REDÉFINIR L'ENLISEMENT POUR LA RÉSOLUTION SLDNF

Jean-Hugues Réty

INRIA-Rocquencourt, B.P. 105, 78153 Le Chesnay cedex, France.
E-mail : Jean-Hugues.Rety@inria.fr

17 mars 1995

Résumé

La résolution SLDNF est la sémantique opérationnelle standard pour les programmes normaux. Afin d'en assurer la correction, une restriction (condition de sécurité) sur le choix des littéraux négatifs est nécessaire. Il s'ensuit une notion d'enlisement. L'apparition d'un enlisement est généralement considérée comme une erreur de programmation. Cependant cette notion est trop faible : un calcul (pour un programme et un but) peut s'enliser bien qu'il existe un arbre SLDNF. Dans ce cas, un autre calcul peut ne pas s'enliser. L'existence d'un enlisement dépend de la stratégie. Nous proposons une notion plus forte d'enlisement - que nous appelons *enlisement fort* - qui ne dépend que de l'existence d'un arbre SLDNF (étendu) et non pas d'une stratégie particulière. La résolution SLDNF à laquelle on intègre l'enlisement fort peut être implantée efficacement. De plus, en application d'un résultat de Drabent ([Dra94]), celle-ci est complète (et correcte) par rapport à la sémantique de Kunen pour une large classe de programmes caractérisée de manière déclarative : les programmes non fortement enlisés.

Mots clés : Programme normal, Résolution SLDNF, Enlisement.

1 Introduction

La résolution SLDNF est une extention de la résolution SLD intégrant la négation par l'échec proposée par Clark ([Cla79]). Afin d'en assurer la correction, l'application d'une restriction sur le choix des littéraux négatifs (condition de sécurité) est nécessaire : un littéral négatif ne peut être choisi que s'il est clos. Il s'ensuit une notion d'enlisement. L'apparition d'un enlisement est généralement considérée comme une erreur de programmation.

Des recherches ont été menées afin de construire des programmes qui ne s'enlisent pas. Celles-ci ont principalement pris deux directions : la définition de classes

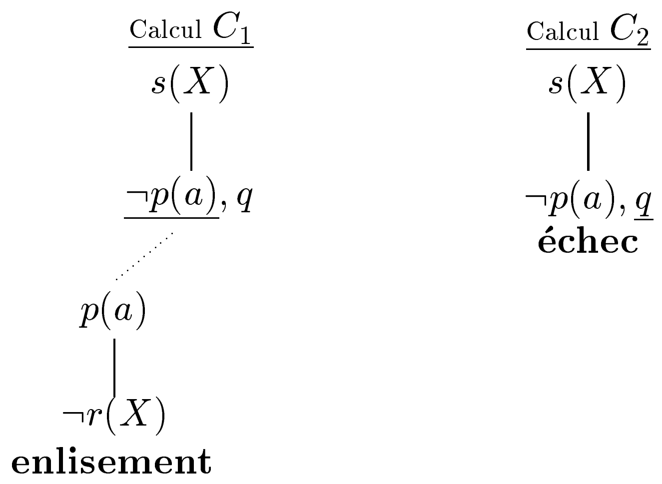

Figure 1 : Calculs pour l'exemple 1.1

(syntaxiques) de programmes qui ne s'enlisent pas et la construction d'outils d'analyse de non enlisement (e.g. [BM90], [MSD90]). Nous pensons que les problèmes rencontrés proviennent essentiellement de la notion d'enlisement elle même qui n'est pas satisfaisante.

L'enlisement est une notion opérationnelle. Un problème majeur est qu'il dépend de la règle de calcul et de la règle de choix de clause (cf exemples 1.1 et 4.1). Etant donnés un programme et un but, un calcul peut s'enliser tandis qu'un autre ne s'enlise pas. Cela rend la propriété de non enlisement difficile à établir. D'autre part, les implantations stoppent l'éxécution d'un calcul dès l'apparition d'un enlisement. L'exemple 1.1 montre que dans certains cas, il est néanmoins possible d'obtenir toutes les réponses.

Exemple 1.1 *Considérons le programme normal $P1$:*
$s(X) \leftarrow \neg p(a), q.$
$p(a) \leftarrow \neg r(X).$

et le but normal $B1$:
$\leftarrow s(X).$

La figure 1 montre deux calculs avec deux règles de calcul différentes pour $P1 \bigcup \{B1\}$. Le choix de $\neg p(a)$ (calcul C_1) dans le but $\neg p(a), q$ conduit à un enlisement dans le calcul "subsidiaire". Le choix de q (calcul C_2) conduit à un échec ; le calcul C_2 ne s'enlise pas.

Il existe un arbre SLDNF pour $P1 \bigcup \{B1\}$ mais le calcul C_1 ne le construit pas. On remarquera qu'une réponse correcte aurait pu être obtenue en revenant au noeud $\neg p(a), q$ et en choisant q. Un tel calcul étant alors correct et complet malgré l'existence d'un enlisement.

La notion d'enlisement apparaît donc trop faible : un programme et un but peuvent s'enliser bien qu'il existe un arbre SLDNF.

L'enlisement est habituellement définit de la façon suivante : Soit P un programme normal et B un but normal, l'évaluation de $P \bigcup \{B\}$ s'enlise si un but uniquement composé de littéraux négatifs non clos est atteint. Cette définition se situe au niveau du calcul. Il y a un réel manque d'une notion déclarative d'enlisement (i.e. abstraite du calcul).

D'après la définition standard de la résolution SLDNF, pour certains programmes et certains buts, il n'existe ni arbre SLDNF ni dérivation SLDNF. Il y a trois raisons à cela. Premièrement, les arbres SLDNF étant construits de façon montante avec indéxation du nombre d'appels négatifs par des entiers naturels, les suites infinies d'appels négatifs ne peuvent être prises en compte. Deuxièmement, pour un programme du type $\{p \leftarrow p\}$ et un but $\leftarrow \neg p$ (p étant clos), il n'existe ni réfutation SLDNF ni arbre SLDNF d'échecs finis pour $\leftarrow p$. La troisième raison est l'enlisement : aucune notion déclarative d'enlisement n'existe et, e.g. pour le but $\leftarrow \neg p(X)$ et un programme quelconque, il n'existe pas d'arbre SLDNF (ni de dérivation SLDNF).

Ces problèmes sont mentionnés dans [AB91] et [AP91] et de nouvelles définitions de la résolution SLDNF ont été proposées ([AD92], [MT92], [MN90]). En particulier Apt et Doets ([AD92]) donnent une définition descendante qui est reproduite dans [AB94] et [Doe94]. Aucun de ces papiers n'aborde vraiment le problème de l'enlisement, leur objectif premier étant de traiter les deux premiers points précédemment cités. Seul [AB94] propose une notion déclarative de l'enlisement. Celle ci est directement abstraite de la notion standard d'enlisement et est donc très faible : un programme et un but s'enlisent s'il existe une règle de calcul pour laquelle un calcul peut s'enliser.

Nous proposons dans ce papier une notion d'enlisement plus forte appelée *enlisement fort* qui correspond à l'idée qu'un enlisement n'est pas significatif s'il est possible de continuer le calcul (sans perte de complétude) avec une autre stratégie. L'enlisement fort ne dépend pas de la règle de calcul. Nous prouvons un résultat plus général dont une conséquence particulière est que, pour la résolution SLDNF, les succès ne dépendent pas de la règle de calcul. De plus, l'enlisement fort peut être aisement et efficacement implanté. Nous proposons un schéma d'implantation que nous appelons *calcul étendu*. Une conséquence du résultat principal de [Dra94] est qu'un calcul étendu qui termine et qui n'est pas fortement enlisé est complet par rapport à la sémantique de Kunen. Un calcul étendu peut s'enliser et néanmoins être complet (s'il n'est pas fortement enlisé). Nous pensons que l'absence d'enlisement fort est une propriété souhaitable pour les programmes normaux.

Le chapitre 2 définit l'enlisement fort. Le chapitre 3 établit l'indépendance par rapport à la règle de calcul. Le chapitre 4 traite d'implantation : nous étudions les problèmes liés à l'enlisement et nous discutons de l'implantation de l'enlisement fort.

Une autre approche du problème se trouve dans [Dra94]. Ce papier prouve la complétude de la résolution SLDNF par rapport à la sémantique de Kunen pour les programmes et les buts qui ne s'enlisent pas via une règle de calcul équitable. Drabent remarque que la notion standard d'enlisement et trop générale. Il propose une nouvelle notion d'enlisement - appelée *enlisement sérieux*[1]. Une comparaison entre enlisement fort et enlisement sérieux est faite dans le chapitre 5.
D'autres papiers traitent de négation et de contrôle. A notre connaissance aucun ne tente de redéfinir l'enlisement.

2 L'enlisement fort

Nous définissons une notion d'enlisement que nous appelons *enlisement fort*. Nous avons choisi de baser notre travail sur la définition de la résolution SLDNF de Lloyd ([Llo87]).

Nous començons par définir la notion préliminaire d'*arbre SLDNF entièrement enlisé*. Intuitivement un tel arbre n'a aucune feuille succès mais au moins une feuille enlisement.

Nous ne rappelons pas les définitions de réfutation SLDNF et d'arbre SLDNF d'échecs finis ([Llo87]).

Definition 2.1 *Soient P un programme normal et B un but normal. Un* arbre SLDNF entièrement enlisé de rang 0 *pour $P\bigcup\{B\}$ est un arbre vérifiant les conditions suivantes :*
(a) Chaque noeud de l'arbre est un but normal non vide.
(b) La racine de l'arbre est B.
(c) Seul des littéraux positifs sont choisis aux noeuds de l'arbre.
(d) Soit $\leftarrow L_1, ..., L_m, ..., L_p$ un noeud de l'arbre qui ne soit pas une feuille et supposons que L_m soit un atome et soit choisi. Alors pour chaque (variante de) clause de programme $A \leftarrow M_1, ..., M_q$ telle que L_m et A soient unifiables avec l'upg θ, le noeud a un fils $\leftarrow (L_1, ..., L_{m-1}, M_1, ..., M_q, L_{m+1}, ..., L_p)\theta$.
(e) Soit $\leftarrow L_1, ..., L_p$ une feuille de l'arbre alors soit
(i) chaque L_i est un littéral négatif non clos, soit
(ii) un atome L_m est choisi et il n'y a pas de (variante de) clause de programme de P dont la tête s'unifie avec L_m.
(f) il existe au moins une feuille de de l'arbre vérifiant (e)(i).

Definition 2.2 *Soient P un programme normal et B un but normal. Un* arbre SLDNF entièrement enlisé de rang k+1 *pour $P\bigcup\{B\}$ est un arbre vérifiant les conditions suivantes :*
(a) Chaque noeud de l'arbre est un but normal non vide.
(b) La racine de l'arbre est B.

[1] serious floundering

(c) Soit $\leftarrow L_1, ..., L_m, ..., L_p$ un noeud de l'arbre qui ne soit pas une feuille et supposons que L_m soit choisi. Alors soit

(i) L_m est un atome et pour chaque (variante de) clause de programme $A \leftarrow M_1, ..., M_q$ telle que L_m et A soient unifiables avec l'upg θ, le noeud a un fils $\leftarrow (L_1, ..., L_{m-1}, M_1, ..., M_q, L_{m+1}, ..., L_p)\theta$, soit

(ii) L_m est un littéral négatif clos $\neg A$ et il existe un arbre SLDNF d'échecs finis de rang k pour $P \bigcup \{\leftarrow A_m\}$, auquel cas le seul fils est $\leftarrow L_1, ..., L_{m-1}, L_{m+1}, ..., L_p$.

(d) Soit $\leftarrow L_1, ..., L_p$ une feuille de l'arbre alors soit

(i) chaque L_i est un littéral négatif non clos ou un littéral négatif clos $L_i = \neg A_i$ vérifiant : il existe un arbre SLDNF entièrement enlisé de rang k pour $P \bigcup \{\leftarrow A_i\}$, soit

(ii) un atome L_m est choisi et il n'y a pas de (variante de) clause de programme de P dont la tête s'unifie avec L_m, soit

(iii) un littéral négatif clos $L_m = \neg A_m$ est choisi et il existe une réfutation SLDNF de rang k de $P \bigcup \{\leftarrow A_m\}$.

(e) il existe au moins une feuille de l'arbre vérifiant (d)(i).

Definition 2.3 *Soit P un programme normal et B un but normal. Un* arbre SLDNF entièrement enlisé *pour $P \bigcup \{B\}$ est un arbre SLDNF entièrement enlisé de rang k pour $P \bigcup \{B\}$, pour un certain k.*

Nous étendons la notion d'arbre SLDNF afin de prendre en compte les littéraux négatifs clos pour lesquels il existe un arbre SLDNF entièrement enlisé. La définition suivante ne diffère de celle de [Llo87] que par l'ajout du cas $(d)(iii)$.
La notion de dérivation SLDNF peut être étendue de manière similaire.

Definition 2.4 *Soient P un programme normal et B un but normal. Un* arbre SLDNF *pour $P \bigcup \{B\}$ est un arbre vérifiant les conditions suivantes :*

(a) Chaque noeud de l'arbre est un but normal (éventuellement vide).

(b) La racine de l'arbre est B.

(c) Soit $\leftarrow L_1, ..., L_m, ..., L_p$ $(p \geq 1)$ un noeud de l'arbre qui ne soit pas une feuille et supposons que L_m soit choisi. Alors soit

(i) L_m est un atome et pour chaque (variante de) clause de programme $A \leftarrow M_1, ..., M_q$ telle que L_m et A soient unifiables avec l'upg θ, le noeud a un fils $\leftarrow (L_1, ..., L_{m-1}, M_1, ..., M_q, L_{m+1}, ..., L_p)\theta$, soit

(ii) L_m est un littéral négatif clos $\neg A$ et il existe un arbre SLDNF d'échecs finis pour $P \bigcup \{\leftarrow A_m\}$, auquel cas le seul fils est $\leftarrow L_1, ..., L_{m-1}, L_{m+1}, ..., L_p$.

(d) Soit $\leftarrow L_1, ..., L_p$ $(p \geq 1)$ une feuille de l'arbre alors soit

(i) un atome L_m est choisi et il n'y a pas de (variante de) clause de programme de P dont la tête s'unifie avec L_m, soit

(ii) un littéral négatif clos $L_m = \neg A_m$ est choisi et il existe une réfutation SLDNF de $P \bigcup \{\leftarrow A_m\}$, soit

(iii) chaque L_i est un littéral négatif non clos ou un littéral négatif clos $L_i = \neg A_i$ vérifiant : il existe un arbre SLDNF entièrement enlisé pour $P \bigcup \{\leftarrow A_i\}$.

(e) Les noeuds qui sont la clause vide n'ont pas de fils.

Une feuille qui vérifie (e) (resp. $(d)(i)$ ou $(d)(ii)$) est dite *succès* (resp. *échec*). Une feuille qui vérifie $(d)(iii)$ est dite *enlisement fort*.

Definition 2.5 *Un arbre SLDNF étendu est dit* fortement enlisé *s'il possède (au moins) une feuille enlisement fort.*

3 Indépendance de la règle de calcul

L'indépendance de la règle de calcul est une propriété souhaitable pour l'algorithme non déterministe qu'est la résolution SLDNF. malgré son importance ce problème n'est pas résolu en général.

Puisque nous nous intéressons a une définition étendue de la résolution SLDNF qui intègre une notion d'enlisement, une difficulté supplémentaire provient de cet enlisement. On peut comparer celle-ci aux difficultés rencontrées pour la résolution SLD avec retard ([BFR93],[Nai93]) ou pour la programmation logique concurrente (avec contraintes) ([CFMW93]). Ces papiers étudient l'indépendance de la règle de calcul (du scheduling pour [CFMW93]) et traitent d'une notion de deadlock qui est proche de l'enlisement. En ce qui concernce la résolution SLDNF, le problème est plus complexe de par la présence de dérivations subsidiaires.

La *réponse calculée* d'une feuille non échec d'un arbre SLDNF est la composition des substitutions utilisées le long de la branche.

Proposition 3.1 *Soient P un programme normal et B un but normal. Soient A et A' deux arbres SLDNF étendus pour $P \bigcup \{B\}$.*
S'il existe une feuille non échec H de A avec σ pour réponse calculée, $\langle C_1, ..., C_n \rangle$ étant la suite des variantes de clause utilisées le long de la branche. Alors il existe une feuille non échec H' de A' avec σ' pour réponse calculée, $\langle C'_1, ..., C'_n \rangle$ étant la suite des variantes de clause utilisées le long de la branche, vérifiant:
σ' est une variante de σ,
H' est une variante de H,
$\forall i \in \{1, ..., n\}$ il existe une variante $\tilde{C}_i$ de C_i telle que la suite $\langle C'_1, ..., C'_n \rangle$ soit une permutation de la suite $\langle \tilde{C}_1, ..., \tilde{C}_n \rangle$.

La preuve est basée sur la vision grammaticale de la programmation logique ([DM93]) qui offre un cadre général pour prouver des propriétés statiques de programmes logique. Il sagit d'une extension de la preuve du résultat principal de [BFR93] qui établit l'indépendance de la règle de calcul pour la résolution SLD avec retard.

Nous donnons une formulation très précise de cette indépendance en terme de squelettes (d'arbres de preuve). Nous étendons la notion de squelette avec des feuilles négatives. Nous définissons un ensemble particulier de squelettes (un squelette est un objet qui "résume" les choix de clause d'une dérivation) qui est intrinsèque au programme. Le résultat est alors prouvé par l'existence d'une bijection entre l'ensemble des feuilles non échec d'un arbre SLDNF étendu et cet ensemble de squelettes. La preuve n'est pas détaillée dans ce papier.

Nous pouvons maintenant donner la définition de l'enlisement fort pour un programme et un but :

Definition 3.1 *Soit P un programme normal et B un but normal. $P \bigcup \{B\}$ est* fortement enlisé *s'il existe un arbre SLDNF étendu fortement enlisé pour $P \bigcup \{B\}$.*

Remarque 3.1 *On peut considérer que (pour un programme et un but) l'enlisement fort n'est pas significatif s'il existe un* plein succès *(la réponse calculée n'instancie pas le but). La définition 3.1 peut être modifiée de la façon suivante : $P \bigcup \{B\}$ est* fortement enlisé *s'il existe un arbre SLDNF étendu fortement enlisé sans feuilles plein succès pour $P \bigcup \{B\}$.*

4 À propos d'implantation

La plupart des langages de programmation logique qui implantent la résolution SLDNF (avec la condition de sécurité) stoppent l'exécution d'un programme dès qu'un but ne comportant que des littéraux négatifs non clos est atteint ; un message d'erreur étant émis. Il est clair que même en cas d'enlisement, les réponses déjà calculées et les réponses de la partie non encore calculée de l'arbre sont correctes. Il semble donc légitime de revenir au point de choix précédent l'enlisement afin d'exécuter le reste du calcul. Ceci permet d'obtenir des réponses supplémentaires mais pose le problème de la dépendance de la régle de calcul ainsi que celui de l'incomplétude. Cette incomplétude peut être divisée en deux parties :

La première est l'incomplétude entre la sémantique déclarative et la résolution SLDNF. La sémantique du complété introduite par Clark est trop forte. Kunen en a proposé une version plus faible ([Kun87]). Néanmoins, pour certains programmes et certains buts, la résolution SLDNF demeure incomplète par rapport à la sémantique de Kunen.

La deuxième partie, qui est celle qui nous intéresse ici, est l'incomplétude entre la sémantique opérationnelle théorique (i.e. la résolution SLDNF) et la sémantique opérationnelle pratique (i.e. le calcul). Les raisons de cette incomplétude sont la non terminaison (par rapport au cas de la résolution SLD, le problème de la non terminaison est ici compliqué par les suites infinies d'appels négatifs) et l'enlisement.

Nous définissons dans ce chapitre une nouvelle notion de calcul - appelée *calcul étendu* - qui calcule un arbre SLDNF étendu dès lors qu'elle termine.

Nous commençons par définir ce qu'est un calcul et par discuter des problèmes opérationnels liés à l'enlisement. Puis nous définissons la notion de calcul étendu.

Dans [Llo87] un calcul est défini comme une tentative de construire une dérivation SLDNF. D'un point de vue plus pratique nous pouvons dire qu'un calcul pour la résolution SLDNF se comporte comme un calcul pour la résolution SLD tant qu'un atome est choisi. Lorsqu'un littéral négatif clos $\neg A$ est choisi dans un but H,

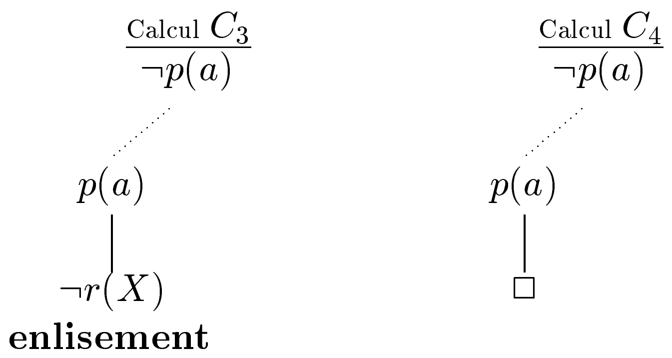

Figure 2 : Calculs pour l'exemple 4.1

un calcul pour $\leftarrow A$ est engagé. Si celui-ci construit un arbre d'échecs finis alors H a un fils avec le littéral $\neg A$ supprimé. S'il construit une réfutation SLDNF alors on conclue à un échec pour H.
Un calcul s'enlise si un but uniquement composé de littéraux négatifs non clos est atteint.

L'exemple du chapitre 1 montre que l'enlisement dépend de la règle de calcul. Un autre problème est que, dans les implantations de la résolution SLDNF, l'enlisement dépend des choix de clauses.

Exemple 4.1 *Considérons le programme normal $P2$:*
$p(a) \ \leftarrow \ \neg r(X).$
$p(a).$

et le but normal $B2$:
$\leftarrow \ \neg p(a).$

La figure 2 montre deux calculs pour $P2 \bigcup \{B2\}$. Le choix de la première clause dans la définition de p conduit à un enlisement (calcul C_3) tandis que le choix de la seconde clause conduit a un échec ; le calcul C_4 ne s'enlise pas.

Dans la résolution SLDNF, il y a apparemment deux choix : le choix du littéral et le choix de la clause. Le choix de la clause se situe au niveau du calcul et détermine le parcours de l'arbre. Cette dépendance est donc hautement non souhaitable.

Il existe un arbre SLDNF pour $P2 \bigcup \{B2\}$ mais le calcul C_3 ne peut le construire. Une réponse correcte pourrait être obtenue en revenant en arrière et en continuant le calcul "subsidiaire" pour $\neg p(a)$. On pourrait alors conclure à un échec pour $P2 \bigcup \{B2\}$, une réfutation de $p(a)$ étant construite ; et ce malgré l'existence d'un enlisement. Remarquons qu'un tel calcul serait de plus complet.

Ces problèmes de dépendance sont bien évidemment non souhaitables. De plus on aimerait une meilleure adéquation entre la résolution SLDNF et le calcul. Les raisons pour lesquelles, dans certains cas, un arbre SLDNF ne peut être construit (bien qu'il en existe un) sont montrées par les exemples 1.1 et 4.1. Il est alors possible d'obtenir une réponse correcte, l'existence d'un enlisement étant transparente.

Nous proposons qu'un calcul pour la résolution SLDNF applique les règles suivantes :

1. choisir dans un but un atome ou un littéral négatif clos *non marqué "enlisé"*.
2. Lorsqu'un but uniquement composé de littéraux négatifs non clos et de littéraux négatifs clos marqués "enlisé" est atteint, revenir en arrière jusqu'au point de choix précédent et continuer le calcul.
3. Lorsqu'un calcul pour un appel négatif $\neg A$ dans un but B construit un arbre (fini) sans succès mais avec au moins une feuille composée uniquement de littéraux négatifs non clos et de littéraux négatifs clos marqués "enlisé" (les règles 1 et 2 ayant été appliquées) (un tel arbre est un arbre SLDNF entièrement enlisé), marquer $\neg A$ "enlisé" dans B et choisir un autre littéral dans B (si cela est possible).

Nous appelons *calcul étendu* un calcul qui applique ces règles.
Nous définissons maintenant la notion d'enlisement fort dans un calcul étendu.

Definition 4.1 *Dans un calcul étendu, un but* fortement enlisé *est un but qui vérifie :*
- il n'est pas dans la portée d'un appel négatif (i.e. il fait partie du calcul principal).
- les littéraux qui le composent sont soit des littéraux négatifs non clos soit des littéraux négatifs clos marqués "enlisé".

Definition 4.2 *Un calcul étendu est* fortement enlisé *s'il possède (au moins) un but fortement enlisé.*

Exemple 4.2 *Aucun calcul étendu pour l'exemple 1.1 ou pour l'exemple 4.1 s'enlise fortement.*

La notion de calcul étendu est en adéquation avec la résolution SLDNF étendue : tout calcul étendu terminé construit un arbre SLDNF étendu. De plus un calcul étendu terminé qui n'est pas fortement enlisé (il peut y avoir des buts enlisés) construit un arbre SLDNF (au sens de [Llo87]). Les buts enlisés qui ne sont pas fortement enlisés ne sont donc pas significatifs. Dans ce contexte, l'enlisement ne doit plus être considéré comme une erreur. Eviter l'apparition d'enlisements peut être vu comme une question d'optimisation. Le problème est de construire des programmes qui ne s'enlisent pas fortement.

Nous avons montré au chapitre 3 que l'enlisement fort est indépendant de la règle de calcul. Contrairement aux nouvelles définitions de la résolution SLDNF, la définition de [Llo87] (et donc notre définition étendue) construit les arbres et dérivations subsidiaires en une seule étape. La règle de calcul ne concerne donc que l'arbre

SLDNF lui-même (qui est un quelque sorte réduit à l'arbre principal). Cependant le résultat d'indépendance peut être appliqué aux calculs étendus terminés.

Nous appelons *réponse enlisement fort* (d'un calcul étendu) un couple composé de la réponse calculée à une feuille enlisement fort et du but associé a cette feuille. Nous appelons *réponse succès* la réponse calculée à une feuille succès.

Nous avons en corollaire de la proposition 3.1 :

Corollaire 4.1 *Soit P un programme normal et B un but normal.*
Deux calculs étendus terminés de $P \bigcup \{B\}$ calculent les mêmes réponses succès et les mêmes reponses enlisement fort (à un renommage près).

En conséquence, deux calculs terminés (pour un programme et un but) calculent les mêmes réponses succès.

En l'absence d'enlisement le coût supplémentaire d'une implémentation de la résolution SLDNF étendue est nul : un calcul étendu qui ne s'enlise pas est un calcul.

5 Comparaison avec l'enlisement sérieux

Drabent propose dans [Dra94] une nouvelle notion d'enlisement qu'il appelle *enlisement sérieux*[2]. Il remarque que la notion d'équité doit être étendue. Pour un but $\leftarrow \neg A, \bar{L}$ ($\bar{L}$ étant une suite de littéraux), si le calcul de $\leftarrow A$ ne termine pas, le choix de $\neg A$ rend impossible le choix ultérieur d'un littéral de $\bar{L}$. Drabent définit (par l'intermédiaire d'un formalisme de négation constructive) une notion d'arbre SLDNF1 pré-échec[3] pour laquelle, dans ce cas, une dérivation pour $\leftarrow A$ et une dérivation pour $\leftarrow \bar{L}$ peuvent être construites simultanément. La définition de l'enlisement sérieux utilise ces arbres. Contrairement à l'enlisement fort, l'enlisement sérieux est défini par rapport à une règle de calcul.
Ce chapitre énonce un résultat de comparaison entre l'enlisement fort et l'enlisement sérieux. La notion d'équité dont il est question dans les propositions suivantes est liée aux arbres SLDNF1 pré-échec.

Nous avons besoin d'une notion de règle de calcul sûre (e.g. [Llo87]) mais pour des raisons techniques, nous utilisons une version légèrement modifiée :

Definition 5.1 *Une règle de calcul est sûre si elle vérifie :*
Si un littéral négatif non clos est choisi dans un but alors ce but ne contient que des littéraux négatifs non clos.

Proposition 5.1 *Soit P un programme normal et B un but normal. Soit R une règle de calcul sûre et équitable. Si $P \bigcup \{B\}$ est sérieusement enlisé pour R alors $P \bigcup \{B\}$ est fortement enlisé.*

[2] serious floundering
[3] SLDNF1 pre-failed tree

Schéma de preuve :
Par induction sur la profondeur de l'enlisement sérieux.
Soit B un but normal.
- S'il y a enlisement sérieux de profondeur 0 via R alors il existe une branche non échec finie d'un arbre SLDNF1 pré-échec pour $P \bigcup \{B\}$ via R dont le dernier but ne contient que des littéraux négatifs non clos (R est sûre).
Cette branche est une dérivation SLDNF fortement enlisée. Donc $P \bigcup \{B\}$ est fortement enlisé.
Si B est clos, il n'existe pas de réfutation SLDNF (def. de l'enlisement sérieux) donc il existe un arbre SLDNF entièrement enlisé de rang 0.
- S'il y a enlisement sérieux de profondeur k via R alors il existe une branche non échec finie d'un arbre SLDNF1 pré-échec pour $P \bigcup \{B\}$ via R dont le dernier but ne contient que des littéraux négatifs non clos ou des littéraux négatifs clos $\neg A_i$ tels qu'il existe un enlisement sérieux de profondeur k-1 pour A_i (R est sûre et équitable).
Par hypothèse d'induction, pour tout A_i il existe un arbre SLDNF entièrement enlisé (de rang k-1) pour $\leftarrow A_i$. Cette branche est une dérivation SLDNF étendue fortement enlisée. Donc $P \bigcup \{B\}$ est fortement enlisé.
Si B est clos, il existe une réfutation SLDNF. Donc il existe un arbre SLDNF entièrement enlisé de rang k.□

Proposition 5.2 *Soit P un programme normal et B un but normal. Soit R une règle de calcul sûre et équitable. S'il existe un arbre SLDNF étendu pour $P \bigcup \{B\}$ et $P \bigcup \{B\}$ n'est pas fortement enlisé alors $P \bigcup \{B\}$ n'est pas sérieusement enlisé pour R.*

Preuve :
Conséquence directe de la proposition 5.1.□

En conséquence de cette proposition, un calcul étendu terminé qui n'est pas fortement enlisé (sa règle de calcul est sûre et équitable) n'est pas sérieusement enlisé pour R.

Avec le résultat principal de [Dra94] nous avons (intuitivement) :

Proposition 5.3 *Soit P un programme normal et B un but normal. Un calcul étendu non fortement enlisé pour $P \bigcup \{B\}$ est complet par rapport à la sémantique de Kunen.*

La réciproque de la proposition 5.1 est vraie (i.e. si $P \bigcup \{B\}$ ne s'enlise pas sérieusement alors $P \bigcup \{B\}$ ne s'enlise pas fortement). L'enlisement sérieux et l'enlisement fort sont donc en un certain sens équivalents. Cependent l'enlisement fort est défini de manière plus simple (l'enlisement sérieux est défini par rapport à une règle de calcul).

En pratique, nous avons besoin de conditions de non enlisement fort. Dans notre approche, établir qu'un programme et un but ne s'enlisent pas fortement est simplement effectué par construction d'un arbre SLDNF étendu non fortement enlisé.

De plus nous donnons aussi un schéma d'implantation efficace pour la resolution SLDNF étendue.

6 Conclusion

Construire des programmes qui ne s'enlisent pas est un problème majeur de la résolution SLDNF. Cependant la notion standard d'enlisement n'est pas satisfaisante. D'une part elle est trop faible : certains enlisements ne sont pas significatifs. D'autre part il sagit d'une notion liée au calcul (et qui dépend de la règle de calcul). Une nouvelle définition abstraite du calcul est nécessaire.

Nous proposons une notion d'enlisement plus forte - l'enlisement fort - qui est défini de façon déclarative et peut être implantée efficacement. Nous pensons apporter ainsi une meilleure compréhension de ce qu'est l'enlisement dans la résolution SLDNF.

Des travaux récents ([AD92], [MT92], [MN90]) abordent le problème de donner une définition complète (dans le sens où pour un programme et un but il existe toujours un arbre SLDNF) de la résolution SLDNF. Cependant ces papiers ne traitent pas complètement le problème de l'enlisement. L'enlisement fort peut être incorporé à ces travaux afin d'obtenir une définition complète de la résolution SLDNF avec une notion déclarative d'enlisement indépendante de la règle de calcul .

Il est possible d'affaiblir la condition de sécurité ([Llo87], [MN90]). Définir l'enlisement fort pour une condition de sécurité plus faible peut s'effectuer par exemple à partir de la définition de la résolution SLDNF de [MN90].

En l'absence d'enlisement une implantation de la résolution SLDNF intégrant l'enlisement fort serait aussi efficace qu'une implantation standard. Dans certains cas un calcul étendu (i.e. un calcul pour une implantation intégrant l'enlisement fort) peut ne pas s'enliser fortement alors qu'une implantation standard s'enlise. Un calcul étendu qui termine et qui n'est pas fortement enlisé (il peut y avoir des buts enlisés) est correct et complet par rapport à la sémantique de Kunen. On peut donc dire que le calcul étendu est plus complet que le calcul standard.

Nous pensons que dans notre contexte, un enlisement (qui n'est pas un enlisement fort) ne doit plus être considéré comme une erreur. Eviter l'enlisement est un problème d'optimisation, comme éviter certaines branches échec. Le problème est maintenant de construire des programmes qui ne s'enlisent pas fortement. Les programmes non fortement enlisés forment une large classe caractérisée de façon déclarative. Il devrait être possible de construire des outils efficaces d'analyse de non enlisement fort.

Remerciements

L'auteur voudrait remercier Wlodzimierz Drabent pour ses commentaires sur une version préliminaire de ce papier.
Merci à Pierre Deransart, Gérard Ferrand et Jan Maluszynski pour de nombreuses discussions toujours fructueuses.

Références

[AB94] K. R. Apt, R. Bol. Logic Programming and Negation: A Survey. A paraître dans Journal of Logic Programming.

[AB91] K. R. Apt, M. Bezem. Acyclic programs. *New generation computing*, 29(3) pages 335-363, 1991.

[AD92] K. R. Apt, K. Doets. A new definition of SLDNF-Resolution. *Report CS-R9242, CWI*, 1992.

[AP91] K. R. Apt, D. Pedreschi. Proving termination of general Prolog programs. *Actes de International Conference on Theoretical Aspects of Computer Software*, Lecture notes in Computer Science 526, pages 265-289, Berlin, 1991. Springer-Verlag.

[BFR93] M. Bergère, G. Ferrand, J-H. Réty. Indépendance de la règle de calcul pour la résolution SLD avec retard. *Actes des troisièmes Journées Francophones de Programmation en Logique*, 1994.

[BM90] R. Barbuti, M. Martelli. Recognizing non-floundering logic programs and goals. International Journal on Foundations of Computer Science, 1(2) pages 151-163, 1990.

[CFMW93] M. Codish, M. Falaschi, K. Mariott, W. Winsborough. Efficient Analysis of Concurrent Constraint Logic Programs. *Actes de ICALP'93*, pages 633-644, 1993.

[Cla79] K. L. Clark. Predicate logic as a computational formalism. *Research Report DOC 79/59*, Imperial college, Dept. of computing, London, 1979.

[DM93] P. Deransart, J. Maluszinski. A Grammatical View of Logic Programming. *MIT Press*, 1993.

[Doe94] K. Doets. From Logic to Logic Programming. *MIT Press*, 1994.

[Dra94] W. Drabent. Completeness of SLDNF-Resolution for Non-Floundering Queries. Soumis, 1994.

[Kun87] K. Kunen. Negation in Logic Programming. *Journal of Logic Programming*, 4, pp. 289-308, 1987.

[Llo87] J. W. Lloyd. Foundations of Logic Programming. *Springer Verlag, 2nd ed.*, 1987.

[MN90] J. Maluszynski, U. Nilsonn. Logic programming and Prolog. *Wiley*, New-York, 1990.

[MSD90] K. Marriott, H. Sondergaard, P. Dart. A characterization of non-floundering logic programs, Debray et Hermenegildo ed., Actes de NA-CLP, pages 661-680, Austin 1990, *MIT Press 1990.*

[MT92] M. Martelli, C. Tricomi. A new SLDNF-tree. *Information Processing Letters* 43 (1992) pages 57-62.

[Nai93] L. Naish. Coroutining and the Construction of terminating logic programs. *Australian Computer Science Communications*, 15 (1), pp. 181-190, 1993.