TERMINATION ANALYSERS FOR FREE!

Aziem Chawdhary

Department of Computer Science, Queen Mary, University of London

Introduction

We are currently working on the development of termination analysers for imperative programs. These are tools that prove as many termination lemmas as possible as quickly as possible. These termination lemmas are of the form:

"the program location X within a loop of recursive function cannot be visited infinitely often during executions that stay within the loop or current recursive stack depth".

With our method, we can take any abstract interpretation based program analysis and produce a termination analysis using the results of the program analysis. Because we can use any abstract interpretation, we are producing many different termincation analysers, by converting invariance analyses into termination analyses. Although the technique we propose is sound for any analysis based on abstract interpretation, this does not mean any analysis will produce a useful termination analysis. We require the domains of the analysis to be *rich*, such as the Octagon domain. An example domain that would not produce a useful termination analyser is the sign domain. Our technique uses the ideas from Terminator and Transition Invariants to try and prove the termination of programs using arbitary program analyses.

Foundations

The foundation of our technique is based upon *Transition Invariants* [6], which is a binary relation approximating the pairs of reachable states (u, v) such that u is reachable and v is reachable from u via one or more program steps. Let $\longrightarrow_P$ be the transition relation for the program $\mathcal{P}$. Let $\longrightarrow_P^+$ be $\longrightarrow_P$'s transitive closure. A transition invariant T is defined to be a relation such that $\longrightarrow_P^+ \subseteq T$.

We construct these transition invariants for every cutpoint in the program and check if it is well-founded [5]. A relation R is well-founded if it has no infinite sequences $s_0, s_1, \ldots$ with respect to R

We use the results from other program analyses to construct the transition invariants we need to check termination. In essence, we get termination provers for free!

Outline of the Algorithm

We now outline the algorithm very briefly. Given a program P and an program analysis D_a on domain D, we perform the following:

1. Run the analysis D_a on P
2. For every cutpoint in the program
   (a) We seed the analysis result at that cutpoint. Seeding basically converts an invariant into a relation between states. We do this by adding auxillary variables and equalities into the invariant. For example, if we have an invariant $x - y >= 0$, then seeding will introduce two new variables x_s, y_s and add the equalities $x_s = x \wedge y_s = y$ to get $x_s = x \wedge y_s = y \wedge x - y >= 0$. The intuition is that x_s, y_s record the variables value before we do the next step.
   (b) Having seeded, we now run the analysis again on the seeded invariant to get another invariant. The values of x, y may have changed, but we know the relationship between their current values and their previous values because of the auxillary variables we introduced. Thus we have a relation from the previous state at the cutpoint and the current state.
   (c) We then check that the second generated invariant is well-founded. If any of the seeded invariants are not well-founded, then we can conclude that the program may diverge. On the other hand, if all the seeded invariants are well-founded, then the program must be terminating.

Example Termination Analysis

```
51    while(x>a && y>b) {
52
53        if (nondet()) {
54            do {
55
56                x = x - 1;
57
58            } while (nondet())
59
61        } else {
62            y = y - 1;
63
64        }
65
66    }
```

We present a small, informal example. In the example we use a program invariance analysis supporting finite disjunctive completion over the Octagon domain. The inner loop in the example above does not guarantee termination, due to the non-deterministic choice. However, it is not possible for the program to visit location 52 infinitely often. This property is called l-termination property. If each cutpoint in the program has the l-termination property, then the program terminates.

Example Termination Analysis

Location	Invariant
⋮	⋮
50	⋮
52	$x \geq a+1 \wedge y \geq b+1$
55	$x \geq a+1 \wedge y \geq b+1$
57	$x \geq a \wedge y \geq b+1$
59	$x \geq a \wedge y \geq b+1$
63	$x \geq a+1 \wedge y \geq b$
65	$(x \geq a \wedge y \geq b+1) \vee (x \geq a+1 \wedge y \geq b)$
67	⋮
⋮	⋮

We run the invariance analysis using the Octagon domain and finite disjunctive completeion, which might compute the following over-approximation of the reachable state space for this program indexed by locations (program counters). Each row in the table represents an invariant for the related program location. An abstract state can be seen to denote a set of concrete states. For example, at line 52 the invariant represents the states:

Example Termination Analysis

The next step is to seed the invariants at the cutpoint. In this example we are trying to prove that location 52 is not visited inifnetly often. We now seed the invariant at location 52. This involves adding seed variables to the invariant:

$$(x \geq a+1 \wedge y \geq b+1 \wedge x_s = x \wedge y_s = y \wedge b_s = b \wedge c_s = c)$$

This can be thought of as a binary relation on program states in the following way:

$$\begin{aligned}\{(s,t) \mid\ & \mathsf{pc}(s) = \mathsf{pc}(t) = 52 \\ & \wedge\ s(\mathtt{x}) = t(\mathtt{x}) \\ & \wedge\ s(\mathtt{y}) = t(\mathtt{y}) \\ & \wedge\ s(\mathtt{a}) = t(\mathtt{a}) \\ & \wedge\ s(\mathtt{b}) = t(\mathtt{b}) \\ & \wedge\ t(\mathtt{x}) \geq t(\mathtt{a}) + 1 \\ & \wedge\ t(\mathtt{y}) \geq t(\mathtt{b}) + 1 \\ \}\end{aligned}$$

Notice that we're using x_s to represent the value of x in s, and x to represent the value of x in t.

Example Analysis

Finally, we run our program analysis with this state as the starting state, which gives us a set of transition invariants

⋮	⋮
50	⋮
52	$(x \geq a+1 \wedge y \geq b+1 \wedge x_s \geq x+1 \wedge y_s \geq y \wedge a_s = a \wedge b_s = b) \vee (x \geq a+1 \wedge y \geq b+1 \wedge x_s \geq x \wedge y_s \geq y+1 \wedge a_s = a \wedge b_s = b) \vee (x \geq a-1 \wedge y \geq b+1 \wedge x_s \geq x+1 \wedge y_s \geq y+1 \wedge a_s = a \wedge b_s = b)$
53	⋮
⋮	⋮

One important aspect of this trick is that the analysis is not aware of our intended meaning of variables like x_s and y_s: it simply thinks of them as symbolic constants. It does not know that the states are representing relations.
Due to the result in [6], all the disjuncts at location 52 are well-founded. Thus, we have proven that location 52 will not be visited infinitely often.

The Good and the Bad

1. The Good
   (a) We have a 'family' of completely automatic termincation analysers, which work without the need for ranking functions
   (b) The analysis uses simple domains such as the Octagon domain
   (c) We reuse all the heuristics provided by the program analysis such as widening, narrowing.
   (d) If the termination proof does not succeed, we have a good starting point from a TERMINATOR style termination proof
   (e) The technique is fast!
   (f) The tehnique is robust with respect to arbitarily nested loops, as we are simple using the standard program analysis techniques to prove relationships between states at cutpoints.
2. The Bad
   (a) We require disjunctive domains. This in essence means that we have sets of values at program points rather than one all-encompassign value. We have developed heurisitics to convert a domain into a 'disjunctive domain' of sorts suitable for our analysis.

References and Acknowlegdements

References

[1] B. Cook, A. Podelski, and A. Rybalchenko. Abstraction refinement for termination. In *SAS'2005: Static Analysis Symposium*, volume 3672 of *LNCS*, pages 87–101. Springer, 2005.

[2] B. Cook, A. Podelski, and A. Rybalchenko. Termination proofs for systems code. In *PLDI'06: Programming Language Design and Implementation*, 2006.

[3] B. Cook, A. Podelski, and A. Rybalchenko. Terminator: Beyond safety. In *CAV'06: Conference on Computer Aided Verification*, 2006.

[4] A. Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 2006. (to appear).

[5] A. Podelski and A. Rybalchenko. A complete method for the synthesis of linear ranking functions. In *VMCAI'2004: Verification, Model Checking, and Abstract Interpretation*, volume 2937 of *LNCS*, pages 239–251. Springer, 2004.

[6] A. Podelski and A. Rybalchenko. Transition invariants. In *LICS'2004: Logic in Computer Science*, pages 32–41. IEEE, 2004.

[]

This is ongoing work with Prof Peter O'Hearn and Dr. Dino Distefano at Queen Mary, University of London and Dr Josh Berdine and Dr Byron Cook at Microsoft Research, Cambridge.