Program Termination analysis using MAX-SMT*

Daniel Larraz[1], Albert Oliveras[1], Enric Rodríguez-Carbonell[1], and Albert Rubio[1]

1 Universitat Politècnica de Catalunya, Barcelona, Spain

Abstract

We show how Max-SMT can be exploited in constraint-based program termination proving. The generation of a ranking function is expressed as a Max-SMT optimization problem where constraints are assigned different weights. As a result, quasi-ranking functions –functions that almost satisfy all conditions for ensuring well-foundedness– are produced in a lack of ranking functions. This allows our method to progress in the termination analysis where other approaches would get stuck. Moreover, Max-SMT makes it easy to combine the process of building the termination argument with the usually necessary task of generating supporting invariants. The method has been implemented in a prototype and succesfully tested on a wide set of programs showing its potential in practice.

1 Introduction

Proving termination is necessary to ensure total correctness of programs. Still, termination bugs are difficult to trace and are hardly notified: as they do not arise as system failures but as unresponsive behavior, when faced to them users tend to restart their devices without reporting to software developers. Due to this, approaches for proving termination of imperative programs have regained an increasing interest in the last decade [1, 2, 3, 4].

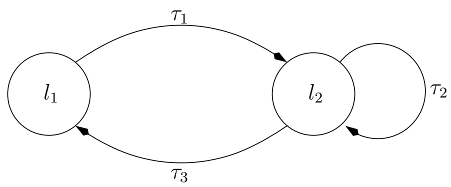
One of the major difficulties in these methods is that often *supporting invariants* are needed. In [5], by formulating both invariant and ranking function synthesis as constraint problems, both can be solved simultaneously, so that only the necessary supporting invariants for the targeted ranking functions –namely, *lexicographic linear ranking functions*– need to be discovered.

Based on this idea, we present a Max-SMT constraint-based approach for proving termination. The crucial observation in our method is that, although our goal is to show that transitions cannot be executed infinitely by finding a ranking function or an invariant that disables them, if we only discover an invariant, or a *quasi-ranking function* that almost fulfills all needed properties for well-foundedness, we have made some progress: either we can remove *part of a transition* and/or we have improved our knowledge on the behavior of the program. A natural way to implement this idea is by considering that some of the constraints are *hard* (the ones guaranteeing invariance) and others are *soft* (those guaranteeing well-foundedness) in a Max-SMT framework. Moreover, by giving different weights to the constraints we can set priorities and favor those invariants and (quasi-) ranking functions that lead to the furthest progress.

The technique has been implemented in our prototype of C++ analyzer CppInv. Thanks to our tool, we have proved termination of a wide set of programs, which have been taken from the programming learning environment Jutge.org [6] and from benchmark suites in the literature [7].

* This work has been partially supported by the Spanish MEC/MICINN under grantTIN 2010-68093-C02-01

```
int main () {
  int x, y, z;
  ℓ1 : while (y ≥ 1) {
    x−−;
    ℓ2 : while (y < z) {
       x++;
       z−−;
    }
    y = x + y;
  }
}
```



$$\Theta(\ell_1) \equiv \mathit{true} \qquad \Theta(\ell_2) \equiv \mathit{false}$$

$$\begin{array}{lllll}
\rho_{\tau_1}: & y \geq 1, & x' = x - 1, & y' = y, & z' = z \\
\rho_{\tau_2}: & y < z, & x' = x + 1, & y' = y, & z' = z - 1 \\
\rho_{\tau_3}: & y \geq z, & x' = x, & y' = x + y, & z' = z
\end{array}$$

Figure 1 Program and its transition system.

2 Encoding Termination using MAX-SMT

In this paper we model imperative programs by means of *transition systems*. See Fig. 1 for an example of a program together with the corresponding transition system. Note that primed versions of the variables represent the values of the variables after the transition and that Θ is a map from locations to formulas characterizing the initial values of the variables. From now on we assume that variables take integer values and programs are linear, i.e., the initial conditions Θ and transition relations ρ are described as conjunctions of linear inequalities.

An important class of invariant maps is that of *inductive invariant maps*:

▶ **Definition 1.** An invariant map μ is said to be *inductive* if:

- **[Initiation]** For every location ℓ: $\Theta(\ell) \models \mu(\ell)$
- **[Consecution]** For every transition $\tau = (\ell, \ell', \rho)$: $\mu(\ell) \wedge \rho \models \mu(\ell')'$.

The basic idea of the approach we follow for proving program termination [8] is to argue by contradiction that no transition is infinitely executable. First of all, it is obvious that disabled transitions (i.e., that can never be executed) cannot be infinitely executable. Moreover, one just needs to focus on transitions joining locations in the same strongly connected component (SCC): if a transition is executed over and over again, then its pre and post locations must belong to the same SCC. So let us assume that one has found a *ranking function* for such a transition τ, according to the following definition:

▶ **Definition 2.** Let $\tau = (\ell, \ell', \rho)$ be a transition such that ℓ and ℓ' belong to the same SCC, denoted by C. A function R is said to be a *ranking function* for τ if:

- **[Boundedness]** $\rho \models R \geq 0$
- **[Strict Decrease]** $\rho \models R > R'$
- **[Non-increase]** For every $\hat{\tau} = (\hat{\ell}, \hat{\ell}', \hat{\rho})$ such that $\hat{\ell}, \hat{\ell}' \in C$: $\hat{\rho} \models R \geq R'$

Note that boundedness and strict decrease *only* depend on τ, while non-increase depends on *all* transitions in the SCC.

Similarly to [5], we consider linear invariant and linear ranking function templates and take the following constraints from the definitions of inductive invariant and ranking function:

Initiation:	For ℓ:	$\mathbb{I}_\ell \stackrel{def}{=} \Theta(\ell) \vdash I_\ell$
Disability:	For $\tau = (\ell, \ell', \rho)$:	$\mathbb{D}_\tau \stackrel{def}{=} I_\ell \wedge \rho \vdash 1 \le 0$
Consecution:	For $\tau = (\ell, \ell', \rho)$:	$\mathbb{C}_\tau \stackrel{def}{=} I_\ell \wedge \rho \vdash I'_{\ell'}$
Boundedness:	For $\tau = (\ell, \ell', \rho)$:	$\mathbb{B}_\tau \stackrel{def}{=} I_\ell \wedge \rho \vdash R \ge 0$
Strict Decrease:	For $\tau = (\ell, \ell', \rho)$:	$\mathbb{S}_\tau \stackrel{def}{=} I_\ell \wedge \rho \vdash R > R'$
Non-increase:	For $\tau = (\ell, \ell', \rho)$:	$\mathbb{N}_\tau \stackrel{def}{=} I_\ell \wedge \rho \vdash R \ge R'$

Finally, let L and T be the sets of locations and transitions in the SCC under consideration, respectively. Let also P be the set of transitions that are *p*ending to be proved finitely executable. Then we construct the following constraint system, which is later on transformed into an SMT problem over linear and non-linear arithmetic:

$$\bigwedge_{\ell \in L} \mathbb{I}_\ell \wedge \bigwedge_{\tau \in T} (\mathbb{D}_\tau \vee \mathbb{C}_\tau) \wedge \bigvee_{\tau \in P} (\mathbb{D}_\tau \vee (\mathbb{B}_\tau \wedge \mathbb{S}_\tau)) \wedge ((\bigwedge_{\tau \in P} \mathbb{N}_\tau) \vee \bigvee_{\tau \in P} \mathbb{D}_\tau).$$

The first two conjuncts guarantee that an invariant map is computed; the other two, that at least one of the pending transitions can be discarded. Notice that, if there is no disabled transition, we ask that *all* transitions in P are non-increasing, but only that at least *one* transition in P (the next to be removed) is both bounded and strict decreasing. Note also that for finding invariants one has to take into account *all* transitions in the SCC, even those that have already been proved to be finitely executable: otherwise some reachable states might not be covered, and the invariant generation would become unsound. Hence in our termination analysis we consider two transition systems: the original transition system for invariant synthesis, whose transitions are T and which remains all the time the same; and the *termination transition system*, whose transitions are P, i.e, where transitions already shown to be finitely executable have been removed. This duplication is similar to the *cooperation graph* of [7].

The idea is to consider the constraints guaranteeing invariance as *hard*, so that any solution to the constraint system will satisfy them, while the rest are *soft*. Let us consider propositional variables $p_\mathbb{B}$, $p_\mathbb{S}$ and $p_\mathbb{N}$, which intuitively represent if the conditions of boundedness, strict decrease and non-increase in the definition of ranking function are violated respectively, and corresponding weights $\omega_\mathbb{B}$, $\omega_\mathbb{S}$ and $\omega_\mathbb{N}$. We consider now the next constraint system (where soft constraints are written $[\cdot, \omega]$, and hard ones as usual):

$$\bigwedge_{\ell \in L} \mathbb{I}_\ell \wedge \bigwedge_{\tau \in T} \Big(\mathbb{D}_\tau \vee \mathbb{C}_\tau\Big) \wedge \bigvee_{\tau \in P} \Big(\mathbb{D}_\tau \vee \big((\mathbb{B}_\tau \vee p_\mathbb{B}) \wedge (\mathbb{S}_\tau \vee p_\mathbb{S})\big)\Big) \wedge$$
$$\Big((\bigwedge_{\tau \in P} \mathbb{N}_\tau) \vee \bigvee_{\tau \in P} \mathbb{D}_\tau \vee p_\mathbb{N}\Big) \wedge [\neg p_\mathbb{B}, \omega_\mathbb{B}] \wedge [\neg p_\mathbb{S}, \omega_\mathbb{S}] \wedge [\neg p_\mathbb{N}, \omega_\mathbb{N}].$$

Note that, since all constraints are fulfiled, ranking functions have cost 0, and (if no transition is disabled) functions that fail in any of the conditions are penalized with the respective weight. Thus, the Max-SMT solver looks for the best solution and gets a ranking function if feasible; otherwise, the weights guide the search to get invariants and quasi-ranking functions that satisfy as many conditions as possible.

Hence this Max-SMT approach allows recovering information even from problems that would be unsatisfiable in the initial method. This information can be exploited to perform dynamic trace partitioning [9] as follows. Assume that the optimal solution to the above Max-SMT formula has been computed, and let us consider a transition $\tau \in P$ such that $\mathbb{D}_\tau \vee ((\mathbb{B}_\tau \vee p_\mathbb{B}) \wedge (\mathbb{S}_\tau \vee p_\mathbb{S}))$ evaluates to true in the solution. Then we distinguish several cases depending on the properties satisfied by τ and the computed function R:

- If τ is disabled then it can be removed.

- If R is non-increasing and satisfies boundedness and strict decrease for τ, then τ can be removed too: R is a ranking function for it.
- If R is non-increasing and satisfies boundedness for τ but not strict decrease, one can split τ in the termination transition system into two new transitions: one where $R > R'$ is added to τ, and another one where $R = R'$ is enforced. Then the new transition with $R > R'$ is automatically eliminated, as R is a ranking function for it. Equivalently, this can be seen as adding $R = R'$ to τ. Now, if the solver could not prove R to be a true ranking function for τ because it was missing an invariant, this transformation will guide the solver to find that invariant so as to disable the transition with $R = R'$.
- If R is non-increasing and satisfies strict decrease for τ but not boundedness, the same technique from above can be applied: it boils down to adding $R < 0$ to τ.
- If R is non-increasing but neither strict decrease nor boundedness are fulfilled for τ, then τ can be split into two new transitions: one with $R < 0$, and another one with $R \geq 0 \wedge R = R'$.
- If R does not satisfy the non-increase property, then it is rejected; however, the invariant map from the solution can be used to strengthen the transition relations for the following iterations of the termination analysis.

Note this analysis may be worth applying on other transitions τ in the termination transition system apart from those that make $\mathbb{D}_\tau \vee ((\mathbb{B}_\tau \vee p_\mathbb{B}) \wedge (\mathbb{S}_\tau \vee p_\mathbb{S}))$ true. E.g., if R is a ranking function for a transition τ but fails to be so for another one τ' because strict decrease does not hold, then, according to the above discussion, τ' can be strengthened with $R = R'$.

On the other hand, working in this iterative way requires imposing additional constraints to avoid getting to a standstill. Namely, in the case where non-increase does not hold and so one would like to exploit the invariant, it is necessary to impose that the invariant is not redundant.

Another advantage of this Max-SMT approach is that by using different weights we can express priorities over conditions. Since, as explained above, violating the property of non-increase invalidates the computed function R, it is convenient to make $\omega_\mathbb{N}$ the largest weight. On the other hand, when non-increase and boundedness are fulfilled but not strict decrease an equality is added to the transition, whereas when non-increase and strict decrease are fulfilled but not boundedness just an inequality is added. As we prefer the former to the latter, in our implementation we set $\omega_\mathbb{B} > \omega_\mathbb{S}$.

Further refinements are possible. E.g., the termination transition system can also be used for generating properties that are guaranteed to eventually hold at a location for some computations. More specifically, we devised the following light-weight approach for generating what we call *termination implications*. In a nutshell, for each location ℓ a new linear inequality template J_ℓ is introduced and the following constraint is imposed: $\bigwedge_{\tau=(\hat{\ell},\ell,\rho)\in P}(\mathbb{D}_\tau \ \vee \ I_{\hat{\ell}} \wedge \rho \vdash J'_\ell)$. The rationale is that, if we find a property J_ℓ that is implied by all transitions going into ℓ and ℓ is finally reached, then J_ℓ must hold. Then this termination implication can be propagated forward to the transitions going out from ℓ, i.e., J_ℓ can be conjoined to $I_\ell \wedge \rho$ in the termination transition system. Finally, additional constraints are imposed to ensure that new termination implications are not redundant with the already computed invariants and termination implications.

▶ **Example 3.** Let us show a termination analysis of the program in Fig. 1. In the first round, the solver finds the invariant $y \geq 1$ at ℓ_2 and the ranking function z for τ_2. While $y \geq 1$ can be added to τ_3 (resulting into a new transition τ'_3), the ranking function allows eliminating τ_2 from the termination transition system.

In the second round, the solver cannot find a ranking function. However, thanks to the Max-SMT formulation, it can produce the quasi-ranking function x, which is non-increasing and strict decreasing for τ_1, but not bounded. This quasi-ranking function can be used to split transition τ_1 into two new transitions $\tau_{1.1}$ and $\tau_{1.2}$ as follows:

$$\rho_{\tau_{1.1}}: \quad \boldsymbol{x \geq 0}, \quad y \geq 1, \quad x' = x - 1, \quad y' = y, \quad z' = z$$
$$\rho_{\tau_{1.2}}: \quad \boldsymbol{x < 0}, \quad y \geq 1, \quad x' = x - 1, \quad y' = y, \quad z' = z$$

Then $\tau_{1.1}$ is immediately removed, since x is a ranking function for it.

In the third and final round, the termination implication $x < 0$ is generated at ℓ_2, together with the ranking function y for transition τ_3'. Note that the termination implication is crucial to prove the strict decrease of y for τ_3', and that the previously generated invariant $y \geq 1$ at ℓ_2 is needed to ensure boundedness. Now τ_3' can be removed, which makes the graph acyclic. This concludes the termination proof.

3 Conclusion

The method presented here has been implemented in the tool CppInv[1].

This tool has been proved competitive in comparison with the new version of T2, which according to the results given in [7] is performing much better when proving termination than most of the existing tools.

For a full description of the method, its implementation and the experimental evaluation, see [10].

References

1. D. Dams, R. Gerth, O. Grumberg, A heuristic for the automatic generation of ranking functions, in: Workshop on Advances in Verification, 2000, pp. 1–8.
2. M. Colón, H. Sipma, Synthesis of linear ranking functions, in: TACAS, Vol. 2031 of Lecture Notes in Computer Science, Springer, 2001, pp. 67–81.
3. A. Podelski, A. Rybalchenko, A complete method for the synthesis of linear ranking functions, in: VMCAI, Vol. 2937 of Lecture Notes in Computer Science, Springer, 2004, pp. 239–251.
4. A. Tiwari, Termination of linear programs, in: CAV, Vol. 3114 of Lecture Notes in Computer Science, Springer, 2004, pp. 70–82.
5. A. R. Bradley, Z. Manna, H. B. Sipma, Linear ranking with reachability, in: CAV, Vol. 3576 of Lecture Notes in Computer Science, Springer, 2005, pp. 491–504.
6. J. Petit, O. Giménez, S. Roura, Jutge.org: an educational programming judge, in: SIGCSE, ACM, 2012, pp. 445–450.
7. M. Brockschmidt, B. Cook, C. Fuhs, Better termination proving through cooperation, in *CAV*, 2013, to appear.
8. M. Colón, H. Sipma, Practical methods for proving program termination, in: CAV, Vol. 2404 of Lecture Notes in Computer Science, Springer, 2002, pp. 442–454.
9. L. Mauborgne, X. Rival, Trace partitioning in abstract interpretation based static analyzers, in: M. Sagiv (Ed.), European Symposium on Programming (ESOP'05), Vol. 3444 of Lecture Notes in Computer Science, Springer-Verlag, 2005, pp. 5–20.
10. D. Larraz, A. Oliveras, E. Rodríguez-Carbonell, A. Rubio, Proving termination of imperative programs using max-smt, in *FMCAD*, 2013, to appear.

[1] CppInv, together with all benchmarks used in the experimental evaluation, is available at `www.lsi.upc.edu/~albert/cppinv-term-bin.tar.gz`.