# Implementing cTI: a constraint-based left-termination inference tool for LP

S. Hoarau

*IREMIA*
*Université de La Réunion, FRANCE*
*Email:* `seb@univ-reunion.fr`

F. Mesnard

*IREMIA*
*Université de La Réunion, FRANCE*
*Email:* `fred@univ-reunion.fr`

U. Neumerkel

*Institut für Computersprachen*
*Technische Universität Wien, AUSTRIA*
*Email:* `ulrich@complang.tuwien.ac.at`

**Abstract**

We present the implementation of cTI, a system for bottom-up termination inference. Termination inference is a generalization of termination analysis/checking. The architecture of cTI is presented and some optimizations and performance results for medium-sized programs are discussed.

## 1 Introduction

Termination is a crucial aspect of program verification. For logic programs, the problem is of particular importance because there is *a priori* no syntactic restriction on queries. Termination has been the subject of many works in the last fifteen years in the logic programming community [31,2,26]. The research efforts can be divided in two groups (a survey is given in [17]). The first group characterizes termination (e.g. [1]) with undecidable criteria. The second weakens those criteria to obtain computable sufficient conditions (e.g. [32]). Our approach belongs to the latter stream. Our main innovation compared to other works of termination analysis (e.g. [32,21,14,7,13]) is to provide a framework where we can *infer* sufficient universal termination conditions from the text of any Prolog program. Inference means that we adopt a bottom-up approach to termination. *There is no need to define a class of queries of interest.* All other works we are aware of require to add such a class. Moreover, such classes can be easily simulated within our framework.

Currently the only requirement we impose on programs is that they must not create infinite rational terms. Hence we only consider NSTO programs [16,15] that can be safely executed with any standard complying system or an execution with occurs check. Our system, called cTI, has been realized with SICStus Prolog and can be used at `http://www.complang.tuwien.ac.at/cti`. cTI is also integrated in the LP environment GUPU [25]. A formal justification of the algorithms underlying cTI can be found in [23] while [5] is a more conceptual paper, describing applications of termination inference and works in progress. In this paper, we present some insights, optimizations, and run times for three main algorithms that lie at the heart of cTI.

## 2 An Overview of cTI

Syntactic informations are often too weak to reason about non-trivial programs. Some semantics informations is required. For this reason our analyzer uses three main constraint structures [20]: Herbrand terms (CLP($\mathcal{H}$)) for the initial program $P$, non-negative integers (CLP($\mathbb{N}$)) and booleans (CLP($\mathcal{B}$)) for approximating $P$. The correspondence between these structures relies on *approximations* [23], which are a simple form [18] of abstract interpretation [9,10]. We present our method to infer termination conditions by using the predicates app/3, app3/4, and nrev/3.

```
app([], Xs, Xs).
app([X|Xs], Ys, [X|Zs]) ←
    app(Xs, Ys, Zs).
```

```
nrev([], []).
nrev([X|Xs], Ys) ←
    nrev(Xs, Zs),
    app(Zs, [X], Ys).
```

```
app3(Xs, Ys, Zs, Us) ←
    app(Xs, Ys, Vs),
    app(Vs, Zs, Us).
```

(i) The initial Prolog program $P$ is mapped to $P^{\mathbb{N}}$, a program in CLP($\mathbb{N}$) using an approximation based on a symbolic norm. In our example, we use the term-size norm:

$$\|t\|_{\mathrm{Term-Size}} = \begin{cases} 1 + \sum_{i=1}^{n} \|t_i\|_{\mathrm{Term-Size}} & \text{if } t = f(t_1, \ldots, t_n), n > 0 \\ 0 & \text{if } t \text{ is a constant} \\ t & \text{if } t \text{ is a variable} \end{cases}$$

E.g.$\|f(0,0)\|_{\mathrm{Term-Size}} = 1$. All non-monotonic elements of the program are approximated by monotone constructs. E.g., Prolog's unsound negation \+X is approximated by X;true. It is maintained that if a goal in $P^{\mathbb{N}}$ is terminating, then also the corresponding goals in $P$ terminate.

```
appℕ(0, Xs, Xs).
appℕ(1+X+Xs,Ys,1+X+Zs) ←
    appℕ(Xs, Ys, Zs).
```

```
nrevℕ(0, 0).
nrevℕ(1+X+Xs,Ys) ←
    nrevℕ(Xs, Zs),
    appℕ(Zs, 1+X, Ys).
```

$\mathrm{app3}_{\mathbb{N}}/4$ same as app3/4

(ii) In $\mathbb{N}$ we compute a model of all predicates. The model describes with a finite conjunction of linear equalities and inequalities the relations between the arguments of a goal (inter-argument relations IR) that hold for every solution. The actual computation is performed with CLP($\mathbb{Q}$). In our example we are able to determine the least model. In general, however, only a less precise model is determined. For recursive predicates, the actual source of potential non-termination, we compute linear strictly decreasing level mappings (see section 4) called $\mu$'s. For instance, the meaning of $\mu^{\mathbb{N}}_{\text{app}}$ is: for each recursive call to app/3, the first and the third argument decrease.

| (least) models | level mappings |
|---|---|
| $\text{IR}^{\mathbb{N}}_{\text{app}}(x,y,z) \equiv z = x + y$ | ${\mu^{\mathbb{N}}_{\text{app}}}^1(x,y,z) \equiv x$ |
| | ${\mu^{\mathbb{N}}_{\text{app}}}^2(x,y,z) \equiv z$ |
| $\text{IR}^{\mathbb{N}}_{\text{nrev}}(x,y) \equiv x = y$ | $\mu^{\mathbb{N}}_{\text{nrev}}(x,y) \equiv x$ |
| $\text{IR}^{\mathbb{N}}_{\text{app3}}(x,y,z,u) \equiv u = x + y + z$ | — |

(iii) $P^{\mathbb{N}}$ is mapped to $P^{\mathcal{B}}$, a program in CLP($\mathcal{B}$). Here 1 means that an argument is bound w.r.t. the considered norm. Note that the obtained program no longer maintains the same termination property. Its sole purpose is to determine the actual dependencies of boundedness within the program. The simplified structure permits us to compute always the least model. In a single boolean term all previously computed linear level mappings are represented.

| | | |
|---|---|---|
| $\text{app}_{\mathcal{B}}$(1, Xs, Xs).<br>$\text{app}_{\mathcal{B}}$(1∧X∧Xs,Ys,1∧X∧Zs) ←<br>$\text{app}_{\mathcal{B}}$(Xs, Ys, Zs). | $\text{nrev}_{\mathcal{B}}$(1, 1).<br>$\text{nrev}_{\mathcal{B}}$(1∧X∧Xs,Ys) ←<br>$\text{nrev}_{\mathcal{B}}$(Xs, Zs),<br>$\text{app}_{\mathcal{B}}$(Zs, 1∧X, Ys). | $\text{app3}_{\mathcal{B}}$/4<br>same as<br>app3/4 |

| least models | level mappings |
|---|---|
| $\text{IR}^{\mathcal{B}}_{\text{app}}(x,y,z) \equiv (x \wedge y) \Leftrightarrow z$ | $\mu^{\mathcal{B}}_{\text{app}}(x,y,z) \equiv x \vee z$ |
| $\text{IR}^{\mathcal{B}}_{\text{nrev}}(x,y) \equiv x \Leftrightarrow y$ | $\mu^{\mathcal{B}}_{\text{nrev}}(x,y) \equiv x$ |
| $\text{IR}^{\mathcal{B}}_{\text{app3}}(x,y,z,u) \equiv (x \wedge y \wedge z) \Leftrightarrow u$ | — |

The meaning of the boolean model is: when any call app(X,Y,Z) terminates sucessfully, then Z is finite iff X and Y are finite and, for nrev/2, when any proof of a call nrev(X,Y) terminates, X is finite iff Y is finite w.r.t. the used norm. Now, if we know (i.e. using level-mappings) that a call to app/3 terminates, then there is a finite number of successes and we may ensure the third argument is *bound* iff the two first arguments are *bound* w.r.t. the norm.

(iv) Finally, using all previously determined informations, $P^{\mathcal{B}}$ is translated into a system of boolean formulæ(see theorem 2.1) that ensures the propagation of the decreasing conditions of the level mappings through the call graph. The resolution of this system (computation of its greatest fixpoint by means of a boolean $\mu$-solver [8]) gives, for each predicate symbol, a boolean term called a *termination condition*.

$$\begin{array}{ll} Pre_{\mathrm{app}}(x,y,z) & \equiv x \vee z \\ Pre_{\mathrm{nrev}}(x,y) & \equiv x \\ Pre_{\mathrm{app3}}(x,y,z,u) & \equiv (x \wedge y) \vee (x \wedge u) \end{array}$$

So any call $\leftarrow$ C, app(X,Y,Z), where C is a CLP($\mathcal{H}$) constraint, left-terminates if X or Z are *ground* in C etc. The main result on which cTI is based on is the following Theorem 2.1, which ensures correctness of the left-termination condition.

**Theorem 2.1 ([22])** *Let $P$ be a program, $p$ and $q$ be two predicate symbols of $P$. Assume that $p$ is defined by $m_p$ rules $r_k$: $p(\tilde{x}) \leftarrow c_k,\ p_{k,1}(\tilde{x}_{k,1}), \ldots p_{k,n_k}(\tilde{x}_{k,n_k})$ and for each $q \notin \bar{p}$ and appearing in the rules defining $\bar{p}$, a left-termination condition $Pre_q$ has been computed. If the set of boolean terms $\{Pre_p\}_{p\in\bar{p}}$ verifies:*

$$\forall p \in \bar{p} \begin{cases} Pre_p(\tilde{x}) \rightarrow_{\mathcal{B}} \mu_p^{\mathcal{B}}(\tilde{x}), \\ [\forall 1 \leq k \leq m_p,\ \forall 1 \leq j \leq n_k, \\ \qquad \Big(Pre_p(\tilde{x}) \wedge c_k^{\mathcal{B}} \wedge \bigwedge_{i=1}^{j-1} Post_{p_{k,i}}(\tilde{x}_{k,i})\Big) \rightarrow_{\mathcal{B}} Pre_{p_{k,j}}(\tilde{x}_{k,j})] \end{cases}$$

*then $\{Pre_p\}_{p\in\bar{p}}$ is a left-termination condition for $\bar{p}$.*

*Running cTI*

Table 2 (page 5) presents timings of cTI using some standard benchmarks [1].

We have chosen twelve medium-sized well-known logic programs. Most of them are taken from [4] except CREDIT, PLAN and MINISSAEXP. Table 1 gives the following measures:

- *#facts* and *#rules*: the number of facts (unit clauses) and rules (non-unit clauses) in the program;
- *#cycles*: the number of strongly connected components (sccs, i.e. cycles of mutually recursive predicate symbols) in the call graph;
- *length*: the number of predicate symbols in the longest cycle in the call graph;
- *#vars*: the average arity of the predicate symbols of the longest cycle in the call graph.

The columns of Table 2 indicate the time for TNA analysis (section 5), the ratio of TNA arguments found in the program, the time for computing a model

[1] collected by Naomi Lindenstrauss, `http://www.cs.huji.ac.il/ naomil` and also available at `http://www.complang.tuwien.ac.at/cti/bench`.

Table 1
Informations about analyzed programs.

| Program | #facts | #rules | #cycles | length | #vars |
|---|---|---|---|---|---|
| ANN | 101 | 99 | 44 | 2 | 4 |
| BID | 24 | 26 | 20 | 1 | 3 |
| BOYER | 63 | 78 | 25 | 2 | 5 |
| BROWSE | 4 | 29 | 15 | 1 | 5 |
| CREDIT | 33 | 24 | 24 | 1 | 2 |
| MINISSAEXP | 37 | 223 | 100 | 5 | 4 |
| PEEPHOLE | 72 | 80 | 11 | 2 | 4 |
| PLAN | 12 | 17 | 16 | 1 | 3 |
| QPLAN | 63 | 87 | 38 | 3 | 8 |
| RDTOK | 7 | 57 | 12 | 4 | 5 |
| READ | 15 | 75 | 17 | 7 | 7 |
| WARPLAN | 43 | 68 | 33 | 3 | 6 |

$\mathrm{IR}_{\mathbb{N}}$ (section 3), determining of the level mappings $\mu$ (section 4), computing the least model $\mathrm{IR}_{\mathcal{B}}$ (section 3), inferring the left termination condition $\mathrm{TC}_{\mathrm{l}}$ (see [8]) and the total runtime. The timings given are average execution times over ten iterations.

Table 2
cTI 0.22 on a Pentium II, 300 MHz, 64Mo, SICStus 3.8.3, 3.3 Mlips.

| times in [ms] program | TNA | res_TNA | $\mathrm{IR}_{\mathbb{N}}$ | $\mu$ | $\mathrm{IR}_{\mathcal{B}}$ | TC | Total |
|---|---|---|---|---|---|---|---|
| ANN | 113 | 8% | 8790 | 14183 | 4936 | 1270 | 29292 |
| BID | 30 | 31% | 1476 | 1190 | 333 | 206 | 3235 |
| BOYER | 83 | 14% | 9683 | 1606 | 820 | 290 | 12482 |
| BROWSE | 33 | 2% | 1673 | 3340 | 450 | 190 | 5686 |
| CREDIT | 30 | 41% | 1113 | 700 | 323 | 146 | 2312 |
| MINISSAEXP | 230 | 19% | 12043 | 19570 | 3383 | 1253 | 36479 |
| PEEPHOLE | 76 | 13% | 5723 | 24593 | 1920 | 680 | 32992 |
| PLAN | 20 | 24% | 2043 | 1090 | 336 | 233 | 3722 |
| QPLAN | 116 | 21% | 8026 | 16953 | 4566 | 1733 | 31394 |
| RDTOK | 80 | 19% | 3430 | 3730 | 903 | 490 | 8633 |
| READ | 120 | 10% | 5886 | 20826 | 1773 | 25106 | 53711 |
| WARPLAN | 63 | 14% | 4736 | 6796 | 1153 | 293 | 13041 |
| mean % of the total | 0% | - | 28% | 49% | 9% | 14% | |

## 3 Fixpoint Computations

As explained in section 2, cTI determines models of two versions of the initial program: $P^{\mathbb{N}}$, the CLP($\mathbb{N}$) version, and $P^{\mathcal{B}}$, the CLP($\mathcal{B}$) version. For this purpose, we developed an abstract immediate consequence operator $U_P$. This operator is quite similar to the well-known $T_P$. Proofs can be found in [18] (in french).

### *3.1 The algorithm*

The key of our abstract computation is the notion of *rational interpretation* for a predicate symbol $p$:

**Definition 3.1** Let $P$ be a program and $p$ be a predicate symbol of $P$. We call a rational interpretation of $p$ an equivalence of the form: $p(\tilde{x}) \leftrightarrow c$ where $c$, a disjunction of conjunctions of atomic constraints, is a finite formula s.t. $vars(c) \subseteq \tilde{x}$. We extend this notion to $P$: a rational interpretation of $P$ is a set $I$ containing exactly one interpretation for each predicate symbol $p$ of $P$.

We write $\mathcal{I}$ the set of all rational interpretations. We want to compute a rational interpretation which is a model of $P$.

**Definition 3.2** $U_P$ is a function on $\mathcal{I}$ defined for any rational interpretation $I$ of a program $P$ by:

$$\begin{aligned} U_P(I) = \{\mathtt{p}(\tilde{x}) \leftrightarrow c \mid\ & c \equiv \textstyle\bigvee_{cl \in P} \left(\exists_{-\tilde{x}}(c_0 \wedge \textstyle\bigwedge_{1 \leq i \leq n} c_i)\right), \\ & cl \equiv \mathtt{p}(\tilde{x}) \leftarrow c_0 \diamond \mathtt{p}_1(\tilde{x}_1), \ldots, \mathtt{p}_n(\tilde{x}_n) \text{ and} \\ & \forall i \in [\![1; n]\!], \mathtt{p}_i(\tilde{x}_i) \leftrightarrow c_i \in I\} \end{aligned}$$

We define the successive powers of $U_P$ as usual.

**Proposition 3.3** $U_P$ *is monotone and continuous.*

Now let us establish a link between the meaning of a program $P$ and the $U_P$ operator. First, we give a ground semantics of a rational interpretation:

**Definition 3.4** Let $I$ be a rational interpretation, we define the semantics of $I$ by: $[I] = \{p(\tilde{d}) \mid p(\tilde{x}) \leftrightarrow c \in I, \tilde{d} \in \tilde{D}_\chi, \models_\chi c(\tilde{d})\}$ where $D_\chi$ is the domain of computation.

**Proposition 3.5** *For all* $n \in \mathbb{N}$, *we have* $T_P \uparrow n \subseteq [U_P \uparrow n]$.

Figure 1 presents an algorithm for the $U_P$ operator.

A corollary of propositions 3.3 and 3.5 states that the least fixpoint of $T_P$ is included the least fixpoint of $U_P$.

### *3.2 Widenings*

In general, lfp($U_P$) is not reachable in finite time. With a *widening* operator ($\nabla$) [10] convergence of the $U_P$ operator can be enforced. In cTI, we use widening for the computations in $\mathbb{N}$, whereas the boolean model is finitely reachable. Widening has a major impact on precision and speed of the computation. (We realized a *generic* fixpoint calculator for both CLP($\mathbb{Q}$) and CLP($\mathcal{B}$) [6,19,24]).

**function** $\texttt{Up}(P, I) : J$

**Require:** $P$ is a program, $I$ is a rational interpretation of $P$.

**Ensure:** $J$ is a rational interpretation of $P$.

1: $J \leftarrow \emptyset$ ;
2: **for all** clause$(p(\tilde{x}) \leftarrow c \ \diamond \ p_1(\tilde{x}_1), \ldots, p_n(\tilde{x}_n)) \in P$ **do**
3:     **for** $i \leftarrow 1$ **to** $n$ **do**
4:         **let** $p_i(\tilde{x}_i) \leftrightarrow c_i \in I$ ;
5:     **end for**
6:     **let** $p(\tilde{x}) \leftrightarrow c' \in J$ ;
7:     $c \leftarrow \bigvee(c', \Pi_{\tilde{x}}(c_1 \wedge \ldots \wedge c_n))$ ;
8:     $J \leftarrow update(J, p(\tilde{x}) \leftrightarrow c))$ ;
9: **end for**
10: **return** $J$ ;

Fig. 1. An abstract $T_P$-like operator.

A simple widening ($\nabla_1$) on a system of linear inequalities is due to Cousot and Cousot [10]. Here is an equivalent definition [27]:

**Definition 3.6** Let $S_1$ and $S_2$ be two sets of linear inequalities defining two polyhedra in $\mathbb{R}^n$. Then:

$$S_1 \ \nabla_1 \ S_2 = \{\beta \in S_1 \mid S_2 \Rightarrow \beta\}$$

Cousot and Cousot in [11] propose an improved version of $\nabla_1$, namely $\nabla_2$ which has been simplified for the sake of efficiency in [28]:

**Definition 3.7** Let $S_1$ and $S_2$ be two sets of linear inequalities defining two polyhedra in $\mathbb{R}^n$. Then:

$$\begin{aligned} S_1 \ \nabla_2 \ S_2 = {} & \{\beta \in S_1 \mid S_2 \Rightarrow \beta\} \bigcup \\ & \{\gamma \in S_2 \mid S_1 \Rightarrow \gamma \bigwedge \exists \beta \in S_1((S_1 - \{\beta\}) \cup \{\gamma\}) \Rightarrow \beta\} \end{aligned}$$

Comparisons of the impact of $\nabla_1$ and $\nabla_2$ on the accuracy of cTI are currently investigated. Figure 2 presents an algorithm for successive iterations of $U_P$ until it reaches a fixpoint.

### *3.3 Optimization*

The generic fixpoint computation engine is used for both $\mathbb{N}$ and $\mathcal{B}$. Making it as efficient as possible is therefore quite important. Currently all unit clauses defining the predicate symbols of the analyzed scc are taken into account in a first pass. Non-unit clauses are processed thereafter. Table 3 shows the impact of this optimization comparing cTI 0.19 and the optimized version 0.22. Note that we also replace the union operator of line 7 of the algorithm presented

**function** `itere_Up`$(P, max)$ : $I_n$

**Require:** $P$ is a CLP program, max is a non-negative integer.

**Ensure:** $I_n$ is a model of $P$, s.t. $\mathrm{lfp}(\mathtt{Up}) \subseteq I_n$ and $\mathtt{Up}(P, I_n) = I_n$.

$n \leftarrow 0$ ;<br>
$I_n \leftarrow \emptyset$ ;<br>
**repeat**<br>
  $I \leftarrow \mathtt{Up}(P, I_n)$ ;<br>
  $n \leftarrow n + 1$ ;<br>
  **if** $n \leq max$ **then**<br>
    $I_n \leftarrow I$ ;<br>
  **else**<br>
    $I_n \leftarrow I_{n-1} \nabla I$ ;<br>
  **end if**<br>
**until** $I_n \subseteq I_{n-1}$<br>
**return** $I_n$ ;

Fig. 2. An algorithm to reach a super set of lfp(Up).

in Fig. 1 by a convex hull (in both versions), which can be easily encoded in CLP($\mathbb{Q}$) using a technique proposed in [12] (see also [3]).

Table 3
Comparison between cTI 0.19 and cTI 0.22.

| times in [ms] | $\mathrm{IR}_{\mathbb{N}}$ | | | $\mathrm{IR}_{\mathcal{B}}$ | | |
|---|---|---|---|---|---|---|
| Programs | 0.19 | 0.22 | gain | 0.19 | 0.22 | gain |
| ANN | 8850 | 6340 | 28% | 4900 | 4900 | 0% |
| BID | 1470 | 950 | 35% | 330 | 330 | 0% |
| BOYER | 9640 | 8140 | 16% | 820 | 820 | 0% |
| BROWSE | 1700 | 1170 | 31% | 450 | 460 | -2% |
| CREDIT | 1110 | 800 | 28% | 320 | 330 | -3% |
| MINISSAEXP | 12040 | 8210 | 32% | 3330 | 3390 | -2% |
| PEEPHOLE | 5710 | 3360 | 41% | 1910 | 1910 | 0% |
| PLAN | 2030 | 980 | 52% | 340 | 340 | 0% |
| QPLAN | 8000 | 5880 | 27% | 4670 | 4550 | 3% |
| RDTOK | 3420 | 1990 | 42% | 890 | 900 | -1% |
| READ | 6360 | 4690 | 26% | 2270 | 1790 | 21% |
| WARPLAN | 4750 | 3270 | 31% | 1160 | 1150 | 1% |
| average | - | - | 32% | - | - | 1% |
| max. | - | - | 52% | - | - | 21% |

# 4 Computing level-mappings

*Level mappings* are a key concept of many approaches to termination. They map ground atoms to natural numbers. K. Sohn and A. Van Gelder developed in 1991 an algorithm (SVG in short, see [30]) based on linear programming which ensures the existence of linear level mappings. We present an extension of this algorithm for the automatic generation of level mappings. Since this method, despite its power, does not seem to be very well-known in the context of termination analysis, we recall it after some preliminaries. Our extension to SVG is presented thereafter.

## *4.1 Preliminaries*

We consider pure CLP($\mathbb{N}$) programs, with three predefined symbols for constraints: $=$, $\geq$, and $\leq$ with their usual meaning. Those programs are abstractions of (constraint) logic programs using (fixed or inferred) norms. We assume that clauses are written in flat form: $p_0(\tilde{x}_0) \leftarrow c_0, p_1(\tilde{x}_1), c_1, \ldots, c_{l-1}, p_l(\tilde{x}_l), c_l$, with $i \neq j \rightarrow \tilde{x}_i \cap \tilde{x}_j = \phi$. In order to simplify this presentation, *we disallow mutually recursive predicates*. Within cTI itself, there is no such restriction. Note that we frequently switch to CLP($\mathbb{Q}^+$) as some problems in this structure are computationally cheaper (e.g. satisfiability), trading precision for speed. From now on, we write CLP for CLP($\mathbb{N}$) or CLP($\mathbb{Q}^+$). Section 3 shows how we can compute a model $M$ for a CLP program $P$, where each predicate $p(\tilde{x})$ is defined as a (finite) conjunction of CLP constraints. We use this model to simplify the program $P$.

**Definition 4.1** Let $M_P$ be a model of the CLP program $P$. The definition of a predicate $p$ is *simplified wrt $M$* when, for the clauses defining $p/n$, we add to the right of each predicate $q(\tilde{x})$ its meaning $c_q(\tilde{x})$ relative to $M_P$. Moreover, those predicates $q/m \neq p/n$ which appear in the bodies are replaced by *true* (e.g. the dummy constraint $0 = 0$). Hence we end with a finite set of CLP clauses of the form: $p(\tilde{x}_0) \leftarrow c_0, p(\tilde{x}_1), c_1, \ldots, c_{l-1}, p(\tilde{x}_l), c_l$. The simplified program is denoted $P_M^{simpl}$.

We are interested in the automatic discovery of linear level mappings.

**Definition 4.2** Let $p/n$ be a recursive predicate symbol of a CLP program $P$. A *linear level mapping* $\mu$ for $p(x_1, \ldots, x_n)$ is a linear relation $\sum_{i=1}^{n} \mu_i x_i$, where the coefficients $\mu_i$ are non-negative integers.

Such linear level mappings should satisfy a property ensuring their usefulness for left-termination:

**Definition 4.3** A linear level mapping $\mu$ for $p$ is *valid wrt $P_{M_P}^{simpl}$* if for each recursive clause defining $p$ in $P_M^{simpl}$, say $p(\tilde{x}_0) \leftarrow c_0, p(\tilde{x}_1), c_1, \ldots, c_{l-1}, p(\tilde{x}_l), c_l$, for $k = 0$ to $l-1$, $\wedge_{i=0}^{k} c_i \rightarrow \mu^T \tilde{x_0} \geq 1 + \mu^T \tilde{x_k}$, where $\mu^T$ denotes the transposed vector of $\mu$.

### *4.2 The algorithm SVG*

Let us first quickly review the algorithm of Sohn and Van Gelder. It aims at checking the existence of one valid linear level mapping. SVG starts with a pure CLP program $P$ and a constrained goal. A top-down boundedness analysis (see [23,24]) reveals the calling modes of each predicate. Arguments are detected as either bounded (denoted $b$) or unbounded ($u$). A CLP model $M$ is computed and $P$ is simplified to $P_M^{simpl}$. Then SVG examines each recursive procedure $p/n$ in turn (the precise order does not matter). Let us symbolically define the level mapping for $p(x_1, \ldots, x_n)$ as $\mu^T \tilde{x} = \sum_{1 \leq i \leq n} \mu_i^{u\ or\ b} x_i$ where $\mu_i^u = 0$ is $x_i$ is labelled as unbounded wrt the calling mode of $p/n$ and $\mu_i^b \geq 0$ if $x_i$ is labelled as bounded. Each clause $r_i$ is processed. For one such clause, $l$ simplified rules (for $k = 0$ to $k = l-1$) are constructed: $p(\tilde{x_0}) \leftarrow \wedge_{0 \leq j \leq k} c_j, p(\tilde{x_k})$. One can assume that the constraint $C_{ij} = \wedge_{0 \leq j \leq k} c_j$ is satisfiable, already projected onto $\tilde{x_0} \cup \tilde{x_k}$, only contains inequalities of the form $\leq$, and implies $\tilde{x_0} \geq 0$ and $\tilde{x_k} \geq 0$. Such a simplified rule gives rise to the following (pseudo-)linear programming problem

$$minimize\ \theta = \mu^T(\tilde{x_0} - \tilde{x_k})\ subject\ to\ C_{ij} \tag{1}$$

A valid linear level mapping $\mu$ exists (at least for this recursive call of this clause) if $\theta^* \geq 1$ where $\theta^*$ denotes the minimum of the objective function. Unfortunately, because of the symbolic constants $\mu$, (1) is *not* a linear programming problem.

The *clever* idea of the authors is to consider its dual form:

$$maximize\ \eta = \tilde{y}\beta\ subject\ to\ \tilde{y} \geq 0 \wedge \tilde{y}A \geq (\mu, -\mu) \tag{2}$$

By duality theory (see [29] for instance), we have $\theta^* = \eta^*$. Now, the authors observe that $\mu$ appears linearly in the dual problem (it is not true for (1)) because no $\mu_i$ appears in $A$. Hence (2) can be rewritten, by adding $\eta \geq 1$ and $\tilde{\mu^b} \geq 0 \wedge \tilde{\mu^u} = 0$, as $S_{ij}$, a set of linear inequations. If the conjunction $S_p = \wedge_{i,j} S_{ij}$ for each recursive call and for each clause defining $p/n$ is satisfiable, then there exists a valid linear level mapping for $p/n$.

### *4.3 An extension of SVG*

Instead of checking satisfiability of $S_p$, we can project it onto $\mu$ (we do not need the top-down boundeness analysis explained subsection 4.2, all arguments are assumed bounded). Hence we get in one constraint *all* the valid linear level mappings. It remains to compute the maximal elements of $\Pi_\mu(S_p)$, given the partial order: $\mu^1 \succeq \mu^2$ if $\forall i \in [\![1; n]\!] \mu_i^1 \neq 0 \rightarrow \mu_i^2 \neq 0$.

**Example 4.4** For app/3, let $\mu(x,y,z) = ax+by+cz$. We have $\Pi_\mu(S_p) = \{a+c \geq 1\}$. There are two maximal elements: $\mu^1(x,y,z) = x$ and $\mu^2(x,y,z) = z$.

In some sense, given a model for a program, this extension is complete. But a more precise model can lead to more maximal elements. Hence the precision of the inferred CLP model is important. From an implementation point of view, this algorithm heavily relies on the costly projection operator. We found that a good strategy is to project constraints as soon as possible.

## 5 Inferring termination neutral arguments

Predicates frequently contain arguments that have no influence on universal left termination. Differences are a frequent pattern with this property. In most uses of differences (e.g. list differences), one argument of the difference is termination neutral.

**Definition 5.1** [Termination neutral argument (TNA)] Let $G$ be a goal $\leftarrow p(\ldots, x_i, \ldots)$, where the $i$-th argument is a variable $x_i$ that occurs only at this position and let $\theta$ be a substitution s.t. $\mathrm{DOM}(\theta) = x_i$. The argument $i$ is a left-termination neutral argument (TNA), if

$$\forall\theta : G\theta \,\mathrm{left-terminates} \Leftrightarrow G \,\mathrm{left-terminates}$$

Since the TNA property is independent of a particular norm, the analysis can be performed prior to any other analysis. The information is currently used to accelerate the search for level mappings: if the $i$-th argument of $p$ is detected as TNA, we set its corresponding coefficient $\mu_i$ to 0 before computing $\mu$. Table 4 summarizes the gain we obtain when applying the TNA analysis on the six most demanding programs.

**Example 5.2** The second argument of app/3, the third of app3/4, and the second of nrev/2 are termination neutral. For app/3, we can safely state $\mu(x,y,z) = ax + cz$.

To determine some termination neutral arguments we represent each argument of a predicate with a boolean 0-1 variable where 1 means termination neutral and 0 means that the argument may influence termination. Constraints are imposed that tell which arguments may influence termination. The actual result is computed by labeling for a maximal satisfying assignment.

We consider a clause (with nontrivial head unifications explicit) to consist of the following possibly empty sequences of goals $S_1 \circ S_2 \circ S_3 \circ S_4$. $S_1$ contains any goals, $S_2$ is a possibly nonterminating goal, $S_3$ are always terminating goals, $S_4$ is an always failing goal followed by any other goals. $S_3$ and $S_4$ are ignored in the analysis. Head variables that occur in $S_1$ may influence termination. All these variables are set to 0. A head variable $H$ that first occurs in $S_2$ may influence termination only if the argument $B$ where it occurs in is not termination neutral. The constraint $\neg B \Rightarrow \neg H$ ensures that if $B$ is false (may influence termination), then $H$ is false, too. The following predicate shows all constraints imposed for our three predicates. A labeling process finds the desired answer.

tnaargs([app($A_1, A_2, A_3$), app3($B_1, B_2, B_3, B_4$), nrev($C_1, C_2$)]) $\leftarrow$
% app/3 clause 1: $[] \circ [] \circ [A_1 = [], A_2 = A_3] \circ []$
% app/3 clause 2:
% $[A_1 = [X|Xs], A_3 = [X|Zs]] \circ [\mathrm{app}(Xs, A_2, Zs)] \circ [] \circ []$
$A_1 = 0, \neg A_2 \Rightarrow \neg A_2, A_3 = 0,$
% app3/4:
% $[\mathrm{app}(B_1, B_2, Vs)] \circ [\mathrm{app}(Vs, B_3, B_4)] \circ [] \circ []$
$B_1 = 0, B_2 = 0, \neg A_2 \Rightarrow \neg B_3, \neg A_3 \Rightarrow \neg A_4,$
% nrev/2 clause 1: $[] \circ [] \circ [C_1 = [], C_2 = []] \circ []$
% nrev/2 clause 2:
% $[C_1 = [X|Xs], \mathrm{nrev}(Xs, Zs)] \circ [\mathrm{app}(Zs, [X], C_2)] \circ [] \circ []$ % not optimal!
% $[C_1 = [X|Xs]] \circ [\mathrm{nrev}(Xs, Zs)] \circ [\mathrm{app}(Zs, [X], C_2)] \circ []$ % ideal
$C_1 = 0, \neg A_3 \Rightarrow \neg C_2$

For app/3 and app3/4 the results are optimal. However, in nrev/2 we are not able to infer that the second argument is termination neutral. While the second goal app/3 is in fact always terminating —cTI is also able to prove this— we do not have this information right at hand at this point in time prior to the actual termination inference. In fact, the primary purpose of TNA analysis within cTI is to speed up the subsequent inference process, sacrificing precision for speed. The analysis performed in this manner is typically faster than parsing the corresponding program text with read/1 using SICStus' tokenizer written in C. It has never been observed[2] to take longer than parsing the program twice.

Table 4
Impact of TNA on level mappings computation.

| Program | with TNA | no TNA | gain |
|---|---|---|---|
| ANN | 14120 | 14442 | 2 % |
| MINISSAEXP | 19578 | 21222 | 8 % |
| PEEPHOLE | 25938 | 25602 | -1 % |
| QPLAN | 16590 | 16888 | 2 % |
| RDTOK | 3772 | 4034 | 7 % |
| READ | 20862 | 21292 | 2 % |
| arith. mean | 16810 | 17247 | 3 % |
| max. gain | - | - | 8 % |

[2] cTI's WWW interface provides an option "TNA only" to perform this comparison

## 6 Conclusion

We have presented the main algorithms of cTI, our bottom-up left-termination inference tool for logic programs and given some running for medium-sized logic programs. The analysis requires three fixpoint computations and the inference of well-founded orders. We have described some optimizations and measured their impacts.

Still, more work is required to cope with certain kinds of programs like CHAT, as suggested by P. Tarau. A more detailed look to this program shows that it contains a predicate symbol with 14 arguments and one scc of 30 mutually recursive predicate symbols with 8 arguments per predicate symbol on the average. Our current implementation, using the constraint solvers of SICStus Prolog, is neither able to compute a numeric model nor the boolean model in reasonable time. Table 2 indicates that the bottleneck is situated in the numeric computations. We therefore consider to replace the generic solvers by more specialized tools (polyhedra manipulations, a a simplex solver optimized for projections, and a more efficient BDD package).

## References


[1] K.R. APT. *From Logic Programming to Prolog.* Prentice Hall, 1997.

[2] K.R. APT and D. PEDRESCHI. Studies in pure prolog: Termination. In J.W. LLOYD, editor, *Proc. of the Symp. in Computational Logic*, pages 150–176. Springer, 1990.

[3] F. BENOY and A. KING. Inferring argument size relationships with CLP($\mathbb{R}$). In *Logic Program Synthesis and Transformation.* Springer-Verlag, 1997.

[4] F. BUENO, M. GARCIA DE LA BANDA, and M. HERMENEGILDO. Effectiveness of global analysis in strict independence-based automatic program parallelization. In *Proceedings of International Symposium on Logic Programming*, pages 320–336, 1994.

[5] S. BURCKEL, S. HOARAU, F. MESNARD, and U. NEUMERKEL. cti: Bottom-up termination inference for logic programs. submitted for publication, 2000.

[6] M. CARLSSON. Boolean constraints in sicstus prolog. Technical Report T91:09, Swedish Institute of Computer Science, 1994.

[7] M. CODISH and C. TABOCH. A semantics basis for termination analysis of logic programs. *Journal of Logic Programming*, 41:103–123, 1999.

[8] S. COLIN, F. MESNARD, and A. RAUZY. Constraint logic programming and mu-calculus. *ERCIM/COMPULOG Workshop on Constraints*, 1997.

[9] P. COUSOT and R. COUSOT. Abstract interpretation: a unifed lattice model for static analysis of programs by construction or approximation of fixpoints.

*Conference Record of the 4th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, 1977.

[10] P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *Journal of Logic Programming*, 13:103–179, 1992.

[11] P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation. In *Lecture Notes of Computer Science*, volume 631. Springer-Verlag, 1992. Proceedings of PLILP'92.

[12] B. De Backer and H. Beringer. A CLP language handling disjunctions of linear constraints. In *Proc. of ICLP'93*, pages 550–563. MIT Press, 1993.

[13] S.K. Debray and N.W. Lin. Cost analysis of logic programs. ACM Transactions on Programming Languages and Systems, 15(5):826–875, November 1993.

[14] S. Decorte. *Enhancing the power of termination analysis of logic programs through types and constraints.* PhD thesis, Katholieke Universiteit Leuven, 1997.

[15] P. Deransart, A. Ed-Dbali, and L. Cervoni. *Prolog: The standard, reference manuel.* Springer-Verlag, 1996.

[16] P. Deransart, G. Ferrand, and M. Téguia. NSTO programs (not subject to occur-check). *Proc. of the Int. Logic Programming Symp.*, pages 533–547, 1991.

[17] D. De Schreye and S. Decorte. Termination of logic programs : the never-ending story. *Journal of Logic Programming*, 19-20:199–260, 1994.

[18] S. Hoarau. *Inférer et compiler la terminaison des programmes logiques avec contraintes.* PhD thesis, Université de La Réunion, 1999.

[19] C. Holzbaur. Ofai clp(q,r) manual, edition 1.3.3. Technical Report TR-95-09, Austrian Research Institute, 1995.

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

[21] N. Lindenstrauss and Y. Sagiv. Automatic termination analysis of logic programs. In *Proceedings of the 14th ICLP*, pages 63–77, 1997.

[22] F. Mesnard. *Etude de la terminaison des programmes logiques avec contraintes, au moyen d'approximations.* PhD thesis, Université Paris VI, 1993.

[23] F. Mesnard. Inferring left-terminating classes of queries for constraint logic programs by means of approximations. In *Proceedings of JICSLP'96*, pages 7–21. The MIT Press, 1996.

[24] F. Mesnard. Entailment and projection for CLP($\mathcal{B}$) and CLP($\mathbb{Q}$) in SICStus Prolog. *1st International Workshop on Constraint Reasoning for Constraint Programming*, 1997.

[25] U. Neumerkel. GUPU: A prolog course environment and its programming methodology. In M. Maher, editor, *Proc. of JICSLP'96*, page 549. MIT Press, 1996. http://www.complang.tuwien.ac.at/ulrich/gupu/.

[26] S. Ruggieri. *Verification and validation of logic programs.* PhD thesis, Università di Pisa, 1999.

[27] H. Sağam. *A Toolkit for Static Analysis of Constraint Logic Programs.* PhD thesis, University of Bristol, 1997.

[28] H. Sağlam and J. Gallagher. Static analysis of logic programs using CLP as a meta-language. Technical Report CSTR-96-003, University of Bristol, Department of Computer Science, 1996.

[29] A. Schrijver. *Theory of linear and integer programming.* John Wiley and Sons, 1986.

[30] K. Sohn and A. Van Gelder. Termination detection in logic programs using argument sizes. In *Proceedings of PODS'91*, pages 216–226, 1991.

[31] T. Vasak and J. Potter. Characterization of terminating logic programs. In *Proceedings of the 1986 IEEE Symposium on Logic Programming*, 1986.

[32] K. Verschaetse and D. De Schreye. Deriving termination proofs for logic programs, using abstract procedures. In *Proceedings of the 8th ICLP*, pages 301–315, 1991.