l'ordre alphabétique est choisie. Détaillons maintenant sur des exemples plusieurs sessions de *UREVEAL*.

4.4 Exemples

Nous allons donner tout d'abord l'exemple classique à savoir la complétion, (ici basique), de la théorie des groupes, que l'on obtient avec *UREVEAL* (**paragraphe 4.4.1**).

L'exemple suivant concerne un cas où la complétion basique est plus 'économique' que la complétion usuelle (**paragraphe 4.4.2**).

Nous donnons ensuite deux exemples d'unification générale : le premier en spécifiant la relation 'est inférieur à' sur un ensemble totalement ordonné et le second dans une théorie où un opérateur est distributif à droite et à gauche par rapport à un autre opérateur (**paragraphe 4.4.3**).

Le dernier exemple montre comment il peut être envisagé d'utiliser *UREVEAL* pour valider des spécifications contraintes, avec l'exemple du problème dit de huit reines en échecs (**paragraphe 4.4.4**).

4.4.1 La complétion basique de la théorie des groupes

Nous allons donner ici l'ensemble du fichier de sortie décrivant notre complétion des groupes sous *UREVEAL*. Ceci permet de voir en détail la génération des paires critiques, et de convaincre que les étapes de simplification sont bien basiques.

```
                  Welcome to UREVEAL - 1.0, March - 1993.

       ---------    Built at the LIFO - Orleans (Fr.)   --------
                    ---------------------------------
                       For help type 'help' or '?'


-> terminal
 Please type your equations, separated by a ';'. Terminate with a ';'
 on a new line.
 x * e = x;
 x * i(x) = e;
 (x * y) * z = x * (y * z);;

 (1) [] x * e = x[[empty]]
 (2) [] x * i(x) = e[[empty]]
 (3) [] (x * y) * z = x * (y * z)[[empty]]


-> * has status 'left_to_right'

-> The precedence is : e < * < i
```

```
-> completion

------------ Beginning Processing ------------


 *** NEW * Rule ***  :   (1) [] x * e -> x[[empty]]

 *** NEW * Rule ***  :   (2) [] x * i(x) -> e[[empty]]

 *** NEW * Rule ***  :   (3) [] (x * y) * z -> x * (y * z)[[empty]]

 ... SUPERPOSING ... :    (1) [] x * e -> x[[empty]]

 ... SUPERPOSING ... :    (2) [] x * i(x) -> e[[empty]]

 ... SUPERPOSING ... :    (3) [] (x * y) * z -> x * (y * z)[[empty]]

 *** NEW * Rule ***  :   (4) [Deduced From: 3 1]
             x * (y * z) -> x * z[[ y =n e]]

 *** NEW * Rule ***  :   (5) [Deduced From: 3 2]
             x * (y * z) -> e[[ z =n i(x * y)]]

 *** NEW * Rule ***  :   (6) [Deduced From: 3 2]
             x * (y * z) -> e * z[[ y =n i(x)]]

 ... SUPERPOSING ... :    (4) [Deduced From: 3 1]
             x * (y * z) -> x * z[[ y =n e]]

 *** NEW * Rule ***  :   (7) [Deduced From: 4 2: Simplif_With: 1]
             x * y -> x[[ y =n i(e)]]

 ... SUPERPOSING ... :    (7) [Deduced From: 4 2: Simplif_With: 1]
             x * y -> x[[ y =n i(e)]]

 *** NEW * Rule ***  :   (8) [Deduced_From: 7 3]
             x * (y * z) -> x * z[[ y =n i(e)]]

 ... SUPERPOSING ... :    (5) [Deduced From: 3 2]
             x * (y * z) -> e[[ z =n i(x * y)]]

 *** NEW * Rule ***  :   (9) [Deduced_From: 5 4: Simplif_With: 1]
             x * y -> x[[ y =n z * i(e * z)]]

 *** NEW * Rule ***  :   (10) [Deduced From: 5 3: Simplif_With: 3]
             x * (y * z) -> e[[ z =n u * i(x * (y * u))]]

 *** NEW * Rule ***  :   (11) [Deduced From: 5 3]
             x * (y * (z * u)) -> e[[ u =n i(x * (y * z))]]
```

```
*** NEW * Rule ***  :   (12) [Deduced_From: 5 3: Simplif_With: 3]
          x * (y * (z * u)) -> e * u[[ z =n i(x * y)]]

... SUPERPOSING ... :    (6) [Deduced From: 3 2]
          x * (y * z) -> e * z[[ y =n i(x)]]

*** NEW * Rule ***  :   (13) [Deduced From: 6 2: Simplif_With: 1]
          e * x -> y[[ x =n i(i(y))]]

*** NEW * Rule ***  :   (14) [Deduced_From: 6 4: Simplif_With: 8]
          x * (e * y) -> x * y[[empty]]

    SIMPLIFYING  4 with 14 ...



*** NEW * Rule ***  :   (15) [Deduced From: 6 5: Simplif_With: 1]
          e * x -> y[[ x =n z * i(i(y) * z)]]

*** NEW * Rule ***  :   (16) [Deduced_From: 6 6: Simplif_With: 14]
          e * x -> y * z[[ x =n i(i(y)) * z]]

... SUPERPOSING ... :    (13) [Deduced From: 6 2: Simplif_With: 1]
          e * x -> y[[ x =n i(i(y))]]

*** NEW * Rule ***  :   (17) [Deduced_From: 13 3]
          x * (y * z) -> u * z[[ x =n e ^  y =n i(i(u))]]

... SUPERPOSING ... :    (14) [Deduced_From: 6 4: Simplif_With: 8]
          x * (e * y) -> x * y[[empty]]

*** NEW * Rule ***  :   (18) [Deduced From: 14 13]
          x * y -> x * z[[ y =n i(i(z))]]

    SIMPLIFYING  13 with 18 ...

*** NEW * Rule ***  :   (19) [ 13: Simplif_With: 18]
          e * x -> x[[empty]]

    SIMPLIFYING  14 with 19 ...
    SIMPLIFYING  6 with 19 ...

*** NEW * Rule ***  :   (20) [ 6: Simplif_With: 19]
          x * (y * z) -> z[[ y =n i(x)]]

... SUPERPOSING ... :    (19) [ 13: Simplif_With: 18]
          e * x -> x[[empty]]

*** NEW * Rule ***  :   (21) [Deduced From: 19 2]
          x -> e[[ x =n i(e)]]
```

```
    SIMPLIFYING  7 with 21 ...

... SUPERPOSING ... :    (21) [Deduced From: 19 2]
          x -> e[[ x =n i(e)]]

... SUPERPOSING ... :    (18) [Deduced From: 14 13]
          x * y -> x * z[[ y =n i(i(z))]]

*** NEW * Rule ***  :   (22) [Deduced From: 18 2]
          x * y -> e[[ x =n i(y)]]

*** NEW * Rule ***  :   (23) [Deduced From: 18 19: Simplif_With: 19]
          x -> y[[ x =n i(i(y))]]

    SIMPLIFYING  18 with 23 ...

... SUPERPOSING ... :    (23) [Deduced From: 18 19: Simplif_With: 19]
          x -> y[[ x =n i(i(y))]]

... SUPERPOSING ... :    (22) [Deduced From: 18 2]
          x * y -> e[[ x =n i(y)]]

*** NEW * Rule ***  :   (24) [Deduced_From: 22 3: Simplif_With: 19]
          x * (y * z) -> z[[ x =n i(y)]]

... SUPERPOSING ... :    (24) [Deduced_From: 22 3: Simplif_With: 19]
          x * (y * z) -> z[[ x =n i(y)]]

*** NEW * Rule ***  :   (25) [Deduced From: 24 5: Simplif_With: 1]
          x -> y[[ x =n z * i(u * z) ^  y =n i(u)]]

*** NEW * Rule ***  :   (26) [Deduced From: 24 3]
          x * (y * (z * u)) -> u[[ x =n i(y * z)]]

... SUPERPOSING ... :    (25) [Deduced From: 24 5: Simplif_With: 1]
          x -> y[[ x =n z * i(u * z) ^  y =n i(u)]]

*** NEW * Rule ***  :   (27) [Deduced_From: 25 24]
          x -> y * z[[ x =n i(u * v) ^  z =n i(u) ^  y =n i(v)]]

    SIMPLIFYING  25 with 27 ...

*** NEW * Rule ***  :   (28) [ 25: Simplif_With: 27]
          x -> y[[ x =n z * (i(z) * i(u)) ^  y =n i(u)]]

    SIMPLIFYING  5 with 27 ...

*** NEW * Rule ***  :   (29) [ 5: Simplif_With: 27]
          x * (y * z) -> e[[ z =n i(y) * i(x)]]

    SIMPLIFYING  28 with 20 ...
```

```
    SIMPLIFYING  8 with 19 ...
    SIMPLIFYING  9 with 1 ...

... SUPERPOSING ... :    (20) [ 6: Simplif_With: 19]
              x * (y * z) -> z[[ y =n i(x)]]

... SUPERPOSING ... :    (27) [Deduced_From: 25 24]
              x -> y * z[[ x =n i(u * v) ^  z =n i(u) ^  y =n i(v)]]

    SIMPLIFYING  29 with 20 ...
    SIMPLIFYING  15 with 19 ...
    SIMPLIFYING  16 with 19 ...
    SIMPLIFYING  17 with 19 ...
    SIMPLIFYING  26 with 3 ...
    SIMPLIFYING  10 with 20 ...
    SIMPLIFYING  11 with 20 ...
    SIMPLIFYING  12 with 19 ...

*** Empty Set of Rules to Process *** :   2.19 sec :

------------ END OF SEARCH ------------

Your System is Complete !


(21) [Deduced From: 19 2]
              x -> e[[ x =n i(e)]]

(1) [] x * e -> x[[empty]]

(23) [Deduced From: 18 19: Simplif_With: 19]
              x -> y[[ x =n i(i(y))]]

(2) [] x * i(x) -> e[[empty]]

(22) [Deduced From: 18 2]
              x * y -> e[[ x =n i(y)]]

(19) [ 13: Simplif_With: 18]
              e * x -> x[[empty]]

(24) [Deduced_From: 22 3: Simplif_With: 19]
              x * (y * z) -> z[[ x =n i(y)]]

(20) [ 6: Simplif_With: 19]
              x * (y * z) -> z[[ y =n i(x)]]

(27) [Deduced_From: 25 24]
              x -> y * z[[ x =n i(u * v) ^  z =n i(u) ^  y =n i(v)]]

(3) [] (x * y) * z -> x * (y * z)[[empty]]
```

```
 GOODBYE !

-> quit
```

Ce script nous montre comment *UREVEAL* complète de façon basique les groupes. Il est peut-être un peu surprenant que l'on obtienne un système final de 10 règles, (exprimées sous une forme contrainte pour des besoin de surréduction), identique au système obtenu par la complétion usuelle. Ce système peut nous servir dans l'état, sans avoir besoin de propager les contraintes, pour résoudre, par surréduction contrainte, des problèmes d'unification modulo la théorie des groupes. Malgré une simplification plus importante par la méthode décrite dans [BGSL93], on obtient également le même ensemble final. Il serait intéressant d'évaluer la différence au niveau du nombre de règles intermédiaires engendrées.

4.4.2 Comparaison complétion basique et complétion usuelle

La basicité a pour but d'atténuer le nombre de possibilités de faire des paires critiques au niveau de la complétion. Mais lorsque la complétion est basique nous faisons moins de simplification que dans une complétion normale; nos comparaisons ont souvent abouti à une meilleur efficacité en temps, de la complétion usuelle. Les tests ont été faits avec le logiciel *REVEAL*. Mais il existe des exemples où la complétion usuelle crée deux fois plus de règles intermédiaires que la complétion basique, (ces règles intermédiaires étant par la suite simplifiées). Examinons un de ces exemples.

```
                   Welcome to UREVEAL - 1.0, March - 1993.

       ---------    Built at the LIFO - Orleans (Fr.)   --------
                    ---------------------------------
                       For help type 'help' or '?'


-> terminal
 Please type your equations, separated by a ';'. Terminate with a ';'
 on a new line.
 g(f(x)) = g(x);
 f(i1(y)) = a;
 i1(h(z)) = i2(z);
 i2(h(z)) = b;;

 (1) [] g(f(x)) = g(x)[[empty]]
 (2) [] f(i1(x)) = a[[empty]]
 (3) [] i1(h(x)) = i2(x)[[empty]]
 (4) [] i2(h(x)) = b[[empty]]

-> The precedence is : a < b < i2 < i1 < f

-> completion
```

```
------------ Beginning Processing ------------
......
------------ END OF SEARCH ------------

 Your System is Complete !

 (7) [Deduced From: 6 4]
            f(b) -> a[[empty]]

 (2) [] f(i1(x)) -> a[[empty]]

 (4) [] i2(h(x)) -> b[[empty]]

 (6) [Deduced_From: 3 2]
            f(i2(x)) -> a[[empty]]

 (9) [Deduced_From: 7 1]
            g(x) -> g(a)[[ x =n b]]

 (1) [] g(f(x)) -> g(x)[[empty]]

 (3) [] i1(h(x)) -> i2(x)[[empty]]

 (5) [Deduced_From: 2 1]
            g(x) -> g(a)[[ x =n i1(y)]]

 (8) [Deduced_From: 6 1]
            g(x) -> g(a)[[ x =n i2(y)]]

 GOODBYE !
-> quit
```

Nous obtenons le même système final en complétion usuelle qu'en complétion basique. Par superposition entre les règles `(1)` et `(2)`, nous créons en complétion usuelle la règle `g(i1(x)) --> g(a)`, qui elle-même peut être superposée avec `(3)`, on obtient une nouvelle équation qui est aussitôt simplifiée. Alors que dans la complétion basique, en superposant `(1)` et `(2)`, on obtient la règle `g(x) -> g(a)[[ x =n i1(y)]]` qui ne se superpose pas avec `(3)` puisque la position pour faire la paire critique est protégée. La complétion basique calcule 4 paires critiques, alors que la complétion usuelle en calcule 6 pour ensuite en simplifier 2.

Le système final obtenu par complétion basique comportera toujours, au minimum, le même nombre de règles que celui obtenu par complétion usuelle. En effet, la simplification étant moins puissante du côté basique, laissera dans certains cas, des règles redondantes si elles avaient été sans contrainte. Tandis que toutes les règles créées en position non basique, par la complétion usuelle seront simplifiées avant la fin de la procédure. En ne regardant que l'aspect complétion, il nous semble que la basicité de la méthode n'améliore pas globalement les performances de la procédure. Mais la complétion que l'on réalise dans notre procédure, n'est pas une fin en soi. Elle a pour but l'obtention d'un système d'équations, qui va nous servir à résoudre des problèmes d'unification, grâce à la surréduction contrainte. Le fait de compléter notre système de façon basique, va nous permettre d'obtenir un ensemble de règles contraintes, (que l'on a prouvé confluent sur les termes clos) tout en conservant les parties protégées. Avoir des parties protégées dans le système final a pour but de réduire l'espace de recherche de notre surréduction contrainte.

Résolvons maintenant quelques problèmes d'unification avec *UREVEAL*.

4.4.3 Exemples d'unification par surréduction contrainte

Dans l'exemple suivant nous allons spécifier la relation 'est inférieur à' dans un ensemble totalement ordonné à quatre éléments. Il provient du programme Prolog suivant :

```
inf(a,b) <-
inf(b,c) <-
inf(c,d) <-
inf(x,z) <- inf(x,y), inf(y,z)
```

Ce programme Prolog a été traduit en un système de réécriture en suivant la méthode de [BH92]. Nous avons rajouté la définition du *et* logique. *UREVEAL* termine, comme nous allons le voir ci-après, en donnant l'ensemble complet des solutions, alors que le programme Prolog tel que, ne termine pas à cause de la stratégie en profondeur d'abord. Dans ce cas l'utilisation d'un second symbole de prédicat pour définir '*inf*' permet de faire terminer l'exemple en Prolog.

Nous avons au départ les 12 règles suivantes :

$$
\begin{array}{lcrcl}
(e_1) & : & inf(x,x) & \doteq & faux[\![\ ]\!] \\
(e_2) & : & inf(b,a) & \doteq & faux[\![\ ]\!] \\
(e_3) & : & inf(c,b) & \doteq & faux[\![\ ]\!] \\
(e_4) & : & inf(d,c) & \doteq & faux[\![\ ]\!] \\
(e_5) & : & inf(a,b) & \doteq & vrai[\![\ ]\!] \\
(e_6) & : & inf(b,c) & \doteq & vrai[\![\ ]\!] \\
(e_7) & : & inf(c,d) & \doteq & vrai[\![\ ]\!] \\
(e_8) & : & et(inf(x,z),et(inf(x,y),inf(y,z))) & \doteq & et(inf(x,y),inf(y,z))[\![\ ]\!] \\
(e_9) & : & et(vrai,x) & \doteq & x[\![\ ]\!] \\
(e_{10}) & : & et(faux,x) & \doteq & faux[\![\ ]\!] \\
(e_{11}) & : & et(x,vrai) & \doteq & x[\![\ ]\!] \\
(e_{12}) & : & et(x,faux) & \doteq & faux[\![\ ]\!]
\end{array}
$$

où la précédence sur les opérateurs est $et > inf > a > b > c > d > vrai > faux$. Si on veut connaître tous les éléments qui sont inférieurs à a, on exécute la commande 'p_narrow' avec l'unificande $inf(a,x) = vrai$. *UREVEAL* termine et donne au cours de la complétion, les trois solutions suivantes :

```
*** NEW * Solution ***  :  [[ x =n b]]

*** NEW * Solution ***  :  [[ x =n c]]

*** NEW * Solution ***  :  [[ x =n d]]
```

Le système complété final comporte 45 règles.

L'exemple suivant montre que même lorsque la complétion ne termine pas, *UREVEAL* permet d'obtenir rapidement des solutions. Soient deux opérateurs f et g tels que f est distributif à droite et à gauche par rapport à g. Ceci se traduit par les deux équations suivantes :

$$
\begin{array}{lcrcl}
(e_1) & : & f(x,g(y,z)) & \doteq & g(f(x,y),f(x,z))[\![\ ]\!] \\
(e_1) & : & f(g(x,y),z) & \doteq & g(f(x,z),f(y,z))[\![\ ]\!]
\end{array}
$$

Essayons de résoudre dans cette théorie l'unificande $f(a,x) = f(x,y)$. La complétion ne termine pas mais *UREVEAL* donne rapidement les solutions suivantes :

```
*** NEW * Solution ***  :  [[ x =n a ^  y =n a]]
*** NEW * Solution ***  :  [[ x =n g(a,a) ^  y =n a]]
*** NEW * Solution ***  :  [[ x =n g(g(a,a),a) ^  y =n a]]
*** NEW * Solution ***  :  [[ x =n g(a,g(a,a)) ^  y =n a]]
*** NEW * Solution ***  :  [[ x =n g(g(g(a,a),a),a) ^  y =n a]]
*** NEW * Solution ***  :  [[ x =n g(g(a,g(a,a)),a) ^  y =n a]]
*** NEW * Solution ***  :  [[ x =n g(g(a,a),g(a,a)) ^  y =n a]]
*** NEW * Solution ***  :  [[ x =n g(a,g(g(a,a),a)) ^  y =n a]]
*** NEW * Solution ***  :  [[ x =n g(a,g(a,g(a,a))) ^  y =n a]]
```

Avec ces quelques solutions on peut voir facilement la structure d'une solution quelconque.

4.4.4 Utilisation de la surréduction pour valider des spécifications contraintes

L'exemple que nous allons traiter maintenant montre comment nous pouvons utiliser *UREVEAL* pour valider des tests avec des spécifications contraintes. Nous allons spécifier le problème des huit reines sur un échiquier [CY92]. Il s'agit de trouver les différentes possibilités de placer huit reines sur un échiquier de façon qu'aucune reine ne se trouve sur la même ligne, la même colonne ni la même diagonale qu'une autre. Nous allons donner ici une spécification simple du problème, qui permette de tester des solutions trouvées.

Définissons maintenant la spécification associée au problème. Pour cela nous avons la constante $echiquier\_vide$ représentant un échiquier 8×8 où toutes les cases sont vides, et la fonction ens_reine qui permet d'ajouter une reine sur un échiquier. Avec ces deux fonctions nous pouvons décrire toutes les configurations possibles pour un échiquier. Pour tester si une configuration est solution, nous ajoutons la fonction est_sure qui teste si un échiquier sur lequel sont positionnées des reines et une nouvelle position sont sûres, c'est-à-dire si deux reines quelconques sont positionnées : ni sur la même ligne, ni sur la même colonne, ni sur la même diagonale.

Nous avons les équations contraintes suivantes où xe représente une variable échiquier, xl et xc des variables entières pour représenter les lignes et les colonnes.

$$
\begin{aligned}
est\_sure(echiquier\_vide, xl, xc) &\doteq true[\![1 \leq xl \leq 8 \;\wedge\; 1 \leq xc \leq 8]\!] \\
est\_sure(ens\_reine(xe, xl1, xc1), xl2, xc2) &\doteq est\_sure(xe, xl2, xc2) \;\; et \;\; est\_sure(xe, xl1, xc1) \\
&\quad [\![xl1 \neq xl2 \;\wedge\; xc1 \neq xc2 \;\wedge\; xl1 + xc1 \neq xl2 + xc2 \\
&\qquad \wedge\, xl1 - xc1 \neq xl2 - xc2]\!] \\
est\_sure(ens\_reine(xe, xl1, xc1), xl2, xc2) &\doteq false[\![xl1 = xl2]\!] \\
est\_sure(ens\_reine(xe, xl1, xc1), xl2, xc2) &\doteq false[\![xc1 = xc2]\!] \\
est\_sure(ens\_reine(xe, xl1, xc1), xl2, xc2) &\doteq false[\![xl1 + xc1 = xl2 + xc2]\!] \\
est\_sure(ens\_reine(xe, xl1, xc1), xl2, xc2) &\doteq false[\![xl1 - xc1 = xl2 - xc2]\!]
\end{aligned}
$$

Une solution possible est la suivante :

⊗							
						⊗	
				⊗			
							⊗
	⊗						
			⊗				
					⊗		
		⊗					

Pour tester cette solution il suffit d'évaluer l'expression :

$$est\_sure(ens\_reine(ens\_reine(ens\_reine(ens\_reine(ens\_reine(ens\_reine(\\ ens\_reine(echiquier\_vide, 1, 1), 2, 7), 3, 5), 4, 8), 5, 2), 6, 4), 7, 6), 8, 3)$$

et l'on doit trouver *true*.

En l'état *UREVEAL* n'a pas de type prédéfini, et n'accepte que des contraintes symboliques; or là nous avons des contraintes algébriques et des contraintes booléennes. Nous pouvons tout de même spécifier ce problème et ainsi tester des solutions avec *UREVEAL*. Pour cela nous allons intégrer les contraintes dans les équations et donc spécifier les entiers positifs et négatifs avec l'addition et la soustraction de manière 'ad hoc'. Nous utilisons le 'moins' unaire. Nous avons les équations suivantes :

```
s(p(x))=x;
p(s(x))=x;
0 + x = x;
s(x) + y = s(x + y);
p(x) + y = p(x + y);
x + 0 = x;
x + s(y) = s(x + y);
x + p(y) = p(x + y);
(x + y) + z = x + (y + z);
m(0) = 0;
m(s(x)) = p(m(x));
m(p(x)) = s(m(x));
m(m(x))=x;
m(x) + x = 0;
x + m(x) = 0;
x + (m(x) + z) = z;
m(x) + (x + z) = z;
m(x + y) = m(y) + m(x);
```

Nous spécifions aussi les booléens avec le 'et' et la conditionnelle 'si alors sinon' :

```
true et x = x;
x et true = x;
false et x = false;
x et false = false;

si(true,x,y)=x;
si(faux,x,y)=y;
```

Notre spécification de la fonction `diff`, qui retourne `true` quand ses deux arguments sont différents et `false` quand ils sont égaux, n'est pas complète car nous utilisons à la fois les entiers positifs et négatifs. Elle est suffisante pour tester les solutions possibles (les reines doivent être positionnées sur l'échiquier).

```
diff(s(x),y) = diff(x,p(y));
diff(p(x),y) = diff(x,s(y));
diff(x,x) = false;
diff(0,s(0)) = true;
diff(0,s(s(0))) = true;
diff(0,s(s(s(0)))) = true;
```

```
diff(0,s(s(s(s(0))))) = true;
diff(0,s(s(s(s(s(0)))))) = true;
diff(0,s(s(s(s(s(s(0))))))) = true;
diff(0,s(s(s(s(s(s(s(0)))))))) = true;
diff(0,s(s(s(s(s(s(s(s(0))))))))) = true;
diff(0,s(s(s(s(s(s(s(s(s(0)))))))))) = true;
diff(0,s(s(s(s(s(s(s(s(s(s(0))))))))))) = true;
diff(0,s(s(s(s(s(s(s(s(s(s(s(0)))))))))))) = true;
diff(0,s(s(s(s(s(s(s(s(s(s(s(s(0))))))))))))) = true;
diff(0,s(s(s(s(s(s(s(s(s(s(s(s(s(0)))))))))))))) = true;
diff(0,s(s(s(s(s(s(s(s(s(s(s(s(s(s(0))))))))))))))) = true;
diff(0,p(0)) = true;
diff(0,p(p(0))) = true;
diff(0,p(p(p(0)))) = true;
diff(0,p(p(p(p(0))))) = true;
diff(0,p(p(p(p(p(0)))))) = true;
diff(0,p(p(p(p(p(p(0))))))) = true;
diff(0,p(p(p(p(p(p(p(0)))))))) = true;
```

Les cases étant numérotées de `0` à `s(s(s(s(s(s(s(0)))))))`, les sommes et les différences ne dépasseront pas `p(p(p(p(p(p(p(0)))))))` ou `s(s(s(s(s(s(s(s(s(s(s(s(s(s(0))))))))))))))`. Pour le problème en lui-même, nous spécifions la fonction `est_sure` avec les deux équations suivantes. Nous supposons, là encore, que chaque position fait partie de l'échiquier.

```
est_sure(evide, xl, xc) = true;
est_sure(ens_reine(xe,xl1,xc1), xl2, xc2) =
  si(diff(xl1,xl2) et diff(xc1,xc2) et diff(xl1 + xc1, xl2 + xc2)
                                et diff(xl1 + m(xc1), xl2 + m(xc2)),
     est_sure(x,xl2,xc2) et est_sure(x,xl1,xc1) , false);
```

Avec cet ensemble d'équations, la précédence suivante sur les opérateurs, `0 < s < p < + < m < true < false < et < diff < si < evide < ens_reine < est_sure` et un statut de gauche à droite pour les opérateurs `+` et `diff`, *UREVEAL* peut tester toute solution avec la restriction que l'utilisateur n'entre que des positions se situant sur l'échiquier. Ainsi en testant :

```
unif
est_sure(ens_reine(ens_reine(ens_reine(ens_reine(ens_reine(ens_reine(
ens_reine(evide,0,0),s(0),s(s(s(s(s(s(0))))))),s(s(0)),s(s(s(s(0))))),
s(s(s(0))),s(s(s(s(s(s(s(0)))))))),s(s(s(s(0)))),s(0)),s(s(s(s(s(0))))),
s(s(s(0)))),s(s(s(s(s(s(0))))))),s(s(s(s(s(0)))))),s(s(s(s(s(s(s(0)))))))),
s(s(0))) = true;
;
```

On obtient comme ensemble complet de solution l'identité, donc le test est 'bon', c'est une bonne configuration. En revanche il n'y a pas de solution pour l'unificande suivant, la configuration qu'il représente n'est donc pas une solution du problème des huit reines, (la première et la troisième reines sont sur la même colonne).

```
unif
est_sure(ens_reine(ens_reine(ens_reine(ens_reine(ens_reine(ens_reine(
ens_reine(evide,0,0),s(0),s(s(s(s(s(s(0))))))),s(s(0)),0),
s(s(s(0))),s(s(s(s(s(s(s(0)))))))),s(s(s(s(0)))),s(0)),s(s(s(s(s(0))))),
s(s(s(0)))),s(s(s(s(s(s(0)))))),s(s(s(s(s(0)))))),s(s(s(s(s(s(s(0))))))),
s(s(0))) = true;
;
```

L'ensemble des solutions ici est vide.

4.5 Conclusion

Le logiciel *UREVEAL* comporte toutes les fonctions nécessaires à l'implémentation de notre méthode générale d'unification. On dispose entre autre de la génération de paires critiques basiques, et de la simplification par réécriture contrainte, ce qui nous permet d'avoir une procédure de complétion sans échec basique. Nous disposons aussi d'une procédure de surréduction contrainte avec laquelle nous résolvons des problèmes d'unification. Ces deux procédures principales nous ont permis de tester, tout au long de l'élaboration de notre algorithme, la faisabilité pratique, les performances et les limites de notre méthode d'unification générale.

Les améliorations prévues pour *UREVEAL* peuvent être séparées en deux branches. Nous devons accroître l'efficacité et les performances de la surréduction contrainte, et augmenter les possibilités d'utilisation de *UREVEAL* en acceptant de nouveaux types de contraintes comme des contraintes algébriques. Pour les performances nous pensons mettre en place des stratégies pour faire diminuer l'espace de recherche tout en restant complet, comme par exemple l'introduction d'un graphe d'opérateurs, l'utilisation d'un 'indexing' pour la simplification, la mise en place d'une stratégie de surréduction comme la LSE stratégie [KB91, BKW92] adaptée à un cadre contraint. Pour le second point nous devrons certainement prédéfinir des théories dans *UREVEAL* comme par exemple les entiers ou les booléens. Dans les deux cas, les améliorations pratiques seront accompagnées de travaux théoriques pour conserver la correction et la complétude de notre méthode.

Une partie de notre travail futur est donc de rendre réellement performant *UREVEAL*, avec l'aide de stratégies diverses et d'algèbres prédéfinies.

Conclusion

L'unification reste un domaine central de recherche en démonstration automatique. La surréduction est une méthode générale et complète, d'unification dans les systèmes de réécriture confluents et noethériens. Dans ce travail, nous avons combiné le mécanisme de complétion, qui permet d'obtenir de tels systèmes, et le mécanisme de surréduction, pour obtenir une nouvelle méthode d'unification, générale et complète. Nous avons travaillé dans le cadre contraint, ce qui nous a semblé être le formalisme idéal, pour optimiser la méthode en intégrant la notion de basicité : nous stockons les parties protégées d'un terme dans une contrainte associée au terme. Dans une seconde partie, nous avons décrit des stratégies basées sur des graphes d'opérateurs, ou de termes, et ayant pour but d'améliorer l'efficacité et la terminaison de la surréduction.

La complétion est basée sur la génération de paires critiques entre les équations. Cette méthode n'est efficace que si elle est accompagnée d'étapes de simplification, permettant de limiter le nombre de nouvelles équations engendrées. Mais en complétion basique, avec la simplification usuelle, nous perdons la complétude de notre méthode pour la surréduction. Pour contrer cela, il fallait interdire certaines étapes de simplifications, qui sont des étapes permises dans d'autres approches "purement" orientée vers la complétion. Notre travail présente de manière explicite une notion de filtre contraint qui est à la base de notre définition de la réécriture contrainte. Cette réécriture nous permet d'avoir des étapes de simplification pendant la complétion basique de notre ensemble d'équations initiales. Ces étapes de simplification sont efficaces, nous avons pu le constater avec notre logiciel *UREVEAL*, et elles vérifient les deux propriétés suivantes :

- une partie protégée, dans une équation contrainte, ne sera pas déprotégée par une étape de simplification, ce qui garantit la basicité de la méthode,
- une partie non protégée dans une équation contrainte ne sera pas protégée par une étape de simplification, ce qui est nécessaire pour la complétude de la méthode.

Parallèlement à cela, nous avons défini une surréduction basique dans un cadre contraint. Les unificateurs ne sont pas calculés pendant l'étape de surréduction, mais ils sont stockés dans les contraintes.

La procédure générale d'unification est décrite par un système de règles d'inférence, combinant la complétion basique et la surréduction basique. Partant d'une théorie équationnelle quelconque, d'un ordre de simplification et d'une équation à résoudre, notre procédure calcule, au bout d'un temps fini et en général tout a fait raisonnable, toute solution donnée. Ceci reste vrai même lorsque le système d'équations initial n'admet pas de système confluent et noethérien fini, car les étapes de surréduction sont faites en même temps que la complétion de notre système.

Travaillant dans un cadre contraint, il était logique d'accepter que les équations initiales puissent être contraintes. Dans cette thèse nous avons uniquement traité le cas de contraintes symboliques (contraintes égalitaires ou contraintes d'ordre). Une équation contrainte représente un ensemble d'équations. Pour obtenir cet ensemble d'équations, qui est indispensable pour la complétude de notre méthode, nous devons parfois instancier les variables de l'équation contrainte par les différentes solutions de la contrainte. Nous donnons une méthode, dite de propagation minimale, pour générer, à partir d'une équation contrainte, un ensemble minimum d'équations nécessaires à notre méthode d'unification. Dans certains cas, l'ensemble obtenu peut tout de même être infini. Cela ne provoque pas un blocage de notre procédure puisque les propagations (des contraintes initiales) sont faites par étapes (finies) pendant la procédure.

Cette méthode d'unification est implantée dans le logiciel *UREVEAL*, où nous avons également la possibilité de réaliser seulement une complétion basique d'un ensemble d'équations. Après de nombreux essais, il ne semble pas que la notion de basicité améliore l'efficacité pratique de la procédure de complétion. Dans notre travail, la complétion basique d'un ensemble d'équations n'est pourtant pas un but en soi. Nous utilisons les règles que nous engendrons pendant la complétion, pour faire des étapes de surréduction. En générant des équations avec des parties protégées pendant les étapes de complétion basique, nous réduisons le nombre d'étapes de surréduction, et améliorons ainsi son efficacité. Ceci dit, notre méthode d'unification reste bien évidemment valable même si nous remplaçons les étapes de complétion basique par de la complétion usuelle.

Dans cette thèse, nous avons aussi présenté deux stratégies pour améliorer l'efficacité et la terminaison de la surréduction. Elles sont toutes deux basées sur la notion combinatoire de graphe : l'une sur un graphe d'opérateurs et l'autre sur un graphe de termes. La seconde est meilleure que la première, mais beaucoup plus compliquée à mettre en place. Nous construisons un graphe à partir de l'ensemble d'équations décrivant la théorie équationnelle et l'équation à résoudre, puis grâce à ce graphe, nous éliminons des branches inutiles dans les étapes de surréduction. C'est ce que nous appelons stratégie de surréduction dirigée par un graphe. Nous avons montré comment il était possible d'intégrer ces améliorations dans nos étapes de surréduction contrainte. Mais une meilleure synthèse serait certes obtenue en considérant les restrictions imposées par l'utilisation d'un graphe, comme un nouveau type de contrainte. Ceci permettrait de conserver la procédure d'unification intacte, et faciliterait l'introduction de nouvelles stratégies. Ce travail reste pour nous un objectif pour l'avenir.

Le problème majeur lié aux méthodes d'unifications générales est le problème d'efficacité et de terminaison dans la pratique. *UREVEAL* nous a permis de tester ces deux points, au cours de l'élaboration de notre procédure d'unification. Dans la majorité des cas, les premières solutions d'un problème sont calculées assez rapidement. Mais ensuite l'espace de recherche devient trop grand pour pouvoir être géré de façon satisfaisante. C'est pourquoi nous envisageons maintenant de concentrer une partie de nos efforts à faire diminuer le nombre de branches créées par les étapes de surréduction. Pour cela nous avons pensé à :

- introduire le graphe d'opérateurs dans *UREVEAL*, ce qui permettra de supprimer un certain nombre de branches de surréduction inutiles,

- introduire de nouvelles stratégies de surréduction ayant pour but de supprimer les branches redondantes pour calculer une seule fois chaque solution,
- optimiser la simplification dans les étapes de complétion basique avec des méthodes permettant de choisir plus rapidement les règles simplificatrices.

Ces travaux futurs, qui ont pour but d'optimiser notre méthode, nous semblent réalisables car ils correspondent à l'adaptation, au cadre contraint, de méthodes déjà existantes. Nous avons aussi pensé à augmenter les capacités de notre méthode en acceptant de nouveaux types de contraintes dans les équations initiales, comme par exemple des contraintes faisant intervenir des entiers ou des booléens. Pour cela, nous devrons inclure des théories prédéfinies et disposer d'algorithmes de résolution de contraintes dans ces théories. Ces travaux, plus long à réaliser que les précédents, seront sans doute indispensables si l'on souhaite voir la surréduction contrainte comme un outil de validation de spécifications contraintes.

Bibliographie

[AABH90] S. ANANTHARAMAN, N. ANDRIANARIVELO, M. P. BONACINA, and J. HSIANG. Sbr3 : A refutational prover for equational theorems. *In S. KAPLAN and M. OKADA, editors*, *Proceedings 2nd international workshop on conditional and typed rewriting systems, montreal (canada)*, volume 516 of *Lecture Notes in Computer Science*. Spring-Verlag, 1990.

[AH90] S. ANANTHARAMAN and J. HSIANG. An automated proof of the Moufang identities in alternative rings. *Journal of automated reasoning*, 6:79–109, 1990.

[AHM89] S. ANANTHARAMAN, J. HSIANG, and J. MZALI. Sbreve2: A term rewriting laboratory with (AC-)unfailing completion. *In N. DERSHOWITZ, editor*, *Proceedings 3rd conference on rewriting techniques and applications, chapel hill (north carolina, usa)*, volume 355 of *Lecture Notes in Computer Science*, pages 533–537. Spring-Verlag, April 1989.

[AND91] N. ANDRIANARIVELO. Démonstration automatique dans les theorie de Horn. *Thèse de Doctorat d'Etat*, *Université d'Orléans*, 1991.

[BAC87] L. BACHMAIR. Proof methods for equational theories. PhD thesis, *University of Illinois*, *Urbana-Champaign*, 1987. *Revised version, august 1988.*

[BCL87] A. BEN CHERIFA and P. LESCANNE. Termination of rewriting systems by polynomial interpretations and its implementation. *Science of computer programming*, 9(2):137–160, October 1987.

[BDH86] L. BACHMAIR, N. DERSHOWITZ, and J. HSIANG. Orderings for equational proofs. *In Proceedings 1st ieee symposium on logic in computer science, cambridge (ma, usa)*, pages 346–357. IEEE, 1986.

[BG91] L. BACHMAIR and H. GANZINGER. Rewrite-based equational theorem proving with selection and simplification. *Technical Report* MPI-I-91-208, *Max-planck institut für informatik, saarbrücken*, 1991.

[BGLS92] L. BACHMAIR, H. GANZINGER, C. LYNCH, and W. SNYDER. Basic paramodulation and superposition. *In Proceedings 11th international conference on automated deduction, albany (ny, usa)*, 1992.

[BGM87] P. G. BOSCO, E. GIOVANNETTI, and G. MOISO. Refined strategies for semantic unification. *In Proceedings of tapsoft'87*, volume 150 of *Lecture Notes in Computer Science*, pages 276–290. Spring-Verlag, 1987.

[BGSL93] L. BACHMAIR, H. GANZINGER, W. SNYDER, and C. LYNCH. Basic paramodulation. *Technical Report* MPI-I-93-236, *Max-planck institut für informatik, saarbrücken*, September 1993.

[BH92] M. P. BONACINA and J. HSIANG. On rewrite programs : Semantics and relationship with prolog. *Journal of logic programming*, 14:155–180, 1992.

[BKW92] A. BOCKMAYR, S. KRISCHER, and A. WERNER. An optimal narrowing strategy for general canonical systems. *In M. RUSINOWITCH and J. L. RÉMY, editors, Proceedings 3rd international workshop on conditional and typed rewriting systems, pont-à-mousson (france)*, pages 247–251, July 1992.

[BKW93] A. BOCKMAYR, S. KRISCHER, and A. WERNER. Narrowing strategies for arbitrary canonical rewrite systems. *Technical Report* 22-93, *Universitat karlsruhe*, July 1993.

[BOU89] A. BOUDET. A new combination technique for ac unification. *Internal report* 494, *Lri, ORSAY (FRANCE)*, June 1989.

[BS] F. BAADER and J. H. SIEKMANN. Unification theory. *In D. M. GABBAY, C. J. HOGGER, and J. A. ROBISON, editors*, *Handbook of logic in artificial intelligence and logic programming. Oxford university press*, *LONDON. To appear.*

[CHA90] J. CHABIN. Surréduction dirigée par un graphe d'opérateurs. formulation équitable par des système d'équations et implantation. *Rapport de DEA*, *ORLÉANS*, 1990.

[CHR89] J. CHRISTIAN. High performance permutative completion. *Technical report* ACT-AI-303-89, *Mcc*, August 1989. *Phd thesis.*

[CR91a] J. CHABIN and P. RÉTY. Narrowing directed by a graph of terms. *In R. V. BOOK, editor*, *Proceedings 4th conference on rewriting techniques and applications, como (italy)*, volume 488 of *Lecture Notes in Computer Science*, pages 112–123. Spring-Verlag, April 1991.

[CR91b] J. CHABIN and P. RÉTY. Narrowing directed by a graph of terms (including proofs). *Technical Report* 91-1, *Laboratoire d'Informatique Fondamentale d'Orléans*, 1991.

[CR92] J. CHABIN and P. RÉTY. Directed narrowing including merging. *Technical Report* 92-1, *Laboratoire d'Informatique Fondamentale d'Orléans*, 1992.

[CY92] S. CALABRETTO and P. YIM. La programmation logico-fonctionnelle avec contraintes : Un outil pour la validation des specifications algebriques avec contraintes. *In Proceedings of the 1st journées francophones de programmation en logique, lille (france)*, pages 232–250, May 1992.

[DER82] N. DERSHOWITZ. Orderings for term-rewriting systems. *Theoretical computer science*, 17:279–301, 1982.

[DJ90] D. DOUGHERTY and P. JOHANN. An improved general E-unification method. *In M. E. STICKEL, editor*, *Proceedings 10th international conference on automated deduction, kaiserslautern (germany)*, volume 449 of *Lecture Notes in Computer Science*, pages 261–275. Spring-Verlag, July 1990.

[DJ92] D. DOUGHERTY and P. JOHANN. An improved general E-unification method. *Journal of symbolic computation*, 14:303–320, 1992.

[DS87] N. DERSHOWITZ and G. SIVAKUMAR. Solving goals in equational langages. *In S. KAPLAN and J-P. JOUANNAUD, editors*, *Proceedings of the first international workshop on conditional and typed rewriting systems*, volume 308 of *Lecture Notes in Computer Science*, pages 45–55. Spring-Verlag, 1987.

[DS88] N. DERSHOWITZ and G. SIVAKUMAR. Goal-directed equation solving. *In Proceedings of the seventh national conference on artificial intelligence*, pages 166–170, *ST. PAUL, MN (USA)*, August 1988.

[FAY78] M. FAY. First order unification in equational theories. PhD thesis, *UNIVERSITY OF CALIFORNIA*, *SANTA CRUZ*, May 1978. *Master thesis 78-5-002*.

[FAY79] M. FAY. First order unification in equational theories. *In Proceedings 4th workshop on automated deduction, austin (texas, usa)*, pages 161–167, 1979.

[FG84] R. FORGAARD and JOHN V. GUTTAG. Reve: A term rewriting system generator with failure-resistant KNUTH-BENDIX. *Technical Report* , *Mit-lcs*, 1984.

[FH83] F. FAGES and G. HUET. Unification and matching in equational theories. *In Proceedings fifth colloquium on automata, algebra and programming, l'aquila (italy)*, Lecture Notes in Computer Science. Spring-Verlag, 1983.

[GAN91] H. GANZINGER. A completion procedure for conditional equations. *Journal of symbolic computation*, 11:51–81, 1991.

[GS89] J. GALLIER and W. SNYDER. Complete sets of transformations for general E-unification. *Theoretical computer science*, 67(2-3):203–260, October 1989.

[HER30] J. HERBRAND. Recherches sur la théorie de la démonstration. *Travaux de la soc. des sciences et des lettres de varsovie, classe iii*, 33(128), 1930.

[HER86] A. HEROLD. Narrowing techniques applied to idempotent unification. *Technical Report* SEKI-REPORT SR-86-16, *Seki*, August 1986.

[HM88] J. HSIANG and J. MZALI. SBREVE user's guide. *Technical Report* , *Lri*, *ORSAY, FRANCE*, 1988.

[HR87] J. HSIANG and M. RUSINOWITCH. On word problem in equational theories. *In TH. OTTMANN, editor*, *Proceedings of 14th international colloquium on automata, languages and programming, karlsruhe (germany)*, volume 267 of *Lecture Notes in Computer Science*, pages 54–71. Spring-Verlag, 1987.

[HUL80a] J.-M. HULLOT. Canonical forms and unification. *In W. BIBEL and R. KOWALSKI, editors*, *Proceedings 5th international conference on automated deduction, les arcs (france)*, volume 87 of *Lecture Notes in Computer Science*, pages 318–334. Spring-Verlag, July 1980.

[HUL80b] J.-M. HULLOT. Canonical forms and unification. *In Proceedings 5th international conference on automated deduction, les arcs (france)*, pages 318–334, July 1980.

[HUL80c] J.-M. HULLOT. Compilation de formes canoniques dans les théories équationelles. *Thèse de Doctorat de Troisième Cycle*, *UNIVERSITÉ DE PARIS SUD*, *ORSAY (FRANCE)*, 1980.

[KB70] DONALD E. KNUTH and P. B. BENDIX. Simple word problems in universal algebras. *In J. LEECH, editor*, *Computational problems in abstract algebra*, pages 263–297. *Pergamon press*, *OXFORD*, 1970.

[KB91] S. KRISCHER and A. BOCKMAYR. Detecting redundant narrowing derivations by the LSE-SL reducibility test. *In R. V. BOOK, editor*, *Proceedings 4th conference on rewriting techniques and applications, como (italy)*, volume 488 of *Lecture Notes in Computer Science*, pages 74–85. Spring-Verlag, April 1991.

[KIR84] C. KIRCHNER. A new equational unification method: A generalization of Martelli-Montanari algorithm. *In R. SHOSTAK, editor, Proceedings 7th international conference on automated deduction, napa valley (ca, usa)*, volume 170 of *Lecture Notes in Computer Science*. Spring-Verlag, 1984.

[KIR85] C. KIRCHNER. Méthodes et outils de conception systématique d'algorithmes d'unification dans les théories équationnelles. *Thèse de Doctorat d'Etat, Université de Nancy I*, 1985.

[KIR86] C. KIRCHNER. Computing unification algorithms. *In Proceedings 1st ieee symposium on logic in computer science, cambridge (ma, usa)*, pages 206–216, 1986.

[KIR87] CLAUDE KIRCHNER. Methods and tools for equational unification. *Technical Report , Crin-inria, BP 239, VANDŒUVRE LÈS NANCY CEDEX, FRANCE*, December 1987.

[KK86] C. KIRCHNER and H. KIRCHNER. Reveur-3: Implementation of a general completion procedure parametrized by built-in theories and strategies. *Science of computer programming*, 20(8):69–86, 1986.

[KKR90] C. KIRCHNER, H. KIRCHNER, and M. RUSINOWITCH. Deduction with symbolic constraints. *Revue d'intelligence artificielle*, 4(3):9–52, 1990. *Special issue on automatic deduction*.

[LAN79] D. S. LANKFORD. Mechanical theorem proving in field theory. *Technical Report , Louisiana tech. university*, 1979.

[LB79] D. S. LANKFORD and A. BALLANTYNE. The refutation completeness of blocked permutative narrowing and resolution. *In Proceedings 4th workshop on automated deduction, austin (texas, usa)*, volume 87 of *Lecture Notes in Computer Science*, pages 53–59. Spring-Verlag, 1979.

[LES83] P. LESCANNE. Computer experiments with the REVE term rewriting systems generator. *In Proceedings of 10th acm symposium on principles of programming languages*, pages 99–108. Association for Computing Machinery, 1983.

[LIM92] S. LIMET. La surréduction dirigée par un graphe d'opérateurs. *Rapport de DEA, ORLÉANS*, 1992.

[MM82] A. MARTELLI and U. MONTANARI. An efficient unification algorithm. *Acm transactions on programming languages and systems*, 4(2):258–282, 1982.

[MMR86] A. MARTELLI, C. MOISO, and G. F. ROSSI. An algorithm for unification in equational theories. *In Proceedings of the ieee symposium on logic programming*, pages 180–186, September 1986.

[MMR87] A. MARTELLI, C. MOISO, and G-F. ROSSI. Lazy unification algorithms for canonical rewrite systems. *In H. AÏT-KACI and M. NIVAT, editors, Colloquium on the resolution of equations in algebraic structures, AUSTIN (TEXAS, USA)*, 1987. MCC AND INRIA.

[MZA86] J. MZALI. Méthodes de filtrage équationnel et de preuve automatique de théorèmes. *Thèse de Doctorat d'Université, Université de Nancy I*, 1986.

[NR92a] R. NIEUWENHUIS and A. RUBIO. Basic superposition is complete. *In B. KRIEG-BRÜCKNER, editor, Proceedings of esop'92*, volume 582 of *Lecture Notes in Computer Science*, pages 371–389. Spring-Verlag, 1992.

[NR92b] R. NIEUWENHUIS and A. RUBIO. Theorem proving with ordering constrained clauses. *In Proceedings 11th international conference on automated deduction, albany (ny, usa)*, volume 617 of *Lecture Notes in Artificial Intelligence*, pages 477–491. Spring-Verlag, June 1992.

[OS90] A OHSUGA and K. SAKAI. Complete equational unification based on an extension of the knuth-bendix completion procedure. *In K. Schulz, editor*, *Proc. of the first international workshop on word equations and related topics*, volume 572 of *Lecture Notes in Computer Science*, pages 197–209. Spring-Verlag, October 1990.

[PLO72] G. PLOTKIN. Building-in equational theories. *Machine intelligence*, 7:73–90, 1972.

[RÉT87] P. RÉTY. Improving basic narrowing. *In P. LESCANNE, editor*, *Proceedings 2nd conference on rewriting techniques and applications, bordeaux (france)*, volume 256 of *Lecture Notes in Computer Science*, pages 228–241, *BORDEAUX (FRANCE)*, May 1987. Spring-Verlag.

[RÉT88] P. RÉTY. Méthodes d'unification par surréduction. *Thèse de Doctorat d'Université*, *Université de Nancy I*, March 1988.

[ROB65] J. A. ROBINSON. A machine-oriented logic based on the resolution principle. *Journal of the association for computing machinery*, 12:23–41, 1965.

[RUS87] M. RUSINOWITCH. Démonstration automatique par des techniques de réécriture. *Thèse de Doctorat d'Etat*, *Université de Nancy I*, 1987. *Also published by intereditions, collection science informatique, directed by g. huet, 1989.*

[SL91] W. SNYDER and C. LYNCH. Basic paramodulation. *Technical Report* , *Boston university*, 1991. *Presented at the 5th international workshop on unification, Barbizon (France).*

[SLA74] J. R. SLAGLE. Automated theorem-proving for theories with simplifiers, commutativity and associativity. *Journal of the association for computing machinery*, 21(4):622–642, 1974.

Résumé

La surréduction est une méthode générale et complète, d'unification dans les systèmes de réécriture confluents et noethériens. Dans ce travail, nous avons combiné le mécanisme de complétion dite ordonnée, qui permet d'obtenir de tels systèmes éventuellement infinis, et le mécanisme de surréduction, pour obtenir une nouvelle méthode d'unification, générale et complète. Nous avons travaillé dans le cadre contraint, ce qui nous a semblé être le formalisme idéal pour optimiser la méthode en intégrant la notion de basicité : nous stockons les parties protégées d'un terme dans une contrainte associée au terme.

Notre procédure d'unification est décrite par un système de règles d'inférence, combinant la complétion ordonnée basique et la surréduction basique. Partant d'une théorie équationnelle quelconque, d'un ordre de simplification et d'une équation à résoudre, notre procédure calcule, au bout d'un temps fini, toute solution donnée. Ceci reste vrai même lorsque le système d'équations initial n'admet pas de système confluent et noethérien fini, car les étapes de surréduction sont faites en même temps que la complétion de notre système. Cette méthode d'unification est implantée dans notre logiciel UREVEAL.

Dans cette thèse, nous avons aussi présenté par ailleurs, deux stratégies pour améliorer l'efficacité et la terminaison de la surréduction. Elles sont toutes deux basées sur la notion combinatoire de graphe : l'une sur un graphe d'opérateurs et l'autre sur un graphe de termes. La seconde est meilleure que la première, mais plus compliquée à mettre en place. Nous construisons un graphe à partir de l'ensemble d'équations décrivant la théorie équationnelle et l'équation à résoudre, puis grâce à ce graphe, nous éliminons des branches inutiles dans les étapes de surréduction. C'est ce que nous appelons surréduction dirigée par un graphe. Ces stratégies d'optimisation ne seront implantées qu'ultérieurement.

Mots clés : résolution d'équations, unification, réécriture, surréduction basique, complétion basique, contrainte, graphe d'opérateurs, graphe de termes.

Abstract

Narrowing is a general and complete tool for unification w.r.t. a canonical rewrite system. In this thesis, we combine ordered completion, which gives us a possibly infinite canonical rewrite system and narrowing, to deduce a complete procedure for general unification. We work in a constrained setup which seems to be an ideal formalism to optimize our method, by adaptinfg the so-called basic strategy : we keep the protected part of a term in a constraint linked to the term.

We give a set of inference rules to describe our procedure, combining ordered basic completion and basic narrowing. Given any equational theory, a simplification ordering, and an equation to be solved, our procedure computes, in finite time, any solution. We do not assume here that the initial set of equations admits a finite canonical system, because steps of narrowing are computed simultaneously w.r.t. the completion steps. This method is implemented in our software UREVEAL.

In this thesis we also study two combinatorial strategies to improve efficiency and termination of narrowing. Both are based on the notion of a graph : first on a graph of operators, the other on a graph of terms. The second is better than the first, but more complicated to run. We build a graph with the set of equations discribing our equationnal theory and the equation to be solved; then with the graph we cut useless narrowing steps. This is called narrowing directed by a graph. Such strategies are not yet built into UREVEAL.

Keywords : equations solving, unification, term rewriting, basic narrowing, basic completion, constraint, graph of operators, graph of terms.