Program Termination analysis using MAX-SMT*

Daniel Larraz¹, Albert Oliveras¹, Enric Rodríguez-Carbonell¹, and Albert Rubio¹

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 successfully 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 grant TIN 2010-68093-C02-01
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: Θ(ℓ) |= μ(ℓ)
- [Consecution] For every transition τ = (ℓ, ℓ′, ρ): μ(ℓ) ∧ ρ |= μ(ℓ′).

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 τ = (ℓ, ℓ′, ρ) 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] ρ ||= R ≥ 0
- [Strict Decrease] ρ ||= R > R′
- [Non-increase] For every ˆτ = (ˆℓ, ˆℓ′, ˆρ) such that ˆℓ, ˆℓ′ ∈ C: ˆρ ||= R ≥ 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:
dynamic trace partitioning \[9\] as follows. Assume that the optimal solution to the above
would be unsatisfiable in the initial method. This information can be exploited to perform
ranking functions that satisfy as many conditions as possible.

Thus, the Max-SMT solver looks for the best solution and gets a ranking
transition is disabled) functions that fail in any of the conditions are penalized with the
soft constraints are written
system (where soft constraints are written
termination analysis we consider two transition systems: the original transition system for
might not be covered, and the invariant generation would become unsound. Hence in our
termination transition system
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 \(p_B, p_S\) and \(p_N\), 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 \(\omega_B, \omega_S\) and \(\omega_N\). We consider now the next constraint system (where soft constraints are written \([\cdot, \omega]\), and hard ones as usual):

\[
\bigwedge_{\ell \in L} I_\ell \land \bigwedge_{\tau \in T} \left( D_\tau \lor C_\tau \right) \land \bigvee_{\tau \in P} \left( D_\tau \lor \left( B_\tau \lor (S_\tau \lor p_B) \right) \right) \land \\
\left( \bigwedge_{\tau \in P} \left( N_\tau \lor p_N \right) \land \left[ -p_B, \omega_B \right] \land \left[ -p_S, \omega_S \right] \land \left[ -p_N, \omega_N \right] \right).
\]

Note that, since all constraints are fulfilled, 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
\(D_\tau \lor (B_\tau \lor p_B) \land (S_\tau \lor p_B)\) evaluates to true in the solution. Then we distinguish several
cases depending on the properties satisfied by \(\tau\) and the computed function \(R\):

- If \(\tau\) is disabled then it can be removed.
If $R$ is non-increasing and satisfies boundedness and strict decrease for $\tau$, then $\tau$ can be removed too: $R$ is a ranking function for it.

If $R$ is non-increasing and satisfies boundedness for $\tau$ but not strict decrease, one can split $\tau$ in the termination transition system into two new transitions: one where $R > R'$ is added to $\tau$, 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 $\tau$. Now, if the solver could not prove $R$ to be a true ranking function for $\tau$ 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 $\tau$ but not boundedness, the same technique from above can be applied: it boils down to adding $R < 0$ to $\tau$.

If $R$ is non-increasing but neither strict decrease nor boundedness are fulfilled for $\tau$, then $\tau$ can be split into two new transitions: one with $R < 0$, and another one with $R \geq 0 \land R = R'$.

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 $\tau$ in the termination transition system apart from those that make $\mathbb{D}_\tau \lor ((\mathbb{B}_\tau \lor \rho_2) \land (\mathbb{S}_\tau \lor \rho_2))$ true. E.g., if $R$ is a ranking function for a transition $\tau$ but fails to be so for another one $\tau'$ because strict decrease does not hold, then, according to the above discussion, $\tau'$ 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 $\omega_\mathbb{D}$ 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 $\omega_\mathbb{D} > \omega_\mathbb{B}$.

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 $\ell$ a new linear inequality template $J_\ell$ is introduced and the following constraint is imposed: $\bigwedge_{\tau=(\ell,\ell,\rho)\in P} (\mathbb{D}_\tau \lor \mathbb{I}_\ell \land \rho \lor J_\ell^\tau)$. The rationale is that, if we find a property $J_\ell$ that is implied by all transitions going into $\ell$ and $\ell$ is finally reached, then $J_\ell$ must hold. Then this termination implication can be propagated forward to the transitions going out from $\ell$, i.e., $J_\ell$ can be conjoined to $I_\ell \land \rho$ 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 $\ell_2$ and the ranking function $z$ for $\tau_2$. While $y \geq 1$ can be added to $\tau_3$ (resulting into a new transition $\tau'_3$), the ranking function allows eliminating $\tau_2$ 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 $\tau_1$, but not bounded. This quasi-ranking function can be used to split transition $\tau_1$ into two new transitions $\tau_{1,1}$ and $\tau_{1,2}$ as follows:

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

$$\rho_{\tau_{1,2}} : \quad x < 0, \quad y \geq 1, \quad x' = x - 1, \quad y' = y, \quad z' = z$$

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 $\ell_2$, together with the ranking function $y$ for transition $\tau'_3$. Note that the termination implication is crucial to prove the strict decrease of $y$ for $\tau'_3$, and that the previously generated invariant $y \geq 1$ at $\ell_2$ is needed to ensure boundedness. Now $\tau'_3$ 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


---

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