# Une caractérisation des arbres SLD en programmation logique avec contraintes

Alexandre Tessier
LIFO, Université d'Orléans, BP 6759, 45067 Orléans Cedex 2
Alexandre.Tessier@lifo.univ-orleans.fr
http://www.univ-orleans.fr/~tessier

## 1 Introduction

L'objectif de ce papier n'est pas de définir une nouvelle sémantique opérationnelle pour les programmes logiques avec contraintes mais seulement de reformuler les sémantiques classiques dans un cadre homogène et clair. Cette reformulation étend la vision grammaticale [5] aux programmes logiques avec contraintes. Elle permet de rendre compte des sémantiques classiques basées sur une interprétation ou une théorie des contraintes [9, 13, 10, 15, 7, 6, 3] ou basées sur des relations abstraites décrivant les solveurs de contraintes [10, 8, 14, 2, 3]. Mais cette comparaison dépasse la cadre de cet article.

Nous mettons en avant les squelettes qui sont intrinsèques au programme. C'est la structure idéale qui rassemblent toute l'information déclarative sur un calcul.

L'effort fourni dans cet reformulation, s'avère rapidement payant pour

- avoir des définitions simples, par exemple, les réponses à un but ou l'arbre SLD selon une règle de calcul ;
- avoir des preuves simples, par exemple, la correction/complétude de la SLD-résolution ou l'indépendance de la règle de calcul ;
- faire le lien avec les sémantiques déclaratives point fixes ou logiques ;
- apporter de nouveaux résultats, comme une caractérisation des branches non échecs des arbres SLD équitables.

## 2 Arbres

Si $R$ est une relation binaire sur $E$ (un sous-ensemble de $E^2$), on note $R^+$ sa clôture transitive, $R^*$ sa clôture réflexive transitive et $R^{-1}$ sa relation réciproque.

**Définition 2.1** *Un* arbre *est un ensemble $E$ muni d'une relation binaire $R$ sur $E$ telle qu'il existe un unique élément $r \in E$, appelé la* racine *de l'arbre, vérifiant : pour tout $e \in E$ : $(e, r) \in R^*$ ; $(r, r) \notin R$ ; pour tout $e \in E \setminus \{r\}$, il existe un unique $e' \in E$, appelé le* père *de $e$, tel que $(e, e') \in R$.*

$E$ est appelé le *domaine* de l'arbre. Les éléments de $E$ sont appelés les *nœuds* de l'arbre. La relation $R$ est appelée *relation de parenté*. Si $(e', e) \in R$ alors $e'$ est un *fils* de $e$. Un nœud $e$ est une *feuille* si pour tout $e' \in E$ : $(e', e) \notin R$. Une *branche* de l'arbre est une suite de nœuds tel que : le premier nœud est la racine ; tout nœud, excepté le premier, est un fils du nœud qui le précède ; la suite est infinie ou le dernier nœud est une feuille. Si $R$ est bien-fondée ($(E, R)$ ne contient pas de branches infinies), on appelle *profondeur* de $(E, R)$ la longueur de sa plus longue branche.

Un *arbre orienté* est un arbre $(E, R)$ muni d'un ensemble d'ordre (strict) $\{<_e\}_{e \in E}$ tel que, pour tout nœud $e$ : $<_e$ est un ordre total (strict) sur l'ensemble des fils de $e$.

Journées du GDR Programmation.
20, 21 et 22 novembre 1996.
Orléans.

**Définition 2.2** *Soit $E$ un ensemble de suites finies d'entiers (i.e. un ensemble de mots sur $\mathbb{N}$). $E$ est un* domaine d'arbre standard *si pour toute suite finie d'entier $N$, pour tout entier $m$ et $n$ : $\varepsilon \in E$ ; si $N \cdot n \in E$ alors $N \in E$ ; si $N \cdot n \in E$ et $m < n$ alors $N \cdot m \in E$.*

*Exemple 1* Soit $E$ un domaine d'arbre standard. Soit $R$ la relation binaire sur $E$ définie par : $(N', N) \in R$ si et seulement si il existe $n \in \mathbb{N}$, $N' = N \cdot n$ ; on montre facilement que $(E, R)$ est un arbre. Il est appelé l'*arbre sur le domaine d'arbre standard $E$*. Sa racine est $\varepsilon$.
Soit $\{<_N\}_{N \in E}$ l'ensemble d'ordres défini par : pour tout $N \in E$, $N \cdot m <_N N \cdot n$ si $m < n$. On constate que $(E, R, \{<_N\}_{N \in E})$ est un arbre orienté. C'est l'*arbre orienté sur le domaine d'arbre standard $E$*. ◇

Un *arbre étiqueté* est un arbre orienté sur un domaine d'arbre standard $E$ muni d'une *fonction d'étiquetage* de $E$ vers un ensemble $F$ dont les éléments sont appelés *étiquettes*. On dit que l'arbre est étiqueté par $F$ et qu'il est enraciné par $f$ si $f$ est l'étiquette de sa racine.

## 3 Programmes logiques avec contraintes

Considérons une fois pour toute quatre ensembles dénombrables qui déterminent le langage des programmes : un ensemble infini de *variables* $V$ ; un ensemble de *symbole de fonction* $\Sigma$ ; un ensemble de symbole de *prédicat de contrainte* $\Pi_c$ ; un ensemble de symbole de *prédicat de programme* $\Pi_p$. On suppose que $\Pi_p \cap \Pi_c = \emptyset$. On note $var(F)$ l'ensemble des variables ayant une occurrence libre dans l'expression $F$ construite sur $(V, \Sigma, \Pi_c \cup \Pi_p)$.

Un *atome* est une formule atomique $p(x_1, \ldots, x_n)$ construite sur $(V, \Pi_p)$, où $x_1, \ldots, x_n$ sont $n$ variables distinctes. Pour ne pas alourdir, il est noté $p(\tilde{x})$.

L'ensemble des *contraintes* $CONST$ est un sous-ensemble du langage du premier ordre construit sur $(V, \Sigma, \Pi_c)$. On suppose qu'il contient la constante logique *vrai*, les formules atomiques du langage et qu'il est clos par renommage des variables, conjonction et quantification existentielle. Pour simplifier les notations, on confond un ensemble fini de contraintes avec la conjonction de ses éléments. Si $c$ est une contrainte et $p(\tilde{x})$ un atome, on note $\exists_{-p(\tilde{x})} c$ la clôture existentielle de $c$ sauf sur les variables de $\tilde{x}$.

Une *clause* est un triplet, noté $a \leftarrow c \,\square\, A$, où $a$ est un atome, $c$ est une contrainte et $A$ est une suite finie d'atomes. $head(a \leftarrow c \,\square\, A) = a$ est appelé la tête de la clause et $store(a \leftarrow c \,\square\, A) = c$ est appelé la contrainte de la clause.

Un *programme* $P$ est une famille de clauses. L'ensemble d'index pour $P$ est noté $index(P)$. Un *nom de clause* est un élément de $index(P)$. La notion de nom de clause permet de différencier deux occurrences d'une même clause dans un programme. Nous verrons la simplicité apportée par cette notion indispensable pour la sémantique opérationnelle. La *définition* du prédicat de programme $p$ dans le programme $P$ est la famille des clauses de $P$ dont la tête a $p$ pour symbole de prédicat ; cette famille est indexée par un sous-ensemble $index(P, p)$ de $index(P)$.

Un *but* est un atome $a$, qui est noté en pratique $\leftarrow a$ (pour des raisons (pré)historiques). Nous considérons des buts atomiques plutôt que des buts généraux parce qu'un but général $\leftarrow c \,\square\, A$ définit une nouvelle relation $p$ sur ses variables libres et il est toujours possible de considérer un but atomique $\leftarrow p(\tilde{x})$, où $\tilde{x} = var(c \,\square\, A)$, en ajoutant un nouveau prédicat $p$ à $\Pi_p$ dont la définition dans le programme contient l'unique clause $p(\tilde{x}) \leftarrow c \,\square\, A$. De plus, les réponses à un but général ne sont construites qu'en regroupant les réponses à des buts atomiques. En procédant ainsi nous simplifions avantageusement les diverses définitions.

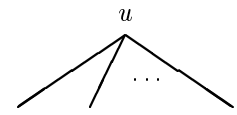
## 4 Sémantique opérationnelle

Nous supposons, pour l'ensemble de cette section, qu'un programme logique avec contraintes $P$ est fixé.

### 4.1 Squelettes

**Définition 4.1** *Un* squelette *$S$ est un arbre orienté sur un domaine d'arbre standard $E_S$, étiqueté par $index(P) \cup \Pi_p$, muni d'une fonction d'étiquetage $label_S$, tel que pour tout nœud $N \in E_S$ :*

- *si $label_S(N) \in \Pi_p$ alors $N$ est une feuille ;*




où $u$ est le nom de la clause $a \leftarrow c \,\square\, p_1(\tilde{x}_1)p_2(\tilde{x}_2)\cdots p_n(\tilde{x}_n)$

Figure 1: $sq(u)$

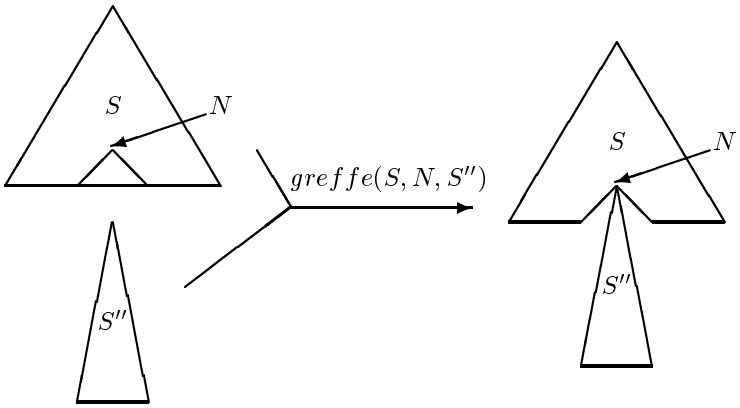



Figure 2: $greffe(S, N, S'')$

- *si* $label_S(N) \in index(P)$ *est le nom de la clause* $a \leftarrow c \,\square\, p_1(\tilde{x}_1)\cdots p_n(\tilde{x}_n)$ *alors* $N$ *a* $n$ *fils* $N_1, \ldots, N_n$ $(N_i = N \cdot (i-1))$*, et chaque fils* $N_i$ *est étiqueté par :* $p_i$ *ou un élément de* $index(P, p_i)$*.*

Soit $S$ un squelette. On note $E_S$ son domaine d'arbre standard ; $label_S$ sa fonction d'étiquetage ; $undef(S)$ l'ensemble des nœuds de $E_S$ étiquetés par des éléments de $\Pi_p$, c'est l'ensemble des *nœuds indéfinis* de $S$ ; $def(S)$ l'ensemble des autres nœuds (ils sont étiquetés par des noms de clause de $index(P)$), c'est l'ensemble des *nœuds définis* de $S$ ; $root(S)$ sa racine. Soit $N$ un nœud de $E_S$, le *symbole de prédicat associé* au nœud $N$ est $label_S(N)$ si $label_S(N) \in \Pi_p$ ou $p$ si $label_S(N) \in index(P, p)$. $S$ est un squelette pour $p$ si le symbole de prédicat associé à sa racine est $p$. $S$ est un *squelette complet* si $undef(S) = \emptyset$.

Pour tout nom de clause $u \in index(P)$, on note $sq(u)$ le squelette dont la racine est étiqueté par $u$ et a ses fils étiqueté par des éléments de $\Pi_p$ (voir Fig. 1). Pour tout symbole de prédicat $p \in \Pi_p$, on note $sq(p)$ le squelette enraciné par $p$ (son domaine est $E_{sq(p)} = \{\varepsilon\}$).

La *greffe* du squelette $S''$ sur le nœud $N$ de $S$ est le squelette $S' = greffe(S, N, S'')$, défini seulement si le symbole de prédicat associé à $N$ dans $S$ est le symbole de prédicat associé à la racine de $S''$ : (voir Fig. 2)

- $E_{S'} = \{N' \mid N' \in E_S, N$ n'est pas un préfixe de $N'\} \cup \{N \cdot N'' \mid N'' \in E_{S''}\}$ ;
- pour tout $N' \in E_{S'}$, si $N' = N \cdot N''$ alors $label_{S'}(N') = label_{S''}(N'')$ sinon $label_{S'}(N') = label_S(N')$ (en particulier, l'étiquette de $N$ est celle de la racine de $S''$).

On dit que $S''$ est le squelette *enraciné* en $N$ dans $greffe(S, N, S'')$.

Nous définissons maintenant une fonction de renommage pour un squelette qui permet de choisir pour chacun de ses nœuds une variante de clause. Cela permet d'associer à un squelette un système de contraintes, pour définir une notion de "squelette satisfiable".

**Définition 4.2** *Soit* $S$ *un squelette. Une* fonction de renommage *pour le squelette* $S$ *est une fonction* $choice_S$ *(de* $def(S)$ *vers l'ensemble des clauses renommées de* $P$*) telle que :*

1. *pour tout nœud* $N \in def(S)$*, il existe un renommage* $\theta$*, tel que* $choice_S(N) = (a \leftarrow c \,\square\, A)\theta$*, où* $a \leftarrow c \,\square\, A$ *est la clause de nom* $label_S(N)$ *;*

2. *pour tout nœud* $N \in def(S)$, *soit* $choice_S(N) = a \leftarrow c \,\square\, a_0 \cdots a_n$, *pour tout* $i = 0, \ldots, n$ : *si* $N \cdot i \in def(S)$ *alors* $head(choice_S(N \cdot i)) = a_i$ ;

3. *pour toute paire* $(N_1, N_2)$ *de nœuds distincts de* $def(S)$, *si* $choice_S(N_1) = a_1 \leftarrow c_1 \,\square\, A_1$ *et* $choice_S(N_2) = a_2 \leftarrow c_2 \,\square\, A_2$ *alors* $(var(c_1 \,\square\, A_1) \setminus var(a_1)) \cap (var(c_2 \,\square\, A_2) \setminus var(a_2)) = \emptyset$.

Étant donné un squelette $S$ de profondeur strictement positive, et une fonction de renommage $choice_S$ pour $S$, pour tout nœud $N$ de $E_S$, on appelle l'atome associé à $N$ : $head(choice_S(N))$ si $N \in def(S)$ ; $a_i$ si $N \in undef(S)$, $N = N' \cdot i$ et $choice_S(N') = a \leftarrow C \,\square\, a_0 \cdots a_i \cdots a_n$.

On associe à tout squelette $S$ et toute fonction de renommage $choice_S$ pour $S$ le système de contraintes : $const(S, choice_S) = \{store(choice_S(N)) \mid N \in def(S)\}$. Notons que si $S$ est fini alors $const(S, choice_S)$ est une contrainte et si la racine de $S$ est indéfinie alors $const(S, choice_S) = \emptyset = vrai$.

Soit $S$ un squelette. $choice_S$ est une fonction de renommage pour $S$ et $\leftarrow p(\tilde{x})$ si $S$ est $sq(p)$ ou si $head(choice_S(root(S))) = p(\tilde{x})$. Notons que $choice_S$ est une fonction de renommage pour $S$.

Si $S$ est un squelette fini et $choice_S$ une fonction de renommage pour $S$ et $\leftarrow a$, on note $AC(S, a)$ la contrainte $AC(S, a) = \exists_{-a} const(S, choice_S)$.

### 4.2 Critère de rejet

Le *critère de rejet* est une relation sur $CONST$. Il exprime le fait que certaines contraintes sont jugées sans intérêt pour l'utilisateur (parce qu'elles sont, par exemple, toujours fausses).

Les propriétés naturelles supposées pour un critère de rejet $RC$ sont, pour toute contrainte $c \in RC$ : $c\theta \in RC$, pour tout renommage $\theta$ ; $c \wedge c' \in RC$, pour toute contrainte $c'$ ; et $\emptyset \notin RC$.

Le critère de rejet n'est, en général, pas quelconque ; il est le plus souvent défini à partir d'une relation déjà existante (qui doit satisfaire les propriétés), par exemple, l'interprétation ou la théorie sous-jacente des contraintes, ou encore le solveur de contraintes (incomplet) du système.

Du critère de rejet $RC$ on déduit une relation $RC'$ sur l'ensemble des squelettes de la manière suivante : $S \in RC'$ si il existe un sous-ensemble fini $c$ de $const(S, choice_S)$, où $choice_S$ est une fonction de renommage pour $S$, tel que $c \in RC$. Cette propriété ne dépend pas de $choice_S$ : $RC$ est fermé par renommage et si $choice_S$ et $choice'_S$ sont deux fonctions de renommage pour $S$ alors il existe un renommage $\theta$ tel que $const(S, choice_S)\theta = const(S, choice'_S)$. Notons que tout $sq(p)$, $p \in \Pi_p$, n'est pas rejeté.

On suppose fixé, jusqu'à la fin de cet article, un critère de rejet $RC$. La sémantique opérationnelle est paramétrée par ce critère de rejet.

On appelle *état de calcul* (selon $RC$), ou plus simplement état, un squelette qui n'appartient pas à $RC'$.

En fait, les définitions de fonction de renommage pour un squelette et critère de rejet pour un système de contraintes ne sont qu'une étape technique intermédiaire pour définir les états de calcul. Dans la suite, seule la notion d'état compte pour définir la SLD-résolution et les réponses à un but.

**Lemme 4.3** *Si* $greffe(S, N, S')$, *où* $N \in undef(S)$, *est un état alors* $S$ *est un état.*

*Preuve.* Soit $choice_{S''}$ une fonction de renommage pour $S'' = greffe(S, N, S')$. La restriction de $choice_{S''}$ à $def(S)$, notée $choice_S$ est une fonction de renommage pour $S$ : tout nœud $N$ de $def(S)$ est étiqueté par le même non de clause dans $S$ et dans $greffe(S, N, S')$. Donc $const(S, choice_S) \subseteq const(S'', choice_{S''})$ et $\mathcal{P}_f(const(S, choice_S)) \subseteq \mathcal{P}_f(const(S'', choice_{S''}))$, où $\mathcal{P}_f(E)$ est l'ensemble des parties finies de $E$. S'il n'existe aucune partie finie de $const(S'', choice_{S''})$ rejetée alors il n'existe aucune partie finie de $const(S, choice_S)$ rejetée. D'où, si $greffe(S, N, S')$ est un état, $N \in undef(S)$, alors $S$ est un état. ∎

### 4.3 Réponses

**Définition 4.4** *Une* réponse *au but* $\leftarrow a$ *est un état fini complet* $S$ *pour* $\leftarrow a$. $AC(S, a)$ *est alors une* contrainte réponse *pour le but* $\leftarrow a$.

**Lemme 4.5** *Soit* $S$ *est une réponse. Pour tout nœud* $N$ *de* $E_S$, *le squelette enraciné en* $N$ *dans* $S$ *est une réponse.*

*Preuve.* Soit $S$ une réponse. Soit $S'$ un squelette enraciné en un nœud $N$ de $E_S$. Il est évident que $S'$ est fini et complet. Si $choice_S$ est une fonction de renommage pour $S$ alors $choice_{S'}$, définie par : pour tout nœud $N'$ de $S'$, $choice_{S'}(N') = choice_S(N \cdot N')$, est une fonction de renommage pour $S'$. Comme $const(S', choice_{S'}) \subseteq const(S, choice_S)$, $S'$ n'est pas rejeté. C'est donc une réponse. ■

On comprend mieux maintenant le rôle primordial joué par les réponses au buts "atomiques". Une réponse à un but général peut se définir comme une paire composée de la contrainte du but général et d'une forêt de réponses aux atomes présents dans le but général. La notion de paire (contrainte, forêt de squelettes) rejetée se définit facilement en définissant la notion de fonction de renommage pour une telle paire, puis la notion de système de contraintes associé à cette paire.

Les réponses aux buts atomiques caractérisent complètement la sémantique opérationnelle du programme.

### 4.4 Résolution

La construction descendante des réponses à un but $\leftarrow p(\tilde{x})$ consiste à construire des états obtenus progressivement en greffant des squelettes aux feuilles indéfinies, jusqu'à obtenir des états complets. On se limitera bien entendu à la construction d'états pour $p$. Supposons que nous ayons déjà construit un état $S$ pour $p$, deux cas se présentent : $S$ est complet, alors $S$ est une réponse ; $S$ est incomplet, alors $S$ a au moins une feuille indéfinie. On progresse vers une réponse en greffant sur cette feuille indéfinie un autre squelette pour obtenir un nouvel état. Le plus petit squelette que l'on peut greffer à une feuille indéfini $N$ d'un état $S$ pour progresser vers une réponse est un $sq(u)$, où $u \in index(P, label_S(N))$, tel que $greffe(S, N, sq(u))$ est un état. Et le plus petit squelette qui permet de démarrer cette construction est $sq(p)$.

**Définition 4.6** *Soit $\hookrightarrow$ la relation binaire sur l'ensemble des états, appelée* relation de transition *entre états de calcul, définie par : $(S, S') \in \hookrightarrow$ (dans la suite nous notons $S \hookrightarrow S'$) si il existe une feuille $N \in undef(S)$ et un nom de clause $u \in index(P, label(N))$ tel que $S' = greffe(S, N, sq(u))$. On dit que $greffe(S, N, sq(u))$* dérive *de $S$ (par la feuille $N$).*

$\hookrightarrow$ définit un système de transition entre deux états de calcul.

**Définition 4.7** *La* SLD-résolution *[11, 1] est une résolution descendante qui, étant donné un symbole de prédicat $p$, construit une suite d'états de la manière suivante :*

1. *le premier état est $sq(p)$ ;*
2. *à partir d'un état $S$ : si $S$ est complet alors $S$ est une* réponse calculée *; sinon, l'état suivant est un état $S'$ qui dérive de $S$.*

$S$ est un *état final* s'il n'existe aucun état qui en dérive, i.e. pour tout état $S'$ : $(S, S') \notin \hookrightarrow$. Alors $S$ est un *état succès* si $S$ est complet, sinon c'est un *état échec*.

On appelle *SLD-dérivation* pour $p \in \Pi_p$ (ou pour le but $\leftarrow p(\tilde{x})$) toute suite (finie ou infinie) d'états $S_0 \cdots S_i \cdots$ telle que $S_0 = sq(p)$, chaque état $S_i$, $i > 0$ dérive du précédent (i.e. $S_{i-1} \hookrightarrow S_i$) et la suite est infinie ou le dernier état est un état final. Une SLD-dérivation succès (respectivement échec) est une SLD-dérivation finie dont l'état final est un état succès (respectivement échec).

**Lemme 4.8** *La SLD-résolution a les deux propriétés suivantes :*

1. *toute réponse calculée est une réponse ;*
2. *elle calcule toutes les réponses.*

*Preuve.*

1. La SLD-résolution ne construit que des états finis, donc s'ils sont complets, ils sont des réponses.
2. Considérons une réponse $S$ pour $p$. Le parcours en largeur de $S$ fournit une suite de nœud $N_1, \ldots, N_n$. On définit la suite d'états $S_0, S_1, \ldots, S_n$, telle que :

- $S_0 = sq(p)$, où $p$ est le symbole de prédicat associé à la racine de $S$ ;
- pour tout $i = 1, \ldots, n$, $def(S_i) = \{N_1, \ldots, N_i\}$, et, pour tout $N \in def(S_i)$, $label_{S_i}(N) = label_S(N)$.

Il est facile de vérifier que $S_0, S_1, \ldots, S_n$ est une SLD-dérivation et $S_n = S$. Par conséquent, $S$ est une réponse calculée.

■

On peut distinguer deux niveaux d'indéterminisme dans la SLD-résolution :

1. le choix de la feuille indéfinie sur laquelle est greffé le squelette ;
2. le choix du squelette qui sera greffé (le choix du nom de clause qui étiquette la racine du squelette).

Le premier correspond à la manière de construire une réponse alors que le second correspond à la réponse qui est construite. Pour se rapprocher d'une résolution effective, nous allons réduire l'indéterminisme. Pour cela on fixe le choix de la feuille indéfinie ; il ne restera plus qu'à explorer toutes les solutions possibles engendrées par les noms de clauses possibles enracinant le squelette greffé sur la feuille choisie.

**Définition 4.9** *On appelle* règle de calcul *une fonction $r$ qui, pour tout état incomplet $S$, fournit une feuille de $undef(S)$.*

*Remarque.* Une règle de calcul est parfois définie comme une fonction qui a tout but (général) associe un atome du but. Cette définition est nettement moins générale que la notre, puisque le contexte ayant conduit au but courant n'est pas pris en compte. Pour deux états différents représentant un même but (voir Fig. 4), la règle de calcul peut choisir deux feuilles différentes. ◇

**Définition 4.10** *Étant donné une règle de calcul $r$, on définit la relation $\hookrightarrow_r$ incluse dans $\hookrightarrow$ comme suit : $S \hookrightarrow_r S'$ si il existe un nom de clause $u \in index(P, label_S(r(S)))$ ($label_S(r(s)) \in \Pi_p$ puisque $r(S) \in undef(S)$) tel que $S' = greffe(S, N, sq(u))$. Une SLD-dérivation selon la règle de calcul $r$ se défini comme une SLD-dérivation en remplaçant partout $\hookrightarrow$ par $\hookrightarrow_r$.*

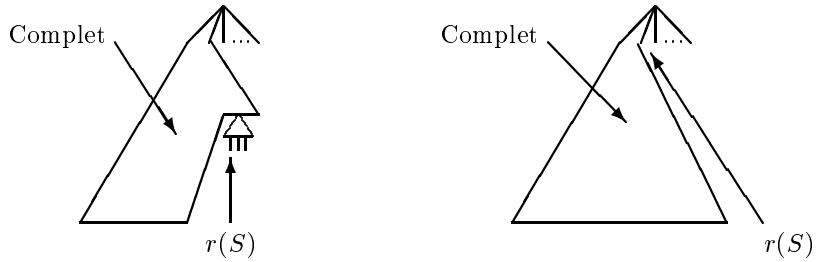
**Lemme 4.11** *Les réponses calculées sont indépendantes de la règle de calcul.*

*Preuve.* Soit $r$ une règle de calcul. On montre que $S$ est une réponse calculée selon $r$ si et seulement si $S$ est une réponse. $\Leftarrow$ Soit $S$ une réponse. Considérons la SLD-dérivations selon $r$ définie par : le premier état est $S_0 = sq(p)$, où $p$ est le symbole de prédicat associé à la racine de $S$ ; si $S_i$ est un état de la SLD-dérivation alors l'état suivant est $S_{i+1} = greffe(S_i, r(S_i), sq(label_S(r(S_i))))$. Cette SLD-dérivation selon $r$ est finie et son état finale est $S$. $\Rightarrow$ Le Lemme 4.8 montre que toute réponse calculée est une réponse. ■

Une SLD-dérivation $U = S_1 S_2 \cdots S_i \cdots$ est

- *équitable* si elle est finie ou si pour tout état $S_i$ de $U$, pour toute feuille $N \in undef(S_i)$, il existe un état $S_j$ de $U$ ($i \leq j$) où $N = r(S_j)$ ;
- *sans co-routinage* si pour tout état incomplet $S_i$ de $U$, $r(S_i)$ est une feuille indéfinie de l'ensemble des feuilles indéfinies les plus profondes.

Une règle de calcul $r$ est *équitable* (resp. *sans co-routinage*) si toute SLD-dérivation selon $r$ est équitable (resp. sans co-routinage). La *règle de calcul standard* est la règle $r$ qui pour tout état $S$ choisie la feuille indéfinie "la plus à gauche" de $S$ : $r(S) = N$ si pour tout $N' \in undef(S)$, $N' \neq N$, il existe un préfixe $M$ de $N$ et un préfixe $M'$ de $N'$ tel que $M' \leq M$. La règle de calcul standard est sans co-routinage (Voir Fig. 3).



La feuille indéfinie la plus à gauche est aussi la plus profonde.

Figure 3: Règle de calcul standard

### 4.5 Arbre de recherche SLD

**Définition 4.12** *Soit $r$ une règle de calcul et $p$ un symbole de prédicat de programme.*
*Soit $\hookleftarrow_r^p$ la relation incluse dans $(\hookrightarrow_r)^{-1}$ définie par $S' \hookleftarrow_r^p S$ si et seulement si $S \hookrightarrow_r S'$, où $S$ et $S'$ sont des états pour $p$. C'est la restriction de $(\hookrightarrow_r)^{-1}$ aux états pour $p$. Elle est aussi notée $\hookleftarrow_r^a$ si $a$ est un atome dont le symbole de prédicat est $p$.*

**Lemme 4.13** *Soit $r$ une règle de calcul. Soit $E_r^p$ l'ensemble d'états $E_r^p = \{S \mid S(\hookleftarrow_r^p)^* sq(p)\}$.*
*$(E_r^p, \hookleftarrow_r^p)$ est un arbre. Il est appelé l'arbre SLD pour $p$ (ou pour le but $\leftarrow p(\tilde{x})$) selon la règle de calcul $r$.*

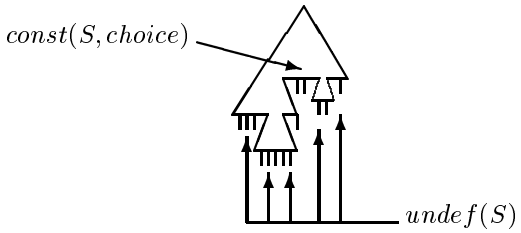
*Preuve.* Montrons que la relation $\hookleftarrow_r^p$ a les propriétés d'un arbre sur le domaine $E_r^p$ :

- $sq(p)$ est la racine de l'arbre, d'après la définition de $E_r^p$ ;
- $(sq(p), sq(p)) \notin \hookleftarrow_r^p$, d'après la définition de $\hookrightarrow_r$ ;
- pour tout $S \in E_r^p$, différent de $sq(p)$, il existe exactement un $S' \in E_r^p$, tel que $S \hookleftarrow_r^p S'$ :
  - $S'$ existe, d'après la définition de $E_r^p$ ;
  - il est unique : considérons deux SLD-dérivation selon $r$ ayant un état $S$ en commun, différent de $sq(p)$ ; elle ont au moins un autre état en commun $S''$ (par exemple $sq(p)$), la feuille choisie en $S''$ est $r(S'')$ dans les deux SLD-dérivations. Dans les deux SLD-dérivations les squelettes greffés en $r(S'')$ sont identiques, sinon le nœud $r(S'')$ aurait des étiquettes différentes dans les états qui dérivent de $S''$ dans chacune des deux SLD-dérivations et il n'y aurait aucune chance pour aboutir plus loin au même état $S$ ; en itérant le processus, on constate que les états qui précèdent $S$ dans chacune des deux SLD-dérivations sont identiques.

■

*Remarque.* Une branche de l'arbre SLD $(E_r^p, \hookleftarrow_r^p)$ est *exactement* une SLD-dérivation selon $r$ pour $p$ (voir les définitions de SLD-dérivation et de branche d'un arbre).
Notre définition des arbres SLD révèle la nature même de ces arbres. Elle montre qu'ils ne peuvent se définir qu'étant donné une règle de calcul.
Les définitions classiques n'ont jamais mis en avant la structure de ces arbres, elles ne précisent que la nature de leurs étiquettes.
Notons que les arbres SLD ne sont pas des arbres étiquetés.
Pour retrouver les arbres SLD classiques étiquetés par des buts généraux, on peut définir une fonction d'étiquetage qui associe à tout état un but construit à partir du système de contrainte associé à l'état et des atomes associés au feuilles indéfinis de l'état. Pour cela il faut définir une fonction de renommage qui s'étend d'un état à un état qui en dérive.
Un but peut étiqueter plusieurs nœuds d'un arbre SLD alors que les états, qui sont les nœuds, n'ont qu'une seule occurrence (grâce aux noms de clause). ◇

Une branche succès (respectivement échec) est une branche finie qui termine par une feuille succès (respectivement échec). L'arbre SLD ne contient que trois types de branches : les branches succès *qui sont* les




On associe au squelette $S$, ci-dessus, et à la fonction de renommage $choice_S$ pour $S$, le but général :

$$\leftarrow const(S, choice_S) \square a_1 \cdots a_n$$

où chaque $a_i$ est l'atome associé à un nœud indéfini de $S$ par $choice_S$.

Figure 4: But général associé à un squelette incomplet

SLD-dérivation succès selon $r$ ; les branches échecs *qui sont* les SLD-dérivation échecs selon $r$ ; les branches infinies *qui sont* les SLD-dérivation infinies selon $r$.

Un arbre SLD est *équitable* (respectivement *sans co-routinage*, *standard*) si ces branches sont équitables (respectivement sans co-routinage, standard), i.e. la restriction de $r$ à l'ensemble des nœuds de l'arbre est équitable (respectivement sans co-routinage, standard). En particulier, tout arbre SLD fini est équitable.

**Lemme 4.14** *L'ensemble des feuilles succès d'un arbre SLD pour $p$ est l'ensemble des réponses pour $p$.*

*Preuve.* D'après les définitions. ■

**Lemme 4.15** *Soit $S$ un squelette et $r$ une règle de calcul. Les propositions suivantes sont équivalentes :*

1. *$S$ est une réponse pour le but $\leftarrow a$.*
2. *$S$ est une réponse calculée pour le but $\leftarrow a$.*
3. *$S$ est une réponse calculée pour le but $\leftarrow a$ selon la règle de calcul $r$.*
4. *$S$ est une feuille succès de l'arbre SLD pour $\leftarrow a$ selon la règle de calcul $r$.*

*Preuve.* L'équivalence entre *1*, *2* et *3* a déjà été montrée.
*4* est équivalent à *3* car il existe une bijection entre les branches de l'arbre SLD selon $r$ pour le but $\leftarrow a$ et les SLD-dérivations selon $r$ pour le but $\leftarrow a$ (voir la preuve du Lemme 4.13). ■

**Corollaire 4.16** *Les succès d'un arbre SLD sont indépendants de la règle de calcul.*

*Remarque.* Il est possible d'orienter un arbre SLD en fixant un ordre sur les nom de clauses, i.e. un ordre total sur les $index(P, p)$. Les fils d'un nœud $S$ sont alors naturellement ordonnés par l'ordre sur les noms des clauses qui enracinent les squelettes greffés en $r(S)$. ◇

Nous avons vu qu'étant donné un but $\leftarrow a$, il existe une bijection entre les branches succès de deux arbres SLD pour $\leftarrow a$. Il existe un autre type de branches qui sont en bijection et cette bijection s'exprime facilement quand les arbres SLD sont équitables.

**Lemme 4.17** *Soit $r$ une règle de calcul équitable. Les branches infinies de $\hookleftarrow_r^p$ sont en bijection avec les états complets infinis pour $p$.*

*Preuve.* Soit $(E_r^p, \hookleftarrow_r^p)$ un arbre SLD équitable.

- Considérons une branche infinie de $\hookleftarrow_r^p$, c'est-à-dire une SLD-dérivation infinie équitable $U = S_1 S_2 \cdots S_i \cdots$ pour $p$.

  Remarquons, d'une part, que si $N \in def(S_i) \cap def(S_j)$ alors $label_{S_i}(N) = label_{S_j}(N)$ et, d'autre part, que la dérivation étant équitable, toute feuille indéfinie d'un de ses états devient définie dans un autre.

  Soit $S$ l'arbre sur le domaine d'arbre standard $E_S = \bigcup_{S_i \in U} def(S_i)$ dont la fonction d'étiquetage $label_S$ est, pour tout $N \in E_S$, $label_S(N) = label_{S_i}(N)$, où $N \in def(S_i)$.

  Pour tout nœud $N$ de $S$, soit $a \leftarrow c \,\square\, a_1 \cdots a_n$ la clause de nom $label_S(N)$, $N$ a bien $n$ fils et le symbole de prédicat associé à chacun est bien le symbole de prédicat de l'atome $a_i$ correspondant. $S$ est donc un squelette pour $p$ et de plus il est complet ($label_S$ fait correspondre à tout nœud de $E_S$ un nom de clause). Il est aussi infini car $U$ est infinie et chaque étape de dérivation défini un nœud qui était indéfini.

  Soit $choice_S$ une fonction de renommage pour $S$, comme nous l'avons vu dans la preuve du Lemme 4.3, $choice_S$ est une fonction de renommage pour chacun des $S_i \in U$. Pour toute partie finie $C$ de $const(S, choice_S)$, il existe un $S_i \in U$ tel que $C \subseteq const(S_i, choice_S)$, donc $S$ est un état. C'est un état complet infini pour $p$.

- Considérons un état $S$ infini et complet pour $p$, et montrons qu'il est possible de construire un SLD-dérivation $U$, selon $r$, pour $p$, infinie et qui "calcule" $S$, i.e. une branche infinie de $\hookleftarrow_r^p$ :
  - le premier état est $S_1 = sq(p)$ ;
  - supposons qu'on ait déjà construit la suite d'état $S_1 \cdots S_i$, préfixe de la branche $U$ de $\hookleftarrow_r^p$, $S_{i+1} = greffe(S_i, r(S_i), label_S(r(S_i)))$ est un état incomplet fils de $S_i$ dans $\hookleftarrow_r^p$.

  On définit ainsi une branche infinie $U$ de $\hookleftarrow_r^p$ qui, comme elle est équitable, calcule l'état $S$.

■

**Corollaire 4.18** *Soit $r_1$ et $r_2$ deux règles de calcul équitables. Les branches non échecs de $\hookleftarrow_{r_1}^p$ et $\hookleftarrow_{r_2}^p$ sont en bijection.*

*Preuve.* Elles sont en bijection avec les états (finis ou infinis) complets pour $p$. ■

**Corollaire 4.19** *S'il existe un arbre SLD fini pour le but $\leftarrow a$ alors tout arbre SLD équitable pour le but $\leftarrow a$ est fini.*

**Définition 4.20** *Un arbre SLD est d'*échec fini *s'il est fini et s'il n'a aucune feuille succès.*

*Remarque.* La vraie notion intéressante est celle d'arbre SLD fini. Il n'y a guère plus à dire si le nombre de feuilles succès est 0 que s'il est 1 ou 7 (même dans le cadre d'une sémantique déclarative logique). ◇

**Corollaire 4.21** *S'il existe un arbre SLD d'échec fini pour le but $\leftarrow a$ alors tout arbre SLD équitable pour le but $\leftarrow a$ est d'échec fini.*

*Preuve.* Soit $\leftarrow a$ un but ayant un arbre SLD d'échec fini. Le corollaire précédent assure que tout arbre SLD équitable pour $\leftarrow a$ est fini : le cas où l'arbre SLD est d'échec fini est un cas particulier. Le Lemme 4.14 assure que tout arbre SLD pour $\leftarrow a$ n'a pas de feuille succès. ■

### 4.6 $\vee$-réponses

On note $success(a)$ l'ensemble des réponses pour le but $\leftarrow a$. Notons que $success(a)$ rend compte de la multiplicité d'une contrainte réponse pour $\leftarrow a$.

**Définition 4.22** *La* $\vee$-réponse *pour le but $\leftarrow a$ est $success(a)$ s'il existe une règle de calcul $r$ telle que l'arbre SLD $\hookleftarrow_r^a$ est fini.*
*La* $\vee$-contrainte réponse *pour le but $\leftarrow a$ est alors le multi-ensemble $\{AC(S, a) \mid S \in success(a)\}$.*

D'un point de vue opérationnel, seuls les calculs finis présentent un intérêt. C'est la raison pour laquelle nous avons défini les $\vee$-contraintes réponses sous la condition d'existence d'un arbre SLD fini. Dans la définition de contrainte réponse nous n'avions aucune hypothèse sur l'existence d'un arbre SLD fini. Malgré tout nous considérions tout de même des calculs finis, mais il s'agit d'un autre niveau de calcul : les SLD-dérivations.

Nous avons donc défini deux niveaux de réponses :

- les états finis complets ;
- l'ensemble des feuilles succès des arbre SLD finis ;

qui correspondent à deux notions de calcul bien distincts.

La définition de $\vee$-contrainte réponse correspond exactement aux définitions de réponses données, par exemple, dans [4, 12].

## 5 Conclusion

Nous avons reformulé simplement la sémantique opérationnelle des programmes logiques avec contraintes.

Nous avons décrit un ensemble dont les éléments sont appelés états. Sur cette ensemble nous avons défini très facilement une relation appelée relation de transition entre état. Enfin, nous avons montré que cette relation a une structure d'arbre : c'est l'arbre SLD.

Notons que nous utilisons les deux notions d'arbres : les arbres SLD sont des arbres au sens général et les états sont des arbres au sens arbres étiquetés.

Nous pouvons trouver de nombreuses applications à ce cadre, comme par exemple, le diagnostic déclaratif d'erreur, la négation constructive ou tout travail concernant la sémantique opérationnelle manipulant des arbres SLD.

## References


[1] K. R. Apt and M. H. Van Emden. Contributions to the Theory of Logic Programming. *Journal of the ACM*, 29(3):841–862, 1982.

[2] Roberto Bagnara, Marco Comini, Francesca Scozzari, and Enea Zaffanella. The AND-compositionality of CLP computed answer constraints. In Paqui Lucio, Maurizio Martelli, and Marisa Navarro, editors, *Joint Conference on Declarative Programming*, pages 355–366, 1996.

[3] Philippe Codognet. Programmation logique avec contraintes : une introduction. In *Journées Francophones de Programmation en Logique*, volume 14 of *Technique et Science Informatique*, pages 665–692. Hermes, 1995.

[4] Alain Colmerauer. Specifications of prolog iv. Draft, 1996.

[5] Pierre Deransart and Jan Małuszyński. *A Grammatical View of Logic Programming*. MIT Press, 1993.

[6] François Fages. Programmation Logique avec Contraintes. École jeunes chercheurs, GDR programmation du CNRS, 1995.

[7] Maurizio Gabbrielli and Giorgio Levi. Modeling answer constraints in Constraint Logic Programs. In Vijay A. Saraswat and K. Ueda, editors, *International Conference on Logic Programming*, pages 238–252. MIT Press, 1991.

[8] Roberto Giacobazzi, Saumya K. Debray, and Giorgio Levi. Generalized Semantics and Abstract Interpretation for Constraint Logic Programs. *Journal of Logic Programming*, 25(3):191–248, 1995. (Short version available in International Conference on Fifth Generation Computer, pp 581-591, 1992).

[9] Joxan Jaffar and Jean-Louis Lassez. Constraint Logic Programming. In $14^{th}$ *ACM Symposium on Principles of Programming Languages*, pages 111–119, 1987. (Full paper available as Department of Computer Science, Monash University Technical Report 86/73).

[10] Joxan Jaffar and Michael J. Maher. Constraint Logic Programming: a survey. *Journal of Logic Programming*, 19-20:503–581, 1994.

[11] R. A. Kowalski. Predicate Logic as a Programming Language. In *Information Processing*, pages 569–574, 1974.

[12] John W. Lloyd. Declarative Programming in Escher. Technical Report CSTR-95-013, Department of Computer Science, University of Bristol, 1995.

[13] Michael J. Maher. A Logic Programming view of CLP. In Warren, editor, *International Conference on Logic Programming*, pages 737–753. MIT Press, 1993.

[14] Maarten H. Van Emden. Value Constraints in the CLP scheme. In *International Logic Programming Symposium, post-conference workshop on Interval Constraints*, 1995.

[15] Pascal Van Hentenryck. Constraint Logic Programming. *Knowledge Engineering Review*, 6(3):151–194, 1991. (Also available as Brown University Technical Report CS-91-05).