An Improved Non-Termination Criterion for Binary Constraint Logic Programs

Etienne Payet   Fred Mesnard

IREMIA,
université de la Réunion, Océan Indien, France

Workshop on Logic-based methods in Programming Environments, 2005

Where Is It?


AFRICA
North Atlantic Ocean
ALGERIA
LIBYA
S A H A R A
EGYPT
MAURITANIA
MALI
NIGER
CHAD
SUDAN
ETHIOPIA
SOMALIA
YEMEN
SAUDI ARABIA
TURKEY
IRAN
RUSSIA
KAZAKHSTAN
FRANCE
SPAIN
CONGO
DEM. REP. OF THE CONGO
TANZANIA
ANGOLA
ZAMBIA
MOZAMBIQUE
MADAGASCAR
SOUTH AFRICA
South Atlantic Ocean
Scale 1:51,400,000

Outline

Outline

Termination

There exists various web-interfaced termination analyzers for Prolog, e.g.

- *cTI* (*ISO-Prolog*)
- TALP
- TermiLog
- TerminWeb

They check or infer termination conditions for *universal termination* of Prolog programs.

Non-Termination

There is also at least one web-interfaced non-termination tool for *pure* Prolog programs:

- *nTI*

It generates classes of queries for which *existential non-termination* is insured: there exists an infinite branch in the search tree.

Optimal Termination Condition

The Idea:

When cTI and nTI produce complementary results, we hold optimal termination conditions for the given program wrt our language defining classes of queries.

An Example (1)

`ways(A,Cs,N)` iff

- N is the number of ways to change
- a given amount of money A
- using a fixed set Cs of coins values

NB: suggested by Mike Codish.

An Example (2)

```
add(0,X,X).
add(s(X),Y,s(Z)) :-
    add(X,Y,Z).

ways(A,[],0).
ways(0,Cs,s(0)).
ways(s(Amount),[C|Coins],N) :-
    add(C,NewAmount,s(Amount)),
    ways(s(Amount),Coins,N1),
    ways(NewAmount,[C|Coins],N2),
    add(N1,N2,N).
ways(s(Amount),[C|Coins],N) :-
     add(s(Amount),s(D),C),
     ways(s(Amount),Coins,N).
```

An Example (3)

cTI:

- term_cond(add(A,B,C),C+A)
- term_cond(ways(A,B,C),0)

What's wrong???

An Example (3)

cTI:

- term_cond(add(A,B,C),C+A)
- term_cond(ways(A,B,C),0)

What's wrong???

An Example (4)

Let's do an optimal termination check with *precision* $= 2$:

- ok for add/3:
  termConds=[[1],[3]],
  nonTermQueries=[[2]-add(s(A),B,s(C))],
  undecidedModes=[]
- problem with ways/3:
  termConds=[],
  nonTermQueries=[
  [1,2,3]-ways(s(A),[0],B),
  ...
  undecidedModes=[]

Oops ... ways(s(t_1),[0],t_2) loops for any term t_1 and term t_2.

An Example (4)

Let's do an optimal termination check with *precision* $= 2$:

- ok for add/3:
  termConds=[[1],[3]],
  nonTermQueries=[[2]-add(s(A),B,s(C))],
  undecidedModes=[]
- problem with ways/3:
  termConds=[],
  nonTermQueries=[
  [1,2,3]-ways(s(A),[0],B),
  ...
  undecidedModes=[]

Oops ... ways(s(t_1),[0],t_2) loops for any term t_1 and term t_2.

An Example (5)

```
ways(A,[],0).
ways(0,Cs,s(0)).
ways(s(Amount),[C|Coins],N) :-
    C=s(_),
    add(C,NewAmount,s(Amount)),
    ways(s(Amount),Coins,N1),
    ways(NewAmount,[C|Coins],N2),
    add(N1,N2,N).
ways(s(Amount),[C|Coins],N) :-
     add(s(Amount),s(D),C),
     ways(s(Amount),Coins,N).
```

An Example (6)

Let's redo an optimal termination check with *precision* $= 3$:

- ok for add/3
- ok for ways/3:
  termConds=[[1,2]],
  nonTermQueries=[
  [1,3]-ways(s(A),[s(0)|B],C),
  [2,3]-ways(s(A),[s(0)],B)
  undecidedModes=[]

An Example (7)

Hence, cTI + nTI *may* provide some means to:

- debug programs
- get a complete knowledge about the termination behaviour of programs.

Outline

1. Motivation
   - Termination/Non-termination in (C)LP
   - Previous Work
2. Our Contribution
   - Preliminary Definitions
   - Main Result

The Binary Unfoldings of a Logic Programs (1)

[Gabbrielli & Giacobazzi, 89] [Codish & Taboch, 99]

- A T_P-like operator : T_P^{bin}
- Input: a pure logic program P
- Output: $lfp(T_P^{bin}) = P^{bin}$ a possibly infinite set of facts and *binary* clauses

Property

Q, an atomic query, left-terminates wrt P

iff

Q terminates wrt P^{bin}

The Binary Unfoldings of a Logic Programs (2)

- compute $P^{bin}_{precision} = T^{bin}_{P} \uparrow precision$
- generalize the lifting lemma to infer classes non-terminating atomic queries from $P^{bin}_{precision}$
- hence we hold classes of non-terminating atomic queries for P

Outline

Preliminary Definitions (1)

We consider *ideal* CLP.

Definition (Set Described by a Query)

The set of atoms that is described by a query $S := \langle p(\tilde{t}) \mid d \rangle$ is $Set(S) = \{p(v(\tilde{t})) \mid \mathcal{D}_{\mathcal{C}} \models_v d\}$.

Definition (More General)

We say that a query S' is *more general than* a query S if $Set(S) \subseteq Set(S')$.

Lifting Theorem

Theorem (Lifting)

Consider a derivation step $S \underset{r}{\Longrightarrow} T$ *and a query* S' *that is more general than* S.
Then, there exists a derivation step $S' \underset{r}{\Longrightarrow} T'$ *where* T' *is more general than* T.

Preliminary Definitions (2)

Definition (Set of Positions)

- A *set of positions*, denoted by τ, is a function that maps each predicate symbol p to a subset of $[1, arity(p)]$.
- Let τ be a set of positions. Then, $\overline{\tau}$ is the set of positions defined as: for each predicate symbol p, $\overline{\tau}(p) = [1, arity(p)] \setminus \tau(p)$.

Preliminary Definitions (3)

Definition (Projection)

Let τ be a set of positions and p a predicate symbol of arity n.

- The *projection of p on τ* is the predicate symbol denoted by p_τ. Its arity equals the number of elements of $\tau(p)$.
- Let $\tilde{t} := (t_1, \ldots, t_n)$ be a sequence of n terms. The *projection of $\tilde{t}$ on τ*, denoted by $\tilde{t}_\tau$ is the sequence $(t_{i_1}, \ldots, t_{i_m})$ where $\{i_1, \ldots, i_m\} = \tau(p)$ and $i_1 \leq \cdots \leq i_m$.
- Let $A := p(\tilde{t})$ be an atom. The *projection of A on τ*, denoted by A_τ, is the atom $p_\tau(\tilde{t}_\tau)$.
- The *projection of a query $\langle A \mid d \rangle$ on τ*, denoted by $\langle A \mid d \rangle_\tau$, is the query $\langle A_\tau \mid d \rangle$.

Preliminary Definitions (4)

Definition (Filter)

- A *filter*, denoted by Δ, is a pair (τ, δ) where τ is a set of positions and δ is a function that maps each predicate symbol p to $\langle p_\tau(\tilde{u}) \mid d \rangle$ where $\mathcal{D}_\mathcal{C} \models \exists d$ and $\tilde{u}$ is a sequence of $arity(p_\tau)$ terms.
- Let $\Delta := (\tau, \delta)$ be a filter and S be a query. Let $p := rel(S)$. S *satisfies* Δ if $Set(S_\tau) \subseteq Set(\delta(p))$.
- Let $\Delta := (\tau, \delta)$ be a filter and S and S' be two queries. S' is Δ*-more general than* S if $S'_{\overline{\tau}}$ is more general than $S_{\overline{\tau}}$ and S' satisfies Δ.

First Result

Definition (Derivation Neutral)

Δ is *DN* for r if for each derivation step $S \underset{r}{\Longrightarrow} T$ and each query S' that is Δ-more general than S, there exists a derivation step $S' \underset{r}{\Longrightarrow} T'$ where T' is Δ-more general than T.

Theorem

Let Δ be a filter that is DN for r.
If $\langle B \mid c\rangle$ is Δ-more general than $\langle H \mid c\rangle$ then $\langle H \mid c\rangle$ loops with respect to $\{r\}$.

Preliminary Definitions (5)

Definition (Local Variables)

Let $r := p(\tilde{X}) \leftarrow c \diamond q(\tilde{Y})$ be a rule. The set of *local variables* of r is denoted by $local\_var(r)$ and is defined as:
$local\_var(r) := Var(c) \setminus (Var(\tilde{X}) \cup Var(\tilde{Y}))$.

Definition (sat)

Let $S := \langle p(\tilde{u}) \mid d \rangle$ be a query and $\tilde{s}$ be a sequence of $arity(p)$ terms. Then, $sat(\tilde{s}, S)$ denotes a formula of the form $\exists_{Var(S')}(\tilde{s} = \tilde{u}' \wedge d')$ where $S' := \langle p(\tilde{u}') \mid d' \rangle$ is any variant of S variable disjoint with $\tilde{s}$.

Outline

1. Motivation
   - Termination/Non-termination in (C)LP
   - Previous Work
2. Our Contribution
   - Preliminary Definitions
   - Main Result

DNlog = DN

Definition (Logical Derivation Neutral)

A filter $\Delta := (\tau, \delta)$ is *DNlog* for $r := p(\tilde{X}) \leftarrow c \diamond q(\tilde{Y})$ if

$$\mathcal{D}_\mathcal{C} \models c \rightarrow \forall_{\tilde{X}_\tau} \left[sat(\tilde{X}_\tau, \delta(p)) \rightarrow \exists_\mathcal{Y}[sat(\tilde{Y}_\tau, \delta(q)) \wedge c] \right]$$

where $\mathcal{Y} := Var(\tilde{Y}_\tau) \cup local\_var(r)$.

Theorem

Assume $\mathcal{C}$ enjoys the following property: for each $\alpha \in D_\mathcal{C}$, there exists a ground $\Sigma_\mathcal{C}$-term a such that $[a] = \alpha$.
Δ is DN for r iff Δ is DNlog for r.

Summary

- For constraint filtered derivations: DN = DNlog
- it strictly generalizes our previous criteria defined in SAS'02, SAS'04, and TOPLAS'06.

- Implementation:
  - SAS'02: CLP(H), filter: positions+*true*
  - SAS'04: CLP(Q), filter: positions+*true*
  - TOPLAS'06: CLP(H), filter: positions+*constraint*
  - WLPE'05: ?