Indépendance de la règle de calcul pour la résolution SLD avec retard*

Michel Bergère†, Gérard Ferrand†‡, Jean-Hugues Réty§

Résumé

Nous étudions la notion d'*enlisement* pour la résolution SLD avec retard. Une conséquence particulière de notre principal résultat est que l'existence d'un enlisement ne dépend pas de la règle de calcul. Cette conséquence particulière est très similaire à certains résultats déjà connus dans le domaine de la programmation logique concurrente. Cependant notre principal résultat est une extension du résultat classique d'indépendance de la règle de calcul : nous montrons que, étant donné un programme et un but définis, deux arbres SLD ont, en un certain sens, les mêmes feuilles succès et les mêmes feuilles enlisement. Nous donnons une explication très précise de cette indépendance en terme d'*arbres de preuve* : des *arbres de preuve partiels* définis de façon intrinsèque au programme (et donc ne dépendant pas de la règle de calcul) sont associés aux feuilles enlisement. Nous commençons par prouver un résultat préliminaire sur des systèmes d'équations afin d'énoncer et de prouver notre résultat.

Mots clefs : Programme Défini, Résolution SLD, Retard, enlisement, Règle de calcul, Arbre de preuve, Equation.

1 Introduction

Il existe pour la *Résolution SLD* un résultat classique ([15, 7]) appelé *Indépendance de la règle de calcul* dans [18, 13] (voir aussi [1]) qui peut être formulé en terme d'*arbres SLD* de la façon suivante : Etant donné un *programme défini* et un but deux arbres SLD ont, en un certain sens, les mêmes *feuilles succès*. L'ensemble de ces feuilles succès ne dépend donc pas de la règle de calcul (la règle de calcul définit l'*atome choisi* qui est candidat à l'unification avec une tête de clause, ainsi une règle de calcul particulière spécifie un arbre SLD particulier). De plus cet ensemble de feuilles succès correspond aux *réponses les plus générales*.

Il est possible de donner une explication plus précise de ce résultat d'indépendance en terme d'*arbres de preuve*. Cette vue est développée dans [13] (elle est aussi décrite dans [14]). Un exemple particulier de son utilité se trouve dans [12, 13] où elle est utilisée pour énoncer et pour prouver certaines propriétés qui ne dépendent pas d'une stratégie particulière. Un arbre de preuve est étiqueté par des *atomes* et peut être vu comme un "assemblage" d'instances de clauses ([7, 28]). Chaque branche succès d'un arbre SLD est le calcul d'un *arbre de preuve*. Les clauses utilisées sur cette branche succès définissent le *squelette* de l'arbre de preuve. En fait cet arbre de preuve calculé est l'arbre de preuve *le plus général* sur ce squelette (ceci provient de l'utilisation de *l'unificateur le plus général (upg)* à chaque noeud de la branche succès). Ainsi la résolution SLD peut être vue comme une méthode *descendante* de calcul des arbres de preuve les plus généraux. Cependant les notions d'arbre de preuve et d'arbre de preuve le plus général ne dépendent pas d'une stratégie particulière. Plus précisément, un programme défini et un but définissent intrinsèquement un ensemble d'arbres de preuve les plus généraux.

*Cette recherche a été financée en partie par le CNRS (GRECO Programmation SCALP).

†Université d'Orléans, Département d'Informatique, LIFO BP 6759, 45067 Orléans cedex 2, France, {**Bergere,Ferrand**}**@univ-orleans.fr**

‡also INRIA - Rocquencourt.

§INRIA - Rocquencourt, B.P. 105, 78153 Le Chesnay cedex, France, **Jean-Hugues.Rety@inria.fr**

La résolution SLD est un *algorithme non déterministe complet* pour calculer cet ensemble. Cet algorithme est paramètré par la règle de calcul : chaque règle de calcul définit un tel algorithme non déterministe complet dont les calculs sont décrits par un arbre SLD. L'indépendance de la règle de calcul provient de l'existence d'une bijection entre l'ensemble des feuilles succès d'un arbre SLD et l'ensemble des arbres de preuve les plus généraux.

Prolog standard utilise la règle de calcul qui choisit toujours le littéral le plus à gauche. Cependant les avantages d'une règle de calcul plus souple (programmation plus déclarative, correction des appels négatifs) sont bien connus. Certains langages de programmation en logique comme MU-Prolog, NU-Prolog, Prolog-II, SEPIA, Gödel, utilisent une règle de calcul partiellement spécifiée par une notion de *retard* : certains atomes sont retardés (i.e. ne peuvent pas être choisis) jusqu'à ce qu'ils soient suffisament instanciés ([25, 16, 11]). Ceci peut être vu comme du *coroutinage*. Il s'ensuit une notion d'*enlisement* lorsque tous les atomes sont retardés (*dealock* du point de vue du coroutinage). Aussi nous allons devoir considérer la résolution SLD avec retard et des arbres SLD avec d'éventuelles *feuilles enlisement*.

Dans ce papier nous énonçons et nous prouvons, pour la résolution SLD avec retard, une extension du résultat classique d'indépendance (de la règle de calcul) : Nous prouvons que deux arbres SLD possèdent, en un certain sens, les mêmes feuilles succès *et les mêmes feuilles enlisement*. Il sagit donc d'un résultat théorique qui montre que même en cas d'enlisement, l'ensemble des feuilles succès et l'ensemble des feuilles enlisement sont intrinsèquement définis par le programme et le but puisqu'ils ne dépendent pas de la règle de calcul.

De plus nous donnons une explication de ce résultat d'indépendance en terme d'*arbres de preuve* : celui-ci provient, d'une part d'une bijection entre l'ensemble des feuilles succès d'un arbre SLD et l'ensemble des arbres de preuve les plus généraux, et d'autre part d'une bijection entre l'ensemble des feuilles enlisement d'un arbre SLD et un nouvel ensemble d'*arbres de preuve partiels*. Ce nouvel ensemble est définit intrinsèquement par un programme et un but. Cette vision des choses est développée dans [13] (cependant notre papier est auto-contenu).

Afin d'énoncer et de prouver notre résultat, un *système d'équations* est associé à chaque arbre de preuve (partiel) et nous commençons par prouver un résultat concernant les systèmes d'équations en général.

Une approche similaire est utilisée dans [22, 6, 5] afin d'étudier une autre extension de la résolution SLD appelée résolution S-SLD dans laquelle une unification étendue est utilisée. L'objet de ces papiers est l'appel de procédures externes. De tels appels doivent être clos. Aussi, notre notion de retard est plus générale (les conditions de retard (d'un terme) sont plus précises). De plus, l'enlisement ne survient pas nécessairement au niveau des équations originelles. Il peut aussi apparaître au niveau des nouvelles équations entre sous termes (S-Unification). Un résultat d'indépendance est aussi prouvé dans ce contexte mais par des méthodes de preuve différentes. Ce résultat et le nôtre sont similaires mais ne semblent pas être conséquence l'un de l'autre.

Une conséquence particulière de notre résultat est que l'existence d'un enlisement ne dépend pas de la règle de calcul. Ceci est très proche de certains résultats déjà connus concernant *l'analyse de suspension* pour les *programmes logiques concurrents* ([8, 9, 10]). Ces résultats utilisent l'*interprétation abstraite* et prouvent que la correction de l'analyse de suspension est indépendante du scheduling. En particulier l'approche de [10] est basée sur des abstractions d'arbres SLD. [9] montre comment l'analyse de *suspension* peut être modifiée afin d'obtenir une analyse d'*enlisement* par utilisation d'un résultat de [29]. (voir aussi [30], [29] et [30] supposent certaines restrictions sur la règle de calcul).

En comparaison avec ces travaux, notre principale contribution est un résultat théorique général qui est une extension de l'Indépendance de la Règle de Calcul sous une forme très précise en terme d'arbres de preuve. De plus, nous mettons en évidence le fait que, d'un point de vue théorique, certains détails embarassants concernants variables, renommages, variantes de clauses, ... peuvent être éliminés en considérant la résolution SLD comme un calcul de *squelettes* (d'arbres de preuve) et en reléguant l'utilisation de variables aux *systèmes d'équations*. De plus, nous associons aux feuilles enlisement des arbres SLD des (squelettes) d'arbres de preuve partiels intrinsèquement définis par le programme et le but. Aussi ceux-ci peuvent s'avérer utiles

pour un *diagnostic déclaratif* d'enlisement (si l'enlisement est considéré comme une erreur de programmation).

Le papier est organisé comme suit : La section 2 traite de systèmes d'équations et de retard indépendamment de la considération de programmes logiques. La section 3 applique les résultats de la section 2 indépendamment de la considération de la résolution SLD. La section 4 applique les résultats des sections 2 et 3 à la résolution SLD avec retard. La section 5 est une conclusion. Pour des raisons de place, les preuves des sections 2 et 3 ont été supprimées : elles sont disponibles dans la version complète de ce papier ([4]).

2 Unification et systèmes d'équations avec retard

Nous supposons que le lecteur est familier avec les notions élémentaires concernant l'unification ([26, 20, 18, 1]).

La notion de *terme* est définie de façon classique étant donné une signature et un ensemble VAR de variables. Soit $TERM$ l'ensemble des termes. Une *substitution* est une application σ de VAR dans $TERM$ étendue de façon classique de $TERM$ dans $TERM$. L'*instance* d'un terme t par une substitution σ est notée σt.

Une *équation* est un couple (ordonné) de termes noté $t = t'$.

L'*instance* d'une équation $t = t'$ par une substitution σ est l'équation $\sigma t = \sigma t'$ (en général nous appliquons de façon évidente les substitutions à toute structure comportant des variables).

Une *solution* de l'équation $t = t'$ est une substitution σ telle que les termes σt et $\sigma t'$ soient syntaxiquement égaux.

Un *système d'équations* est un famille finie $S = (e_i)_{i \in I}$ où chaque e_i est une équation. Une solution du système $S = (e_i)_{i \in I}$ est une substitution qui est solution de chaque équation $e_i, i \in I$.

Etant donné $J \subseteq I$, la *restriction* de S à J est le système $S|_J = (e_i)_{i \in J}$.

Rappelons qu'une méthode pour résoudre un système est de résoudre les équations dans un certain ordre (algorithme d'unification). Plus formellement :

Definition 2.1 *Un calcul pour le système* $S = (e_i)_{i \in I}$ *est une suite d'*états de calcul $(J_k, \mu_k)_{k=0,\cdots,m}$ *où* (J_{k+1}, μ_{k+1}) derive *de* (J_k, μ_k), *un* état de calcul *étant un couple* (J, μ) *où* $J \subseteq I$ *et* μ *est soit une* substitution *soit le symbole "échec".*

*L'*état de calcul initial (J_0, μ_0) *est défini par* $J_0 = \emptyset, \mu_0 = \{\ \}$ *(substitution identité).*

L'état (J', μ') derive *de l'état* (J, μ) *si :*

(1) $\mu \neq$ *échec*

(2) Il existe un $j \in I$ *tel que :*

- *(i)* $J' = J \cup \{j\}$ *et* $j \notin J$
- *(ii) Si l'équation* μe_j *est sans solution, alors* $\mu' =$ *échec, sinon il existe un* upg *(unificateur le plus général)* μ'' *de cette équation vérifiant* $\mu' = \mu'' \circ \mu$.

Un état (J, μ) *est* terminal *si aucun état n'en dérive.*

Un calcul est terminé *si son dernier état est terminal.*

Remarque 2.1 *L'état* (J, μ) *est* terminal *ssi l'une au moins des deux conditions suivantes est satisfaite :*

(i) $\mu =$ *échec*

(ii) $J = I$

Il est bien connu que dans un calcul $(J_k, \mu_k)_{k=0,\cdots,m}$, μ_k est un *upg* de $S|_{J_k}$ pour chaque $k < m$. De plus, si le calcul est terminé, le système admet une solution si et seulement si $\mu_m \neq échec$. Dans ce cas μ_m est un *upg* du système.

Nous nous intéressons maintenant à une notion de *calcul avec retard* : certains calculs ne sont pas autorisés car certaines équations ne peuvent pas être résolues tant qu'elles ne satisfont pas une condition. Cette condition concerne l'instantiation du *membre gauche* de l'équation. Plus formellement :

Definition 2.2 *Soit $R \subseteq TERM$ supposé* fermé par substitution *(i.e. toute instance d'un terme de R est un terme de R). Une équation $t = t'$ est dite* retardée *(par rapport à R) si $t \notin R$.*

La notion de calcul avec retard *(par rapport à R) peut être formalisée comme précédemment mais nous devons modifier la condition (2)(i) de la définition 2.1 : nous ajoutons que l'équation μe_j est* non retardée.

Remarquons que la seule condition posée sur R est d'être fermé par substitution. Cette définition très générale de la notion de retard permet d'appliquer directement nos résultats à MU-Prolog, NU-Prolog, SEPIA, Gödel.

Exemple 2.1 *Soit $S = (e_i)_{i=1,2,3}$ le système*
(1) $f(X, Y) = f(a, b)$
(2) $g(X) = g(a)$
(3) $g(Y) = g(b)$
et $R = \{f(t, t') | t, t'$ sans variable$\} \cup \{g(t) | t$ possiblement avec variables$\}$.
Un calcul avec retard par rapport à R ne peut commencer par (1) (i.e. J_1 ne peut pas être $\{1\}$).
(1) est retardée jusqu'a ce que X et Y soient suffisament instanciés par (2) et (3).

Remarque 2.2 *Nous devons modifier dans la remarque 2.1 la seconde condition pour qu'un état (J, μ) soit terminal. Celle-ci devient :*

(ii) Pour tout $j \in I - J$ l'équation μe_j est retardée. ($I = J$ est un cas particulier).

Il est évident qu'un calcul avec retard terminé dont le dernier état (J_m, μ_m) satisfait soit $\mu_m = échec$ soit $J_m = I$ est un cas particulier de calcul (sans retard) et que le résultat (aucune solution ou *upg*) est le même pour les deux notions de calcul.

notre objectif est ici d'étudier le résultat (i.e. l'état terminal (J_m, μ_m)) dans les autres cas (i.e. lorsque $\mu_m \neq échec$ et $J_m \neq I$). μ_m est alors un *upg* de $S|_{J_m}$.

La question est la dépendance de l'ensemble terminal J_m par rapport au calcul avec retard : lorsque plusieurs équations ne sont pas retardées, quel est l'influence du choix de l'une d'entre elles sur les instanciations futures des équations qui sont encore retardées.

Par la suite, un ensemble *R fermé par substitution* est supposé donné.

Definition 2.3 *La* substitution σ libère *le terme t (pour R) si $\sigma t \in R$. Le* système d'équations S libère *le terme t si toutes les solutions de S libèrent t. Le* système d'équations S libère *l'équation $t = t'$ si S libère t.*

Remarquons qu'un système non soluble libère n'importe quel terme (n'importe quelle équation), et qu'un système qui admet une solution libère un terme t (une équation $t = t'$) ssi son *upg* libère t.

La remarque suivante est immédiate :

Remarque 2.3 *Puisque R est fermé par substitution, si t et t' sont variants l'un de l'autre (le même terme à un renommage près), alors $t \in R$ ssi $t' \in R$.*

Si les couples (S, t) et (S', t') sont variants l'un de l'autre alors S libère t ssi S' libère t'.

Ainsi la relation "S libère t" est invariante par renommage des variables.

Théorème 2.1 *Soit S, le système $(e_i)_{i\in I}$. Il existe un* plus petit *ensemble J inclus dans I tel que ($S|_J$ libère $e_j \Rightarrow j \in J$). C'est aussi le plus petit ensemble vérifiant ($S|_J$ libère $e_j \Leftrightarrow j \in J$).*

Esquisse de preuve : cet ensemble est défini inductivement par les règles $j \leftarrow J$ où $S|_J$ libère e_j. Remarquons que $j \leftarrow \emptyset$ est une règle ssi e_j est une équation $t = t'$ vérifiant $t \in R$.

Definition 2.4 *Ce plus petit ensemble J inclus dans I est appelé le* domaine décidable *du système S. Son complémentaire dans I est appelé le* domaine suspendu *de S. La restriction de S à son domaine décidable est appelée le* le sous-système décidable de S.

S est partiellement d-soluble *(par rapport à R) si son sous-système décidable admet une solution. Si de plus S coincide avec son sous-système décidable (i.e. son domaine suspendu est vide) alors S est dit* (complètement) d-soluble.

Remarque 2.4 *Si S, le système $(e_i)_{i\in I}$, n'est pas partiellement d-soluble, il coïncide avec son sous-système décidable (son domaine décidable est I). Ceci est en accord avec l'idée selon laquelle, dans un tel cas, même si certaines équations restent retardées, nous pouvons considérer que le système n'admet pas de solution.*

Théorème 2.2 *Soit $(J_0, \mu_0), \cdots, (J_m, \mu_m)$ un* calcul avec retard terminé *(définitions 2.2 et 2.1). Alors :*

(i) si $\mu_m \neq$ échec, alors J_m est le domaine décidable *du système,*

(ii) si $\mu_m =$ échec, alors tous les calculs terminés se terminent par échec.

Ce théorème **résoud la question précédente** concernant la dépendance de l'ensemble terminal J_m par rapport au calcul avec retard. Il sagit d'un résultat d'indépendance qui, avec la définition 2.6, donne à cet ensemble terminal un sens (domaine décidable) qui ne dépend pas de la notion de calcul.

3 Programmes définis, squelettes et solvabilité avec retard

Nous supposons que le lecteur est familier avec les terminologies, notations et résultats standards de la programmation logique ([18, 1]).

Un langage du premier ordre est supposé donné. Les membres des équations que nous allons considérer sont des *atomes* (formules atomiques). Soit $ATOM$ l'ensemble des atomes. (En fait la différence entre termes et atomes, i.e. entre symboles de fonction et de prédicat, n'est significative que pour la sémantique logique. Elle n'est pas significative pour la compréhension de base de la sémantique opérationnelle des langages de type PROLOG. Cette vision de la programmation logique est aussi celle de [17] dans le paradigme des *propositions en tant que types*.)

Par la suite un *programme défini* et un *but* sont supposés fixés.

Il est pratique de formaliser un *programme défini* par une famille finie $\{Cr\}_{r\in N}$ où chaque C_r est une *clause définie* $a_0 \leftarrow a_1, ..., a_n$ ($a_i \in ATOM$). Le symbole r est le *nom* de la clause C_r et n est l'*arité* de r.

De même le but est formalisé par une clause $\leftarrow a_1, \cdots, a_n$ ($a_i \in ATOM$) et un symbole g ($g \notin N$) qui est le nom du but, n est l'arité de g.

Nous considérons un autre symbole noté '?', d'arité 0.

Definition 3.1 *Un* squelette *(partiel) est un arbre orienté fini, étiqueté par $N \bigcup \{g, ?\}$ vérifiant :*

- *le degré d'un noeud (nombre de fils) est l'arité de son étiquette,*

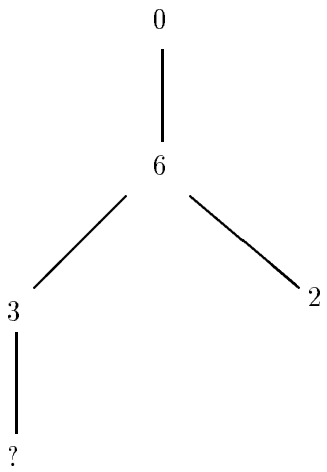

Figure 1 : Squelette pour l'exemple 3.1

- *la racine est l'unique occurence de g.*

Un noeud d'un squelette est dit indéterminé *si son étiquette est '?'. Les autres noeuds sont dits* définis. *Un noeud est* propre *s'il est défini mais n'est pas la racine.*

L'ensemble des noeuds indéterminés (resp. définis, propres) d'un squelette s est noté $indet(s)$ *(resp.* $def(s)$, $prop(s)$*).*

Un squelette s est dit total *si* $indet(s) = \emptyset$.

Exemple 3.1 *Considérons le programme et le but suivants (nous prenons les entiers comme noms de clauses,* $g = 0$*).*
$0 :\leftarrow r(X, c)$
$1 : p(a, b) \leftarrow$
$2 : p(b, c) \leftarrow$
$3 : p(X, Y) \leftarrow q(X)$
$4 : q(f(X)) \leftarrow q(X)$
$5 : q(d) \leftarrow$
$6 : r(X, Y) \leftarrow p(X, Z), p(Z, Y)$

La Figure 1 montre un squelette.

Remarque 3.1 *(i) Un squelette peut aussi être défini comme un* terme *sur une signature dont l'ensemble de symboles est* $N \cup \{g, ?\}$.

(ii) Les noeuds indéterminés sont des feuilles (l'arité de '?' est 0) et les autres feuilles sont des occurences de noms de faits (clause avec un corps vide).

(iii) Un squelette est complètement défini par ses noeuds propres.

Definition 3.2 *Un* renommage *pour un squelette s est une famille* $(C'_u)_{u \in def(s)}$ *où :*

(i) si $u \in prop(s)$ *et u est étiqueté par r,* C'_u *est une* variante *de la clause* C_r,

(ii) si u est la racine, C'_u *est exactement la clause but* $\leftarrow a_1, \cdots, a_n$,

(iii) si $u \neq v$, C'_u *et* C'_v *n'ont aucune variable en commun.*

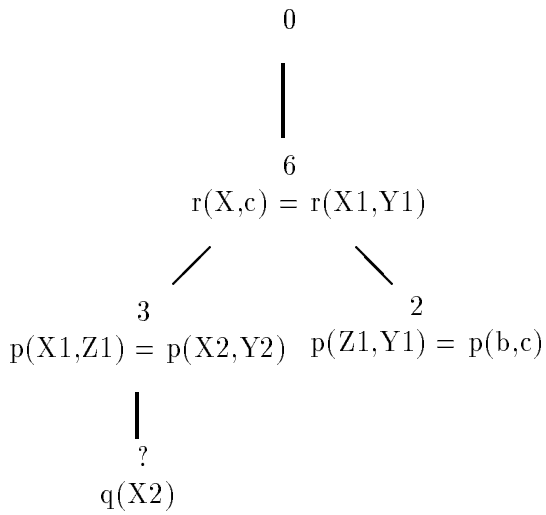

Figure 2 : système et atome de l'exemple 3.2

Definition 3.3 *Soit s un squelette équipé d'un renommage $(C'_u)_{u \in def(s)}$.*

Pour chaque noeud non racine u, nous définissons un atome a_u *et, si de plus $u \in prop(s)$, nous définissons aussi une* équation e_u *de la façon suivante :*

Soit v le noeud père de u, u est le i^{me} fils de v ($1 \leq i \leq$ arité de l'étiquette de v).

Soit a le $i^{ème}$ membre du corps de la clause C'_v.

L'atome a_u est a.

De plus, si $u \in prop(s)$ (alors C'_u est défini) l'équation e_u est $a =$ tête de C'_u.

$(e_u)_{u \in prop(s)}$ est appelé le système d'équations *associé à s et le renommage.*

*Soit σ une solution de ce système. Si nous remplaçons dans un squelette l'étiquette de chaque noeud excepté la racine par l'atome σa_u, nous obtenons un nouvel arbre appelé l'*arbre de preuve partiel *défini sur s par le renommage et la solution σ.*

Notons que si $u \in prop(s)$ l'étiquette de u dans l'arbre de preuve partiel est l'atome $\sigma a_u = \sigma($tête de $C'_u)$.

L'arbre de preuve partiel est appelé arbre de preuve (total) *si s est total (i.e. $indet(s) = \emptyset$).*

Exemple 3.2 *(suite de l'exemple 3.1). La Figure 2 montre les atomes et le système associés au squelette de la Figure 1 (et à un renommage évident). Notons que les* indices *des équations sont implicites (ce sont les noeuds du squelette). La Figure 3 montre un arbre de preuve partiel sur ce squelette.*

On voit aisément que les arbres de preuve sont exactement les arbres vérifiant la condition suivante :

Soient u un noeud et $u_1, \cdots, u_n$ les fils de u. Soient $a_1, \cdots, a_n$ les étiquettes de $u_1, \cdots, u_n$. Si u n'est pas la racine alors son étiquette est un atome a et $a \leftarrow a_1, \cdots, a_n$ est une instance d'une clause du programme. Si u est la racine alors $\leftarrow a_1, \cdots, a_n$ est une instance du but (l'étiquette de la racine reste le symbole g).

Il sagit d'une notion classique d'arbre de preuve en programmation logique ([7, 28, 13, 14]).

Remarquons que pour chaque arbre de preuve a il existe au moins un squelette (total) s tel que a soit un arbre de preuve sur s. Cependant, pour un squelette donné, il n'existe un arbre de preuve que si le système associé admet une solution.

Definition 3.4 *Dans les définitions précédentes d'un arbre de preuve, si la solution σ est un* upg *du système associé alors l'arbre de preuve est dit* arbre de preuve le plus général *sur le squelette s.*

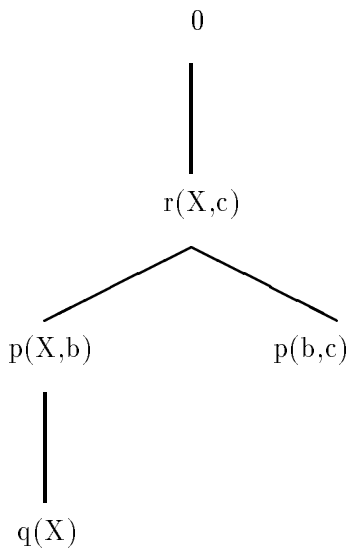

Figure 3 : arbre de preuve partiel pour l'exemple 3.2

Exemple 3.3 *(Suite). L'arbre de preuve de la Figure 3 est un arbre de preuve le plus général sur le squelette précédent (Figure 1).*

Il est facile de vérifier que tout arbre de preuve sur s est une instance d'un arbre de preuve le plus général sur s et que, sur un même squelette, deux arbres de preuve les plus généraux sont variants l'un de l'autre.

Soit $a_1, \cdots, a_n$ les étiquettes des fils de la racine dans un arbre de preuve le plus général. $\leftarrow a_1, \cdots, a_n$ est une instance du but par l'upg σ (voir plus haut). σ est une *réponse la plus générale* pour le programme et le but (voit (ii) de la définition 3.2 d'un renommage). Afin de calculer les réponses les plus générales, un système de programmation en logique doit calculer les arbres de preuve les plus généraux. La résolution SLD peut être vue comme l'une des manières (descendante) de calculer les arbres de preuve les plus généraux. Celle-ci génère le squelette et résouds le système dans un même temps (choisir une clause = générer un noeud du squelette, unifier l'atome choisi avec la tête de clause = résoudre une équation du système). Cependant la notion d'arbre de preuve ne dépend pas d'une stratégie particulière. L'utilisation des arbres de preuve est un moyen pratique d'énoncer et d'étudier des propriétés qui ne dépendent de la stratégie. Cette vision des choses est développée dans [13, 14].

Nous sommes ici motivés par la résolution SLD avec retard i.e. certains atomes candidats à l'unification avec une tête de clause sont retardés (ne peuvent pas être choisis) car insuffisamment instanciés. Si tous les atomes sont retardés, la recherche d'arbres de preuve est stoppée avec seulement un arbre de preuve partiel *("enlisement")*. Nous allons étudier ces arbres de preuve partiels, qui sont des arbres de preuve partiels les plus généraux.

Remarque 3.2 *Etant donné un squelette s, la totalité de l'information concernant un possible arbre de preuve le plus général sur s est contenue dans s. En effet la solvabilité du système ne dépend pas du renommage et, si le système est soluble, l'arbre de preuve le plus général est unique à un renommage près.*

Nous nous servons de cette observation pour limiter le plus possible l'utilisation explicite de variables.

Nous considérons maintenant la formalisation de la notion de retard par un ensemble $R \subseteq ATOM$ (voir section précédente).

Soit s un squelette équipé d'un renommage et S le système d'équations associé. La d-solubilité (section précédente, definition 2.6) de S ne dépend pas d'un renommage particulier (un autre renommage donnerait une autre variante du système d'équations)

Definition 3.5 *Un* squelette *est dit* d-soluble *si, étant donné un renommage, son système d'équations associé est d-soluble.*

Remarquons qu'un squelette d-soluble n'est pas nécessairement total (il peut posséder des feuilles indéterminées).

Si u est une feuille indéterminée et a est l'atome associé à u, la relation : "S libère a" ne dépend pas d'un renommage particulier.

Definition 3.6 *Une feuille indéterminée u d'un squelette s est dite* suspendue *si, étant donné un renommage quelconque, le système d'équations associé à s ne libère pas l'atome associé à u.*

Nous notons $susp(s)$ l'ensemble des feuilles indéterminées suspendues de s

Definition 3.7 *Un squelette est* d-complet *s'il est d-soluble et si toutes ses feuilles indéterminées sont suspendues.*

Exemple 3.4 *(suite). Considérons R défini par l'union des ensembles : $\{q(t)|t$ sans variable$\}$, $\{p(t,t')|t,t'$ possiblement avec variables$\}$, $\{r(t,t')|t,t'$ possiblement avec variables$\}$.*

Le squelette de la Figure 1 (voir aussi Figures 2,3) est d-complet.

Nous pouvons remarquer que si nous ajoutons au système S associé au squelette d-complet s une quelconque équation $a_u = \cdots$, où a_u est l'atome associé à $u \in indet(s) = susp(s)$, alors le sous-système décidable du système augmenté est toujours S.

Remarque 3.3 *Si $R = ATOM$, un squelette est d-complet ssi il ne possède aucune feuille indéterminée et son système d'équations associé est* soluble *(sans notion de retard) i.e. ssi il est le squelette d'un arbre de preuve classique.*

Il est évident que pour tout R, un squelette sans feuilles indéterminées est un cas particulier de squelette d'un arbre de preuve classique.

Definition 3.8 *Soient s, s' deux squelettes, s est une* approximation *de s', notée $s \leq s'$, si :*

- *$prop(s) \subseteq prop(s')$,*
- *pour tout noeud* propre *u de s, l'étiquette de u dans s et dans s' est la même.*

Remarquons que $\leq$ est un *ordre partiel.*

Notation 3.1 *Les notations et le lemme suivants seront utiles.*

- *Si r est un nom de clause dont l'arité est n, nous notons $tree(r)$ l'arbre dans lequel la racine est étiquetée par r et dont les autres noeuds sont n feuilles étiquetées par '?'.*
- *Si $u \in indet(s)$, nous notons $s[u \leftarrow tree(r)]$ le résultat de la greffe de $tree(r)$ (r étant un nom de clause d'arité n) au noeud u de s (i.e. dans $s[u \leftarrow tree(r)]$ le noeud u est étiqueté par r et possède n fils étiquetés par '?', les autres noeuds restant comme dans s).*

Remarquons que s est une approximation de $s[u \leftarrow tree(r)]$.

La notion de squelette d-complet est définie en dehors de toute référence à un calcul descendant. Le lemme technique suivant est un résultat clé qui servira dans la section 4 à relier cette notion à la résolution SLD. La difficulté est de prouver qu'il est toujours possible de résoudre (avec retards) le système d'équations par une stratégie descendante. En effet, la définition d'un squelette d-soluble dit que tout calcul résoud toutes les équations mais une stratégie descendante interdit le choix de certaines équations. Le problème sera de prouver qu'une stratégie descendante ne peut pas s'enliser alors qu'il reste des équations non retardées que l'on ne peut pas choisir car situées plus bas dans l'arbre.

Lemme 3.1 *Soient s' un squelette d-complet et s une approximation d-soluble de s'.*

(i) Si $u \in (prop(s') \bigcap indet(s)) - susp(s)$ alors $s[u \leftarrow tree(\text{étiquette de } u \text{ dans } s')]$ est encore une approximation d-soluble de s'.

(ii) Si $s' \neq s$ alors $(prop(s') \bigcap indet(s)) - susp(s) \neq \emptyset$

Esquisse de preuve : Un renommage pour s' est fixé. Notons S' le système d'équations associé à s' et à ce renommage, et S le système $S'|_{prop(s)}$.

(i) $prop(s[u \leftarrow tree(\text{étiquette de } u \text{ dans } s')]) = prop(s) \bigcup \{u\}$ et S libère e_u car $u \notin susp(s)$. De plus, si μ est un *upg* de S alors μe_u admet une solution (sinon s' ne serait pas d-soluble).

(ii) Considérons la partition $\{J, K, L\}$ de $prop(s')$ définie par :

$$J = prop(s) \ , \ K = prop(s') \bigcap indet(s) \ , \ L = prop(s') - (J \bigcup K)$$

Remarquons que si $s' \neq s$ alors $K \neq \emptyset$ (sinon $prop(s) = prop(s')$ et $s' = s$).

Puisque s' est d-soluble, son domaine décidable est $prop(s')$. Aussi, par définition, $prop(s')$ est le plus petit (et le seul) sous-ensemble M de $prop(s')$ vérifiant ($S|_M$ libère $u \Longrightarrow u \in M$). Donc il existe au moins un noeud $u \in K$ tel que e_u soit libéré par $S'|_{J \bigcup L}$ ($J \bigcup L$ est strictement inclus dans $prop(s')$, par conséquent il ne vérifie pas l'implication précédente). Notons $t = t'$ l'équation e_u. $S'|_L$ ne possède aucune variable en commun avec S et t. Aussi $S'|_{J \bigcup L}$ possède un *mgu* μ et S possède un *mgu* ν tels que $\mu t = \nu t \in R$. D'où t (et donc e_u) est libéré par le système S. D'où $u \notin susp(s)$.

Remarquons que, en conséquence de ce lemme, une approximation s d'un squelette d-complet s' avec $s \neq s'$ ne peut pas être un squelette d-complet.

Une telle approximation n'est pas nécessairement d-soluble mais :

- l'approximation tree(g) de s' est trivialement d-soluble car elle ne posséde aucune équation.
- Pour toute approximation d-soluble s de s', il existe un u dans $(prop(s') \bigcap indet(s)) - susp(s)$ tel que $s_1 = s[u \leftarrow tree(\text{étiquette de } u \text{ dans } s')]$ est une approximation d-soluble de s' verifiant $s < s_1 \leq s'$.

4 Résolution SLD avec retard

Motivés par la résolution SLD, nous allons étendre à notre contexte particulier la notion de dérivation SLD.

Dans la résolution SLD classique, il existe apparamment deux choix avant d'unifier l'atome choisi et une tête de clause : le choix de l'atome et le choix de la clause. Cependant un arbre SLD classique est une formalisation implicite du choix de l'atome. Dans notre contexte ce choix devient essentiellement le choix d'une feuille indéterminée dans un squelette.

Definition 4.1 *Une* fonction de sélection *est une fonction $\mathcal{F}$ definie pour tout* squelette s d-soluble *mais* non d-complet *et telle que $\mathcal{F}(s) \in indet(s) - susp(s)$.*

Une fonction de sélection $\mathcal{F}$ étant fixée, nous définissons la notion de *dérivation* (par rapport à $\mathcal{F}$) avec retard (sur les squelettes d-solubles) :

Si s et s' sont deux squelettes d-solubles, s' *dérive* de s s'il existe une clause de nom r vérifiant $s' = s[\mathcal{F}(s) \leftarrow tree(r)]$.

Une *dérivation* pour le but g est une suite (finie ou infinie) $s_0, s_1, \cdots$ telle que $s_0 = tree(g)$ et s_{i+1} dérive de s_i pour tout i.

Une dérivation finie $(s_0, \cdots, s_n)$ est dite *depuis* s_0 *vers* s_n.

Une dérivation *terminée* est une dérivation finie qui ne peut pas être étendue. Le squelette terminal d'une telle dérivation est :

soit un squelette d-complet (total ou partiel),

soit un squelette d-soluble s qui possède au moins une feuille indéterminée et non suspendue, $\mathcal{F}(s)$, et pour laquelle, pour chaque nom de clause r, le squelette $s[\mathcal{F}(s) \leftarrow tree(r)]$ n'est pas d-soluble.

Une dérivation *succès* (resp. *enlisement*) est une dérivation terminée vers un squelette d-complet qui est total (resp. non total).

Une dérivation *échec* est une dérivation terminée qui n'est ni succès ni enlisement.

Remarque 4.1 *Etant donné un renommage, soit a l'atome associé à la feuille* $\mathcal{F}(s)$ *de s.*

Pour tout nom de clause r, le système associé au squelette $s[\mathcal{F}(s) \leftarrow tree(r)]$ *est le système associé à s augmenté de l'équation* $(a =$ *tête de* $(renommée)C_r)$.

Soit μ *un upg du système d'équations associé à s. La condition pour un squelette d-soluble d'être terminal dans une dérivation échec peut s'énoncer comme suit : Pour tout nom de clause r, l'équation* $(\mu a =$ *tête de* $(renommée)C_r)$ *n'admet pas de solution (nous pouvons supposer que* μ *ne modifie pas les variables de* $(renommée)C_r)$.

La dérivation SLD avec retard est décrite par cette précédente notion de dérivation. Il est facile de voir que pour toute dérivation pour g $(s_0, s_1, \cdots)$ il existe une suite de renommages qui possède les propriétés suivantes : Nous pouvons définir, avec les renommages, la suite $(t_0, t_1, \cdots)$ où chaque t_i est un arbre de preuve partiel le plus général sur le squelette s_i et, si l_i est la liste de tous les atomes qui sont étiquette d'une feuille $u \in indet(s_i)$ de t_i, alors $(l_0, l_1, \cdots)$ est la suite des *résolvantes* de la dérivation SLD. La *règle de calcul* est trivialement spécifiée par la fonction de sélection $\mathcal{F}$. De plus le choix de la *clause d'entrée* est spécifié par une suite de noms de clauses définie par $s_{i+1} = s_i[\mathcal{F}(s_i) \leftarrow tree(r_{i+1})]$.

Lemme 4.1 *L'application* $(s_0, \cdots, s_n) \mapsto s_n$, *où* $(s_0, \cdots, s_n)$ *est une dérivation dans le système de transition est* injective.

Preuve: Remarquons d'abord que si $(s_0, \cdots, s_n)$ est une dérivation alors tout s_i est une approximation de s_n. Remarquons aussi que si $(s_0, \cdots, s_n, s_{n+1})$ est aussi une dérivation alors $s_n \neq s_{n+1}$.

Considérons maintenant deux dérivations $s_0, s_1, \cdots$ et $s'_0, s'_1 \cdots$ depuis $s = s_0 = s'_0$ telles que $s_i = s'_i$ pour $i < k$ mais $s_k \neq s'_k$. Il existe un noeud de s_k et s'_k (le noeud $\mathcal{F}(s_{k-1}) = \mathcal{F}(s'_{k-1})$) étiqueté par deux noms de clauses différents dans s_k et s'_k. Ainsi s_k et s'_k ne peuvent pas être des approximations du même squelette s_n.

Notre principal résultat est que cette application est une *bijection* entre l'ensemble des *dérivations succès* du système de transition et l'ensemble des *squelettes d-complets*. Il reste à prouver :

Théorème 4.1 *Pour* toute *fonction de sélection* $\mathcal{F}$, *pour tout* squelette d-complet s, *il existe une dérivation (par rapport à* $\mathcal{F}$*) pour g vers s.*

Preuve : Le lemme 3.1 nous permet de définir la dérivation suivante où chaque s_k est un squelette qui est une approximation de s:

$s_0 = tree(g)$,

si $s_k \neq s$ alors $s_{k+1} = s_k[\mathcal{F}(s_k) \leftarrow tree(\text{étiquette de } \mathcal{F}(s_k) \text{ dans } s)]$

Il existe un k pour lequel $s_k = s$. Ainsi, nous avons défini une dérivation depuis s_0 vers s.

Ce résultat peut être transposé en termes d'arbres SLD avec retard. Les étiquettes de ces arbres sont des squelettes d-solubles. Cela est détaillé dans la version longue de ce papier ([4]).

Un arbre SLD avec retard est en fait un cas particulier de la notion générale d'arbre SLD avec branches incomplètes utilisé dans [19].

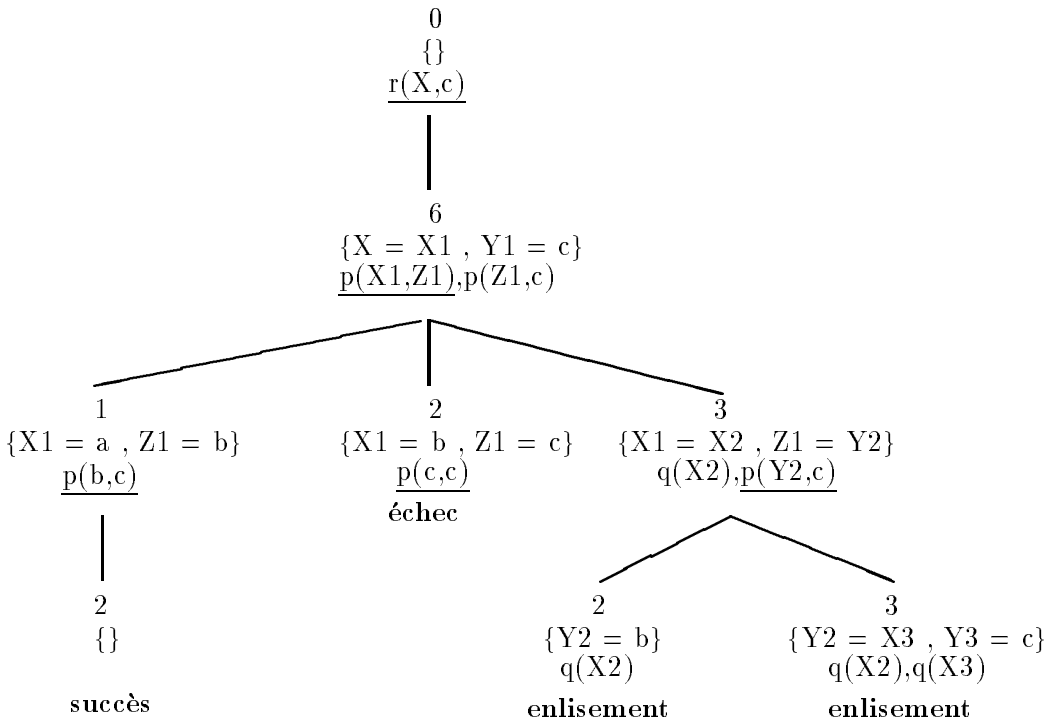

Figure 4 : représentation SLD d'un arbre de recherche pour l'exemple 4.1

Les dérivations terminées ou infinies peuvent être identifiées avec les branches de l'arbre. Il existe donc quatre type de branches : *infinies*, *succès* lorsque le squelette terminal est d-complet et total, *enlisement* lorsque le squelette terminal est d-complet et non total,et *échec*.

En pratique, nous utilisons une représentation similaire à la représentation habituelle d'un arbre SLD et nous représentons la fonction de sélection en *soulignant* l'atome μa (notation de la remarque 4.1).

IL existe trois sortes de feuilles : *succès* lorsque la liste des atomes est vide, *échec* lorsqu'un atome est choisi, *enlisement* lorsque la liste des atomes est non vide mais qu'aucun atome n'est choisi parce qu'ils sont tous retardés.

Exemple 4.1 *(suite). La Figure 4 montre un exemple d'arbre SLD avec retard.*

Le corollaire suivant est la transcription du théorème 4.1 en termes d'arbres SLD avec retard.

Corollaire 4.1 *Soit* $\mathcal{S}$ *l'*arbre de recherche *défini par la* fonction de sélection $\mathcal{F}$. *Soit* $\phi_{\mathcal{F}}$ *l'application* $l \mapsto s$, *où* l *est une* feuille non échec *(i.e. succès ou enlisement) de* $\mathcal{S}$ *et* s *est le* squelette d-complet *qui est l'étiquette de* l *dans* $\mathcal{S}$. $\phi_{\mathcal{F}}$ *est* bijective.

Soient $\mathcal{S}_1$ *et* $\mathcal{S}_2$ *deux* arbres de recherche *définis par deux* fonctions de sélection $\mathcal{F}_1$ *et* $\mathcal{F}_2$. *L'application* $(\phi_{\mathcal{F}_2})^{-1} \circ \phi_{\mathcal{F}_1}$ *est* bijective *entre les feuilles non échec de* $\mathcal{S}_1$ *et les feuilles non échec de* $\mathcal{S}_2$. *Une feuille non échec de* $\mathcal{S}_1$ *et une feuille non échec de* $\mathcal{S}_2$ *qui se correspondent par cette application possèdent les mêmes étiquettes (*squelettes d-complets*) dans* $\mathcal{S}_1$ *et dans* $\mathcal{S}_2$.

Les *squelettes d-complets* sont définis en dehors de toute stratégie. Aussi notre résultat est un résultat de correction et de complétude par rapport aux squelettes d-complets. Plus précisément, chaque arbre de recherche est correct et complet.

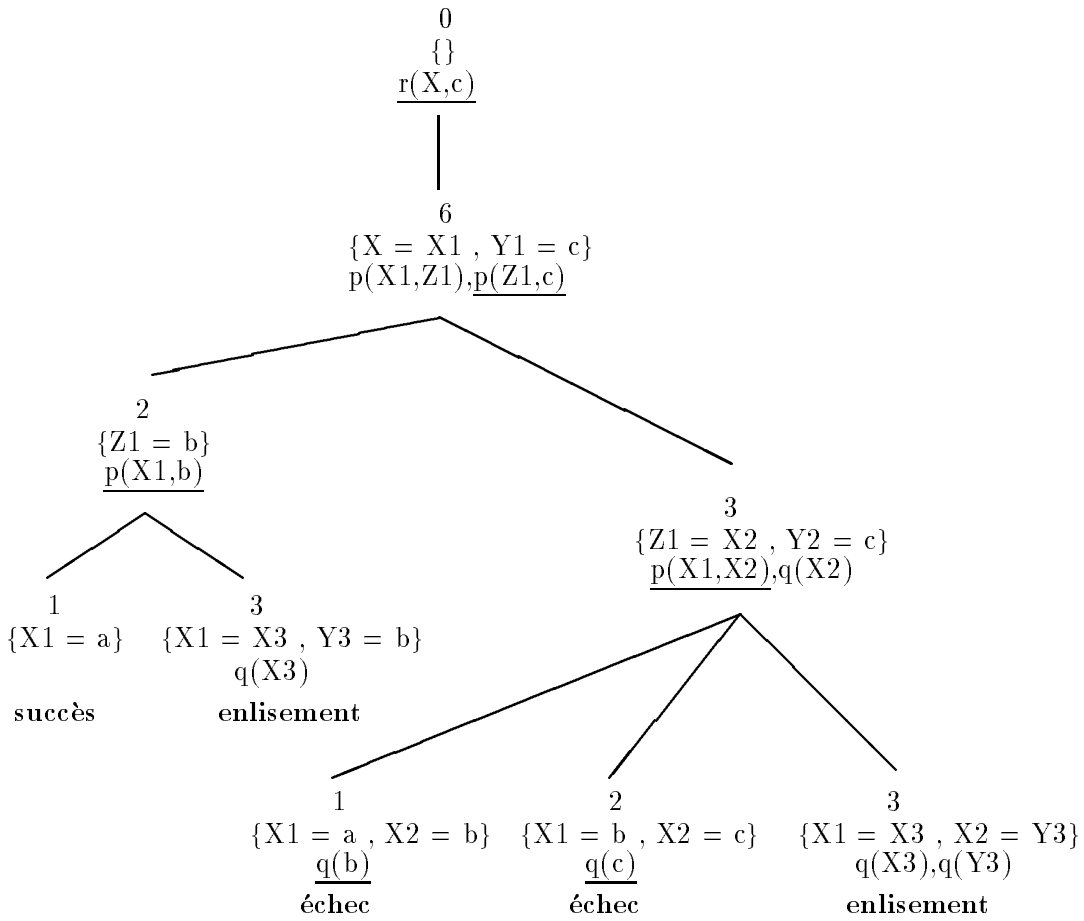

Figure 5 : représentation SLD d'un arbre de recherche pour l'exemple 4.2

Nous pouvons dire que deux arbres SLD avec retard possèdent, en un certain sens, (à un renommage près) les mêmes *feuilles succès*, (et donc les mêmes réponses calculées), *et* les *mêmes feuilles enlisement*. Cette indépendance provient de la correspondance entre ces feuilles et les squelettes d-complets.

Sans notion de retard (i.e. $R = ATOM$), ce résultat donne une formulation précise du résultat bien connu : *Indépendance de la Règle de Calcul* (théorème 9.2 dans [18]) Il sagit aussi d'une formulation de la proposition 2.5 dans [13]. Nous avons étendu ces résultats à la notion de retard.

En conséquence, si un arbre SLD avec retard n'a aucune feuille enlisement alors aucun arbre SLD avec retard n'a de feuille enlisement. En d'autre termes, l'enlisement ne dépend pas de la règle de calcul.

Exemple 4.2 *(suite). La Figure 5 montre un autre arbre SLD avec retard (i.e. une autre fonction de sélection). Nous voyons la même feuille enlisement, $q(X)$, sur cet arbre et sur celui de la Figure 4. L' (unique) squelette correspondant est celui de l'exemple 3.1 (Figures 1,2,3).*

Nous retrouvons aussi la même feuille enlisement $q(X), q(Y)$, et la même feuille succès.

5 Conclusion

Nous avons étendu, pour la notion de retard, le théorème d'indépendance de la règle de calcul de la résolution SLD et nous avons donné une explication de cette indépendance en terme

d'*arbres de preuve*. Il sagit d'une extension des idées développées dans [13].

La dérivation SLD est une notion opérationnelle. Au contraire, la notion d'arbre de preuve ne dépend pas d'un comportement opérationnel particulier. Celle-ci spécifie ce qui est calculé par une dérivation SLD (une dérivation SLD est seulement un moyen de calculer un arbre de preuve). Un squelette détermine la combinaison des clauses utilisées par une dérivation SLD. De plus les *squelettes d'*arbres de preuve partiels qui sont associés aux enlisements sont intrinsèquement définis par le programme et le but.

Lorsqu'un enlisement survient, une substitution σ est calculée mais un but B (insuffisamment instancié) reste non résolu. Nous pouvons voir σ comme une réponse contrainte par B (ou partiellement évaluée au sens de [19]). Nous avons montré que l'ensemble de ces réponses contraintes est intrinsèquement défini par le programme et le but. Elles possèdent un sens déclaratif évident : $B \Rightarrow \sigma(but)$ est une conséquence logique du programme.

Il est aisé de voir comment notre formalisation de la notion de retard peut être appliquée dans le but d'étudier la règle de calcul flexible de certains langages de programmation en logique, tels MU-Prolog, NU-Prolog, Prolog-II, SEPIA, Gödel. Par exemple Gödel considère l'enlisement comme une erreur de programmation et stoppe le calcul quand un enlisement survient, avec envoi d'un message d'erreur. notre résultat montre que l'apparition d'un tel enlisement n'influence pas les feuilles non échec (et en particulier les feuilles succès) qui restent à calculer. Aussi il pourrait être intéressant d'afficher la réponse contrainte et de revenir en arrière.

De plus notre caractérisation de l'enlisement indépendante de la règle de calcul peut s'avérer utile pour étudier des méthodes de validation : méthodes pour prouver l'absence d'enlisement (à comparer à l'*analyse de suspension* par *interprétation abstraite* pour les *programmes logiques concurrents* [8, 9, 10]). D'autre part cette caractérisation peut être utile pour une sorte de *diagnostic déclaratif* dans le but de trouver la cause de l'apparition d'un enlisement.

Une implantation correcte de la résolution SLDNF (incluant la *condition de sécurité*, voir [18]) nécessite de retarder certains appels négatifs et donne aussi lieu à une notion d'enlisement. Des travaux ont été menés afin d'analyser cet enlisement ([3, 23]). Il est possible d'appliquer notre résultat d'indépendance à un contexte plus général comportant les deux formes d'enlisement pour la résolution SLDNF (travail à venir). Pour cela une définition plus précise de la notion d'arbre SLDNF (telle que dans [2, 24]) que la définition classique ([18]) est nécessaire.

remerciements : nous remercions les référés anonymes d'une précédente version de ce papier pour leurs commentaires et suggestions.

Références

[1] K. Apt. Introduction to Logic Programming, in Handbook of Theoretical Computer Science, volume B, chapter 10, pages 491-574 (J. van Leeuwen ed.). *Elsevier*, 1990.

[2] k. Apt, K. Doets. A new definition of SLDNF-Resolution. *Report CS-R9242, CWI*, 1992.

[3] R. Barbuti, M. Martelli. Recognizing non-floundering logic programs and goals. International Journal on Foundations of Computer Science, 1(2) pages 151-163, 1990.

[4] M. Bergère, G. Ferrand, J. H. Réty. Independence of the Computation Rule for SLD-Resolution with Delay. *Research Report LIFO 93-9*, Orléans, 1993.

[5] S. Bonnier. A formal basis for Horn clause logic with external polymorphic functions. *PhD Thesis 276*, Linköping University, 1992.

[6] J. Boye. S-SLD-Resolution, an operational semantics for logic programs with external procedures. In *Proceedings of the 3rd international symposium on Programming language implementation and logic programming*, pp 383-393, Passau, LNCS 528, Springer-Verlag, 1991.

[7] K.L. CLark. Predicate Logic as a Computational Formalism. *Research Report DOC 79/59, Imperial College, London*, 1979.

[8] M. Codish, M. Falaschi, K. Marriott. Suspension analysis for concurrent logic programs, in K. Furukawa ed., Proc. 8th ICLP, pages 331-345, Paris 1991, *MIT Press 1991*

[9] M. Codish, M. Falaschi, K. Marriott. Suspension analyses for concurrent logic programs. RR ESPRIT project ParForce, 1993.

[10] C. Codognet, P. Codognet, M. Corsini. Abstract interpretation for concurrent logic languages, in Debray and Hermenegildo ed., Proc. NACLP, pages 215-232, Austin 1990. *MIT Press 1990*

[11] A. Colmerauer. Theoretical model of PROLOG II, in Logic Programming and its applications (M. Van Caneghem, D. Warren ed.) *Ablex Publishing Corp.*, pages 3-31, 1986.

[12] P. Deransart, G. Ferrand, M. Teguia. NSTO Programs (Not Subject To Occur-check). *ILPS'91*, pages 533-547, (V. J. Saraswat, K. Ueda ed.) *MIT Press*, 1991.

[13] P. Deransart, J. Maluszinski. A Grammatical View of Logic Programming. *MIT Press, 1993.*

[14] G. Ferrand. Basic Concepts of Logic Programming and PROLOG. (ICLP'91 tutorial) *RR INRIA*, 1991.

[15] R. Hill. LUSH-Resolution and its completeness. *DCL Memo 78, Dpt of Artificial Intelligence, Univ. of Edinburgh*, 1974.

[16] P.M. Hill, J. W. Lloyd. The Gödel Programming Language. CSTR-92-27, *Dept of C.S. University of Bristol*, 1992.

[17] G. Huet. A uniform approach to type theory, in Logical Foundations of Functional Programming, chapter 16, pages 337-397, (G. Huet ed.). *Addison-Wesley* , 1990.

[18] J. W. Lloyd. Foundations of Logic Programming. *Springer Verlag, 2nd ed.*, 1987.

[19] J.W. Lloyd, J.C. Shepherdson. Partial evaluation in Logic Programming. *J. Logic Programming*, 1991:11, pages 217-242.

[20] J.L. Lassez, M.J. Maher, K. G. Marriot. Unification Revisited, in Foundations of Deductive Databases and Logic Programming (J. Minker, Ed.). *Morgan Kaufmann Publishers*, pages 587–625, 1988.

[21] M.J. Maher. A Logic Programming view of CLP. *ICLP'93* , pages 737-753, *Warren ed.*, 1993.

[22] J. Maluszynski, S. Bonnier, J. Boye, F. Kluzniak, A. Kagedal and U. Nilsson. Logic Programs with External Procedures. In Apt, de Bakker, Rutten (eds.) *Logic Programming Languages : Constraints, functions and objects.* MIT Press, 1993.

[23] K. Marriott, H. Sondergaard, P. Dart. A characterization of non-floundering logic programs, in Debray and Hermenegildo ed., Proc. NACLP, pages 661-680, Austin 1990, *MIT Press 1990.*

[24] M. Martelli, C. Tricomi. A new SLDNF-tree. *Information Processing Letter* 43 (1992) pages 57-62.

[25] L. Naish. Negation and Control in Prolog, *Springer LNCS 238*, 1986.

[26] J.A. Robinson. A Machine Oriented Logic based on the Resolution Principle. *JACM 12(1)*, pages 23-41, 1965.

[27] E. Y. Shapiro. The family of Concurrent Logic Programming Languages, *ACM computing surveys*, Vol 21 (3), 1989.

[28] L. Sterling, E.Y. Shapiro. The Art of PROLOG, v*MIT Press*, 1986.

[29] K. Yelick, J. Zachary. Moded type systems for logic programs, in Proc. 6th POPL, pages 116-124, ACM, 1989.

[30] J. Zachary, K. Yelick. Moded type systems to support abstraction, in F. Pfenning ed., Types in Logic Programming, Chapter 8, pages 229-243. *MIT Press 1992.*