Using CLP Simplifications to Improve Java Bytecode Termination Analysis

Fausto SPOTO, Lunjin LU, and Fred MESNARD

Dipartimento di Informatica, Università di Verona, Italy
Oakland University, U.S.A.
IREMIA, université de la Réunion, France

March 2009

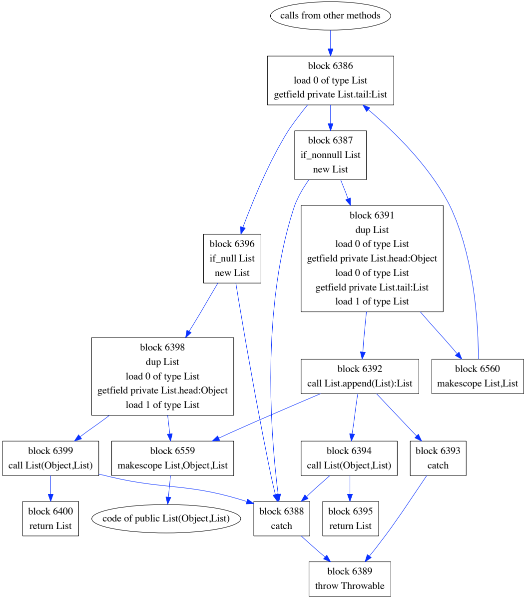
Introduction

JuliaWeb is a termination analyzer for Java Bytecode:
`http://julia.scienze.univr.it/termination`

In this paper:

- we propose a set of simplifications of the *CLP* programs generated by the termination analysis in JuliaWeb;
- we prove those transformations correct w.r.t. termination;
- we experiment with those transformations.

From Java Bytecode to CLP



From Java Bytecode to CLP

Example

```
block6391(IL0,IL1,IS0) :-
    {IS0=OS1, IL1=OS4, IS0=OS0, IL1=OL1, IL0=OL0,
     OS3 >= 0, OS2 >= 0, IL0-OS3 >= 1, IL0-OS2 >= 1},
    block6392(OL0,OL1,OS0,OS1,OS2,OS3,OS4).

block6391(IL0,IL1,IS0) :-
    {IS0=OS1, IL1=OS4, IS0=OS0, IL1=OL1, IL0=OL0,
     OS3 >= 0, OS2 >= 0, IL0-OS3 >= 1, IL0-OS2 >= 1},
    block6560(OL0,OL1,OS0,OS1,OS2,OS3,OS4).
```

- predicates are named `block`*i* or `entry`*j*;
- two arrows connect block 6391 with blocks 6392 and 6560;
- two local variables `L0` and `L1` are in scope: `L0` implements `this` and `L1` `other`;
- at the beginning of block 6391, there is only one stack element `S0`, while there are 5 at its end.

CLP($\mathbb{Z}_{Lin}$): Operational semantics

Let $\vec{m}, \vec{n} \in \mathbb{Z}^*$ and C be the clause

$$p(\vec{i}) \text{ :- } c, q(\vec{o})$$

where c is a linear constraint over the variables $\vec{i} \cup \vec{o}$.

- $q(\vec{n})$ *is derived from* $p(\vec{m})$ using C, written $p(\vec{m}) \rightarrow^C q(\vec{n})$, if there is a solution θ of $c[\vec{i} \mapsto \vec{m}]$ such that $q(\vec{n}) = q(\vec{o})\theta$.
- A *derivation* of $p_0(\vec{n}_0)$ is $p_0(\vec{n}_0) \rightarrow p_1(\vec{n}_1) \rightarrow \cdots \rightarrow p_k(\vec{n}_k)$ such that $p_{i+1}(\vec{n}_{i+1})$ is derived from $p_i(\vec{n}_i)$ for all $0 \leq i < k$.
- A *resolution* is a maximal derivation.

NB: our semantics is *ground* CLP.

CLP($\mathbb{Z}_{Lin}$): Termination

- An entry p *terminates* in a program P if, for every $\vec{n} \in \mathbb{Z}^*$, all resolutions of $p(\vec{n})$ by using the clauses of P, with predicates in the strongly connected component of p, are finite. Otherwise, p *diverges*.
- Let P_1 and P_2 be programs. P_1 *terminates more* than P_2, written $P_1 \sqsupseteq P_2$, if whenever an entry of P_1 terminates in P_1, it also terminates in P_2.
- P_1 and P_2 are *termination-equivalent*, written $P_1 \equiv P_2$, if P_1 terminates more than P_2 and vice versa.

NB: our definition formalizes a *loop-local* notion of termination.

PS1: Removing clauses outside loops

A clause $p(\vec{i})$:- $c, q(\vec{o})$ *occurs in a loop* if p and q belong to the same strongly connected component of predicates.

Proposition

Let P be a program and P_s be the same program deprived of those clauses that do not occur in a loop. Then $P \equiv P_s$.

PS2: Removing clauses by unfolding

If a program contains clauses $p(\vec{m})$:- $c_1, q(\vec{n})$ and $q(\vec{v})$:- $c_2, r(\vec{w})$, we can *unfold* them into the clause $p(\vec{m})$:- $c_1 \wedge c_2 \wedge \vec{n} = \vec{v}, r(\vec{w})$.

Done systematically for all occurrences of q on the right of the clauses of P, and followed by the removal of the clauses defining q, we say that we *unfold q away from P*.

Proposition

Let P be a program and q a non-entry predicate in P with no clause of the form $q(\vec{n})$:- $c, q(\vec{m})$. Let P_s be P where q has been unfolded away. Then $P \equiv P_s$.

PS3: Removing unsupported or subsumed clauses

$p(\vec{i})$:- $c, q(\vec{o})$ is *unsupported* if q is undefined.

$p(\vec{i})$:- $c_1, q(\vec{o})$ *subsumes* $p(\vec{i})$:- $c_2, q(\vec{o})$ if c_1 entails c_2.

Proposition

Let P be a program and P_s be P deprived from unsupported or subsumed clauses. Then $P \equiv P_s$.

Removing variables?

Let c be a constraint:

- $c^v \quad = $ the v*-dedicated part* of $c \quad = \exists_{-\{iv,ov\}}.c$
- $c^{-v} = $ the v*-independent part* of $c = \exists_{\{iv,ov\}}.c$

An operation that removes a variable from a predicate:

$$p(iv_1, \dots, iv_n) \ominus v = \begin{cases} p(iv_1, \dots, iv_{i-1}, iv_{i+1}, \dots, iv_n) & \text{if } v \equiv v_i \\ p(iv_1, \dots, iv_n) & \text{otherwise.} \end{cases}$$

The transformation:

$$Comp^{-v} = \{p(\vec{i}) \ominus v \text{ :- } c^{-v}, q(\vec{o}) \ominus v \mid p(\vec{i}) \text{ :- } c, q(\vec{o}) \in Comp\}$$

removes v from a strongly connected component *Comp*.

Removing variables?

Proposition

Let p_0 be an entry diverging in *Comp*. Then p_0 also diverges in $Comp^{-v}$.

In general, *Comp* is *not* termination-equivalent to $Comp^{-v}$, as shown by the counter-example 4.8 of the paper.

In what follows, we identify two cases where removal of a variable *maintains* termination equivalence. Common condition:

- a variable v is *isolated* in a strongly connected component *Comp* if, for every $p(\vec{i})$:- $c, q(\vec{o}) \in Comp$, $c = c^v \wedge c^{-v}$.

PS4: Removing open variables

- An isolated variable v in a strongly connected component *Comp* is *right-open* if, for every $p(\vec{i})$:- $c, q(\vec{o}) \in Comp$, we have that c^v is either *true* or $iv = ov$, or $ov \geq const$, $ov = const$ or $ov \leq const$ (or equivalent), where *const* is an integer constant. *Left-openness* is defined analogously.

Proposition

Let v be right- or left-open in a strongly connected component *Comp*. If an entry diverges in $Comp^{-v}$ then it diverges in *Comp*.

PS4: Removing open variables

Example

L1 is isolated and right-open in the component:

```
entry3880(IL0,IL1) :-
    {IL1 = OL1, OL0 >= 0, IL0 >= 2, IL0 - OL0 >= 1},
    entry3880(OL0,OL1).
```

Hence L1 can be removed:

```
entry3880(IL0) :-
    {OL0 >= 0, IL0 >= 2, IL0 - OL0 >= 1},
    entry3880(OL0).
```

PS5: Removing uniform variables

- An isolated variable v is *uniform* in a strongly connected component *Comp* if there is $x \in \mathbb{Z}$ such that, for every $p(\vec{i})$:- $c, q(\vec{o}) \in Comp$, the valuation $\{iv \mapsto x, ov \mapsto x\}$ is a solution of c^v.

Property

Let a variable v be uniform in a strongly connected component *Comp*. If an entry diverges in $Comp^{-v}$ then it diverges *Comp*.

PS5: Removing uniform variables

Example

```
block3853(IL0,IL1,IL2) :-
    {IL2 - OL2 = -1, IL1 = OL1, IL0 = OL0, IL1 - IL2 >= 1},
    block3853(OL0,OL1,OL2).
block3853(IL0,IL1,IL2) :-
    {IL2 - OL2 = -1, IL1 = OL1, OL0 = 1, IL1 - IL2 >= 2},
    entry3849(OL0,OL1,OL2).
entry3849(IL0,IL1,IL2) :-
    {IL2 = OL2, IL1 = OL1, IL0 = OL0, IL0 >= 1},
    block3853(OL0,OL1,OL2).
```

L0 is isolated. Taking $x = 1$ shows that L0 is uniform. Hence L0 can be removed.

Experiments

program	meth.	orig	loops	fold	subsum	open	unif
`Ack`	5	7.11	0.21	0.21	0.21	0.21	0.21
precision		5	5	5	5	5	5
`BubbleS`	5	19.07	1.55	0.71	0.71	0.49	0.49
precision		3	4	5	5	5	5
`NQueens`	222	-	210.31	156.32	92.29	47.77	34.34
precision		-	171	171	171	171	171
`JLex`	137	-	228.51	335.85	374.82	121.95	81.21
precision		-	84	87	102	102	102
`Kitten`	947	-	200.39	226.79	152.47	93.70	79.35
precision		-	811	827	827	827	827

Conclusion

We have presented:

- some termination-equivalent simplifications of the CLP programs that are automatically generated during termination analysis of Java bytecode programs;
- some real case of analysis showing that these simplifications decrease the time for building a proof by some order of magnitude.