Certifying Safety and Termination Proofs for Integer Transition Systems*

Marc Brockschmidt[1], Sebastiaan J.C. Joosten[2], René Thiemann[2], and Akihisa Yamada[2]

1 **Microsoft Research Cambridge, UK**
2 **University of Innsbruck, Austria**

1 Introduction

A number of recently introduced techniques for proving termination of programs in languages such as Java [7, 8] and C [2, 4, 5] rely on a two-step process, in which the input program is first transformed into an intermediate formal language, and then a standard termination analyzer is used on the intermediate program. These intermediate languages are usually variations of integer transition systems (ITSs), reflecting the pervasive use of built-in integer data types in programming languages. For example, the C program in Figure 1 is translated to the ITS in Figure 2.

```
while (y >= -3)
  y = y - 2;
while (x >= 0) {
  x = x + y;
  y = y - 1;
}
```

Figure 1 Input C program

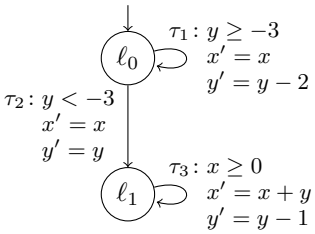

Figure 2 ITS $\mathcal{P}$, the input program as an ITS

Thus, to establish *trustworthiness* of such proofs of termination, two problems need to be tackled. First, the soundness of the translation from the source programming language needs to be proven, using elaborate models capturing the semantics of advanced programming languages. Then, the soundness of termination proofs on ITSs needs to be proven.

In this work, we tackle the second problem by discussing ongoing work to automatically *certify* termination proofs generated for a given ITS. While this continues work on certification of termination proofs of term rewrite systems [9], proving termination of ITSs requires substantially different techniques and introduces new challenges. Most notably, these include the handling of integers, the existence of designated start states of the computation, and the need to support program invariants. To generate program invariants needed for termination proofs, a number of approaches reduce the termination analysis problem to a sequence of program safety problems [1, 3, 10], which are passed to an underlying safety prover.

Thus, we first discuss how to certify safety proofs for a generalization of ITSs in Sect. 3, and then present our ongoing work on certifying termination proofs on top of this in Sect. 4.

* This work was partially supported by FWF project Y757. The authors are listed in alphabetical order regardless of individual contributions or seniority.

42nd Conference on Very Important Topics (CVIT 2016).
Editors: John Q. Open and Joan R. Acces; Article No. 23; pp. 23:1–23:5

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2 Logic Transition Systems

Instead of ITSs, we consider a more general subset of labeled transition systems in which the actions are defined by formulas whose syntax and semantics are specified as follows:

▶ **Definition 1.** A *logic* Λ specifies a typed signature Σ and its interpretation. We denote by $\Lambda_\sigma(V)$ the set of terms of type σ over typed variables from V. An *assignment* α on V assigns each variable in V a value from the corresponding domain.

Further, we assume a type bool and $\wedge : \mathsf{bool} \times \mathsf{bool} \to \mathsf{bool} \in \Sigma$, interpreted as usual. We call a term $\phi \in \Lambda_{\mathsf{bool}}(V)$ a *formula*, and write $\alpha \models \phi$ if ϕ is interpreted to true under assignment α, and $\phi \models \psi$ to denote semantic entailment.

▶ **Definition 2.** A *logic transition system (LTS)* is a tuple $(\Lambda, \mathcal{V}, \mathcal{L}, \ell_0, \mathcal{P})$, where Λ is a logic, $\mathcal{V}$ is the set of *program variables*, $\mathcal{L}$ is the set of *locations*, $\ell_0 \in \mathcal{L}$ is the *initial location*, and $\mathcal{P}$ is the set of transition rules. Here, a *transition rule* is a triple of $\ell, r \in \mathcal{L}$ and a formula $\phi \in \Lambda_{\mathsf{bool}}(\mathcal{V} \cup \mathcal{X} \cup \mathcal{V}')$, written $\ell \xrightarrow{\phi} r$, where $\mathcal{X}$ is a set of auxiliary variables, $\mathcal{V}'$ is the set $\{v' \mid v \in \mathcal{V}\}$, and v' is a new variable called *post* variable for v.

Concerning notation, we write t' (resp. ϕ') for term t (resp. formula ϕ) where all variables v are replaced by v', and we often just write $\mathcal{P}$ for the whole LTS. Note that an LTS can be seen as a labeled transition system, which is usually abbreviated also to LTS.

A *state* is a pair (ℓ, α) of a location $\ell \in \mathcal{L}$ and an assignment α on $\mathcal{V}$. Every assignment α gives rise to an *initial state* (ℓ_0, α). There is a *transition step* from $s = (\ell, \alpha)$ to $t = (r, \beta)$, written $s \to_\mathcal{P} t$, iff $\ell \xrightarrow{\phi} r \in \mathcal{P}$ and $\alpha \cup \beta' \cup \gamma \models \phi$, where β' is the assignment on $\mathcal{V}'$ defined by $\beta'(v') = \beta(v)$, and γ is an arbitrary assignment on the auxiliary variables. If there is a transition sequence $(\ell_0, \alpha_0) \to_\mathcal{P} \cdots \to_\mathcal{P} (\ell_n, \alpha_n)$, then the state (ℓ_n, α_n) and the location ℓ_n is said to be *reachable*.

We define ITSs by fixing Λ to the integer arithmetic, i.e., there is a type int whose domain is $\mathbb{Z}$, and constants, addition, multiplication, (in)equalities etc. are in the signature.

3 Certifying Safety Proofs

The safety of a program means that certain "bad" states cannot be reached, and is usually modeled by adding a single error location $\ell_\lightning \in \mathcal{L}$ that is reached from such bad states. Proving safety then reduces to showing that $\ell_\lightning$ is not reachable.

Safety provers typically work by finding inductive invariants that show the error location is unreachable, or equivalently, that the invariant 'false' holds for the error location. To find such invariants, safety provers usually transform the transition system. Hence, a certifier has to check the soundness of such transformations, i.e., that every execution in the original system can be simulated in the transformed system, besides the validity of invariants.

Our certifier supports safety proofs as produced by the Impact [6] algorithm. Given a LTS $\mathcal{P}$ with error location $\ell_\lightning$, a safety proof takes the form of a graph $\mathcal{G}$ in which nodes (ℓ, ϕ) represent all states (ℓ, α) where $\alpha \models \phi$.

▶ **Definition 3.** A graph over nodes from $\mathcal{L} \times \Lambda_{\mathsf{bool}}(\mathcal{V})$ is a valid *unwinding* for a LTS $\mathcal{P}$ if

- it contains a node (ℓ_0, true);
- every node is categorized as either a *transition node* or a *covered node*;
- for every transition node (ℓ, ϕ) and transition rule $\ell \xrightarrow{\psi} r \in \mathcal{P}$ there is an edge $(\ell, \phi) \longrightarrow (r, \chi)$, called a *transition edge*, such that $\phi \wedge \psi \models \chi'$;

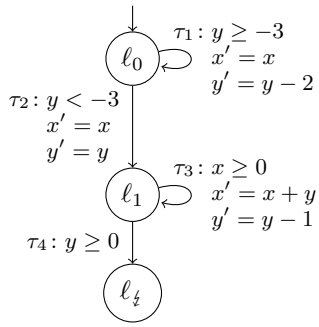

Figure 3 LTS $\mathcal{Q}$, safety ensures $y < 0$ at ℓ_1

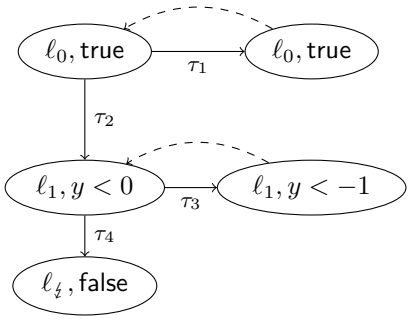

Figure 4 Graph $\mathcal{G}$, a valid unwinding of $\mathcal{Q}$

- for every covered node (ℓ, ϕ) there is an edge $(\ell, \phi) \dashrightarrow (\ell, \chi)$, called a *cover edge*, such that (ℓ, χ) is a transition node and $\phi \models \chi$;
- In every node $(\ell_\lightning, \phi)$ the formula ϕ is unsatisfiable, i.e., $\phi \models \mathsf{false}$.

▶ **Example 4.** Consider again the LTS $\mathcal{P}$ of Figure 2. In order to prove the termination of the second while loop, the invariant $y < 0$ in location ℓ_1 is essential. To verify this invariant, we create a copy of $\mathcal{P}$ as $\mathcal{Q}$, which additionally contains a transition $\ell_1 \xrightarrow{y \geq 0} \ell_\lightning$, cf. Figure 3. Clearly, if $\mathcal{Q}$ is safe, then $y < 0$ holds at location ℓ_1.

To ensure the safety of $\mathcal{Q}$, graph $\mathcal{G}$ in Figure 4 is constructed using the Impact algorithm [6]. Here, the node $\ell_1, y < -1$ is a "covered node", whose safety is proven by referring to node $\ell_1, y < 0$, which "covers" all described program states. Checking validity demands several entailment checks. For instance, for the edge for transition τ_3 we need to ensure

$$\underbrace{y < 0}_{\phi} \wedge \underbrace{x \geq 0 \wedge x' = x + y \wedge y' = y - 1}_{\psi} \models \underbrace{y' < -1}_{\chi'}.$$

We have formally proven that the existence of a valid unwinding ensures safety in Isabelle/HOL.

▶ **Theorem 5** (In Isabelle/HOL). *If $\mathcal{G}$ is a valid unwinding for $\mathcal{P}$, then $\mathcal{P}$ is safe.*

We model unwindings basically following the original definition [6], where nodes and transition edges are specified as a tree, whereas cover edges are given as a separate set. However, this turned out to be unwieldy in the formalization. Our formalization is not restricted to trees (since being a tree or not is irrelevant for soundness), each node has either exactly one cover edge or a list of transition edges, nodes are modeled by some parametric type α, and the location and the invariant of a node are modeled by a function from α to $\mathcal{L} \times \Lambda_{\mathsf{bool}}(\mathcal{V})$.

Whereas Theorem 5 is a statement about the soundness of the technique in [6], we also implemented an executable certifier for safety proofs. It demands that $\mathcal{G}$ is provided in the certificate and validates the various entailments $\phi \models \chi$ within Definition 3. To this end, the certificate also has to contain hints on how to prove each of the entailments.

Most of the formalization of the certifier is generic, i.e., it is not restricted to *integer* transition systems. However, at the moment we only have a formalized entailment checker for linear integer arithmetic. Hence, we arrive at the following soundness theorem for CeTA's safety certifier.

▶ **Theorem 6** (In Isabelle/HOL). *Let $\mathcal{P}$ be an LTS over linear integer formulas. If the certifier accepts a certificate for $\mathcal{P}$, then $\mathcal{P}$ is safe.*

Under `http://cl-informatik.uibk.ac.at/~thiemann/ceta_safety.tgz` we provide a version of CeTA for validating safety proofs. The archive also contains a small hand-written safety proof following Figures 1 and 2 of [6], as well as an automatically generated safety proof by T2 – similar to Figure 4. To this end, we employed the T2 version that is available at `https://github.com/mmjb/T2/tree/cert`.

4 Towards Certifying Termination Proofs

▶ **Definition 7.** An LTS $\mathcal{P}$ is *terminating* if there exist no infinite transition sequence starting from the initial location: $(\ell_0, \alpha_0) \to_{\mathcal{P}} (\ell_1, \alpha_1) \to_{\mathcal{P}} \cdots$.

The cooperation graph technique [1] reduces termination proving to safety checking, and incorporates some insights from termination proving for TRSs, such as deletion of rules in the dependency pair setting.

For certification purpose, it turns out that the full cooperation graph technique need not be formalized; instead, the following notion suffices.

▶ **Definition 8** (Cooperation Problem)**.** We define a set of fresh locations $\mathcal{L}^\sharp = \{\ell^\sharp \mid \ell \in \mathcal{L}\}$. A *cooperation problem* $\mathcal{Q}$ is an LTS over $\mathcal{L} \cup \mathcal{L}^\sharp$ such that every transition rule in $\mathcal{Q}$ is either of form $\ell \xrightarrow{\phi} r$, $\ell \xrightarrow{\phi} r^\sharp$, or $\ell^\sharp \xrightarrow{\phi} r^\sharp$ for $\ell, r \in \mathcal{L}$. We say the cooperation problem is *terminating* if there exists no infinite sequence of form

$$(\ell_0, \alpha_0) \to_{\mathcal{Q}} \cdots \to_{\mathcal{Q}} (\ell_n, \alpha_n) \to_{\mathcal{Q}} (\ell_n^\sharp, \alpha_n) \to_{\mathcal{Q}} (\ell_{n+1}^\sharp, \alpha_{n+1}) \to_{\mathcal{Q}} \cdots$$

where each transition rule $\ell^\sharp \xrightarrow{\phi} r^\sharp$ used after the n-th step must be used infinitely often.

▶ **Theorem 9** (In Isabelle/HOL)**.** *Let $\mathcal{P}$ be an LTS and $\mathcal{Q}$ a cooperation problem such that*

- *for each location $\ell \in \mathcal{L}$, there is a transition rule $\ell \xrightarrow{\phi} \ell^\sharp \in \mathcal{Q}$ where ϕ is a conjunction of identities $x' = x$; and*
- *for each transition rule $\ell \xrightarrow{\phi} r \in \mathcal{P}$, there exist $\ell \xrightarrow{\phi} r \in \mathcal{Q}$ and $\ell^\sharp \xrightarrow{\phi} r^\sharp \in \mathcal{Q}$.*

Then, if $\mathcal{Q}$ is terminating w.r.t. Definition 8, then $\mathcal{P}$ is terminating w.r.t. Definition 7.

The crucial advantage of considering cooperation problems is that one can remove a transition rule $\ell^\sharp \xrightarrow{\phi} r^\sharp$ without affecting termination if the rule cannot be applied infinitely often. This is unsound for original LTSs; consider e.g. a nonterminating LTS consisting of only the two transition rules $\ell_0 \xrightarrow{\mathsf{true}} \ell_1$ and $\ell_1 \xrightarrow{\mathsf{true}} \ell_1$. Clearly, the first transition rule can be applied only once. Nevertheless, if one removes it then ℓ_1 becomes unreachable, and the resulting LTS is terminating.

We are now able to formalize the main termination procedure for cooperation problems, namely transition rule removal with invariants and rank functions. As a first step we only formalize it with $\mathbb{Z}$ as target domain.

▶ **Theorem 10** (In Isabelle/HOL)**.** *Let $\mathcal{P}$ be a cooperation problem over $\mathcal{L} \cup \mathcal{L}^\sharp$. Let $I : \mathcal{L}^\sharp \to \Lambda_{\mathsf{bool}}(\mathcal{V})$ map locations to* invariants*, $R : \mathcal{L}^\sharp \to \Lambda_{\mathsf{int}}(\mathcal{V})$ map locations to* rank functions *(encoded as (linear) integer expressions), $\mathcal{D}$ a set of transition rules and $b \in \mathbb{Z}$ such that*

- *The invariants specified by I are valid, i.e., $\beta \models I(\ell^\sharp)$ whenever $(\ell_0, \alpha) \to_{\mathcal{P}}^* (\ell^\sharp, \beta)$.*
- *For every $\ell^\sharp \xrightarrow{\phi} r^\sharp \in \mathcal{P}$, $I(\ell^\sharp) \wedge \phi \models R(\ell^\sharp) \geq R(r^\sharp)'$.*
- *Transition rules in $\mathcal{D}$ are of the form $\ell^\sharp \xrightarrow{\phi} r^\sharp$ and $I(\ell^\sharp) \wedge \phi \models R(\ell^\sharp) > R(r^\sharp)' \wedge R(\ell^\sharp) \geq b$.*

Then the termination of $\mathcal{P}\setminus\mathcal{D}$ implies the termination of $\mathcal{P}$ (w.r.t. Definition 8), where $\mathcal{P}\setminus\mathcal{D}$ denotes the cooperation problem obtained by removing of all transitions rules in $\mathcal{D}$ from $\mathcal{P}$.

▶ **Example 11.** Consider again the LTS $\mathcal{P}$ of Example 4. We first construct the initial cooperation problem which yields copied versions $\tau_1^\sharp, \tau_2^\sharp, \tau_3^\sharp$ of transition rules τ_1, τ_2, τ_3.

One usually would delete transition $\tau_2^\sharp$ via an SCC-analysis, but this can also be mimicked by Theorem 10: choose $R(\ell_0^\sharp) = 1$, $R(\ell_1^\sharp) = 0$, $b = 0$, and $I(\ell_i^\sharp) = \mathsf{true}$.

Transition $\tau_1^\sharp$ corresponding to the first while loop can also be deleted without invariants: choose $R(\ell_0^\sharp) = y$, $R(\ell_1^\sharp) = 0$, $b = -3$, and $I(\ell_i^\sharp) = \mathsf{true}$.

Finally, transition $\tau_3^\sharp$ demands the invariant from Example 4. We choose $R(\ell_1^\sharp) = x$, $b = 0$, and $I(\ell_1^\sharp) = y < 0$. Hence, besides the invariant one has to validate the entailment

$$\underbrace{y < 0}_{I(\ell_1^\sharp)} \wedge \underbrace{x \geq 0 \wedge x' = x + y \wedge y' = y - 1}_{\phi} \models \underbrace{x}_{R(\ell_1^\sharp)} > \underbrace{x'}_{R(\ell_1^\sharp)'} \wedge \underbrace{x}_{R(\ell_1^\sharp)} \geq \underbrace{0}_{b}.$$

5 Conclusion

Certification establishes trustworthiness of termination and safety proofs of integer transition systems. In our formally verified certifier CeTA, we implemented a mode that certifies safety proofs for ITSs.

We moreover made a fundamental step towards certifying termination proofs for ITSs. Future work includes deciding a machine-readable format of ITS termination proofs, building a parser, and establishing a connection to the safety proof certifier. The last is required in order to certify termination proofs that use invariants. A valid unwinding $\mathcal{G}$ for an LTS $\mathcal{P}$ is used to get the invariants mentioned in Theorem 10, which is the main connection that is still missing for the termination certifier.

References

1 Marc Brockschmidt, Byron Cook, and Carsten Fuhs. Better termination proving through cooperation. In *CAV'13*.

2 Marc Brockschmidt, Byron Cook, Samin Ishtiaq, Heidy Khlaaf, and Nir Piterman. T2: Temporal property verification. In *TACAS'16*.

3 Byron Cook, Andreas Podelski, and Andrey Rybalchenko. Termination proofs for systems code. In *PLDI'06*.

4 Stephan Falke, Deepak Kapur, and Carsten Sinz. Termination analysis of C programs using compiler intermediate languages. In *RTA'11*.

5 Daniel Larraz, Albert Oliveras, Enric Rodríguez-Carbonell, and Albert Rubio. Proving termination of imperative programs using max-SMT. In *FMCAD'13*.

6 Ken McMillan. Lazy abstraction with interpolants. In *CAV'06*.

7 Carsten Otto, Marc Brockschmidt, Christian von Essen, and Jürgen Giesl. Automated termination analysis of Java Bytecode by term rewriting. In *RTA'10*.

8 Fausto Spoto, Fred Mesnard, and Étienne Payet. A termination analyser for Java Bytecode based on path-length. *ACM Transactions on Programming Languages and Systems*, 32(3), 2010.

9 René Thiemann and Christian Sternagel. Certification of termination proofs using CeTA. In *TPHOLs'09*.

10 Caterina Urban, Arie Gurfinkel, and Temesghen Kahsai. Synthesizing ranking functions from bits and pieces. In *TACAS'16*.