A Coinductive Approach to Proving Reachability Properties in Logically Constrained Term Rewriting Systems

Ştefan Ciobâcă[1] and Dorel Lucanu[1]

Alexandru Ioan Cuza University, Romania
{stefan.ciobaca,dlucanu}@info.uaic.ro

Abstract. We introduce a sound and complete coinductive proof system for reachability properties in transition systems generated by logically constrained term rewriting rules over an order-sorted signature modulo builtins. A key feature of the calculus is a circularity proof rule, which allows to obtain finite representations of the infinite coinductive proofs.

1 Introduction

We propose a framework for specifying and proving reachability properties of systems whose behaviour is modelled using transition systems described by logically constrained term rewriting systems (LCTRSs). By reachability properties we mean that a set of target states are reached in all terminating system computations starting from a given set of initial states. We assume transition systems are generated by constrained term rewriting rules of the form

$$l \twoheadrightarrow r \texttt{ if } \phi,$$

where l and r are terms and ϕ is a logical constraint. The terms l, r may contain both uninterpreted function symbols and function symbols interpreted in a builtin model, e.g., the model of booleans and integers. The constraint ϕ is a first-order formula that limits the application of the rule and which may contain predicate symbols interpreted in the builtin model. The intuitive meaning of a constrained rule $l \twoheadrightarrow r \texttt{ if } \phi$ is that any instance of l that satisfies ϕ transitions in one step into a corresponding instance of r.

Example 1. The following set of constrained rewrite rules specifies a procedure for compositeness:

$$\begin{aligned}
&\mathit{init}(n) \twoheadrightarrow \mathit{loop}(n, 2) \texttt{ if } \top,\\
&\mathit{loop}(i \times k, i) \twoheadrightarrow \mathit{comp} \texttt{ if } k > 1,\\
&\mathit{loop}(n, i) \twoheadrightarrow \mathit{loop}(n, i + 1) \texttt{ if } \neg(\exists k.k > 1 \land n = i \times k).
\end{aligned}$$

If n is not composite, the computation of the procedure is infinite.

Given a LCTRS, which serves as a specification for a transition system, it is natural to define the notion of constrained term $\langle t \mid \phi \rangle$, where t is an ordinary term (with variables) and ϕ is a logical constraint. The intuitive meaning of such a term is the set of ground instances of t that satisfy ϕ.

Example 2. The constrained term $\langle init(n) \,|\, \exists u. 1 < u < n \wedge n \ mod \ u = 0 \rangle$ defines exactly the instances of $init(n)$ where n is composite.

A *reachability formula* is a pair of constrained terms $\langle t \,|\, \phi \rangle \Rightarrow \langle t' \,|\, \phi' \rangle$. The intuitive meaning of a reachability formula is that any instance of $\langle t \,|\, \phi \rangle$ reaches, along all terminating paths of the transition system, an instance of $\langle t' \,|\, \phi' \rangle$ that agrees with $\langle t \,|\, \phi \rangle$ on the set of shared variables.

Example 3. The reachability formula

$$\langle init(n) \,|\, \exists u. 1 < u < n \wedge n \ mod \ u = 0 \rangle \Rightarrow \langle comp \,|\, \top \rangle$$

captures a functional specification for the algorithm described in Example 1: each terminating computation starting from a state in which n is composite reaches the state *comp*. Computations that start with a negative number (composite or not) are infinite and therefore vacuously covered by the specification above.

We propose an effective proof system that, given a LCTRS, proves valid reachability formulas such as the one above, assuming an oracle that solves logical constraints. In practice, we use an SMT solver instead of the oracle.

Contributions 1. As computations can be finite or infinite, an inductive approach for reachability is not practically possible. In Section 2, we propose a coinductive approach for specifying transition systems, which is an elegant way to look at reachability, but also essential in handling both finite and infinite executions. 2. We formalize the semantics of LCTRSs as a reduction relation over a particular model that combines order-sorted terms with builtin elements such as integers, booleans, arrays, etc. The new approach, introduced in Section 3, is simpler than the usual semantics for constrained term rewriting systems [18,20,19,14], but it also lifts several technical restrictions that are important for our case studies. 3. We introduce a sound and complete coinductive proof system for deriving valid reachability formulas for transition systems specified by a LCTRS. We present our proof system in two steps: in the first step, we provide a three-rule proof system (Figure 1) for *symbolic execution of constrained terms*. When interpreting the proof system coinductively, its proof trees can be finite or infinite. The finite proof trees correspond to reachability formulas $\langle t \,|\, \phi \rangle \Rightarrow \langle t' \,|\, \phi' \rangle$ where there is a bounded number of symbolic steps between $\langle t \,|\, \phi \rangle$ and $\langle t' \,|\, \phi' \rangle$. The infinite proof trees correspond to proofs of reachability formulas $\langle t \,|\, \phi \rangle \Rightarrow \langle t' \,|\, \phi' \rangle$ that hold for an unbounded number of symbolic steps between $\langle t \,|\, \phi \rangle$ and $\langle t' \,|\, \phi' \rangle$ (obtained, e.g., by unrolling loops). Symbolic execution has similarities to narrowing, but unlike narrowing, where each step computes a possible successor, symbolic execution must consider *all* successors of a state at the same time. 4. The infinite proof trees above cannot be obtained in finite time in practice. In order to derive reachability formulas that require an unbounded number of symbolic steps in finite time, we introduce a fourth proof rule to the system that we call *circularity*. The circularity proof rule can be used to compress infinite proof trees into finite proof trees. The intuition is to use as axioms the goals that are to be proven, when they satisfy a *guardedness* condition. This compression of infinite coinductive trees into finite proof trees via the guardedness condition nicely complements our coinductive approach. This separation between symbolic execution and circularity answers an open question in [21]. 5. We introduce the `RMT` tool, an implementation of the proof system that validates our approach on a number of examples. `RMT` uses an SMT solver to discharge logical constraints. The tool is expressive enough for specifying various transition systems, including operational semantics of programming languages, and proving reachability properties of practical interest and is intended to be the starting point of a library for rewriting modulo builtins, which could have more applications.

Related Work A number of approaches [1,2,14,25,29] to combining rewriting and SMT solving have appeared lately. The rewrite tool Maude [12] has been extended with SMT solving in [25] in order to enable the analysis of open systems. A method for proving invariants based on an encoding into reachability properties is presented in [29]. Both approaches above are restricted to *topmost rewrite theories*. While almost any theory can be written as a topmost theory [22], the encoding can significantly increase the number of transitions, which raises performance concerns. Our definition for constrained term is a generalization of that of constructor constrained pattern used in [29]. In particular [29] does not allow for quantifiers in constraints, but quantifiers are critical to obtaining a complete proof system, as witnessed by their use in the subsumption rule in our proof system ([subs], Figure 1). The approach without quantifiers is therefore not sufficient to prove reachabilities in a general setting.

A calculus for reachability properties in a formalism similar to LCTRSs is given in [1]. However, the notion of reachability in [1] is different from ours: while we show reachability along *all terminating paths of the computation*, [1] solves reachability properties of the form $\exists \widetilde{x}.t(\widetilde{x}) \to^* t'(\widetilde{x})$ (i.e. does there exists an instance of t that reaches, along some path, an instance of t').

Work on constrained term rewriting systems appeared in [20,19,18,14]. In contrast to this approach to constrained rewriting, our semantics is simpler (it does not require two reduction relations), it does not have restrictions on the terms l, r in a rule $l \twoheadrightarrow r \texttt{ if } \phi$ and the constraint is an arbitrary first-order formula ϕ, possibly with quantifiers, which are crucial to obtain symbolic execution in its full generality. Constrained terms are generalized to guarded terms in [2], in order to reduce the state space.

Reachability in rewriting is explored in depth in [13]. The work by Kirchner and others [17] is the first to propose the use of rewriting with symbolic constraints for deduction. Subsequent work [25,20,14] extends and unifies previous approaches to rewriting with constraints. The related work section in [25] includes a comprehensive account of literature related to rewriting modulo constraints.

Our previous work [10,21] on proving program correctness was in the context of the K framework [27]. K, developed by Roşu and others, implements semantics-based program verifiers [11] for any language that can be specified by a rewriting-based operational semantics, such as C [15], Java [4] and JavaScript [23]. Our formalism is not more expressive than that of reachability logic [10] for proving partial correctness of programs in a language-independent manner, but it does have several advantages. Firstly, we make a clear separation between *rewrite rules* (used to define transition systems), for which it makes no sense to have constraints on both the lhs and the rhs, and *reachability formulas* (used to specify reachability properties), for which there can be constraints on both the lhs and the rhs. We provide clear semantics of both syntactic constructs above, which makes it unnecessary to check well-definedness of the underlying rewrite system, as required in [10]. Additionally, this separation, which we see as a contribution, makes it easy to get rid of the top-most restriction in previous approaches. Another advantage is that the proposed proof system is very easy to automate, while being sufficiently expressive to specify real-world applications. Additionally, we work in the more general setting of LCTRSs, not just language semantics, which enlarges the possible set of applications of the technique. We also have several major technical improvements compared to [21], where the proof system is restricted to the cases where unification can be reduced to matching and topmost rewriting. The totality property required for languages specifications, which was quite restrictive, was replaced by a local property in proof rules and all restrictions needed to reduce unification to matching were removed.

In contrast to the work on partial correctness in [11], the approach on reachability discussed here is meant for any LCTRS, not just operational semantics. The algorithm in [11] contains a small source of incompleteness, as when proving a reachability property it is either discharged completely through implication or through circularities/rewrite rules. We allow a reachability rule to be discharged partially by subsumption and partially by other means. Constrained terms are a fragment of Matching Logic (see [26]), where no distinction is made between terms and constraints. Coinduction and circular or cyclic proofs have been proposed in other contexts. For example, circular proof systems have been proposed for first-order logic with inductive predicates in [6] and for separation logic in [5]. In the context of interactive theorem provers, circular coinduction has been proposed as an incremental proof method for bisimulation in process calculi (see [24]). A compositional and incremental approach to coinduction that uses a semantic guardedness check instead of a syntactic check is given in [16].

Paper Structure We present coinductive definitions for execution paths and reachability predicates in Section 2. In Section 3, we introduce logically constrained term rewriting with builtins in an order-sorted setting. In Section 4, we propose a sound and complete coinductive calculus for reachability and a circularity rule for compressing infinite proof trees into finite proof trees. Section 6 discusses the implementation before concluding. The proofs can be found in [8].

2 Reachability Properties: Coinductive Definition

In this section we introduce a class of reachability properties, defined coinductively. A *state predicate* is a subset of states. A *reachability property* is a pair $P \Rightarrow Q$ of state predicates. Such a reachability property is *demonically valid* iff each execution path starting from a state in P eventually reaches a state in Q, or if it is infinite. Since the set of finite and infinite executions is coinductively defined, the set of valid predicates can be defined coinductively as well. Formally, consider a transition system $(M, \rightsquigarrow)$, with $\rightsquigarrow \subseteq M \times M$. We write $\gamma \rightsquigarrow \gamma'$ for $(\gamma, \gamma') \in \rightsquigarrow$. An element $\gamma \in M$ is *irreducible* if $\gamma \not\rightsquigarrow \gamma'$ for any $\gamma' \in M$.

Definition 1 (Execution Path). *The set of* (complete) execution paths *is coinductively defined by the following rules:*

$$\frac{}{\gamma}\ \gamma \in M, \gamma \text{ irreducible} \qquad \frac{\tau}{\gamma_0 \circ \tau}\ \gamma_0 \rightsquigarrow hd(\tau)$$

where the function hd is defined by $hd(\gamma) = \gamma$ and $hd(\gamma_0 \circ \tau) = \gamma_0$.

The above definition includes both the finite execution paths ending in a irreducible state and the infinite execution paths, defined as the greatest fixed point of the associated functional (see [8]).

Definition 2 (State and Reachability Predicates). *A* state predicate *is a subset $P \subseteq M$. A* reachability predicate *is a pair of state predicates $P \Rightarrow Q$. The predicate P is* runnable *if $P \neq \emptyset$ and for all $\gamma \in P$ there is $\gamma' \in M$ s.t. $\gamma \rightsquigarrow \gamma'$.*

A *derivative* measures the sensitivity to change of a quantity. For the case of transition systems, the change of states is determined by the transition relation.

Definition 3 (Derivative of a State Predicate). *The derivative of a state predicate P is the state predicate $\partial(P) = \{\gamma' \mid \gamma \rightsquigarrow \gamma' \text{ for some } \gamma \in P\}$.*

As a reachability predicate specifies reachability property of execution paths, we define when a particular execution path satisfies a reachability predicate.

Definition 4 (Satisfaction of a Reachability Predicate). *An execution path τ* satisfies *a reachability predicate $P \Rightarrow Q$, written $\tau \models^\forall P \Rightarrow Q$, iff $\langle \tau, P \Rightarrow Q\rangle \in \nu\,\widehat{\mathsf{EPSRP}}$, where EPSRP consists of the following rules:*

$$\frac{}{\langle \tau, P \Rightarrow Q\rangle}\ hd(\tau) \in P \cap Q \qquad \frac{\langle \tau, \partial(P) \Rightarrow Q\rangle}{\langle \gamma_0 \circ \tau, P \Rightarrow Q\rangle}\ \gamma_0 \in P, \gamma_0 \rightsquigarrow hd(\tau).$$

The notation $\widehat{\mathsf{EPSRP}}$ stands for the functional of EPSRP and $\nu\widehat{\mathsf{EPSRP}}$ stands for its greatest fixed point (see [8]). We coinductively define the set of *demonically valid reachability predicates* over $(M, \rightsquigarrow)$. This allows to use coinductive proof techniques to prove validity of reachability predicates.

Definition 5 (Valid Reachability Predicates, Coinductively). *We say that $P \Rightarrow Q$ is* demonically valid*, and we write*

$$(M, \rightsquigarrow) \models^\forall P \Rightarrow Q,$$

iff $P \Rightarrow Q \in \nu\,\widehat{\mathsf{DVP}}$, where DVP consists of the following rules:

$$[\![\mathsf{Subsumption}]\!]\ \frac{}{P \Rightarrow Q}\ P \subseteq Q \qquad [\![\mathsf{Step}]\!]\ \frac{\partial(P \setminus Q) \Rightarrow Q}{P \Rightarrow Q}\ P \setminus Q \text{ runnable.}$$

The condition $P \setminus Q$ *runnable* in the second rule is essential to avoid the cases where execution is stuck. These blocking states have no successor in $\partial(P \setminus Q)$ and, in the absense of the condition, we would wrongly conclude that they satisfy $P \Rightarrow Q$. The terminating executions are captured by $[\![\mathsf{Subsumption}]\!]$.

The following proposition justifies our definition of demonically valid reachability predicates.

Proposition 1. *Let $P \Rightarrow Q$ be a reachability predicate. We have $(M, \rightsquigarrow) \models^{\forall} P \Rightarrow Q$ iff any execution path τ* starting from *P ($hd(\tau) \in P$) satisfies $P \Rightarrow Q$.*

3 Logically Constrained Term Rewriting Systems

In this section we introduce our formalism for LCTRSs. We interpret LCTRSs in a model combining order-sorted terms with builtins such as integers, booleans, etc. Logical constraints are first-order formulas interpreted over the fixed model.

We assume a *builtin model* M^{b} for a many-sorted *builtin signature* $\Sigma^{\mathsf{b}} = (S^{\mathsf{b}}, F^{\mathsf{b}})$, where S^{b} is a set of *builtin sorts* that includes at least the sort *Bool* and F^{b} is the S^{b}-sorted set of *builtin function symbols.* We assume that the set interpreting the sort *Bool* in the model M^{b} is $M^{\mathsf{b}}_{Bool} = \{\top, \bot\}$. We use the standard notation M_o for the interpretation of the sort/symbol o in the model M. The set CF^{b}, defined as the set of (many-sorted) first-order formulas with equality over the signature Σ^{b}, is the set of *builtin constraint formulas.* Functions returning *Bool* play the role of predicates and terms of sort *Bool* are atomic formulas. We will assume that the builtin constraint formulas can be decided by an oracle (implemented as an SMT solver).

A *signature modulo builtins* is an order-sorted signature $\Sigma = (S, \leq, F)$ that includes Σ^{b} as a subsignature and such that the only builtin constants in Σ are elements of the builtin model ($\{c \mid c \in F_{\varepsilon,s}, s \in S^{\mathsf{b}}\} = M^{\mathsf{b}}_s$) – therefore the signature might be infinite. By $F_{w,s}$ we denoted the set of function symbols of arity w and result sort s. Σ^{b} is called the *builtin subsignature* of Σ and $\Sigma^{\mathsf{c}} = (S, \leq, (F \setminus F^{\mathsf{b}}) \cup \bigcup_{s \in S^{\mathsf{b}}} F_{\varepsilon,s})$ the *constructor subsignature* of Σ. We let $\mathcal{X}$ be an S-sorted set of variables.

We extend the builtin model M^{b} to an $(S, \leq, F)$-model M^{Σ} defined as follows: • $M^{\Sigma}_s = T_{\Sigma^{\mathsf{c}},s}$, for each $s \in S \setminus S^{\mathsf{b}}$ (M^{Σ}_s is the set of ground constructor terms of sort s, i.e. terms built from constructors applied to builtin elements); • $M^{\Sigma}_f = M^{\mathsf{b}}_f$ for each builtin function symbol $f \in F^{\mathsf{b}}$; • M^{Σ}_f is the term constructor $M^{\Sigma}_f(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$, for each non-builtin function symbol $f \in F \setminus F^{\mathsf{b}}$. By fixing the interpretation of the non-builtin function symbols, we can reduce constraint formulas to built-in constraint formulas by relying on an unification algorithm described in detail in [7]. We also make the standard assumption that $M_s \neq \emptyset$ for any $s \in S$.

Example 4. Let $\Sigma^{\mathsf{b}} = (S^{\mathsf{b}}, F^{\mathsf{b}})$, where $S^{\mathsf{b}} = \{Int, Bool\}$ and F^{b} include the usual operators over *Bool*eans $(\vee, \wedge, \ldots)$ and over the *Int*egers $(+, -, \times, \ldots)$. The builtin model M^{b} interprets the above sorts and operations as expected.

We consider the signature modulo builtins $\Sigma = (S, \leq, F)$, where the set of sorts $S = \{\mathit{Cfg}, \mathit{Int}, \mathit{Bool}\}$ consists of the builtin sorts and an additional sort *Cfg*, where the subsorting relation $\leq \subseteq S \times S = \emptyset$ is empty, and where the set of function symbols F includes, in addition to the builtin symbols in F^{b}, the following function symbols: $\mathit{init} : \mathit{Int} \to \mathit{Cfg}, \mathit{loop} : \mathit{Int} \times \mathit{Int} \to \mathit{Cfg}, \mathit{comp} : \mathit{Cfg}$. We have that $M^{\Sigma}_{\mathit{Cfg}} = \{\mathit{init}(i) \mid i \in \mathbb{Z}\} \cup \{\mathit{loop}(i, j) \mid i, j \in \mathbb{Z}\} \cup \{\mathit{comp}\}$.

The set CF of *constraint formulas* is the set of first-order formulas with equality over the signature Σ. The subset of the builtin constraint formulas is denoted by CF^b. Let $\mathit{var}(\phi)$ denote the set of variables freely occurring in ϕ. We write $M^{\Sigma}, \alpha \vDash \phi$ when the formula ϕ is satisfied by the model M^{Σ} with a valuation $\alpha : X \to M^{\Sigma}$.

Example 5. The constraint formula $\phi \triangleq \exists u.1 < u < n \land n \mathbin{\mathit{mod}} u = 0$ is satisfied by the model M^{Σ} defined in Example 4 and any valuation α such that $\alpha(n)$ is a composite number.

Definition 6 (Constrained Terms). *A* constrained term *φ of sort $s \in S$ is a pair $\langle t \mid \phi \rangle$, where $t \in T_{\Sigma,s}(\mathcal{X})$ and $\phi \in$ CF.*

Example 6. Continuing the previous example, the following is a constrained term: $\langle \mathit{init}(n) \mid \exists u.1 < u < n \land n \mathbin{\mathit{mod}} u = 0 \rangle$.

We consistently use φ for constrained terms and ϕ for constraint formulas.

Definition 7 (Valuation Semantics of Constraints). *The* valuation semantics *of a constraint ϕ is the set $\llbracket \phi \rrbracket \triangleq \{\alpha : X \to M^{\Sigma} \mid M^{\Sigma}, \alpha \vDash \phi\}$.*

Example 7. Continuing the previous example, we have that

$$\llbracket \exists u.1 < u < n \land n \mathbin{\mathit{mod}} u = 0 \rrbracket = \{\alpha : X \to M^{\Sigma} \mid \alpha(n) \text{ is composite}\}.$$

Definition 8 (State Predicate Semantics of Constrained Terms). *The* state predicate semantics *of a constrained term $\langle t \mid \phi \rangle$ is the set*

$$\llbracket \langle t \mid \phi \rangle \rrbracket \triangleq \{\alpha(t) \mid \alpha \in \llbracket \phi \rrbracket\}.$$

Example 8. Continuing the previous example, we have that

$$\llbracket \langle \mathit{init}(n) \mid \exists u.1 < u < n \land n \mathbin{\mathit{mod}} u = 0 \rangle \rrbracket = \{\mathit{init}(n) \mid n \text{ is composite}\}.$$

We now introduce our formalism for logically constrained term rewriting systems. Syntactically, a rewrite rule consists of two terms (the left hand side and respectively the right hand side), together with a constraint formula. As the two terms could *share some variables*, these shared variables should be instantiated consistently in the semantics:

Definition 9 (LCTRS). *A* logically constrained rewrite rule *is a tuple (l, r, ϕ), often written as $l \twoheadrightarrow r$ `if` ϕ, where l, r are terms in $T_{\Sigma}(\mathcal{X})$ having the same sort, and $\phi \in$ CF. A* logically constrained term rewriting system *$\mathcal{R}$ is a set of logically constrained rewrite rules. $\mathcal{R}$ defines an order-sorted* transition relation *$\rightsquigarrow_{\mathcal{R}}$ on M^{Σ} as follows: $t \rightsquigarrow_{\mathcal{R}} t'$ iff there exist a rule $l \twoheadrightarrow r$ `if` ϕ in $\mathcal{R}$, a context $c[\cdot]$, and a valuation $\alpha : X \to M^{\Sigma}$ such that $t = \alpha(c[l])$, $t' = \alpha(c[r])$ and $M^{\Sigma}, \alpha \vDash \phi$.*

Example 9. We recall the LCTRS given in the introduction:

$$\mathcal{R} = \left\{ \begin{array}{l} \mathit{init}(n) \twoheadrightarrow \mathit{loop}(n, 2) \;\texttt{if}\; \top, \\ \mathit{loop}(i \times k, i) \twoheadrightarrow \mathit{comp} \;\texttt{if}\; k > 1, \\ \mathit{loop}(n, i) \twoheadrightarrow \mathit{loop}(n, i + 1) \;\texttt{if}\; \neg(\exists k.k > 1 \land n = i \times k) \end{array} \right\}.$$

A LCTRS $\mathcal{R}$ defines a sort-indexed transition system $(M^{\Sigma}, \rightsquigarrow_{\mathcal{R}})$. As each constrained term φ defines a state predicate $[\![\varphi]\!]$, it is natural to specify reachability predicates as pairs of constrained terms sharing a subset of variables. The shared variables must be instantiated in the same way by the execution paths connecting states specified by the two constrained terms.

Definition 10 (Reachability Properties of LCTRSs). *A* reachability formula $\varphi \Rightarrow \varphi'$ *is a pair of constrained terms, which may share variables. We say that a LCTRS* $\mathcal{R}$ demonically satisfies $\varphi \Rightarrow \varphi'$, *written*

$$\mathcal{R} \models^{\forall} \varphi \Rightarrow \varphi',$$

iff $(M^{\Sigma}, \rightsquigarrow_{\mathcal{R}}) \models^{\forall} [\![\sigma(\varphi)]\!] \Rightarrow [\![\sigma(\varphi')]\!]$ *for each* $\sigma : var(\varphi) \cap var(\varphi') \to M^{\Sigma}$.

Since the carriers sets of M^{Σ} consist of ground terms, σ is both a substitution and a valuation in the definition above. Its role is critical: to ensure that the shared variables of φ and φ' are instantiated by the same values.

Example 10. Continuing the previous example, we have that the reachability formula $\langle \mathit{init}(n) \mid \exists u.1 < u < n \land n \; mod \; u = 0\rangle \Rightarrow \langle \mathit{comp} \mid \top\rangle$ is demonically satisfied by the constrained rule system $\mathcal{R}$ defined in Example 9:

$$\mathcal{R} \models^{\forall} \langle \mathit{init}(n) \mid \exists u.1 < u < n \land n \; mod \; u = 0\rangle \Rightarrow \langle \mathit{comp} \mid \top\rangle.$$

We have checked the above reachability formula against $\mathcal{R}$ mechanically, using an implementation of the approach described in this paper.

4 Proving Reachability Properties of LCTRSs

We introduce two proof systems for proving reachability properties in transition systems specified by LCTRSs. The first proof system formalizes *symbolic execution* in a LCTRS, in the following sense: a reachability formula $\varphi \Rightarrow \varphi'$ can be proven if any execution path starting with φ either reaches a state that is an instance of φ', or is divergent. Note that this intuition holds when the proof system is interpreted coinductively, where infinite proof trees are allowed. Unfortunately, these infinite proof trees have a limited practical use because they cannot be obtained in finite time.

In order to solve this limitation, we introduce a second proof system, which contains an additional inference rule, called *circularity.* The *circularity* rule allows to use the reachability formula to be proved as an axiom. This allows to fold infinite proof trees into finite proof trees, which can be obtained in finite time. Adding the reachability formulas that are to be proved as axioms seems at first to be unsound, but it corresponds to a natural intuition: when reaching a proof obligation that we have handled before, there is no need to prove it again, because the previous reasoning can be reused (possibly leading to an infinite proof branch). However, the circularity rule must be used in a guarded fashion in order to preserve soundness. We introduce a simple criterion to select the sound proof trees.

4.1 Derivatives of Constrained Terms

Our proof system relies on the notion of derivative at the syntactic level:

Definition 11 (Derivatives of Constrained Terms). *The* set of derivatives *of a constrained term $\varphi \triangleq \langle t \,|\, \phi \rangle$ w.r.t. a rule $l \twoheadrightarrow r$ `if` ϕ_{lr} is*

$$\Delta_{l,r,\phi_{lr}}(\varphi) \triangleq \{\langle c[r] \,|\, \phi' \rangle \mid \phi' \triangleq \phi \wedge t = c[l] \wedge \phi_{lr}, \\ c[\cdot] \textit{ an appropriate context and } \phi' \textit{ is satisfiable}\}, \quad (1)$$

where the variables in $l \twoheadrightarrow r$ `if` ϕ_{lr} are renamed such that $var(l, r, \phi_{lr})$ and $var(\varphi)$ are disjoint. If $\mathcal{R}$ is a set of rules, then $\Delta_{\mathcal{R}}(\varphi) = \bigcup_{(l,r,\phi_{lr})\in\mathcal{R}} \Delta_{l,r,\phi_{lr}}(\varphi)$. A constrained term φ is $\mathcal{R}$-derivable *if $\Delta_{\mathcal{R}}(\varphi) \neq \emptyset$.*

Example 11. Continuing the previous examples, we have that

$$\Delta_{\mathcal{R}}(\langle init(n) \,|\, \exists u.1 < u < n \wedge n \; mod \; u = 0 \rangle) = \\ \{\langle loop(n, 2) \,|\, \exists u.1 < u < n \wedge n \; mod \; u = 0 \rangle\}.$$

In the above case, $\Delta_{\mathcal{R}}$ includes only the derivative computed w.r.t. the first rule in $\mathcal{R}$, because the constraints of the ones computed w.r.t. the other rules are unsatisfiable. Intuitively, the derivatives of a constrained term denote all its possible successor configurations in the transition system generated by $\mathcal{R}$.

The symbolic derivatives and the concrete ones are related as expected:

Theorem 1. *Let $\varphi \triangleq \langle t \,|\, \phi \rangle$ be a constrained term, $\mathcal{R}$ a constrained rule system, and $(M^{\Sigma}, \rightsquigarrow_{\mathcal{R}})$ the transition system defined by $\mathcal{R}$. Then $[\![\Delta_{\mathcal{R}}(\varphi)]\!] = \partial([\![\varphi]\!])$.*

Our proof systems allow to replace any reachability formula by an equivalent one. Two reachability formulas, $\varphi_1 \Rightarrow \varphi_1'$ and $\varphi_2 \Rightarrow \varphi_2'$, are *equivalent*, written $\varphi_1 \Rightarrow \varphi_1' \equiv \varphi_2 \Rightarrow \varphi_2'$, if, for all LCTRSs $\mathcal{R}$,

$$\mathcal{R} \models^{\forall} \varphi_1 \Rightarrow \varphi_1' \text{ iff } \mathcal{R} \models^{\forall} \varphi_2 \Rightarrow \varphi_2'.$$

We write $[\![\varphi]\!] \subseteq_{shared} [\![\varphi']\!]$ iff for each $\sigma : var(\varphi) \cap var(\varphi') \to M^{\Sigma}$, we have $[\![\sigma(\varphi)]\!] \subseteq [\![\sigma(\varphi')]\!]$. The next result, used in our proof system, shows that inclusion of the state predicate semantics of two constrained terms can be expressed as a constraint formula, when the shared variables are instantiated consistently.

Proposition 2. *The inclusion $[\![\langle t \,|\, \phi \rangle]\!] \subseteq_{shared} [\![\langle t' \,|\, \phi' \rangle]\!]$ holds if and only if $M^{\Sigma} \models \phi \to (\exists \widetilde{x})(t = t' \wedge \phi')$, where $\widetilde{x} \triangleq var(t', \phi') \setminus var(t, \phi)$.*

4.2 Proof System for Symbolic Execution

The first proof system, DSTEP, derives sequents of the form $\langle t_l \,|\, \phi_l \rangle \Rightarrow \langle t_r \,|\, \phi_r \rangle$. The proof system consists of three proof rules presented in Figure 1 and an implicit structural rule that allows to replace reachability formulas by equivalent

$$[\mathsf{axiom}]\ \frac{}{\langle t_l \mid \bot\rangle \Rightarrow \langle t_r \mid \phi_r\rangle}$$

$$[\mathsf{subs}]\ \frac{\langle t_l \mid \phi_l \wedge \neg(\exists \widetilde{x}.t_l = t_r \wedge \phi_r)\rangle \Rightarrow \langle t_r \mid \phi_r\rangle}{\langle t_l \mid \phi_l\rangle \Rightarrow \langle t_r \mid \phi_r\rangle}\quad \begin{array}{l}\widetilde{x} \triangleq var(t_r, \phi_r) \setminus var(t_l, \phi_l)\\ \exists \widetilde{x}.t_l = t_r \wedge \phi_r \text{ satisfiable}\end{array}$$

$$[\mathsf{der}^{\forall}]\ \frac{\langle t^j \mid \phi^j\rangle \Rightarrow \langle t_r \mid \phi_r\rangle, j \in \{1, \ldots, n\}}{\langle t_l \mid \phi_l\rangle \Rightarrow \langle t_r \mid \phi_r\rangle}\quad \begin{array}{l}\langle t_l \mid \phi_l\rangle \text{ is } \mathcal{R}\text{−derivable and}\\ \phi_l \to \bigvee_{j\in\{1,\ldots,n\}} \exists \widetilde{y}^j.\phi^j \text{ is valid}\end{array}$$

$$\text{where } \Delta_{\mathcal{R}}(\langle t_l \mid \phi_l\rangle) = \{\langle t^1 \mid \phi^1\rangle, \ldots, \langle t^n \mid \phi^n\rangle\} \text{ and } \widetilde{y}^j = var(t^j, \phi^j) \setminus var(t_l, \phi_l)$$

Fig. 1. The $\mathsf{DSTEP}(\mathcal{R})$ Proof System

reachability formulas. The instances of this implicit structural rule are not included in the proof trees. We explain the three rules in the proof system.
• The [axiom] rule discharges goals where the left hand side of the goal does not match any state. As our structural rule identifies equivalent reachability formulas, this rule can be applied to any left-hand side where the constraint is unsatisfiable (equivalent to $\bot$). This rule discharges reachability formulas where there are no execution paths starting from the left-hand side, and therefore there is no need to continue the proof process along this branch.
• The [subs] rule discharges the cases where the left-hand side is an instance of the right-hand side. The constraint $\exists \widetilde{x}.t_l = t_r \wedge \phi_r$ is true exactly when the left-hand side is an instance of the right-hand side, which is ensured by Proposition 2. The proof of the current goal continues only for the cases where the negation of this constraint holds (i.e., the cases where the left-hand side is not included in the right-hand side).
• The [der$^{\forall}$] rule allows to take a symbolic step in the left-hand side of the current goal. It computes all derivatives of the left-hand side; the proof process must continue with each such derivative. Let $\psi \triangleq \phi_l \to \bigvee\{\exists \widetilde{y}^j.\phi^j\}$ be the logical constraint that occurs in the condition of [der$^{\forall}$]. The formula ψ is valid iff, for any instance of $\langle t_l \mid \phi_l\rangle$, there is at least one rule of $\mathcal{R}$ that can be applied to it, meaning that $\mathcal{R}$ is *total for* $\langle t_l \mid \phi_l\rangle$. Summarising, the condition of [der$^{\forall}$] says that $[\![\langle t_l \mid \phi_l\rangle]\!]$ must have at least one successor and furthermore that any instance $\gamma \in [\![\langle t_l \mid \phi_l\rangle]\!]$ has a $\rightsquigarrow_{\mathcal{R}}$-successor.

The following result shows that $\mathsf{DSTEP}(\mathcal{R})$ is sound and complete, modulo an oracle for solving logical constraints.

Theorem 2. *Let $\mathcal{R}$ be a LCTRS. For any reachability formula $\varphi \Rightarrow \varphi'$, we have* $$\mathcal{R} \models^{\forall} \varphi \Rightarrow \varphi' \text{ iff } \varphi \Rightarrow \varphi' \in \nu\, \widehat{\mathsf{DSTEP}}(\mathcal{R}).$$

Example 12. Consider the LCTRS $\mathcal{R}$ defined in Example 9. The proof tree for the reachability formula $\langle init(n) \mid \psi\rangle \Rightarrow \varphi_r$, where $\psi \triangleq \exists u.1 < u < n \wedge n \ mod\ u = 0$ denotes the fact that n is composite and $\varphi_r \triangleq \langle comp \mid \top\rangle$, is infinite:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\langle \mathit{comp} \,|\, \bot \rangle \Rightarrow \varphi_r}\,[\mathsf{axiom}]}{\langle \mathit{comp} \,|\, \psi \wedge \phi_a \rangle \Rightarrow \varphi_r}\,[\mathsf{subs}] \qquad \dfrac{\dfrac{\dfrac{}{\langle \mathit{comp} \,|\, \bot \rangle \Rightarrow \varphi_r}\,[\mathsf{axiom}]}{\langle \mathit{comp} \,|\, \psi \wedge \phi_2 \wedge \phi_b \rangle \Rightarrow \varphi_r}\,[\mathsf{subs}] \quad \vdots}{\langle \mathit{loop}(n,3) \,|\, \psi \wedge \phi_2 \rangle \Rightarrow \varphi_r}\,[\mathsf{der}^\forall]}{\langle \mathit{loop}(n,2) \,|\, \psi \rangle \Rightarrow \varphi_r}\,[\mathsf{der}^\forall]}{\langle \mathit{init}(n) \,|\, \psi \rangle \Rightarrow \varphi_r}\,[\mathsf{der}^\forall]$$

The right branch of the above proof tree is infinite, and:

$$\begin{array}{ll}
\phi_2 \triangleq \neg\exists k. k > 1 \wedge n = 2 \times k & \phi_a \triangleq \mathit{loop}(n,2) = \mathit{loop}(i' \times k', i') \wedge k' > 1 \\
\phi_3 \triangleq \neg\exists k. k > 1 \wedge n = 3 \times k & \phi_b \triangleq \mathit{loop}(n,3) = \mathit{loop}(i' \times k', i') \wedge k' > 1 \\
 & \dots
\end{array}$$

Note that in the presentation of the tree above, we used the structural rule to replace reachability formulas by equivalent reachability formulas as follows:

$$\begin{array}{lcl}
\langle \mathit{comp} \,|\, \psi \wedge \phi_a \wedge \neg(\mathit{comp} = \mathit{comp} \wedge \top) \rangle \Rightarrow \varphi_r & \equiv & \langle \mathit{comp} \,|\, \bot \rangle \Rightarrow \varphi_r, \\
\langle \mathit{comp} \,|\, \psi \wedge \phi_2 \wedge \phi_b \wedge \neg(\mathit{comp} = \mathit{comp} \wedge \top) \rangle \Rightarrow \varphi_r & \equiv & \langle \mathit{comp} \,|\, \bot \rangle \Rightarrow \varphi_r, \\
\langle \mathit{loop}(n',2) \,|\, \top \wedge \mathit{init}(n') = \mathit{init}(n) \wedge \psi \rangle \Rightarrow \varphi_r & \equiv & \langle \mathit{loop}(n,2) \,|\, \psi \rangle \Rightarrow \varphi_r, \\
\langle \mathit{loop}(n', i'+1) \,|\, \psi \wedge \phi_2' \rangle \Rightarrow \varphi_r & \equiv & \langle \mathit{loop}(n,3) \,|\, \psi \wedge \phi_2 \rangle \Rightarrow \varphi_r,
\end{array}$$

where $\phi_2' \triangleq \mathit{loop}(n,2) = \mathit{loop}(n', i') \wedge \neg\exists k. k > 1 \wedge n' = i' \times k$. The ticks appear in the formulas above because, to compute derivatives, we used the following fresh instance of $\mathcal{R}$:

$$\mathcal{R} = \left\{ \begin{array}{l}
\mathit{init}(n') \twoheadrightarrow \mathit{loop}(n',2) \texttt{ if } \top, \\
\mathit{loop}(i' \times k', i') \twoheadrightarrow \mathit{comp} \texttt{ if } k' > 1, \\
\mathit{loop}(n', i') \twoheadrightarrow \mathit{loop}(n', i'+1) \texttt{ if } \neg(\exists k. k > 1 \wedge n' = i' \times k)
\end{array} \right\}.$$

4.3 Extending the Proof System with a Circularity Rule

As we said at the beginning of the section, the use of DSTEP is limited because of the infinite proof trees. The next inference rule is intended to use the initial goals as axioms to fold infinite DSTEP-proof trees into sound finite proof trees.

Definition 12 (Demonic circular coinduction). *Let G be a finite set reachability formulas. Then the set of rules $\mathsf{DCC}(\mathcal{R}, G)$ consists of $\mathsf{DSTEP}(\mathcal{R})$, together with*

$$[\mathsf{circ}]\ \dfrac{\begin{array}{l}\langle t_r^c \,|\, \phi_l \wedge \phi \wedge \phi_r^c \rangle \Rightarrow \varphi_r, \\ \langle t_l \,|\, \phi_l \wedge \neg\phi \rangle \Rightarrow \varphi_r\end{array}}{\langle t_l \,|\, \phi_l \rangle \Rightarrow \varphi_r} \quad \begin{array}{l}\phi \textit{ is } \exists \mathit{var}(t_l^c, \phi_l^c). t_l = t_l^c \wedge \phi_l^c, \\ \langle t_l^c \,|\, \phi_l^c \rangle \Rightarrow \langle t_r^c \,|\, \phi_r^c \rangle \in G\end{array}$$

where $\langle t_l^c \,|\, \phi_l^c \rangle \Rightarrow \langle t_r^c \,|\, \phi_r^c \rangle$ is a rule in G whose variables have been renamed with fresh names.

The idea is that G should be chosen conveniently so that $\mathsf{DCC}(\mathcal{R}, G)$ proves G itself. We call such goals G (that are used to prove themselves) *circularities.* The intuition behind the rule is that the formula ϕ defined in the rule holds when a circularity can be applied. In that case, it is sufficient to continue the current proof obligation from the rhs of the circularity $\langle t_r^c \,|\, \phi_r^c \wedge \phi_l \wedge \phi \rangle$. The cases when ϕ does not hold (the circularity cannot be applied) are captured by the proof obligation $\langle t_l \,|\, \phi_l \wedge \neg\phi \rangle \Rightarrow \varphi_r$.

Of course, not all proof trees under $\mathsf{DCC}(\mathcal{R}, G)$ are sound. The next two definitions identify a class of sound proof trees (cf. Theorem 3).

Definition 13. *Let PT be a proof tree of $\varphi \Rightarrow \varphi'$ under $\mathsf{DCC}(\mathcal{R}, G)$. A* [circ] *node in PT is* guarded *iff it has as ancestor a* [der$^\forall$] *node. PT is* guarded *iff all its* [circ] *nodes are guarded.*

Definition 14. *We write $(\mathcal{R}, G) \vdash^\forall \varphi \Rightarrow \varphi'$ iff there is a proof tree of $\varphi \Rightarrow \varphi'$ under $\mathsf{DCC}(\mathcal{R}, G)$ that is guarded. If F is a set of reachability formulas, we write $(\mathcal{R}, G) \vdash^\forall F$ iff $(\mathcal{R}, G) \vdash^\forall \varphi \Rightarrow \varphi'$ for all $\varphi \Rightarrow \varphi' \in F$.*

The criterion stated by Definition 13 can be easily checked in practice. The following theorem states that the guarded proof trees under DCC are sound.

Theorem 3 (Circularity Principle). *Let $\mathcal{R}$ be a constrained rule system and G a set of goals. If $(\mathcal{R}, G) \vdash^\forall G$ then $\mathcal{R} \vDash^\forall G$.*

Theorem 3 can be used by finding a set of circularities and using them in a guarded fashion to prove themselves. Then the circularity principle states that such circularities hold.

Example 13. In order to prove $\langle init(n) \mid \psi \rangle \Rightarrow \langle comp \mid \top \rangle$, we choose the following set of circularities

$$G = \left\{ \begin{array}{l} \langle init(n) \mid \psi \rangle \Rightarrow \langle comp \mid \top \rangle, \\ \langle loop(n, i) \mid 2 \leq i \wedge \exists u.i \leq u < n \wedge n \mathbin{mod} u = 0 \rangle \Rightarrow \langle comp \mid \top \rangle \end{array} \right\}.$$

The second circularity is inspired by the infinite branch of the proof tree under DSTEP. We will show that $(\mathcal{R}, G) \vdash^\forall G$, and by Theorem 3, it follows that all reachability formulas in G hold in $\mathcal{R}$.

First circularity. To obtain a proof of the circularity $\langle init(n) \mid \psi \rangle \Rightarrow \langle comp \mid \top \rangle$, we replace the infinite subtree rooted at $\langle loop(n, 2) \mid \psi \rangle \Rightarrow \varphi_r$ in Example 12 by the following finite proof tree (that uses [circ]):

$$\dfrac{\dfrac{\dfrac{}{\langle comp \mid \bot \rangle \Rightarrow \varphi_r}\,[\mathsf{axiom}]}{\langle comp \mid \psi \wedge \phi \wedge \top \rangle \Rightarrow \varphi_r}\,[\mathsf{subs}] \qquad \dfrac{}{\langle loop(n, 2) \mid \psi \wedge \neg\phi \rangle \Rightarrow \varphi_r}\,[\mathsf{axiom}]}{\langle loop(n, 2) \mid \psi \rangle \Rightarrow \varphi_r}\,[\mathsf{circ}]$$

where $\phi \triangleq \exists n', i'.loop(n, 2) = loop(n', i') \wedge 2 \leq i' \wedge \exists u.i' \leq u < n' \wedge n' \mathbin{mod} u = 0$.

Second circularity. To complete the proof of G, we have to find a finite proof tree for

$$\langle loop(n, i) \mid 2 \leq i \wedge \exists u.i \leq u < n \wedge n \mathbin{mod} u = 0 \rangle \Rightarrow \langle comp \mid \top \rangle$$

as well. This is also obtained using [circ] as follows:

$$\dfrac{\dfrac{\dfrac{}{\langle comp \mid \bot \rangle \Rightarrow \varphi_r}\,[\mathsf{axiom}]}{\langle comp \mid \psi_i \wedge \psi_a \rangle \Rightarrow \varphi_r}\,[\mathsf{subs}] \qquad \dfrac{T_1 \qquad T_2}{\langle loop(n, i+1) \mid \psi_i \wedge \psi_b \rangle \Rightarrow \varphi_r}\,[\mathsf{circ}]}{\langle loop(n, i) \mid \psi_i \rangle \Rightarrow \langle comp \mid \top \rangle}\,[\mathsf{der}^\forall]$$

where

$$\psi_a \triangleq k' > 1 \wedge \mathit{loop}(n,i) = \mathit{loop}(i' \times k', i'),$$
$$\psi_b \triangleq \neg \exists k.k > 1 \wedge n = i \times k,$$
$$\psi_i \triangleq 2 \le i \wedge \exists u.i \le u < n \wedge n \ \mathit{mod}\ u = 0.$$

The subtree

$$\frac{T_1 \qquad T_2}{\langle \mathit{loop}(n, i+1) \,|\, \psi_i \wedge \psi_b \rangle \Rightarrow \varphi_r}\ [\mathsf{circ}]$$

is:

$$\frac{\dfrac{\dfrac{}{\langle \mathit{comp} \,|\, \bot \rangle \Rightarrow \varphi_r}\ [\mathsf{axiom}]}{\langle \mathit{comp} \,|\, \psi_i \wedge \psi_b \wedge \psi_c \rangle \Rightarrow \varphi_r}\ [\mathsf{subs}] \qquad \dfrac{}{\langle \mathit{loop}(n, i+1) \,|\, \psi_i \wedge \psi_b \wedge \neg\psi_c \rangle \Rightarrow \varphi_r}\ [\mathsf{axiom}]}{\langle \mathit{loop}(n, i+1) \,|\, \psi_i \wedge \psi_b \rangle \Rightarrow \varphi_r,}\ [\mathsf{circ}]$$

where
$\psi_c \triangleq \exists n', i'.\mathit{loop}(n, i+1) = \mathit{loop}(n', i') \wedge 2 \le i' \wedge \exists u.i' \le u < n' \wedge n' \ \mathit{mod}\ u = 0$.
The constraint ψ_c holds when the circularity can be applied and therefore this branch is discharged immediately by [subs] and [axiom]. The other branch, when the circularity cannot be applied, is discharged directly by [axiom], as $\psi_i \wedge \psi_b \wedge \neg\psi_c$ is unsatisfiable (ψ_i says that n has a divisor between i and n, ψ_b says that i is not a divisor of n, and ψ_c that n has a divisor between $i+1$ and n).
Note that in both proof trees of the two circularities in G, in order to apply the [circ] rule, we used the following fresh instance of the second circularity:

$$\langle \mathit{loop}(n', i') \,|\, 2 \le i' \wedge \exists u.i' \le u < n' \wedge n' \ \mathit{mod}\ u = 0 \rangle \Rightarrow \langle \mathit{comp} \,|\, \top \rangle.$$

The proof trees for both goals (circularities) in G are guarded. We have shown therefore that $(\mathcal{R}, G) \vdash^{\forall} G$. By the Circularity Principle (Theorem 3), we obtain that $\mathcal{R} \vDash^{\forall} G$ and therefore

$$\mathcal{R} \vDash^{\forall} \left\{ \begin{array}{l} \langle \mathit{init}(n) \,|\, \exists u.1 < u < n \wedge n \ \mathit{mod}\ u = 0) \rangle \Rightarrow \langle \mathit{comp} \,|\, \top \rangle, \\ \langle \mathit{loop}(n, i) \,|\, 2 \le i \wedge \exists u.i \le u < n \wedge n \ \mathit{mod}\ u = 0 \rangle \Rightarrow \langle \mathit{comp} \,|\, \top \rangle \end{array} \right\}$$

which includes what we wanted to show of our transition system defined $\mathcal{R}$ in the running example.

5 Implementation

We have implemented the proof system for reachability in a tool called `RMT` (for **r**ewriting **m**odulo **t**heories). `RMT` is open source and can be obtained from

`http://github.com/ciobaca/rmt/`.

To prove a reachability property, the `RMT` tool performs a bounded search in the proof system given above. The bounds can be set by the user. We have also tested the tool on reachability problems where we do not use strong enough circularities. In these cases, the tool will not find proofs. A difficulty that appears when a proof fails, difficulty shared by all deductive approaches to correctness, is that it is not known is the specification is wrong or if the circularities are not strong enough. Often, by analyzing the failed proof attempt, the user may have the chance to find a hint for the missing circularities, if any. In addition, proofs might also fail because of the incompleteness of the SMT solver. In addition to the running example, we have used `RMT` on a number of examples, summarized in the table below:

LCTRS	Reachability Property
Computation of $1 + \ldots + n$	Result is $n * (n + 1)/2$
Comp. of $gcd(u, v)$ by *rptd. subtractions*	Result matches builtin *gcd* function
Comp. of $gcd(u, v)$ by *rptd. divisions*	Result matches builtin *gcd* function
Mult. of two naturals by *rptd. additions*	Result matches builtin $\times$ function
Comp. of $1^2 + \ldots + n^2$	Result is $n(n + 1)(2n + 1)/6$
Comp. of $1^2 + \ldots + n^2$ *w/out multiplications*	Result is $n(n + 1)(2n + 1)/6$
Semantics of an IMPerative language	Program computing $1 + \ldots + n$ is correct
Semantics of a FUNctional language	Program computing $1 + \ldots + n$ is correct
Semantics of a FUNctional language	Program computing $1^2 + \ldots + n^2$ is correct

Implementation details. `RMT` contains roughly 5000 lines of code, including comments and blank lines. `RMT` depends only on the standard C++ libraries and it can be compiled by any relatively modern C++ compiler out of the box. At the heart of `RMT` is a hierarchy of classes for representing variables, function symbols and terms. Terms are stored in DAG format, with maximum structure sharing. The `RMT` tool relies on an external SMT solver to check satisfiability of constraints. By default, the only dependency is the `Z3` SMT solver, which should be installed and its binary should be in the system path. A compile time switch allows to use any other SMT solver that supports the SMTLIB interface, such as CVC4 [3]. In order to reduce constraints over the full signature to constraints over the builtin signature, `RMT` uses a unification modulo builtins algorithm (see [7]), which transforms any predicate $t_1 = t_2$ (where the terms t_1, t_2 can possibly contain constructor symbols) into a set of builtin constraints.

6 Conclusion and Future Work

We introduced a coinduction based method for proving reachability properties of logically constrained term rewriting systems. We use a coinductive definition of transition systems that unifies the handling of finite and infinite executions. We propose two proof systems for the problem above. The first one formalizes symbolic execution in LCTRSs coinductively, with possibly infinite proof trees. This proof system is complete, but its infinite proof trees cannot be used in practice as proofs. In the second proof system we add to symbolic execution a circularity proof rule, which allows to transform infinite proof trees into finite trees. It is not always possible to find finite proof trees, and we conjecture that establishing a given reachability property is higher up in the arithmetic hierarchy.

We also proposed a semantics for logically constrained term rewriting systems as transition systems over a model combining order-sorted terms with builtin elements such as booleans, integers, etc. The proposed semantics has the advantage of being simpler than the usual semantics of LCTRSs defined in [20], which requires two reduction relations (one for rewriting and one for computing). The approach proposed here also removes some technical constraints such as variable inclusion of the rhs in the lhs, which is important in modelling open systems, where the result of a transition is non-deterministically chosen by the environment. In addition, working in an order-sorted setting is indispensable in order to model easily the semantics of programming languages.

In fact, proving program properties, like correctness and equivalence, is one application of our method. A tool such as C2LCTRS (`http://www.trs.cm.is.nagoya-u.ac.jp/c2lctrs/`) can be used to convert the semantics of a C program into a LCTRS and then RMT can prove reachability properties of the C program. Additionally, the operational semantics of any language can be encoded as a LCTRS [28] and then program correctness is reducible to a particular reachability formula. But our approach is not limited to programs, as any system that can be modelled as a LCTRS is also amenable to our approach. We define reachability in the sense of partial correctness (i.e., nonterminating executions vacuously satisfy any reachability property). Termination should be established in some other way [18], as it is an orthogonal concern. Our approach to reachability and LCTRSs extends to working modulo AC (or more generally, modulo any set of equations E), but we have not formally presented this to preserve brevity and simplicity. For future work, we would like to test our approach on other interesting problems that arrise in various domains. In particular, it would be interesting to extend our approach to reachability in the context of program equivalence [9]. An interesting challenge is to add defined operations to the algebra underlying the constrained term rewriting systems, which would allow a user to define their own functions, which are not necessarily builtin.

Acknowledgements. We thank the anonymous reviewers for their valuable suggestions. This work was supported by a grant of the Romanian National Authority for Scientific Research and Innovation, CNCS/CCCDI - UEFISICDI, project number PN-III-P2-2.1-BG-2016-0394, within PNCDI III.

References

1. Luis Aguirre, Narciso Martí-Oliet, Miguel Palomino, and Isabel Pita. Conditional Narrowing Modulo SMT and Axioms. In *PPDP 2017*, pages 17–28.
2. Kyungmin Bae and Camilo Rocha. Guarded Terms for Rewriting Modulo SMT. In *FACS 2017*, pages 78–97.
3. Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In *CAV 2011*, pages 171–177.
4. Denis Bogdănaş and Grigore Roşu. K-Java: A Complete Semantics of Java. In *POPL 2015*, pages 445–456.
5. James Brotherston, Nikos Gorogiannis, and Rasmus L. Petersen. A generic cyclic theorem prover. In *APLAS 2012*, pages 350–367.
6. James Brotherston and Alex Simpson. Sequent calculi for induction and infinite descent. *Journal of Logic and Computation*, 21(6):1177–1216, 2011.
7. Ştefan Ciobâcă, Andrei Arusoaie, and Dorel Lucanu. Unification Modulo Builtins. In *WoLLIC 2018.* (to appear).

8. Ştefan Ciobâcă and Dorel Lucanu. A Coinductive Approach to Proving Reachability Properties in Logically Constrained Term Rewriting Systems, 2018. arXiv:1804.08308.
9. Ştefan Ciobâcă, Dorel Lucanu, Vlad Rusu, and Grigore Roşu. A language-independent proof system for full program equivalence. *Formal Asp. Comput.*, 28(3):469–497, 2016.
10. Andrei Ştefănescu, Ştefan Ciobâcă, Radu Mereuţă, Brandon M. Moore, Traian Florin Şerbănuţă, and Grigore Roşu. All-Path Reachability Logic. In *RTA-TLCA 2014*, pages 425–440.
11. Andrei Ştefănescu, Daejun Park, Shijiao Yuwen, Yilong Li, and Grigore Roşu. Semantics-Based Program Verifiers for All Languages. In *OOPSLA 2016*, pages 74–91.
12. Francisco Durán, Steven Eker, Santiago Escobar, Narciso Martí-Oliet, José Meseguer, and Carolyn L. Talcott. Built-in Variant Generation and Unification, and Their Applications in Maude 2.7. In *IJCAR 2016*, pages 183–192.
13. Santiago Escobar, José Meseguer, and Prasanna Thati. Narrowing and Rewriting Logic: from Foundations to Applications. *ENTCS*, 177:5 – 33, 2007.
14. Carsten Fuhs, Cynthia Kop, and Naoki Nishida. Verifying procedural programs via constrained rewriting induction. *ACM TOCL*, 18(2):14:1–14:50, 2017.
15. Chris Hathhorn, Chucky Ellison, and Grigore Roşu. Defining the Undefinedness of C. In *PLDI 2015*, pages 336–345.
16. Chung-Kil Hur, Georg Neis, Derek Dreyer, and Viktor Vafeiadis. The power of parameterization in coinductive proof. In *POPL 2013*, pages 193–206.
17. Claude Kirchner, Helene Kirchner, and Michael Rusinowitch. Deduction with Symbolic Constraints. Technical Report RR-1358, INRIA, 1990.
18. Cynthia Kop. Termination of LCTRSs. *CoRR*, abs/1601.03206, 2016.
19. Cynthia Kop and Naoki Nishida. Constrained Term Rewriting tooL. In *LPAR 2015*, pages 549–557.
20. Cynthia Kop and Naoki Nishida. Term Rewriting with Logical Constraints. In *FroCoS 2013*, pages 343–358.
21. Dorel Lucanu, Vlad Rusu, and Andrei Arusoaie. A generic framework for symbolic execution: A coinductive approach. *J. Symb. Comput.*, 80:125–163, 2017.
22. José Meseguer and Prasanna Thati. Symbolic Reachability Analysis Using Narrowing and Its Application to Verification of Cryptographic Protocols. *Higher-Order and Symbolic Computation*, 20(1-2):123–160, 2007.
23. Daejun Park, Andrei Ştefănescu, and Grigore Roşu. KJS: a Complete Formal Semantics of JavaScript. In *PLDI 2015*, pages 346–356, 2015.
24. Andrei Popescu and Elsa L. Gunter. Incremental pattern-based coinduction for process algebra and its isabelle formalization. In *FOSSACS 2010*, pages 109–127.
25. Camilo Rocha, José Meseguer, and César A. Muñoz. Rewriting modulo SMT and open system analysis. *J. Log. Algebr. Meth. Program.*, 86(1):269–297, 2017.
26. Grigore Roşu. Matching logic. *Logical Methods in Comp. Sci.*, 13(4):1–61, 2017.
27. Grigore Roşu and Traian Florin Şerbănuţă. An Overview of the K Semantic Framework. *Journal of Logic and Algebraic Programming*, 79(6):397–434, 2010.
28. Traian-Florin Şerbănuţă, Grigore Roşu, and José Meseguer. A rewriting logic approach to operational semantics. *Inf. and Comp.*, 207(2):305–340, 2009.
29. Stephen Skeirik, Andrei Ştefănescu, and José Meseguer. A constructor-based reachability logic for rewrite theories. TR. http://hdl.handle.net/2142/95770.