Debugging Systems for Constraint Programming

ESPRIT 22532

Task T.WP2.1: **Declarative Debugging in Constraint Programming**

Delivrable D.WP2.1.M2.1-1

An Adaptation of Negative Declarative Error Diagnosis to any Computation Rule

Bernard Malfon and Alexandre Tessier

LIFO, rue Léonard de Vinci, BP 6759, 45067 Orléans Cedex 2, France
Bernard.Malfon@lifo.univ-orleans.fr, Alexandre.Tessier@inria.fr
http://www.univ-orleans.fr/SCIENCES/LIFO/Members/tessier/

http://discipl.inria.fr/

Abstract

This report is an additional information to the previous deliverable of the task T.WP2.1. It shows how to adapt the declarative diagnosis of negative errors when the SLD tree contains co-routining. It provides a procedure which computes a finite SLD tree according to a depth-first computation rule from a finite SLD tree with co-routining. Next it explains how to adapt negative diagnosis to the case of SLD tree according to a depth-first computation rule. The reader is assumed to know the previous deliverable D.WP2.1.M1.1-1.

1 Introduction

We assume that the reader knows the deliverable [3] from section 3 (page 9) until the end. We take up all the notations of this deliverable (look at the index and the table of notations pages 31–32 in [3]).

The first point (section 2) is to prove that if there exists a finite SLD tree for a goal then there exists a finite SLD tree for the same goal which uses a *depth-first computation rule* and to provide a procedure to build this SLD tree.

Intuitively, a depth-first computation rule is a computation rule such that when the chosen atom in the goal was not been introduced from the previous step then the atoms introduced from the previous step will never be chosen in the derivation. In terms of skeletons, let ν be the chosen node in a state of a SLD derivation. The node chosen at the next step of the derivation is either a child of ν or the undefined children of ν will never be chosen in the derivation.

> *Remark.* The standard computation rule and computation rules without co-routining are particular cases of depth-first computation rule. $\diamond$

Section 3 shows that we can adapt the definition of *negative proof tree*, and deduce from a finite SLD tree according to a depth-first computation rule a negative proof tree on which we can apply negative error diagnosis presented in [3].

Thus, now, negative diagnosis can be used with any computation rule.

2 From Co-Routining to Depth-First

The section shows that from an *arbitrary* finite SLD tree $\hookrightarrow_r^p$ we can compute a finite SLD tree $\hookrightarrow_{r'}^p$ where r' is a *depth-first* computation rule.

Definition 1 *Let us consider the final state s of a finite SLD derivation. The computation rule is depth-first if the sequence of the nodes chosen at each step corresponds to a depth-first traversal of s.*

> *Remark.* The previous definition can be extended to infinite SLD derivation (but this is out of the scope of this paper because we are just interested by finite computations).
> Indeed, it is possible to assiociate an infinite skeleton to each infinite SLD derivation (see [6]). The computation rule is depth-first if the sequence of the nodes chosen at each step corresponds to a depth-first traversal of the infinite skeleton.
> Note that an infinite SLD derivation according to a depth-first computation rule cannot have an associated infinite skeleton with several infinite branch. $\diamond$

A constraint logic program P and a finite SLD tree $\hookrightarrow_r^p$ are assumed to be fixed.

A finite SLD tree $\hookrightarrow_{r'}^p$, where r' is a depth-first computation rule, is defined inductively by the definition of its depth-first computation rule r' as follows:

1. The root of $\hookrightarrow_{r'}^p$ is $sk(p)$ and $r'(sk(p))$ is the root of $sk(p)$.
2. Let us consider a state s of $\hookrightarrow_{r'}^p$. Let s' be the parent of s in $\hookrightarrow_{r'}^p$ and $\nu = r'(s')$. Let μ be the first node of the path from ν to the root of s such that:
   - there exists an undefined child ρ of μ in s;
   - there exists a state s_1 of $\hookrightarrow_r^p$;
   - for each node κ of $def(s_1) \cap def(s)$: $symbol(s, \kappa) = symbol(s_1, \kappa)$;
   - and $\mu = r(s_1)$.

   We define $r'(s) = \mu$.

Remark. Note that when r is without co-routining, the previous definition allows to take $r' = r$. $\diamond$

Lemma 2 *$\hookrightarrow^p_{r'}$ is a finite SLD tree according to the depth-first comptation rule r'.*

3 Negative Diagnosis

The first step is to adapt the definition of *negative proof tree* (see pages 21–23 in [3]).

We modify the definition of extensions of a state s according to a set $g \subseteq undef(s)$ (denoted in by $ans_P(s, g)$ [3]).

The set of extensions of a state s of the SLD tree $\hookrightarrow^p_r$ for a subset $g \subseteq undef(s)$ according to a computation rule r is:

$$ans_P(s, g, r) = \left\{ s' \text{ of } \hookrightarrow^p_r \;\middle|\; \begin{array}{l} s \sqsubseteq s', \\ r(s') \text{ is not a descendant of a member of } g, \\ \text{there is no } s'' \neq s' \text{ in } \hookrightarrow^p_r \text{ such that} \\ s' \sqsubseteq s'' \text{ and} \\ r(s'') \text{ is not a descendant of a member of } g, \end{array} \right\}$$

Remark. Note that when r is without co-routining we have $ans_P(s, g, r) = ans_P(s, g)$. $\diamond$

We define now the formula associated to a state s of $\hookrightarrow^p_r$ and a set $g \subseteq undef(s)$, according to the computation rule r:

$$\mathcal{F}_P(s, g, r) = \forall(\exists_{-var(A_g)} c \wedge A_g \rightarrow \bigvee_{s' \in ans_P(s,g,r)} \exists_{-var(A_g)} c_{s'})$$

where

- for each $s' \in ans_P(s, g, r)$: $\forall(a \leftarrow c_{s'} \wedge A_{s'})$ is the formula proved by s';
- $\forall(a \leftarrow c \wedge A)$ is the formula proved by s;

- A_g is the conjunction of atoms associated to g in s.

Remark. The definition of $\mathcal{F}_P(s, g, r)$ is identical to the definition of $\mathcal{F}_P(s, g)$ (in [3]) except that $ans_P(s, g)$ is replaced by $ans_P(s, g, r)$. $\diamond$

Now, we are able to give the new negative proof tree definition.

We consider the following set of rules: for each state s of $\hookrightarrow_r^p$ (where r is a depth-first computation rule) and for each set $g \subseteq undef(s)$:

- if $g = \emptyset$

$$\frac{\emptyset}{\mathcal{F}_P(s, \emptyset, r)}$$

- if $g = \{r(s)\}$

$$\frac{\{\mathcal{F}_P(s', undef(s') \setminus undef(s), r) \mid s \hookrightarrow_r s'\}}{\mathcal{F}_P(s, \{r(s)\}), r}$$

- if $g \neq \emptyset$ and $g \neq \{r(s)\}$

$$\frac{\{\mathcal{F}_P(s, \{r(s)\}, r)\} \cup \bigcup\{\mathcal{F}_P(s', g \setminus \{r(s)\}, r) \mid s' \in ans_P(s, \{r(s)\}, r)\}}{\mathcal{F}_P(s, g, r)}$$

Remark. Note that the difference between this definition and the definition of [3] (except that $\mathcal{F}_P(s, g)$ and $ans_P(s, g)$ are replaced by $\mathcal{F}_P(s, g, r)$ and $ans_P(s, \{r(s)\}, r)$) is that we do not impose that $r(s) \in g$. $\diamond$

Lemma 3 *To each pair (s, g) corresponds exactly one (negative) proof tree of the formula $\mathcal{F}_P(s, g, r)$.*

Lemma 4 *The proof tree which corresponds to $(p, undef(p))$ is rooted by the negative answer formula to the goal $\forall(\leftarrow p(\tilde{x}))$.*

All the continuation is identical to the end of [3].

4 Conclusion

This paper presents an extension of the technics developped in [3] for declarative error diagnosis to all finite computation using any rule. It is a kind of errata. I should have put page ... read ... instead of ...!

This is realy a good news because the fact that we assumed a computation rule without co-routining for the negative part was a limitation wrt the classical methods used for logic programming [5, 2, 4].

Finaly I would remind the reader of the algorithm proposed in [1] was the starting point of this research.

References

[1] Wlodek Drabent, Simin Nadjm-Tehrani, and Jan Małuszyński. Algorithmic Debugging with Assertions. In Harvey Abramson and M. H. Rogers, editors, *Meta-Programming in Logic Programming*, pages 501–522. MIT Press, 1989.

[2] Gérard Ferrand. Error Diagnosis in Logic Programming: an adaptation of E. Y. Shapiro's method. *Journal of Logic Programming*, 4:177–198, 1987.

[3] Gérard Ferrand and Alexandre Tessier. Clarification of the bases of Declarative Diagnosers for CLP. Deliverable D.WP2.1.M1.1-1, 1997. Debugging Systems for Constraint Programming (ESPRIT 22532).

[4] John W. Lloyd. Declarative Error Diagnosis. *New Generation Computing*, 5(2):133–154,, 1987.

[5] Ehud Y. Shapiro. *Algorithmic Program Debugging.* ACM Distinguished Dissertation. MIT Press, 1982.

[6] Alexandre Tessier. *Approche, en termes de squelettes de preuve, de la sémantique et du diagnostic d'erreur des programmes logiques avec contraintes.* PhD thesis, LIFO, University of Orléans, 1996.

This research was supported in part by LOCO Project, common to University of Orléans and INRIA Rocquencourt.