Proving Reachability Properties on Term Rewriting Systems with Strategies

Thomas Genet and Yann Salmon

IRISA, INRIA and Université Rennes 1, France

Abstract. We aim at defining regular over-approximation of sets of reachable terms for term rewriting systems applied with a strategy. In this ongoing work, we focus on *innermost* strategies which are the evaluation strategy of most functional programming languages. Having an accurate over-approximation of reachable terms for functional programming languages would permit to prove richer unreachability properties, *i.e.* safety properties on such programs.

1 Introduction

Term rewriting systems [BN98], TRS for short, are a convenient way of representing computer programs: a term represents the current stage of the computation and rewriting rules represent allowed steps from one stage to another. We are interested in computing the set of terms reachable by rewriting, *i.e.* the set $R^*(L_0) = \left\{\text{term } t \;\middle|\; \exists s \in L_0, s \to_R^* t\right\}$, for a given finite term rewriting system R and a set of so-called initial terms L_0. Reachability properties on the TRS can be proven using a computable representation of this set or of an approximation thereof. This technique can be applied to Java programs thanks to the translation in [Boi+07] and is used in [GK00] to assess the security of some cryptographic systems. In this paper, we take much simpler examples for the sake of readability.

For example, using an appropriate signature Σ, let R be a term rewriting system defining the natural integers with constant `0` and successor `S`, the usual arithmetical operations `sum` and `mult` as well as the `factorial`. Let L_0 be the initial language, *i.e.* the set of ground terms that we consider as the starting point of our program. Continuing our example, let L_0 be the set of terms of the form `factorial(S(S... S(0)...))`. Verifying that the factorial of a natural is never zero using our formalism amounts to checking that the term `0` is not reachable from terms in L_0 by arbitrary applications of the rules in R, that is $R^*(L_0) \cap \{0\} = \emptyset$.

The general TRS (un)reachability problem, "Does $R^*(L_0) \cap L_b = \emptyset$?", where L_b is a set of terms considered bad, is obviously not decidable, due to the Turing-completeness of the TRS formalism. Circumventing this is usually done by working with regular languages [Com+08], for which computing the intersection and checking emptiness can be done in polynomial time. Restricting

to regular initial (L_0) and bad languages (L_b) is not enough: this does not ensure that $R^*(L_0)$ is regular. The regularity-preservation property of TRS is in turn undecidable [GT95]; it has been proven true for some restricted classes of TRS. For instance, Réty [RV05] computes $R^*_{strat}(L_0)$, the set of descendants of terms in L_0 obtained by applications of rules in R using the strategy *strat*, for the restricted class of TRS called constructor-based and several *strat*.

Instead of restricting the TRS classes, we focus on tree automaton completion, stemming form the work of [Gen98], and more precisely on equational completion [GR10] (or just "completion" for short). This technique aims at computing a sufficiently precise regular over-approximation K of $R^*(L_0)$. Since $K \supseteq R^*(L_0)$, from $K \cap L_b = \emptyset$ follows $R^*(L_0) \cap L_b = \emptyset$. This procedure can be applied to any left-linear TRS, but does not always terminate. Over-approximating eases termination, but it needs to be precise enough to prove unreachability properties: if K is too large then we may have $K \cap L_b \neq \emptyset$ though $R^*(L_0) \cap L_b = \emptyset$. Recently, many techniques for building accurate approximations have been studied: declarative language (equations) for defining approximations [GR10] and automatic approximation refinement [Bou+06; Boi+10]. However, none of those approaches takes the *strategy* used to apply the rules of the TRS into account: they over-approximate $R^*(L_0)$ which is itself an over-approximation of $R^*_{strat}(L_0)$.

This causes imprecisions when over-approximating the set of reachable terms for functional programming languages, such as OCaml [Ler+12] or those used in proof assistants like Coq [BC04] or Isabelle/HOL [NPW02]. Such functional programs can be encoded by TRS applied with a *leftmost innermost strategy*. Our overall objective is to take into account *strat* in the over-approximation of $R^*_{strat}(L_0)$. This work, which is a first step in this direction, aims at taking into account the innermost part of the strategy. We want to refine the equational completion procedure of [GR10] in order to have a better precision in the over-approximation of $R^*_{\text{in}}(L_0)$, the set of descendants of terms in L_0 obtained by applications of rules in R using an innermost strategy. At the end of the paper, we will see that the technique presented here is likely to be adapted to cover the leftmost part of the strategy, but this is ongoing work. A precise approximation of terms reachable by leftmost innermost rewriting would be a simple and elegant alternative to Higher Order Recursive Schemes used for the static analysis of functional programs [OR11].

2 Innermost strategies

In our example TRS R, the term `mult(S(0), plus(S(0), S(0)))` has more than one reducible expression, or redex: it can be rewritten in one step either to the term `plus(mult(0, plus(S(0), S(0))), plus(S(0), S(0)))`, using the rule `mult(S(`x`),`y`)`→`plus(mult(`x`, `y`), `y`)` at the root, or to the term `mult(S(0), S(plus(0, S(0))))`, using the rule `plus(S(`x`), `y`)`→`S(plus(`x`, `y`))`

in the appropriate subterm `plus(S(0), S(0))`. To consistently decide which redex to reduce, implementations of TRS can use a strategy.

Innermost strategies consist in reducing a redex only when none of its strict subterms is a redex itself; in our example, this is the latter reduction. Due to some reductions being disallowed, $R^*_{\text{in}}(L_0) \subseteq R^*(L_0)$, and very often, $R^*_{\text{in}}(L_0) \subset R^*(L_0)$. Note that given a term t, checking that each subterm of t at depth 1 is R-irreducible is sufficient to ensure that a rewriting step may be applied to t under an innermost strategy.

We want to build a correct and reasonably precise over-approximation of $R^*_{\text{in}}(L_0)$. As exposed later, the current equational completion procedure is not strategy-aware. This does not hinder correctness: $K \supseteq R^*(L_0)$ implies $K \supseteq R^*_{\text{in}}(L_0)$. However, the process introduces a potentially great number of terms in K that are irrelevant when we are interested in $R^*_{\text{in}}(L_0)$.

3 Usual completion procedure

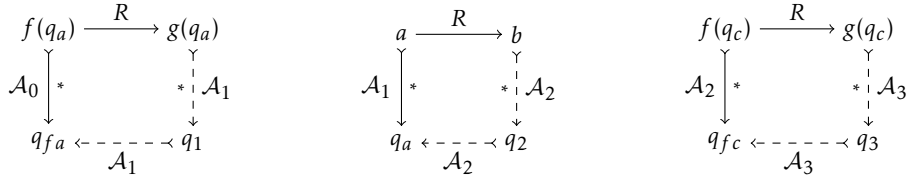
Given a left linear TRS R over signature Σ, the automaton completion procedure takes an automaton $\mathcal{A}_0$ recognising the initial language L_0 and iteratively adds transitions to it to incorporate R-descendants of already recognised terms. It stops when (if) a fixpoint is reached: the language recognised by the produced automaton $\mathcal{A}^*$, $\mathscr{L}(\mathcal{A}^*)$[1], is a superset of L_0 and is R-closed, thus $\mathscr{L}(\mathcal{A}^*)$ is a superset of $R^*(L_0)$. Transitions, and possibly new states, are added to the automaton $\mathcal{A}_i$, where i is some step of the completion process, by resolving so-called critical pairs ($\mathcal{CP}$): a rule $\ell \to r \in R$ instantiated into configurations of $\mathcal{A}_i$ as $\ell\sigma \to r\sigma$, where σ is an arbitrary map from variables of Σ to states of $\mathcal{A}$, extended to terms like a traditional substitution and $\mathcal{A}_i$ recognises $\ell\sigma$ into a state q, but not $r\sigma$. The resolution amounts to adding valid (normalised) transitions and the needed states to let the completed automaton $\mathcal{A}_{i+1}$ recognise $r\sigma$ into q as well.

The procedure does not always terminate, because resolution can create new critical pairs. Merging states of the automaton counters this at the cost of precision. This can be parametrised by linear equations between terms as in [GR10]. An equation $t = s$ instantiated into configurations of $\mathcal{A}$ as $t\sigma = s\sigma$ is said to be applicable if there exist two distinct states q_1 and q_2 such that $\mathcal{A}$ recognises $t\sigma$ into q_1 and $s\sigma$ into q_2. In this case, the states q_1 and q_2 are "merged". Therefore, the equational completion procedure takes as input not only R and $\mathcal{A}_0$, but a set E of such equations as well, and, in case of termination, outputs a fixpoint automaton $\mathcal{A}^*_{R,E}$.

For example, take signature $\Sigma = \{f : 1, g : 1, a : 0, b : 0, c : 0\}$ and set of variables $\{x\}$, $R = \{f(x) \to g(x), a \to b\}$, $L_0 = \{f(a), f(c)\}$ and $E = \varnothing$. Note that

[1] Given an automaton $\mathcal{A}$ and a state q of $\mathcal{A}$, we note $\mathscr{L}(\mathcal{A}, q)$ the set of terms recognised by $\mathcal{A}$ into state q.

$g(a) \in R^*(L_0) \setminus R^*_{\text{in}}(L_0)$. A reasonable automaton to recognise L_0 is $\mathcal{A}_0$ consisting of states q_a, q_c, q_{fa}, q_{fc} and transitions $a \rightarrowtail q_a$, $c \rightarrowtail q_c$, $f(q_a) \rightarrowtail q_{fa}$ and $f(q_c) \rightarrowtail q_{fc}$. We have a critical pair $\mathcal{CP}_1 = (f(x) \to g(x), \sigma = \{x \mapsto q_a\}, q_{fa})$: $\mathcal{A}_0$ recognises $f(q_a)$ into q_{fa} but not $g(q_a)$. Resolving $\mathcal{CP}_1$ builds a new automaton $\mathcal{A}_1$ from $\mathcal{A}_0$ by addition of state q_1 and transitions $g(q_a) \rightarrowtail q_1$ and $q_1 \rightarrowtail q_{fa}$. This is shown in figure 1. The epsilon transition $q_1 \rightarrowtail q_{fa}$ reads no input and is oriented in this way to ensure that the right-hand side of the rule is recognised into the same state as the left-hand side was previously recognised. We also solve $\mathcal{CP}_2 = (a \to b, \sigma = \emptyset, q_a)$ by adding q_2 and $b \rightarrowtail q_2$, $q_2 \rightarrowtail q_a$ and $\mathcal{CP}_3 = (f(x) \to g(x), \sigma = \{x \mapsto q_c\}, q_{fc})$ by adding q_3 and $g(q_c) \rightarrowtail q_3$ and $q_3 \rightarrowtail q_{fc}$. In this example, there are no more critical pairs and the procedure terminates with $\mathcal{A}^*_{R,E} = \mathcal{A}_3$. Assuming the same final states as in $\mathcal{A}_0$, we have indeed $\mathscr{L}(\mathcal{A}_3) = R^*(L_0)$.



(a) $\mathcal{CP}_1$ (b) $\mathcal{CP}_2$ (c) $\mathcal{CP}_3$

Fig. 1. Resolving critical pairs

The correctness theorem of [GR10] states that under the aforementioned hypotheses, $\mathscr{L}(\mathcal{A}^*_{R,E}) \supseteq R^*(\mathscr{L}(\mathcal{A}_0))$. The precision theorem of the same states that under the supplementary hypothesis that $\mathcal{A}_0$ is consistent with R and E, which merely means that the language recognised in each state of $\mathcal{A}_0$ is a subset of some equivalence class of the congruence relation induced by E, and has a common antecedent by R/E, the "R modulo E" rewrite relation, we have $\mathscr{L}(\mathcal{A}^*_{R,E}) \subseteq (R/E)^*(\mathscr{L}(\mathcal{A}_0))$. Our current research aims at designing a procedure to build an automaton $\mathcal{A}^*_{R_{\text{in}},E}$ such that $\mathscr{L}(\mathcal{A}^*_{R_{\text{in}},E}) \subseteq (R_{\text{in}}/E)^*(\mathscr{L}(\mathcal{A}_0))$.

Left-linearity of R is required for correctness: take $\Sigma = \{h : 2, g : 1, a : 0\}$, $R = \{h(x,x) \to g(x)\}$, *ie.* a non left-linear TRS due to the presence of twice the same variable x in the left-hand side of a rule, and take $\mathcal{A}_0$ with transitions $a \rightarrowtail q_1$, $a \rightarrowtail q_2$ and $h(q_1, q_2) \rightarrowtail q_f$. This last transition should intuitively give rise to a critical pair, but does not because the possible instantiations $\ell\sigma$ of the only rule are $h(q_1, q_1)$, $h(q_2, q_2)$ and $h(q_f, q_f)$. $\mathcal{A}_0$ is the fix-point of the completion procedure, but $g(a) \notin \mathscr{L}(\mathcal{A}_0)$ while $g(a) \in R(\{f(a,a)\})$.

4 Current investigations

The completion of some critical pairs corresponds to rewriting steps that do not conform to innermost strategies. In the example of section 3, we do not want to solve $\mathcal{CP}_1$, at least not before $\mathcal{CP}_2$, and even when we do, we would like to distinguish between a and b being recognised into q_a when dealing with $g(q_a)$. How to determine with certainty that every term recognised in a given state is reducible by R? Indeed, if we abstain from solving a critical pair that involves a state representing at least one irreducible term, we risk missing some terms in $R^*_{\text{in}}(L_0)$ and loosing correctness.

A first approach uses the fact that because R is left-linear, IRR(R), the set of R-normal forms, is regular [GT95, Theorem 4]. We define the fibre of a configuration c of automaton $\mathcal{A}$, $\mathscr{F}(\mathcal{A}, c)$, inductively: if c is a state, $\mathscr{F}(\mathcal{A}, c) = \mathscr{L}(\mathcal{A}, c)$; if c is a constant, $\mathscr{F}(\mathcal{A}, c) = \{c\}$; else $c = f(c_1, \ldots, c_k)$ and $\mathscr{F}(\mathcal{A}, c) = \{f(\alpha_1, \ldots, \alpha_k) \mid \forall i \in [\![1\,;k]\!], \alpha_i \in \mathscr{F}(\mathcal{A}, c_i)\}$. It is a regular language. We define a new notion of critical pair $\mathcal{CP} = (\ell \to r, \sigma, q)$ by adding the following restriction: if $\ell\sigma$ is of the form $f(c_1, \ldots, c_k)$ where $c_1, \ldots, c_k$ are configurations, then for all $i \in [\![1\,;k]\!]$, we must have $\mathscr{F}(\mathcal{A}, c_i) \cap \text{IRR}(R) \neq \emptyset$. The fixpoint automaton that this new procedure produces (if any) recognises a R_{in}-closed language which contains L_0, and therefore contains $R^*_{\text{in}}(L_0)$. The proof can be derived from the one of theorem 45 in [GR10]: both are based on the ability to build a critical pair from any valid rewriting step.

However, since building an automaton recognising IRR(R) can be exponential in the size of R, we are also exploring a second approach that would not require IRR(R). The structure of the completed automaton can be used to distinguish between normal forms and reducible terms. On the previous example, we can infer from the automaton that a is not a normal form since it is recognised by q_a and there is an epsilon transition $q_2 \rightarrowtail q_a$. According to completion, this epsilon transition denotes that a is reducible and we can thus mark the path $a \rightarrowtail q_a$ as "forbidden" w.r.t. an innermost completion. It might be possible to check if a critical pair is innermost-compatible by carefully inspecting the recognising path of $l\sigma \rightarrowtail q$. In our example, the path $a \rightarrowtail q_a$ would be forbidden after completion of $\mathcal{CP}_2$. Since the path $g(q_a) \rightarrowtail q_1 \rightarrowtail q_{fa}$ results from the completion of some critical pair ($\mathcal{CP}_1$), $a \rightarrowtail q_a$ would not be taken into account, thereby excluding $g(a)$ from the recognised language and keeping $g(b)$. On the other hand, the path $f(q_a) \rightarrowtail q_{fa}$ does not result from a completion step, so $a \rightarrowtail q_a$ would still be available in this case, causing $f(a)$ to be recognised. This is not surprising, since it belongs to L_0, a subset of $R^*_{\text{in}}(L_0)$. Furthermore, if a state has only forbidden paths pointing thereto, it could be completely avoided as a value when enumerating the possible σ.

This approach might be useful for leftmost-innermost strategies as well, where one only reduces an innermost redex when there is no redex at its left. With $\Sigma = \{h : 2, a : 0, b : 0, c : 0, d : 0\}$, $R = \{a \to b, c \to d\}$, $\mathcal{A} = \{a \rightarrowtail q_a, c \rightarrowtail q_c, h(q_a, q_c) \rightarrowtail q_h\}$, usual completion adds epsilon transitions $q_b \rightarrowtail q_a$ and $q_d \rightarrowtail q_c$ that cause $h(a, d) \rightarrowtail h(q_a, d) \rightarrowtail h(q_a, q_d) \rightarrowtail h(q_a, q_c) \rightarrowtail q_h$.

Indeed, $h(a,d) \in R(\{h(a,c)\})$, but this stems from a non-leftmost rewriting. Marking $a \rightarrowtail q_a$ and $c \rightarrowtail q_c$ as "forbidden" like before would allow us to disregard the aforementioned derivation, because q_a appears to the left of q_c in the configuration and the paths leading to each of them are "forbidden". On the other hand, we would retain $h(b,d) \rightarrowtail h(q_b,d) \rightarrowtail h(q_b,q_d) \rightarrowtail h(q_b,q_c) \rightarrowtail q_h$ because q_b, appearing to the left of q_c, has a non-forbidden path from b to it. This requires more investigation, though.

5 Conclusion and perspectives

We sketched a general way of refining equational completion to adapt it to innermost strategies. We have to give a precise and efficient algorithm to compute the fibre of a configuration that cleverly takes into account the fact that each completion step can modify the fibres. We also plan to assess whether our second approach is correct and to compare its efficiency with the first one, and maybe interleave them.

We have to quantify the gain in precision granted by our proposed procedure and balance it with its supplementary computational cost. It should also be noted that due to our proposed strategy-aware completion procedure resolving less critical pairs than the usual one, its termination is easier to achieve. We want to explore whether this gain is marginal or not.

Finally, we want to adapt the usual completion procedure to different and more expressive strategies so as to build precise over-approximation of reachable terms for other programming languages. Target strategies are the outermost strategy, used for example by Haskell [Has], and declarative strategies like the one used in Tom [Bal+07].

References

[BC04] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Springer Verlag, 2004.

[BN98] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[Bal+07] E. Balland et al. "Tom: Piggybacking Rewriting on Java". In: *RTA'07*. LNCS. Springer, 2007, pp. 36–47.

[Boi+07] Yohan Boichut et al. "Rewriting Approximations for Fast Prototyping of Static Analyzers". In: *RTA*. Vol. 4533. LNCS. Springer, 2007, pp. 48–62.

[Boi+10] Y. Boichut et al. *Fast Equational Abstraction Refinement for Regular Tree Model Checking*. Technical Report. INRIA, 2010. URL: `http://hal.inria.fr/inria-00501487`.

[Bou+06] A. Bouajjani et al. "Abstract Regular Tree Model Checking". In: *ENTCS* 149.1 (2006), pp. 37–48.

[Com+08] Hubert Comon et al. *Tree Automata Techniques and Applications*. 2008.

[GK00] Thomas Genet and Francis Klay. *Rewriting for Cryptographic Protocol Verification (extended version)*. Technical Report 3921. INRIA, 2000.

[GR10] Thomas Genet and Vlad Rusu. "Equational Tree Automata Completion". In: *Journal of Symbolic Computation* 45 (2010), pp. 574–597.

[GT95] Rémy Gilleron and Sophie Tison. "Regular Tree Languages and Rewrite Systems". In: *Fundamenta informaticae* 24 (1995), pp. 157–175.

[Gen98] Thomas Genet. "Decidable Approximations of Sets of Descendants and Sets of Normal Forms". In: *Proc. 9th RTA Conf., Tsukuba (Japan), volume 1379 of LNCS*. Springer-Verlag, 1998, pp. 151–165.

[Has] *The Haskell Programming Language*. `http://www.haskell.org`. 2012.

[Ler+12] X. Leroy et al. *The Objective Caml system release 3.12 – Documentation and user's manual*. INRIA. `http://caml.inria.fr/ocaml/`. 2012.

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*. Vol. 2283. LNCS. Springer, 2002.

[OR11] L. Ong and S. Ramsay. "Verifying higher-order functional programs with pattern-matching algebraic data types". In: *POPL'11*. 2011.

[RV05] Pierre Réty and Julie Vuotto. "Tree automata for rewrite strategies". In: *J. Symb. Comput.* 40.1 (July 2005), pp. 749–794. ISSN: 0747-7171.