HAL
open science

Tree Automata Completion for Static Analysis of Functional Programs

Thomas Genet, Yann Salmon

To cite this version:

Thomas Genet, Yann Salmon. Tree Automata Completion for Static Analysis of Functional Programs. 2013. hal-00780124v2

HAL Id: hal-00780124

https://hal.science/hal-00780124v2

Submitted on 27 May 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Tree Automaton Completion for Static Analysis of Functional Programs

Thomas Genet* Yann Salmon†

May 24, 2013

Tree Automata Completion is a family of techniques for computing or approximating the set of terms reachable by a rewriting relation. For functional programs translated into TRSs, we give a sufficient condition for completion to terminate. Second, in order to take into account the evaluation strategy of functional programs, we show how to refine completion to approximate reachable terms for a rewriting relation controlled by a strategy. In this paper, we focus on innermost strategy which represents the call-by-value evaluation strategy.

1 Introduction

Computing or approximating the set of terms reachable by rewriting finds more and more applications. For a Term Rewriting System (TRS) R and a set of terms $L_0 \subseteq T(\Sigma)$, the set of reachable terms is $R^*(L_0) = \left\{t \in T(\Sigma) \,\middle|\, \exists s \in L_0, s \to_R^* t\right\}$. This set can be computed for specific classes of R but, in general, it has to be approximated. Applications of the approximation of $R^*(L_0)$ are ranging from cryptographic protocol verification [GK00, ABB+05], to static analysis of various programming languages [BGJL07, KO11] or to TRS termination proofs [Mid02, GHWZ05]. Most of the techniques compute such approximations using tree automata as the core formalism to represent or approximate the (possibly) infinite set of terms $R^*(L_0)$. Most of them also rely on a Knuth-Bendix completion-like algorithm to produce an automaton $\mathcal{A}^*$ recognising exactly, or over-approximating, the set of reachable terms. As a result, these techniques can be refered as *tree automata completion* techniques [Gen98, TKS00, Tak04, FGVTT04, BCHK09, GR10, Lis12].

In this paper, we investigate the application of tree automata completion techniques to the static analysis of functional programs. The objective of such an analysis is to over-approximate the set of possible results of higher-order functional programs [OR11, KO11]. First, like the standard Knuth-Bendix completion, tree automata completion is not guaranteed to terminate. For TRSs extracted from functional programs, we show that termination of automata completion is guaranteed using natural constraints on the definition of the approximation. Second, we show that completion can take the rewriting strategy into account, i.e. over-approximate

*IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France, Thomas.Genet@irisa.fr
†IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France, Yann.Salmon@irisa.fr

terms reachable by rewriting under a strategy. In this paper, we focus on the innermost strategy which corresponds to the call-by-value strategy that is at the heart of several functional programming languages, such as Ocaml [LDG+12].

Surprisingly, very little effort has been done on computing or over-approximating $R^*_{strat}(L_0)$, i.e. set of reachable terms when R is applied with a strategy $strat$. To the best of our knowledge, Pierre RÃĭty's work [RV02] is the only one to have tackled this goal. He gives some sufficient conditions on L_0 and R for $R^*_{strat}(L_0)$ to be recognised by a tree automaton $\mathcal{A}^*$, where $strat$ can be the innermost or the outermost strategy. However, those restrictions on R and L_0 are strong and generally incompatible with the analysis of functional programs. In this paper, we define a tree automata completion algorithm over-approximating the set $R^*_{\text{in}}(L_0)$ for all left-linear TRSs R and all regular set of input terms L_0.

This paper is organised as follows: Section 2 recalls some basic notions about TRSs and tree automata. Section 3 defines tree automata completion. Section 4 shows how to guarantee the termination of completion when analysing TRSs obtained from functional programs. Section 5 presents some experiments. Finally, Section 6 explains how to tune completion so as to take innermost strategy into account.

2 Basic notions and notations

2.1 Terms

Definition 1 (Signature).

A signature is a set whose elements are called function symbols. Each function symbol has an arity, which is a natural integer. Function symbols of arity 0 are called constants. Given a signature Σ and $k \in \mathbb{N}$, the set of its function symbols of arity k is noted Σ_k. 1◀

Definition 2 (Term, ground term, linearity).

Given a signature Σ and a set $\mathcal{X}$ whose elements are called variables and such that $\Sigma \cap \mathcal{X} = \varnothing$, we define the set of terms over Σ and $\mathcal{X}$, $T(\Sigma, \mathcal{X})$, as the smallest set such that :

1. $\mathcal{X} \subseteq T(\Sigma, \mathcal{X})$ and
2. $\forall k \in \mathbb{N}, \forall f \in \Sigma_k, \forall t_1, \ldots, t_k \in T(\Sigma, \mathcal{X}), f(t_1, \ldots, t_k) \in T(\Sigma, \mathcal{X})$.

Terms in which no variable appears, i.e. terms in $T(\Sigma, \varnothing)$, are called ground; the set of ground terms is noted $T(\Sigma)$.

Terms in which any variable appears at most once are called linear.[1] 2◀

Definition 3 (Substitution).

A substitution over $T(\Sigma, \mathcal{X})$ is an application from $\mathcal{X}$ to $T(\Sigma, \mathcal{X})$. Any substitution is inductively extended to $T(\Sigma, \mathcal{X})$ by $\sigma(f(t_1, \ldots, t_k)) = f(\sigma(t_1), \ldots, \sigma(t_k))$. Given a substitution σ and a term t, we note $t\sigma$ instead of $\sigma(t)$. 3◀

Definition 4 (Context).

A context over $T(\Sigma, \mathcal{X})$ is a term in $T(\Sigma \cup \mathcal{X}, \{\square\})$ in which the variable $\square$ appears exactly once. A ground context over $T(\Sigma, \mathcal{X})$ is a context over $T(\Sigma)$. The smallest possible context, $\square$, is called the trivial context. Given a context C and a term t, we note $C[t]$ the term $C\sigma_t$, where $\sigma_t : \square \mapsto t$. 4◀

[1] In particular, any ground term is linear.

Definition 5 (Position).
Positions are finite words over the alphabet $\mathbb{N}$. The set of positions of term t, $\mathrm{Pos}(t)$, is defined by induction over t:

1. for all constant c and all variable X, $\mathrm{Pos}(c) = \mathrm{Pos}(X) = \{\Lambda\}$ and
2. $\mathrm{Pos}(f(t_1,\ldots,t_k)) = \{\Lambda\} \cup \bigcup_{i=1}^{k} \{i\}.\mathrm{Pos}(t_i)$. 5 ◀

Definition 6 (Subterm-at-position, replacement-at-position).
The position of the hole in context C, $\mathrm{Pos}_\square(C)$, is defined by induction on C:

1. $\mathrm{Pos}_\square(\square) = \Lambda$
2. $\mathrm{Pos}_\square(f(C_1,\ldots,C_k)) = i.\mathrm{Pos}_\square(C_i)$, where i is the unique integer in $[\![1\,;k]\!]$ such that C_i is a context.

Given a term u and $p \in \mathrm{Pos}(u)$, there is a unique context C and a unique term v such that $\mathrm{Pos}_\square(C) = p$ and $u = C[v]$. The term v is noted $u_{|p}$, and, given another term t, we note $u[t]_p = C[t]$. 6 ◀

2.2 Rewriting

Definition 7 (Rewriting rule, term rewriting system).
A rewriting rule over $(\Sigma,\mathcal{X})$ is a couple $(\ell,r) \in T(\Sigma,\mathcal{X}) \times T(\Sigma,\mathcal{X})$, that we note $\ell \to r$, such that any variable appearing in r also appears in ℓ. A term rewriting system (TRS) over $(\Sigma,\mathcal{X})$ is a set of rewriting rules over $(\Sigma,\mathcal{X})$. 7 ◀

Definition 8 (Rewriting step, redex, reducible term, normal form).
Given a signature $(\Sigma,\mathcal{X})$, a TRS R over it and two terms $s,t \in T(\Sigma)$, we say that s can be rewritten into t by R, and we note $s \to_R t$ if there exist a rule $\ell \to r \in R$, a ground context C over $T(\Sigma)$ and a substitution σ over $T(\Sigma,\mathcal{X})$ such that $s = C[\ell\sigma]$ and $t = C[r\sigma]$.
In this situation, the term s is said to be reducible by R and the subterm $\ell\sigma$ is called a redex of s. A term s that is not reducible by R is a normal form of R. The set of normal forms of R is noted $\textsc{Irr}(R)$.
We note $\to_R^*$ the reflexive and transitive closure of $\to_R$. 8 ◀

Definition 9 (Set of reachable terms).
Given a signature $(\Sigma,\mathcal{X})$, a TRS R over it and a set of terms $L \subseteq T(\Sigma)$, we note $R(L) = \{t \in T(\Sigma) \mid \exists s \in L, s \to_R t\}$ and $R^*(L) = \left\{t \in T(\Sigma) \,\middle|\, \exists s \in L, s \to_R^* t\right\}$. 9 ◀

Definition 10 (Left-linearity).
A TRS R is said to be left-linear if for each rule $\ell \to r$ of R, the term ℓ is linear. 10 ◀

Definition 11 (Constructors and defined symbols, sufficient completeness).
Given a TRS R over $(\Sigma,\mathcal{X})$, there is a partition $(\mathcal{C},\mathcal{D})$ of Σ such that all symbols occurring at the root position of left-hand sides of rules of R are in $\mathcal{D}$. $\mathcal{D}$ is the set of defined symbols of R, $\mathcal{C}$ is the set of constructors. Terms in $T(\mathcal{C})$ are called data-terms. A TRS R over $(\Sigma,\mathcal{X})$ is sufficiently complete if for all $s \in T(\Sigma)$, $R^*(\{s\}) \cap T(\mathcal{C}) \neq \varnothing$. 11 ◀

2.3 Equations

Definition 12 (Equivalence relation, congruence).

A binary relation over some set S is an equivalence relation if it is reflexive, symmetric and transitive.

An equivalence relation $\equiv$ over $T(\Sigma)$ is a congruence if for all $k \in \mathbb{N}$, for all $f \in \Sigma_k$, for all $t_1, \ldots, t_k, s_1, \ldots, s_k \in T(\Sigma)$ such that $\forall i \in [\![1\,;k]\!]$, $t_i \equiv s_i$, we have $f(t_1, \ldots, t_k) \equiv f(s_1, \ldots, s_k)$. 12 ◀

Definition 13 (Equation, $\equiv_E$).

An equation over $(\Sigma, \mathcal{X})$ is a pair of terms $(s,t) \in T(\Sigma,\mathcal{X}) \times T(\Sigma,\mathcal{X})$, that we note $s = t$. A set E of equations over $(\Sigma,\mathcal{X})$ induces a congruence $\equiv_E$ over $T(\Sigma)$ which is the smallest congruence over $T(\Sigma)$ such that for all $s = t \in E$ and for all substitution $\theta : \mathcal{X} \to T(\Sigma)$, $s\theta \equiv_E t\theta$. The classes of equivalence of $\equiv_E$ are noted with $[\cdot]_E$. 13 ◀

Definition 14 (Rewriting modulo E).

Given a TRS R and a set of equations E both over $(\Sigma,\mathcal{X})$, we define the R modulo E rewriting relation, $\to_{R/E}$, as follows. For any $u, v \in T(\Sigma)$, $u \to_{R/E} v$ if and only if there exist $u', v' \in T(\Sigma)$ such that $u' \equiv_E u$, $v' \equiv_E v$ and $u' \to_R v'$.

We define $\to^*_{R/E}$, $(R/E)(L)$ and $(R/E)^*(L)$ for $L \subseteq T(\Sigma)$ as in Definitions 8 and 9. 14 ◀

2.4 Tree automata

Definition 15 (Tree automaton, delta-transition, epsilon-transition, new state).

An automaton over Σ is some $\mathcal{A} = (\Sigma, Q, Q_F, \Delta)$ where Q is a finite set of states, Q_F is a subset of Q whose elements are called final states and Δ a finite set of transitions. A delta-transition is of the form $f(q_1, \ldots, q_k) \rightarrowtail q'$ where $f \in \Sigma_k$ and $q_1, \ldots, q_k, q' \in Q$. An epsilon-transition is of the form $q \rightarrowtail q'$ where $q, q' \in Q$. A configuration of $\mathcal{A}$ is a term in $T(\Sigma, Q)$.

A state $q \in Q$ that appears nowhere in Δ is called a new state. A configuration is elementary if each of its subconfigurations at depth 1 (if any) is a state. A configuration is trivial if it is just a state. 15 ◀

Remark. We simply write $\mathcal{A}$ to denote an automaton, write $Q_{\mathcal{A}}$ for the set of states of $\mathcal{A}$. We assimilate an automaton with its set of transitions. When taking a "new state", we silently expand $Q_{\mathcal{A}}$ if needed. We are only rarely interested in Q_F, the set of final states.

Definition 16.

Let $\mathcal{A} = (\Sigma, Q, Q_F, \Delta)$ be an automaton and let c, c' be configurations of $\mathcal{A}$. We say that $\mathcal{A}$ recognises c into c' in one step, and note $c \underset{\mathcal{A}}{\rightarrowtail} c'$ if there a transition $\tau \rightarrowtail \rho$ in $\mathcal{A}$ and a context C over $T(\Sigma, Q)$ such that $c = C[\tau]$ and $c' = C[\rho]$. We note $\overset{*}{\underset{\mathcal{A}}{\rightarrowtail}}$ the reflexive and transitive closure of $\underset{\mathcal{A}}{\rightarrowtail}$ and, for any $q \in Q$, $\mathscr{L}(\mathcal{A}, q) = \left\{ t \in T(\Sigma) \,\middle|\, t \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} q \right\}$. We extend this definition to subsets of Q and note $\mathscr{L}(\mathcal{A}) = \mathscr{L}(\mathcal{A}, Q_F)$. 16 ◀

Definition 17 (Determinism, Completeness).

An automaton is deterministic if it has no epsilon-transition and for all delta-transitions $\tau \rightarrowtail \rho$ and $\tau' \rightarrowtail \rho'$, if $\tau = \tau'$ then $\rho = \rho'$. An automaton is complete if each of its non-trivial configurations is the left-hand side of some of its transitions. 17 ◀

Definition 18 (Colours).
Transitions may have "colours", like $\mathfrak{R}$ for transition $q \overset{\mathfrak{R}}{\rightarrowtail} q'$, and we denote by $\mathcal{A}^{\not\mathfrak{R}}$ the automaton obtained from $\mathcal{A}$ by removing all transitions coloured with $\mathfrak{R}$. 18 ◀

Definition 19.
Given two states q, q' of some automaton $\mathcal{A}$ and a colour $\mathfrak{E}$, we note $q \underset{\mathcal{A}}{\overset{\mathfrak{E}}{\rightleftarrows}} q'$ when we have both $q \underset{\mathcal{A}}{\overset{\mathfrak{E}}{\rightarrowtail}} q'$ and $q' \underset{\mathcal{A}}{\overset{\mathfrak{E}}{\rightarrowtail}} q$. 19 ◀

Remark. $q \underset{\mathcal{A}}{\overset{\mathfrak{E},*}{\rightleftarrows}} q'$ is stronger than $(q \underset{\mathcal{A}}{\overset{\mathfrak{E},*}{\rightarrowtail}} q' \wedge q' \underset{\mathcal{A}}{\overset{\mathfrak{E},*}{\rightarrowtail}} q)$. $\underset{\mathcal{A}}{\overset{\mathfrak{E},*}{\rightleftarrows}}$ is an equivalence relation over $Q_{\mathcal{A}}$. This relation is extended to a congruence relation over $T(\Sigma, Q)$. The equivalence classes are noted with $[\cdot]_{\mathfrak{E}}$.

Definition 20.
Let $\mathcal{A} = (\Sigma, Q, Q_F, \Delta)$ be an automaton and $\mathfrak{E}$ a colour. We note $\mathcal{A}/\mathfrak{E}$ the automaton over Σ whose set of states is $Q/\mathfrak{E}$, whose set of final states if $Q_F/\mathfrak{E}$ and whose set of transitions is

$$\begin{aligned}&\left\{f([q_1]_{\mathfrak{E}}, \ldots, [q_k]_{\mathfrak{E}}) \rightarrowtail [q']_{\mathfrak{E}} \,\middle|\, f(q_1, \ldots, q_k) \rightarrowtail q' \in \Delta\right\}\\ \cup &\left\{[q]_{\mathfrak{E}} \rightarrowtail [q']_{\mathfrak{E}} \,\middle|\, q \rightarrowtail q' \in \Delta \wedge [q]_{\mathfrak{E}} \neq [q']_{\mathfrak{E}}\right\}.\end{aligned}$$ 20 ◀

Remark. For any configurations c, c' of $\mathcal{A}$, we have $c \underset{\mathcal{A}}{\overset{*}{\rightarrowtail}} c'$ if and only if $[c]_{\mathfrak{E}} \underset{\mathcal{A}/\mathfrak{E}}{\overset{*}{\rightarrowtail}} [c']_{\mathfrak{E}}$. So the languages recognised by $\mathcal{A}$ and $\mathcal{A}/\mathfrak{E}$ are the same.

We now give notations used for pair automata, the archetype of which is the product of two automata.

Definition 21 (Pair automaton).
An automaton $\mathcal{A} = (\Sigma, Q, Q_F, \Delta)$ is said to be a pair automata if there exists some sets Q_1 and Q_2 such that $Q = Q_1 \times Q_2$. 21 ◀

Definition 22 (Product automaton).
Let $\mathcal{A} = (\Sigma, Q, Q_F, \Delta_{\mathcal{A}})$ and $\mathcal{B} = (\Sigma, P, P_F, \Delta_{\mathcal{B}})$ be two automata. The product automaton of $\mathcal{A}$ and $\mathcal{B}$ is $\mathcal{A} \times \mathcal{B} = (\Sigma, Q \times P, Q_F \times P_F, \Delta)$ where

$$\begin{aligned}\Delta = &\left\{f(\langle q_1, p_1\rangle, \ldots, \langle q_k, p_k\rangle) \rightarrowtail \langle q', p'\rangle \,\middle|\, f(q_1, \ldots, q_k) \rightarrowtail q' \in \Delta_{\mathcal{A}} \wedge f(p_1, \ldots, p_k) \rightarrowtail p' \in \Delta_{\mathcal{B}}\right\}\\ &\cup \left\{\langle q, p\rangle \rightarrowtail \langle q', p'\rangle \,\middle|\, q \rightarrowtail q' \in \Delta_{\mathcal{A}}\right\} \cup \left\{\langle q, p\rangle \rightarrowtail \langle q, p'\rangle \,\middle|\, p \rightarrowtail p' \in \Delta_{\mathcal{B}}\right\}.\end{aligned}$$ 22 ◀

Definition 23 (Projections).
Let $\mathcal{A} = (\Sigma, Q, Q_F, \Delta)$ be a pair automaton, let $\tau \rightarrowtail \rho$ be one of its transitions and $\langle q, p\rangle$ be one of its states. We define $\Pi_1(\langle q, p\rangle) = q$ and extend $\Pi_1(\cdot)$ to configurations inductively: $\Pi_1(f(\gamma_1, \ldots, \gamma_k)) = f(\Pi_1(\gamma_1), \ldots, \Pi_1(\gamma_k))$. We define $\Pi_1(\tau \rightarrowtail \rho) = \Pi_1(\tau) \rightarrowtail \Pi_1(\rho)$. We define $\Pi_1(\mathcal{A}) = (\Sigma, \Pi_1(Q), \Pi_1(Q_F), \Pi_1(\Delta))$. $\Pi_2(\cdot)$ is defined on all these objects in the same way for the right components. 23 ◀

Remark. Using $\Pi_1(\mathcal{A})$ amounts to forgetting the precision given by the right components of the states. As a result, $\mathscr{L}(\Pi_1(\mathcal{A}), q) \supseteq \mathscr{L}(\mathcal{A}, \langle q, p\rangle)$.

2.5 Innermost strategies

In general, a strategy over a TRS R is a set of (computable) criteria to describe a certain subrelation of $\to_R$. In this paper, we will be interested in innermost strategies. In these strategies, commonly used to execute functional programs ("call-by-value"), terms are rewritten by always contracting one of the lowest reducible subterms.

Definition 24 (Innermost strategy).
Given a TRS R and two terms s, t, we say that s can be rewritten into t by R with an innermost strategy, and we note $s \to_{R_{\text{in}}} t$, if $s \to_R t$ and each strict subterm of the redex in s is a R-normal form. We define $\to^*_{R_{\text{in}}}$, $R_{\text{in}}(L)$ and $R^*_{\text{in}}(L)$ as in Definitions 8 and 9. 24 ◀

Remark. It is in fact sufficient to check whether subterms of s at depth 1 are in normal form to decide whether s can be rewritten with an innermost strategy.

To deal with innermost strategies, we will have to discriminate normal forms. This is possible within the tree automaton framework when R is left-linear.

Theorem 25 ([CR87]).
Let R be a left-linear TRS. There is a deterministic and complete tree automaton $\mathcal{IRR}(R)$ such that $\mathscr{L}(\mathcal{IRR}(R)) = \textsc{Irr}(R)$ and with a distinguished state p_{red} such that $\mathscr{L}(\mathcal{IRR}(R), p_{\text{red}}) = T(\Sigma) \setminus \textsc{Irr}(R)$. 25 ■

Remark. From determinism and the property of p_{red} follows that for any state p different from p_{red}, $\mathscr{L}(\mathcal{IRR}(R), p) \subseteq \textsc{Irr}(R)$.

3 Classical equational completion

Equational completion of [GR10] is an iterative process on automata that is not guaranteed to terminate. Each iteration comprises two parts: (exact) completion itself, then equational merging. The former tends to incorporate descendants by R of already recognised terms into the recognised language; this leads to the creation of new states. The latter tends to merge states in order to ease termination of the overall process, at the cost of precision of the computed result. Some transition added by equational completion will have colours $\mathfrak{R}$ or $\mathfrak{E}$; it is assumed that the transitions of the input automaton $\mathcal{A}_0$ do not have any colour and that $\mathcal{A}_0$ does not have any epsilon-transition.

3.1 Exact completion

Exact completion is about resolving critical pairs. A critical pair represents a situation where some term is recognised by the current automaton, but not its descendants by R. Its resolution consists in adding transitions to let the descendants be recognised as well. This process can create new critical pairs.

Definition 26 (Critical pair).
A pair $(\ell \to r, \sigma, q)$ where $\ell \to r \in R$, $\sigma : \mathcal{X} \to Q_{\mathcal{A}}$ and $q \in Q_{\mathcal{A}}$ is critical if (see Figure 1(a))

1. $\ell\sigma \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} q$ and

2. $r\sigma \overset{*}{\underset{\mathcal{A}}{\not\rightarrowtail}} q$. 26 ◀

We will want to add transitions $r\sigma \rightarrowtail q'$ and $q' \rightarrowtail q'$. However, doing the former is not generally possible in one step, as $r\sigma$ might be a non-elementary configuration. We will have to normalise this configuration by adding intermediate states. For example, to set $f(g(a,b)) \rightarrowtail q'$, we will first add $a \rightarrowtail q_a$, then, $b \rightarrowtail q_b$, $g(q_a, q_b) \rightarrowtail q_g$ and finally $f(q_g) \rightarrowtail q'$, exploring $f(g(a,b))$ with a postfix traversal. At each of these steps, we will look whether we could reuse an already existing transition; if not, we will add a new state.

Definition 27 ($Norm_{\mathcal{A}}(c,q)$).

Let $\mathcal{A}$ be an automaton, c be a non-trivial configuration and q be any state. Let $\Xi = (\xi_1, \ldots, \xi_K)$ be a postfix traversal of c (ξ_K is the root position). With Δ the set of transitions of $\mathcal{A}$, let us use an auxiliary function:

$$Norm_{\mathcal{A}}(c,q) = NormAux_{\Delta}^{\Xi,1}(c,q). \tag{1}$$

Now let us define $NormAux_{\cdot}^{\Xi,i}$. For i ranging from 1 to $K-1$, for any set of transitions Δ and any configuration d such that $d_{|\xi_i}$ is an elementary configuration, let

$$NormAux_{\Delta}^{\Xi,i}(d,q) = \{d_{|\xi_i} \rightarrowtail q'\} \cup NormAux_{\Delta \cup \{d_{|\xi_i} \rightarrowtail q'\}}^{\Xi,i+1}(d[q']_{\xi_i}, q) \tag{2}$$

where q' is such that $d_{|\xi_i} \rightarrowtail q'$ in $\Delta/\mathfrak{E}$ (or, if there is no such state, q' is a new state, distinct from q). Also, for any set of transitions Δ and any elementary configuration d, let

$$NormAux_{\Delta}^{\Xi,K}(d,q) = \{d \rightarrowtail q\}. \tag{3}$$

27 ◀

Remark. It is necessary to consider equivalence by $\mathfrak{E}$ when searching for an already existing transition. Suppose we work with some $f(q_1)$ and have $q_1 \overset{\mathfrak{E}}{\underset{\Delta}{\rightleftarrows}} q_2$ and $f(q_2) \underset{\Delta}{\rightarrowtail} q'$: we do not want to create a new state here, but reuse q' and set $f(q_1) \rightarrowtail q'$.

Definition 28 (Completion of a critical pair).

A critical pair $CP = (\ell \to r, \sigma, q)$ in automaton $\mathcal{A}$ is completed by first computing $N = Norm_{\mathcal{A}/\mathfrak{R}}(r\sigma, q')$ where q' is a new state, then adding to $\mathcal{A}$ the new states and the transitions appearing in N as well as the transition $q' \overset{\mathfrak{R}}{\rightarrowtail} q$. If $r\sigma$ is a trivial configuration (ie. r is just a variable), only transition $r\sigma \overset{\mathfrak{R}}{\rightarrowtail} q$ is added. 28 ◀

(a) A critical pair

(b) Situation of application

Figure 1: A critical pair and a situation of application of an equation

Definition 29 (Step of completion).

The automaton resulting from completion of a list comprising of one critical pair CP is noted $\mathcal{J}_R^{(CP)}(\mathcal{A})$. For a list of zero critical pair, we set $\mathcal{J}_R^{()}(\mathcal{A}) = \mathcal{A}$ and for any list LCP of critical pairs (of $\mathcal{A}$) whose head is CP and tail is LCP', $\mathcal{J}_R^{LCP}(\mathcal{A}) = \mathcal{J}_R^{LCP'}\left(\mathcal{J}_R^{(CP)}(\mathcal{A})\right)$. A step of completion of automaton $\mathcal{A}$ consists in producing automaton $\mathcal{C}_R(\mathcal{A}) = \mathcal{J}_R^{\mathcal{CP}}(\mathcal{A})$, where $\mathcal{CP}$ is a list of all critical pairs of $\mathcal{A}$. 29 ◀

Example 30.

Let Σ be defined with $\Sigma_0 = \{n, 0\}$, $\Sigma_1 = \{s, a, f\}$, $\Sigma_2 = \{c\}$ where 0 is meant to represent integer zero, s the successor operation on integers, a the predecessor ("antecessor") operation, n the empty list, c the constructor of lists of integers and f intended by some unwise person to be the function on lists that filters out integer zero. Let $R = \{f(n) \to n, f(c(s(X), Y)) \to c(s(X), f(Y)), f(c(a(X), Y)) \to c(a(X), f(Y)), f(c(0, Y)) \to f(Y), a(s(X)) \to X, s(a(X)) \to X\}$. Let $\mathcal{A}_0 = \{n \rightarrowtail q_n, 0 \rightarrowtail q_0, s(q_0) \rightarrowtail q_s, a(q_s) \rightarrowtail q_a, c(q_a, q_n) \rightarrowtail q_c, f(q_c) \rightarrowtail q_f\}$.
We have $\mathscr{L}(\mathcal{A}_0, q_f) = \{f(c(a(s(0)), n))\}$ and $R(\mathscr{L}(\mathcal{A}_0, q_f)) = \{f(c(0, n)), c(a(s(0)), f(n))\}$.

There is a critical pair CP_1 in $\mathcal{A}_0$ with the rule $f(c(a(X), Y)) \to c(a(X), f(Y))$, the substitution $\sigma_1 = \{X \mapsto q_s, Y \mapsto q_n\}$ and the state q_f. It is resolved by adding transitions to recognise $c(a(q_s), f(q_n))$ into q_f. Normalisation finds and reuses the transition $a(q_s) \rightarrowtail q_a$. It has to create a new state q_{N1} such that $f(q_n) \rightarrowtail q_{N1}$, and q_{N2} such that $c(q_a, q_{N1}) \rightarrowtail q_{N2}$. We then add $q_{N2} \overset{\mathfrak{R}}{\rightarrowtail} q_f$, and have produced $\mathcal{J}_R^{(PC_1)}(\mathcal{A}_0)$.

Another critical pair is CP_2 in $\mathcal{A}_0$ with the rule $a(s(X)) \to X$, the substitution $\sigma_2 = \{X \mapsto q_0\}$ and the state q_a. It is resolved by adding to $\mathcal{J}_R^{(PC_1)}(\mathcal{A}_0)$ the transition $q_0 \overset{\mathfrak{R}}{\rightarrowtail} q_a$, producing $\mathcal{J}_R^{(PC_1, PC_2)}(\mathcal{A}_0)$.

There is no more critical pair in $\mathcal{A}_0$: thus $\mathcal{C}_R(\mathcal{A}_0) = \mathcal{J}_R^{(PC_1, PC_2)}(\mathcal{A}_0)$. There is a new critical pair in $\mathcal{C}_R(\mathcal{A}_0)$ with $f(n) \to n$, the empty substitution and state q_{N1}. 30 ◁

3.2 Equational merging

Since completion of a critical pair can create new critical pairs, the process fuels itself, which is problematic for obtaining a fix-point. Equational merging is a way of countering this phenomenon at the cost of precision that is parametrised by equations over $T(\Sigma)$.

Definition 31 (Situation of application of an equation).

Given an equation $s = t$, an automaton $\mathcal{A}$, a $\theta : \mathcal{X} \to Q_{\mathcal{A}}$ and states q_1 and q_2, we say that $(s = t, \theta, q_1, q_2)$ is a situation of application in $\mathcal{A}$ if (see Figure 1(b))

1. $s\theta \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} q_1$,
2. $t\theta \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} q_2$ and
3. $q_1 \overset{\mathfrak{E}}{\underset{\mathcal{A}}{\not\leftrightarrow}} q_2$. 31 ◀

Definition 32 (Application of an equation).
Given $(s = t, \theta, q_1, q_2)$ a situation of application in $\mathcal{A}$, applying the underlying equation in it consists in adding transitions $q_1 \overset{\mathfrak{E}}{\rightarrowtail} q_2$ and $q_2 \overset{\mathfrak{E}}{\rightarrowtail} q_1$ to $\mathcal{A}$. This produces a new automaton $\mathcal{A}'$ and we note $\mathcal{A} \rightsquigarrow_E \mathcal{A}'$. 32 ◀

Remark. In [GR10], q_1 and q_2 were merged by "renaming" q_2 into q_1, ie. removing q_2 from $Q_{\mathcal{A}}$ and replacing every occurrence of q_2 by q_1 in the transitions of $\mathcal{A}$. This is equivalent to applying our method, then considering automaton $\mathcal{A}/_{\mathfrak{E}}$ (see definition 20) and finally choosing a representative (here q_1 for the class $\{q_1, q_2\}$) of each equivalence class of states.

Definition 33 (Simplified automaton).
Given an automaton $\mathcal{A}$ and a set of equations E, we call simplified automaton of $\mathcal{A}$ by E and note $\mathcal{S}_E(\mathcal{A})$ the automaton resulting from the successive application of all applicable equations in $\mathcal{A}$. 33 ◀

Remark ($\rightsquigarrow_E$ is confluent). Indeed, there is a unique automaton $\mathcal{A}^!$ that differs from $\mathcal{A}$ only by its $\mathfrak{E}$-transitions and is such $\mathcal{A}^{\not\mathfrak{R}} \rightsquigarrow_E^* (\mathcal{A}^!)^{\not\mathfrak{R}}$ and there is no more situation of application of equations in $(\mathcal{A}^!)^{\not\mathfrak{R}}$.

Definition 34 (Step of equational completion).
A step of equational completion is the composition of a step of exact completion, then equational simplification: $\mathcal{CE}_{R,E}(\mathcal{A}) = \mathcal{S}_E(\mathcal{C}_R(\mathcal{A}))$. 34 ◀

The following notion is part of an easier discourse about the R/E-coherence notion of [GR10].

Definition 35 (Coherent automaton).
Let $\mathcal{A}_0 = (\Sigma, Q, Q_F, \Delta)$ be a tree automaton and E a set of equations. The automaton $\mathcal{A}_0$ is said to be *E-coherent* if for all $q \in Q$, there exists $s \in \mathscr{L}(\mathcal{A}_0, q)$ such that $\mathscr{L}(\mathcal{A}_0, q) \subseteq [s]_E$. 35 ◀

3.3 Known results

We now recall the two main theorems of [GR10].

Theorem 36 (Correctness).
Let $\mathcal{A}_0$ be some automaton. Assume the equational completion procedure defined above terminates when applied to $\mathcal{A}_0$. Let $\mathcal{A}_*$ be the resulting fix-point automaton. If R is left-linear, then the calculated over-approximation is correct, that is

$$\mathscr{L}(\mathcal{A}_*) \supseteq R^*(\mathscr{L}(\mathcal{A}_0)).$$ 36 ■

We will make usage of E-coherence for the precision theorem.

Lemma 37.
Let $\mathcal{A}_0$ be a E-coherent automaton, R a left-linear TRS and $\mathcal{A}$ be an automaton obtained from $\mathcal{A}_0$ after several steps of equational completion with R, E. Then $\mathcal{A}^{\not\mathfrak{R}}$ is E-coherent and moreover, for all state q of $\mathcal{A}$, there exists $s \in \mathscr{L}(\mathcal{A}^{\not\mathfrak{R}}, q)$ such that $\mathscr{L}(\mathcal{A}, q) \subseteq (R/E)^*(s)$. 37 ■

Such an automaton is said to be R/E-coherent. The intuition behind this is the following: in the tree automaton, $\mathfrak{R}$-transitions represent rewriting steps and transitions of $\mathcal{A}^{\not\mathfrak{R}}$ recognize E-equivalence classes. More precisely, in a R/E-coherent tree automaton, if two terms s, t

are recognized into the same state q in $\mathcal{A}^{\not\mathfrak{R}}$ then they belong to the same E-equivalence class. Otherwise, if at least one $\mathfrak{R}$-transition is necessary to recognize, say, t into q then at least one step of rewriting was necessary to obtain t from s. In [GR10], the following theorem made an assumption of R/E-coherence for $\mathcal{A}_0$, but, given that $\mathcal{A}_0$ does not have any $\mathfrak{R}$-transition, $\mathcal{A}_0$ is R/E-coherent if and only if it is E-coherent.

Theorem 38 (Upper bound).
Let E be a set of equations, $\mathcal{A}_0$ a E-coherent tree automaton and R a left-linear TRS. If $\mathcal{A}$ is an automaton produced from $\mathcal{A}_0$ after several steps of equational completion with R, E, then

$$\mathscr{L}(\mathcal{A}) \subseteq (R/E)^*(\mathscr{L}(\mathcal{A}_0)) \qquad 38 \blacksquare$$

4 Termination of completion for functional programs

Now, we consider functional programs viewed as TRSs. We assume that such TRSs are left-linear, which is a common assumption on TRSs obtained from functional programs [BN98]. In this section, we will restrict ourselves to sufficiently complete constructor-based TRSs and will refer to them as *functional TRSs*. For the moment, types used in the functional program are not taken into account in the TRS, but they will be in a next section. First, we state a sufficient condition for completion to terminate and, next, we will specialize it for functional TRS.

Remark. In this section, we will be interested in counting the states of $\mathcal{A}/\mathfrak{E}$, where $\mathcal{A}$ is produced by equational completion. As we remarked just after definition 32, dealing with $\mathcal{A}/\mathfrak{E}$ amounts to act as if applying an equation would effectively merge (or rename) states. Therefore, we will assimilate $\mathcal{A}/\mathfrak{E}$ with $\mathcal{A}$ and consider that states in relation modulo $\underset{\mathcal{A}}{\overset{\mathfrak{E},*}{\rightleftarrows}}$ are merged. Note that with this convention, $\mathcal{A}^{\not\mathfrak{R}}$ (that is indeed $(\mathcal{A}/\mathfrak{E})^{\not\mathfrak{R}}$) has no epsilon-transition.

4.1 Ensuring termination of completion

In this section, we show that if $T(\Sigma)/_{\equiv_E}$ is finite then completion terminates by proving that completion produces no more new states than E-equivalence classes. Doing so is not possible for an arbitrary E because equational completion does not take reflexivity of $\equiv_E$ into account and there may exist $q \neq q'$ such that $s \underset{\mathcal{A}}{\rightarrowtail} q$ and $s \underset{\mathcal{A}}{\rightarrowtail} q'$.[2] We will enrich E with a set of reflexivity equations that solve this difficulty without altering $\equiv_E$.

Definition 39 (Set of reflexivity equations E^r).
For a given set of symbols Σ, $E^r = \{f(x_1,\ldots,x_k) = f(x_1,\ldots,x_k) \mid k \in \mathbb{N} \wedge f \in \Sigma_k\}$. 39 ◀

Note that for all set of equations E, the relation $\equiv_E$ is the same as $\equiv_{E \cup E^r}$. However, simplification with E^r transforms all automaton into an automaton $\mathcal{A}$ whose states coincide with equivalence classes of $\equiv_E$ and such that $\mathcal{A}^{\not\mathfrak{R}}$ is deterministic, as stated in the following lemmas.

Lemma 40.
Let $\mathcal{A}$ be some automaton, $s = t$ be some equation of E and $\theta : \mathcal{X} \to Q_{\mathcal{A}}$. If $s\theta \underset{\mathcal{S}_E(\mathcal{A})}{\overset{*}{\rightarrowtail}} q$ and $t\theta \underset{\mathcal{S}_E(\mathcal{A})}{\overset{*}{\rightarrowtail}} q'$, then $q \underset{\mathcal{S}_E(\mathcal{A})}{\overset{*}{\rightarrowtail}} q'$ or $q' \underset{\mathcal{S}_E(\mathcal{A})}{\overset{*}{\rightarrowtail}} q$. If $\mathcal{S}_E(\mathcal{A})$ has no epsilon-transition, then $q = q'$. 40 ■

[2] This behaviour is sometimes useful for solving verification problems.

Proof.

This translates the fact that, by definition, there cannot be a situation of application for equation $s = t$ in $\mathcal{S}_E(\mathcal{A})$, and we chose a representation of automata where states in relation modulo $\overset{\mathfrak{E},*}{\underset{\mathcal{A}}{\rightleftarrows}}$ are merged. □

Lemma 41.

For all tree automaton $\mathcal{A}$ without epsilon-transition and all set of equations E, if $E \supseteq E^r$, then $\mathcal{S}_E(\mathcal{A})$ is deterministic. 41 ■

Proof.

Since $\mathcal{A}$ has no epsilon-transition, neither does $\mathcal{S}_E(\mathcal{A})$. We prove this lemma by induction on the terms recognized by $\mathcal{S}_E(\mathcal{A})$. Let a be a constant such that $a \underset{\mathcal{S}_E(\mathcal{A})}{\overset{*}{\rightarrowtail}} q$ and $a \underset{\mathcal{S}_E(\mathcal{A})}{\overset{*}{\rightarrowtail}} q'$: by Lemma 40, $q = q'$. For the inductive case, let $t = f(t_1,\dots,t_k)$ such that $t \underset{\mathcal{S}_E(\mathcal{A})}{\overset{*}{\rightarrowtail}} q$ and $t \underset{\mathcal{S}_E(\mathcal{A})}{\overset{*}{\rightarrowtail}} q'$. By induction hypothesis, for each $i \in [\![1\,;k]\!]$, there exists a unique state q_i such that $t_i \underset{\mathcal{S}_E(\mathcal{A})}{\overset{*}{\rightarrowtail}} q_i$. Hence $f(q_1,\dots,q_k) \underset{\mathcal{S}_E(\mathcal{A})}{\overset{*}{\rightarrowtail}} q$ and $f(q_1,\dots,q_k) \underset{\mathcal{S}_E(\mathcal{A})}{\overset{*}{\rightarrowtail}} q'$. Since $f(x_1,\dots,x_k) = f(x_1,\dots,x_k) \in E^r$, by Lemma 40, $q = q'$. □

Corollary 42.

Let $\mathcal{A}$ be an automaton produced by some steps of equational completion with $E \supseteq E^r$. The automaton $\mathcal{A}^{\mathfrak{R}}$ is deterministic. 42 ■

Definition 43 (Set $E^c_{\mathcal{F}}$ of contracting equations for $\mathcal{F}$).

Let $\mathcal{F} \subseteq \Sigma$. The set of equations $E^c_{\mathcal{F}}$ is *contracting* (for $\mathcal{F}$) if its equations are of the form $u = u_{|p}$ with u linear and $p \neq \Lambda$ and if the set of normals forms of $T(\mathcal{F})$ w.r.t. TRS

$$\overrightarrow{E^c_{\mathcal{F}}} = \left\{ u \to u_{|p} \;\middle|\; u = u_{|p} \in E^c_{\mathcal{F}} \right\}$$

is finite. 43 ◀

Contracting equations define an upper bound on the number of states of a simplified automaton.

Lemma 44.

Simplification of a tree automaton $\mathcal{A}$ using a set E of equations such that $E \supseteq E^r \cup E^c_{\Sigma}$ with E^c_{Σ} contracting ends up on an automaton $\mathcal{S}_E(\mathcal{A})$ having no more states than terms in $\textsc{Irr}(\overrightarrow{E^c_{\Sigma}})$. 44 ■

Proof.

Assume that no term of $\textsc{Irr}(\overrightarrow{E^c_{\Sigma}})$ is recognised by $\mathcal{S}_E(\mathcal{A})$. Then, for all term s such that $s \underset{\mathcal{S}_E(\mathcal{A})}{\overset{*}{\rightarrowtail}} q$, we know that s is not in normal form w.r.t. $\overrightarrow{E^c_{\Sigma}}$. As a result, the left-hand side of an equation of E^c_{Σ} can be applied to s. This means that there exists an equation $u = u_{|p}$, a ground context C and a substitution θ such that $s = C[u\theta]$. Furthermore, since $s \underset{\mathcal{S}_E(\mathcal{A})}{\overset{*}{\rightarrowtail}} q$, we know that $C[u\theta] \underset{\mathcal{S}_E(\mathcal{A})}{\overset{*}{\rightarrowtail}} q$ and that there exists a state q' such that $C[q'] \underset{\mathcal{S}_E(\mathcal{A})}{\overset{*}{\rightarrowtail}} q$ and $u\theta \underset{\mathcal{S}_E(\mathcal{A})}{\overset{*}{\rightarrowtail}} q'$. From

$u\theta \underset{\mathcal{S}_E(\mathcal{A})}{\overset{*}{\rightarrowtail}} q'$, we know that all substerms of $u\theta$ are recognised by at least one state in $\mathcal{S}_E(\mathcal{A})$. Thus, there exists a state q'' such that $u_{|p}\theta \underset{\mathcal{S}_E(\mathcal{A})}{\overset{*}{\rightarrowtail}} q''$. We thus have a situation of application of the equation $u = u_{|p}$ in the automaton. Since $\mathcal{S}_E(\mathcal{A})$ is simplified, we thus know that $q' = q''$. As mentioned above, we know that $C[q'] \underset{\mathcal{S}_E(\mathcal{A})}{\overset{*}{\rightarrowtail}} q$. Hence $C[u_{|p}\theta] \underset{\mathcal{S}_E(\mathcal{A})}{\overset{*}{\rightarrowtail}} C[q'] \underset{\mathcal{S}_E(\mathcal{A})}{\overset{*}{\rightarrowtail}} q$. If $C[u_{|p}\theta]$ is not in normal form w.r.t. $\overrightarrow{E^c_\Sigma}$ then we can do the same reasoning on $C[u_{|p}\theta] \underset{\mathcal{S}_E(\mathcal{A})}{\overset{*}{\rightarrowtail}} q$ until getting a term that is in normal form w.r.t. $\overrightarrow{E^c_\Sigma}$ and recognised by the same state q. Thus, this contradicts the fact that $\mathcal{S}_E(\mathcal{A})$ recognises no term of $\textsc{Irr}(\overrightarrow{E^c_\Sigma})$ are disjoint.

Then, by definition of E^c_Σ, $\textsc{Irr}(\overrightarrow{E^c_\Sigma})$ is finite. Let $\{t_1, \ldots, t_n\}$ be the subset of $\textsc{Irr}(\overrightarrow{E^c_\Sigma})$ recognised by $\mathcal{S}_E(\mathcal{A})$. Let $q_1, \ldots, q_n$ be the states recognising $t_1, \ldots, t_n$ respectively. We know that there is a finite set of states recognising $t_1, \ldots, t_n$ because $E \supseteq E^r$ and Corollary 42 entails that $\mathcal{S}_E(\mathcal{A})^{\mathcal{R}}$ is deterministic. Now, for all term s recognised by a state q in $\mathcal{S}_E(\mathcal{A})$, i.e. $s \underset{\mathcal{S}_E(\mathcal{A})}{\overset{*}{\rightarrowtail}} q$, we can use a reasoning similar to the one carried out above and show that q is equal to one state of $\{q_1, \ldots, q_n\}$ recognising normal forms of $\overrightarrow{E^c_\Sigma}$ in $\mathcal{S}_E(\mathcal{A})$. Finally, there are at most $card(\textsc{Irr}(\overrightarrow{E^c_\Sigma}))$ states in $\mathcal{S}_E(\mathcal{A})$. □

Now it is possible to state the Theorem guaranteeing the termination of completion if the set of equations E contains E^r and a set of contracting equations E^c_Σ.

Theorem 45.
Let $\mathcal{A}$ be a tree automaton, R a left linear TRS and E a set of equations. If $E \supseteq E^r \cup E^c_\Sigma$ with E^c_Σ contracting then completion of $\mathcal{A}$ by R and E terminates. 45 ■

Proof.
For completion to diverge it must produce infinitely many new states. This is impossible with sets of equation E^c_Σ and E^r as shown in Lemma 44. □

Remark. Note that if E contains E^r and a set of contracting equations, $\equiv_E$ is of finite index (there are finitely many equivalence classes in $T(\Sigma)/_{\equiv_E}$). However, finiteness of $T(\Sigma)/_{\equiv_E}$ alone is not enough to guarantee termination of equational completion as defined in Section 3. For instance, if $\Sigma = \{f, g, a\}$ and $E = \{f(x) = g(x), g(x) = x\}$ then $T(\Sigma)/_{\equiv_E}$ is finite but completion of a tree automaton recognising $f(a)$ with $f(x) \to f(f(x))$ will not terminate because terms having g symbols will not be recognised and $g(x) = x$ will not be applied.

For TRSs representing functional programs, defining contraction equations of $E^c_\mathcal{C}$ on $\mathcal{C}$ rather than on Σ is enough to guarantee termination of completion. This is more convenient and also closer to what is usually done in static analysis where abstractions are usually defined on data and not on function applications. Since the TRSs we consider are sufficiently complete, any term of $T(\Sigma)$ can be rewritten into a data-term of $T(\mathcal{C})$. As above, using equations of $E^c_\mathcal{C}$ we are going to ensure that the data-terms of the computed languages will be recognised by a bounded set of states. To lift-up this property to $T(\Sigma)$ it is enough to ensure that $\forall s, t \in T(\Sigma)$ if $s \to_R t$ then s and t are recognised by equivalent states. This is the role of the set of equations E_R.

Definition 46 (E_R).
Let R be a TRS, the set of R-equations is $E_R = \{l = r \mid l \to r \in R\}$. 46 ◀

Theorem 47.
Let $\mathcal{A}$ be a tree automaton, R a sufficiently complete left-linear TRS and E a set of equations. If $E \supseteq E^r \cup E^c_{\mathcal{C}} \cup E_R$ with $E^c_{\mathcal{C}}$ contracting then completion of $\mathcal{A}$ by R and E terminates. 47 ■

Proof.
Firstly, we can use a reasoning similar to the one used in the proof of Lemma 44 to show that the number of states recognising terms of $T(\mathcal{C})$ are in finite number. Let $G \subseteq T(\mathcal{C})$ be the set of normal forms of $T(\mathcal{C})$ w.r.t. $\overrightarrow{E^c_{\mathcal{C}}}$. Since $E \supseteq E^r \cup E^c_{\mathcal{C}}$, like in the proof of Lemma 44, we can show that in any completed automaton, terms of $T(\mathcal{C})$ are recognised by no more states than terms in G. Secondly, since R is sufficiently complete, for all term $s \in T(\Sigma) \setminus T(\mathcal{C})$ we know that there exists a term $t \in T(\mathcal{C})$ such that $s \to^*_R t$. The fact that $E \supseteq E_R$ guarantees that s and t will be recognised by equivalent states in the completed (and simplified) automaton. Since the number of states necessary to recognise $T(\mathcal{C})$ is finite, so is the number of states necessary to recognise terms of $T(\Sigma)$. □

4.2 Ensuring termination of completion for functional TRS with types

To exploit the types of the functional program, we now see Σ as a many-sorted signature whose set of sorts is $\mathcal{S}$. Each symbol $f \in \Sigma$ is associated to a profile $f : S_1 \times \ldots \times S_k \mapsto S$ where $S_1, \ldots, S_k, S \in \mathcal{S}$ and k is the arity of f. Well-sorted terms are inductively defined as follows: $f(t_1, \ldots, t_k)$ is a well-sorted term of sort S if $f : S_1 \times \ldots \times S_k \mapsto S$ and $t_1, \ldots, t_k$ are well-sorted terms of sorts $S_1, \ldots, S_k$, respectively. We denote by $T(\Sigma, \mathcal{X})^{\mathcal{S}}$, $T(\Sigma)^{\mathcal{S}}$ and $T(\mathcal{C})^{\mathcal{S}}$ the set of well-sorted terms, ground terms and constructor terms, respectively. Note that we have $T(\Sigma, \mathcal{X})^{\mathcal{S}} \subseteq T(\Sigma, \mathcal{X})$, $T(\Sigma)^{\mathcal{S}} \subseteq T(\Sigma)$ and $T(\mathcal{C})^{\mathcal{S}} \subseteq T(\mathcal{C})$. We assume that R and E are *sort preserving*, *i.e.* that for all rule $l \to r \in R$ and all equation $u = v \in E$, $l, r, u, v \in T(\Sigma, \mathcal{X})^{\mathcal{S}}$, l and r have the same sort and so do u and v. Again, this assumption on R is natural if R is the translation of a well-typed functional program. We still assume that the (sorted) TRS is sufficiently complete, which is defined in a similar way except that it holds only for well-sorted terms, i.e. for all $s \in T(\Sigma)^{\mathcal{S}}$ there exists a term $t \in T(\mathcal{C})^{\mathcal{S}}$ such that $s \to^*_R t$. We slightly refine the definition of contracting equations as follows. For all sort S, if S has a unique constant symbol we note it c^S.

Definition 48 (Set $E^c_{\mathcal{F},\mathcal{S}}$ of contracting equations for $\mathcal{F}$ and $\mathcal{S}$).
Let $\mathcal{F} \subseteq \Sigma$. The set of well-sorted equations $E^c_{\mathcal{F},\mathcal{S}}$ is *contracting* (for $\mathcal{F}$) if its equations are of the form (a) $u = u_{|p}$ with u linear and $p \neq \Lambda$, or (b) $u = c^S$ with u of sort S, and if the set of normal forms of $T(\mathcal{F})^{\mathcal{S}}$ w.r.t. the TRS

$$\overrightarrow{E^c_{\mathcal{F},\mathcal{S}}} = \left\{ u \to v \;\middle|\; u = v \in E^c_{\mathcal{F},\mathcal{S}} \wedge (v = u_{|p} \vee v = c^S) \right\}$$

is finite. 48 ◀

The termination theorem for completion of the sorted TRSs is close to the previous one except that that it takes into account the refined version of contracting equations and that it needs E-coherence of $\mathcal{A}_0$. This is useful to ensure that terms recognised by completed automata are well-sorted.

Theorem 49.

Let $\mathcal{A}_0$ be a tree automaton recognising well-sorted terms, R a sufficiently complete sort-preserving left-linear TRS and E a sort-preserving set of equations. If $E \supseteq E^r \cup E^c_{\mathcal{C},\mathcal{S}} \cup E_R$ with $E^c_{\mathcal{C},\mathcal{S}}$ contracting and $\mathcal{A}_0$ is E-coherent then completion of $\mathcal{A}_0$ by R and E terminates. [49] ■

Proof.

Let $\mathcal{A}$ be any tree automaton obtained by completion of $\mathcal{A}_0$ by R and E. As in Lemma 44, from finiteness of the set normal forms of $T(\mathcal{C})^{\mathcal{S}}$ w.r.t. $\overrightarrow{E^c_{\mathcal{C},\mathcal{S}}}$, we can obtain finiteness of the set of states recognising terms of $T(\mathcal{C})^{\mathcal{S}}$ in the completed automaton. The only slight difference comes from rules of the form $u = c^S$. If a term $s \in T(\mathcal{C})^{\mathcal{S}}$ is not in normal form w.r.t. $\overrightarrow{E^c_{\mathcal{C},\mathcal{S}}}$ because the rule $u \to c^S$ applies then we have: $s = C[u\sigma] \underset{\mathcal{A}}{\overset{*}{\rightarrowtail}} q$. Thus there exists a state q' such that $u\sigma \underset{\mathcal{A}}{\overset{*}{\rightarrowtail}} q'$. Since c^S is the only constant of sort S and since $u\sigma$ is of sort S, we know that c^S is necessarily a subterm of $u\sigma$. Thus there exists a state q'' such that $c^S \underset{\mathcal{A}}{\overset{*}{\rightarrowtail}} q''$ and since completed automata are simplified, $q' = q''$ and finally $C[c^S] \underset{\mathcal{A}}{\overset{*}{\rightarrowtail}} q$. As in Lemma 44, we can iterate the process until finding a normal form of $\overrightarrow{E^c_{\mathcal{C},\mathcal{S}}}$. This entails the finiteness of the set of states recognising terms of $T(\mathcal{C})^{\mathcal{S}}$ in $\mathcal{A}$. Then, as in the proof of Theorem 47 we can use the fact that $E \supseteq E_R$ to have that terms of $T(\Sigma)^{\mathcal{S}}$ are recognised in $\mathcal{A}$ using a finite set of states. What remains to be proved is that $\mathcal{A}$ recognises only well-sorted terms, i.e. that it recognises no term of $T(\Sigma) \setminus T(\Sigma)^{\mathcal{S}}$. This is true because $\mathcal{A}_0$ is E-coherent, and by Theorem 38, $\mathcal{L}(\mathcal{A}) \subseteq (R/E)^*(\mathcal{L}(\mathcal{A}_0))$. Besides, R and E being sort-preserving, so is R/E. Thus, terms of $\mathcal{L}(\mathcal{A})$ are all well-sorted. □

5 Experiments

All completions were performed using Timbuk and TimbukCEGAR [BBGL12]. All the $\mathcal{IRR}(R)$ automata constructions and automata intersections were performed using Taml. All completion results have been certified by Coq [BC04] using the Coq-extracted completion checker [BGJ08]. All those tools are freely available on Timbuk web site [Tim12].

5.1 An introductory example

```
Ops append:2 rev:1 nil:0 cons:2 a:0 b:0
Vars X Y Z U Xs
TRS R
append(nil,X)->X                          rev(nil)->nil
append(cons(X,Y),Z)->cons(X,append(Y,Z))  rev(cons(X,Y))->append(rev(Y),cons(X,nil))

Automaton A0 States q0 qla qlb qnil qf qa qb Final States q0 Transitions
rev(qla)->q0        cons(qb,qnil)->qlb    cons(qa,qla)->qla    nil->qnil
cons(qa,qlb)->qla   a->qa                 cons(qb,qlb)->qlb    b->qb

Equations E Rules
append(nil,X)=X                           a=a  b=b  nil=nil      cons(X,cons(Y,Z))=cons(Y,Z)
append(cons(X,Y),Z)=cons(X,append(Y,Z))   cons(X,Y)=cons(X,Y)
rev(nil)=nil                              append(X,Y)=append(X,Y)
rev(cons(X,Y))=append(rev(Y),cons(X,nil)) rev(X)=rev(X)
```

In this example, the TRS R encodes the classical *reverse* and *append* functions. The language recognised by automaton $\mathcal{A}_0$ is the set of terms of the form $rev(l_a)$ where l_a can be any non empty list of the form $[a,a,\ldots,b,b,\ldots]$. Note that there is at least one a and one b in the list. We assume that $\mathcal{S} = \{T, list\}$ and sorts for symbols are the following: $a : T$, $b : T$, $nil : list$, $cons : T \times list \mapsto list$, $append : list \times list \mapsto list$ and $rev : list \mapsto list$. Now, to use Theorem 49, we need to prove each of its assumptions. The set E of equations contains E_R, E^r and $E^c_{\mathcal{C},\mathcal{S}}$. The set of Equations $E^c_{\mathcal{C},\mathcal{S}}$ is contracting because the automaton $\mathcal{IRR}(\overrightarrow{E^c_{\mathcal{C},\mathcal{S}}})$ recognises a finite language. This automaton can be computed using Taml: it is the intersection between the automaton $\mathcal{A}_{T(\mathcal{C})^{\mathcal{S}}}$[3] recognising $T(\mathcal{C})^{\mathcal{S}}$ and the automaton $\mathcal{IRR}(\{cons(X, cons(Y,Z)) \to cons(Y,Z)\})$:

States `q2 q1 q0` **Final States** `q0 q1 q2` **Transitions** `b->q2 a->q2 nil->q1 cons(q2,q1)->q0`

It is easy to see that E and R are sort preserving and that $\mathcal{A}_0$ recognises well-sorted terms. We can also prove sufficient completeness of R on $\mathscr{L}(\texttt{A0})$ using, for instance, Maude [CDE+09] or even Timbuk [Gen98] itself. The last assumption of Theorem 49 to prove is that $\mathcal{A}_0$ is E-coherent. This can be shown by remarking that each state $\mathcal{A}_0$ recognises at least one term and that for all state q such that $s \overset{*}{\underset{\mathcal{A}_0}{\rightarrowtail}} q$ and $t \overset{*}{\underset{\mathcal{A}_0}{\rightarrowtail}} q$ then $s \equiv_E t$. For instance $cons(b, cons(b, nil)) \overset{*}{\underset{\mathcal{A}_0}{\rightarrowtail}} q_{lb}$ and $cons(b, nil) \overset{*}{\underset{\mathcal{A}_0}{\rightarrowtail}} q_{lb}$ and $cons(b, cons(b, nil)) \equiv_E cons(b, nil)$. Thus, completion is guaranteed to terminate: after 4 completion steps (7 ms) we obtain the following fixpoint automaton $\mathcal{A}_*$:

States `q5 q7 q8 q11` **Final States** `q11` **Transitions**

```
b -> q8                  nil -> q5                rev(q5) -> q5          append(q5,q11) -> q11
append(q11,q11) -> q11   cons(q7,q5) -> q11       cons(q7,q11) -> q11    cons(q8,q5) -> q11
cons(q8,q11) -> q11      rev(q11) -> q11          a -> q7
```

To restrain the language to normal forms, it is necessary to compute the intersection with $\textsc{Irr}(R)$.Since we are dealing with sufficiently complete TRSs, we know that $\textsc{Irr}(R) \subseteq T(\mathcal{C})^{\mathcal{S}}$. Thus, we can use again the tree automaton $\mathcal{A}_{T(\mathcal{C})^{\mathcal{S}}}$. If we compute the intersection of $\mathcal{A}_*$ with the automaton $\mathcal{A}_{T(\mathcal{C})^{\mathcal{S}}}$ we obtain the automaton:

States `q3 q2 q1 q0` **Final States** `q3` **Transitions**

```
a->q0  nil->q1  b->q2  cons(q0,q1)->q3  cons(q0,q3)->q3  cons(q2,q1)->q3  cons(q2,q3)->q3
```

which recognises any (non empty) flat list of a and b. Thus, our analysis preserved the property that the result cannot be the empty list but lost the order of the elements in the list. This is not surprising if we have a closer look to our set of equations E. In particular, the equation $cons(X, cons(Y,Z)) = cons(X,Z)$ makes $cons(a, cons(b, nil))$ equal to $cons(a, nil)$. It is possible to refine by hand $E^c_{\mathcal{C},\mathcal{S}}$ as follows:

```
cons(a, cons(a,X))=cons(a,X)  cons(b,cons(b,X))=cons(b,X)  cons(a,cons(b,cons(a,X)))=cons(a, X)
```

This set of equations avoids the previous problem. Again E verifies the conditions of Theorem 49 and completion is thus guaranteed to terminate. The result is the automaton:

[3] Such an automaton can be automatically defined. It has one state per sort and one transitions per constructor. For instance, on our example $\mathcal{A}_{T(\mathcal{C})^{\mathcal{S}}}$ will have transitions: $a \to qT$, $b \to qT$, $cons(qT, qlist) \to qlist$ and $nil \to qlist$.

States q0 q1 q5 q7 q8 q11 q17 **Final States** q0 **Transitions**

```
nil -> q5  rev(q5) -> q5  append(q5,q11) -> q11  append(q11,q11) -> q11  cons(q8,q5) -> q11
cons(q8,q11) -> q11       rev(q11) -> q11       append(q5,q17) -> q17   append(q17,q17) -> q17
cons(q7,q5) -> q17        cons(q7,q17) -> q17   cons(q7,q1) -> q1       cons(q7,q11) -> q1
b -> q8                   append(q0,q17) -> q0  append(q11,q17) -> q0   cons(q8,q0) -> q0
cons(q8,q17) -> q0        rev(q1) -> q0         a -> q7
```

This time, intersection with $\mathcal{A}_{T(\mathcal{C})^S}$ gives:

States q4 q3 q2 q1 q0 **Final States** q4 **Transitions**

```
a->q1  b->q3  nil->q0  cons(q1,q0)->q2  cons(q1,q2)->q2  cons(q3,q2)->q4  cons(q3,q4)->q4
```

This automaton exactly recognises lists of the form $[b,b,\ldots,a,a,\ldots]$ with at least one b and one a, as expected. However, equation tuning by hand is difficult. Hopefully, this can be automated using a tree automata completion equipped with a counter-example guided approximation refinement (a.k.a. CEGAR) [BBGL12]. This is what we do in the following example.

5.2 Higher-order function example

Here we show how to take higher-order functions into account and the benefit of a CEGAR for approximation refinement. We choose to illustrate the two aspects on an example taken from [OR11]. The encoding of higher-order functions into first order terms is borrowed from [Jon87]: defined symbols become constants, constructor symbols remain the same, and an additional *application* operator '@' of arity 2 is introduced. For instance, the function *nz* testing if a natural is different from 0 and the higher-order function $filter$ filtering out all elements that do not satisfy a predicate are represented by the following TRS:

$$
\begin{aligned}
&@(nz, zero) \to false\\
&@(nz, s(X)) \to true\\
&@(@(filter, X), nil) \to nil\\
&@(@(filter, X), cons(Y, Z)) \to if(@(X, Y), cons(Y, @(@(filter, X), Z)), @(@(filter, X), Z))\\
&if(true, X, Y) \to X\\
&if(false, X, Y) \to Y
\end{aligned}
$$

The objective of [OR11] is to compute an approximation of all possible results of the function call $(filter\ nz\ l)$ where l is any list of naturals. To respect the presentation used in [OR11], we also give the initial set of terms as a grammar generating terms of the form $(filter\ nz\ l)$ with l any list of naturals. As a result, the initial automaton recognises only a non terminal symbol genS and the TRS contains the rules used to generate the language. The corresponding Timbuk specification is the following, where app stands for @.

Ops app:2 filter:0 zero:0 s:1 nz:0 nil:0 cons:2 if:3 true:0 false:0 genl:1
Vars F X Y Z U Xs X2 X3 Y2 Z2
TRS R

```
app(nz,zero) -> false    app(nz,s(X)) -> true
if(true,X,Y) -> X        if(false,X,Y) -> Y       app(app(filter,X),nil) -> nil
app(app(filter,X),cons(Y,Z)) -> if(app(X,Y),cons(Y,app(app(filter,X),Z)),app(app(filter,X),Z))

genS -> app(app(filter,nz),genl(zero))            genl(X) -> cons(X,genl(zero))
genl(X) -> nil                                    genl(X) -> genl(s(X))
```

Automaton A0 **States** qf **Final States** qf **Transitions** genS -> qf

Equations E **Rules**

```
if(true,X,Y)=X                                  app(X,Y)=app(X,Y)          s(X)=zero
if(false,X,Y)=Y                                 filter=filter              cons(X,Y)=Y
app(nz,zero)=false                              zero=zero  s(X)=s(X)
app(nz,s(X))=true                               nz=nz  nil=nil
app(app(filter,X),nil)=nil                      cons(X,Y)=cons(X,Y)
app(app(filter,X),cons(Y,Z))=                   if(X,Y,Z)=if(X,Y,Z)
  if(app(X,Y2),cons(Y,app(app(filter,X2),Z)),   true=true  false=false
        app(app(filter,X3),Z2))                 genl(X)=genl(X)
```

Automaton Bad **States** ql ql0 q1 qz qnil **Final States** ql0 **Transitions**

```
cons(qz,ql)->ql0   cons(q1,ql)->ql     cons(q1,qnil)->ql  cons(q1,ql0)->ql0  s(qz)->q1
cons(qz,ql0)->ql0  cons(qz,qnil)->ql0  nil->qnil          s(q1)->q1          zero->qz
```

Automaton ATCS **States** qnat qlist **Final States** qnat qlist **Transitions**

```
zero->qnat   s(qnat)->qnat  nil->qlist  cons(qnat,qlist)->qlist
```

As in previous example, we can check that R, E and $\mathcal{A}_0$ respect the assumptions of Theorem 49. Note that contracting equations of $E^c_{\mathcal{C},\mathcal{S}}$ are very drastic: the effect of the equation $s(X) = zero$ is to merge all naturals together and the effect of $cons(X, Y) = Y$ is to scramble all lists together. The Bad automaton defines the language of undesirable terms: any list of naturals where there is at least one 0. As soon as automatic refinement is used, termination is no longer guaranteed. This is a common limitation of all CEGAR techniques like in [OR11]. However, within 58ms TimbukCEGAR achieves 8 completion steps, 5 refinement steps and produces a completed tree automaton that have no term in common with $\mathscr{L}$(Bad). Its product with the automaton $\mathcal{A}_{T(\mathcal{C})^{\mathcal{S}}}$ gives the automaton recognising lists of naturals strictly greater to 0, which is the expected result:

States q3 **Final States** q3 **Transitions**

```
nil->q2   nil->q3  zero->q0  s(q0)->q1  s(q1)->q1  cons(q1,q3)->q3  cons(q1,q2)->q3
```

This automaton recognizes lists of naturals strictly greater than 0 which is the expected language of normal forms of the higher-order call $(filter\ nz\ l)$ with l any list of naturals. Example 8 of [OR11] can be solved in the same way.

Ops app:2 map2:0 kzero:0 kone:0 one:0 zero:0 nil:0 cons:2 genS:0
Vars F X Y Z U X2

TRS R1

```
app(app(app(map2, X), Y), nil) -> nil
app(app(app(map2, X), Y), cons(Z, U)) ->
            cons(app(X, Z), app(app(app(map2, Y), X), U))

app(kzero, X) -> zero
app(kone, X) -> one

genS -> nil
genS -> cons(zero,genS)
```

Set A0

```
app(app(app(map2, kzero), kone),genS)
```

Automaton Bad

```
States qf q0 q1 qok0 qok1 qnil
Final States qf
Transitions
cons(q1,qok1) -> qf      cons(q0,qok0) -> qf      cons(q0,qf) -> qf       cons(q1,qf) -> qf
cons(q0,qok1) -> qok0    cons(q1,qok0) -> qok1    cons(q0,qnil) -> qok0   cons(q1,qnil) -> qok1
nil -> qnil              zero -> q0               one -> q1

Equations Simple
Rules
app(app(app(map2, X), Y), nil) = nil                app(X,Y)=app(X,Y)    cons(X,Y)=Y
app(app(app(map2, X), Y), cons(Z, U)) =             zero=zero
    cons(app(X, Z), app(app(app(map2, Y), X2), U))  one=one
app(kzero, X) = zero                                nil=nil
app(kone, X) = one                                  cons(X,Y)=cons(X,Y)
                                                    kone=kone
                                                    kzero=kzero
```

On this example, timbukCEGAR achieves 4 completion steps and 2 refinement steps and gives in 36ms the following tree automaton with 24 transitions:

```
Automaton Acomp
States q0 q1 q2 q3 q4 q5 q6 q7 q8 q9 q10 q11 q12 q13 q14 q15 q16 q17 q18
Final States q6
Transitions
cons(q10,q16) -> q14    app(q11,q1) -> q12   app(q0,q3) -> q11   app(q1,q8) -> q10
genS -> q9              cons(q8,q9) -> q9    zero -> q8          nil -> q7
app(q4,q9) -> q6        app(q2,q3) -> q4     kone -> q3          app(q0,q1) -> q2
one -> q17              app(q3,q8) -> q17    kzero -> q1         app(q12,q9) -> q16
cons(q17,q6) -> q16     map2 -> q0           q14 -> q6           q10 -> q8
q8 -> q10               q7 -> q16            q7 -> q9            q7 -> q6
```

Again, the intersection of this automaton with $\mathcal{A}_{T(\mathcal{C})}$ results in the automaton:

```
States q5:0 q4:0 q3:0 q2:0 q1:0 q0:0
Final States q5
Transitions
nil -> q1          nil -> q4           nil -> q5           one -> q2           zero -> q3
cons(q3,q1) -> q5  cons(q3,q1) -> q0   cons(q3,q4) -> q0   cons(q2,q0) -> q4   cons(q2,q1) -> q4
cons(q2,q5) -> q4  cons(q3,q4) -> q5
```

This automaton recognizes lists of the form `[zero,one,zero,one...]`, which is the expected result.

6 Adaptation to innermost strategies

6.1 Introduction

The classical equational completion procedure with a left-linear TRS R produces a correct over-approximaton of $R^*(L_0)$ whenever it terminates. As $R^*_{\text{in}}(L_0) \subseteq R^*(L_0)$, this is a correct over-approximation of $R^*_{\text{in}}(L_0)$ as well. Still, we would like to refine this procedure to deal more precisely with R_{in}. Indeed, there are some critical pairs that we would not want to complete because they do not correspond to any *innermost* rewriting situation.

Example 50.

Let us look at Example 30. The rewriting of $f(c(a(s(0)),n))$ into $c(a(s(0)),f(n))$ does not conform to innermost strategy because $a(s(0))$ is not a normal form. We would like to abstain from completing CP_1 of Example 30. 50 ◁

Due to the definition of innermost rewriting, we will need to discriminate between normal forms and terms reducible by R. To do so is possible using the automaton $\mathcal{IRR}(R)$ (see Theorem 25). It is possible to build a product between $\mathcal{A}$ and $\mathcal{IRR}(R)$, the tree automaton recognising the normal forms of R. Let $\mathcal{A}_\circ$ be an automaton recognising the initial language. Completion will start with $\mathcal{A}_0 = \mathcal{A}_\circ \times \mathcal{IRR}(R)$. This automaton enjoys the following property, which we will be useful to prove correctness.

Definition 51 (Consistency with $\mathcal{IRR}(R)$).

A pair automaton $\mathcal{A}$ is said to be consistent with $\mathcal{IRR}(R)$ if, for any configuration c and any state $\langle q,p\rangle$ of $\mathcal{A}$, $\Pi_2(c)$ is a configuration and p is a state of $\mathcal{IRR}(R)$, and if $c \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} \langle q,p\rangle$ then $\Pi_2(c) \overset{*}{\underset{\mathcal{IRR}(R)}{\rightarrowtail}} p$. 51 ◀

The problem is, then, to update this product after each completion step: each completion step produces new states and new transitions that are not in $\mathcal{A}$ and thus not covered by the product. A naive way to do this would be to re-compute the product after each new completion step. However, this is rather inefficient: if N_A is the number of transitions of $\mathcal{A}$ and N_I is the number of transition of $\mathcal{IRR}(R)$, the product automaton can have $N_A \times N_I$ transitions, and this product will be done after each step of completion. Hence, for n steps of completion, the number of transitions can be exponential w.r.t. n, i.e. $N_A \times (N_I)^n$. We propose a more efficient solution where completion only produces *transitions of the product automaton*.

In the next subsections, we will restate the definitions used by equational completion to adapt them to our new framework of pair automata. *The TRS R is always supposed left-linear.* Some parts of them might look tricky, and they are indeed tricks to preserve the property of consistency with $\mathcal{IRR}(R)$.

6.2 Equational simplification

Definition 52 (Situation of application of an equation).

Given an equation $s = t$, an automaton $\mathcal{A}$, a $\theta : \mathcal{X} \to Q_\mathcal{A}$ and states $\langle q_1,p_1\rangle$ and $\langle q_2,p_2\rangle$, we say that $(s = t, \theta, \langle q_1,p_1\rangle, \langle q_2,p_2\rangle)$ is a situation of application in $\mathcal{A}$ if

1. $s\theta \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} \langle q_1,p_1\rangle$,
2. $t\theta \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} \langle q_2,p_2\rangle$,
3. $\langle q_1,p_1\rangle \overset{\mathfrak{E}}{\underset{\mathcal{A}}{\not\rightleftarrows}} \langle q_2,p_2\rangle$ and
4. $p_1 = p_2$. 52 ◀

Definition 53 (Application of an equation).

Given $(s = t, \theta, \langle q_1,p_1\rangle, \langle q_2,p_1\rangle)$ a situation of application in $\mathcal{A}$, applying the underlying equation in it consists in adding transitions $\langle q_1,p_1\rangle \overset{\mathfrak{E}}{\rightarrowtail} \langle q_2,p_1\rangle$ and $\langle q_2,p_1\rangle \overset{\mathfrak{E}}{\rightarrowtail} \langle q_1,p_1\rangle$ to $\mathcal{A}$. 53 ◀

Lemma 54.
Applying an equation preserves consistency with $\mathcal{IRR}(R)$. 54 ■

Proof.
Let $\mathcal{A}$ be a consistent with $\mathcal{IRR}(R)$ automaton whose set of states is Q, let $\mathcal{B}$ result from the adjunction of transition $\langle q_1, p_1\rangle \rightarrowtail \langle q_2, p_1\rangle$ to $\mathcal{A}$ due to the application of some equation. Notice that this is sufficient because of the symmetry between q_1 and q_2. We proceed by induction on k, the number of times the transition $\langle q_1, p_1\rangle \rightarrowtail \langle q_2, p_1\rangle$ occurs in the path $c \overset{*}{\underset{\mathcal{B}}{\rightarrowtail}} \langle q, p\rangle$ where c is a configuration and $\langle q, p\rangle$ is a state of $\mathcal{B}$. If there is no occurrence, then $c \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} \langle q, p\rangle$ and by consistency of $\mathcal{A}$, $\Pi_2(c) \overset{*}{\underset{\mathcal{IRR}(R)}{\rightarrowtail}} p$.

Suppose the property is true for some k and there is a context C on $T(\Sigma, Q)$ such that $c \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} C[\langle q_1, p_1\rangle] \underset{\mathcal{B}}{\rightarrowtail} C[\langle q_2, p_1\rangle] \overset{*}{\underset{\mathcal{B}}{\rightarrowtail}} \langle q, p\rangle$ with the last part of the path using less than k times the new transition. First, there is a configuration c_1 such that $c = C[c_1]$ and $c_1 \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} \langle q_1, p_1\rangle$, and therefore $\Pi_2(c_1) \overset{*}{\underset{\mathcal{IRR}(R)}{\rightarrowtail}} p_1$. Second, by induction hypothesis, $\Pi_2(C[p_1]) \overset{*}{\underset{\mathcal{IRR}(R)}{\rightarrowtail}} p$. Finally, $\Pi_2(c) \overset{*}{\underset{\mathcal{IRR}(R)}{\rightarrowtail}} \Pi_2(C[p_1]) \overset{*}{\underset{\mathcal{IRR}(R)}{\rightarrowtail}} p$. □

Remark. The condition 4 ($p_1 = p_2$) of definition 52 is obviously necessary for this lemma. We will explain why consistency is important after having defined the notion of innermost critical pair (Definition 55).

6.3 Exact completion

Definition 55 (Innermost critical pair).
A pair $(\ell \to r, \sigma, \langle q, p\rangle)$ where $\ell \to r \in R$, $\sigma : \mathcal{X} \to Q_{\mathcal{A}}$ and $\langle q, p\rangle \in Q_{\mathcal{A}}$ is critical if

1. $\ell\sigma \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} \langle q, p\rangle$,
2. there is no p' such that $r\sigma \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} \langle q, p'\rangle$ and
3. for each subconfiguration γ at depth 1 of $\ell\sigma$, the state $\left\langle q_\gamma, p_\gamma\right\rangle$ such that $\gamma \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} \left\langle q_\gamma, p_\gamma\right\rangle$ in the recognition path of condition 1 is with $p_\gamma \neq p_{\text{red}}$. 55 ◀

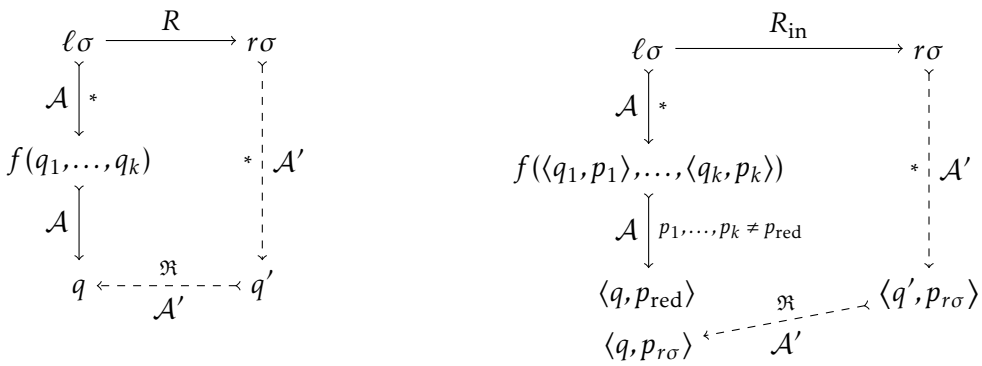
Remark. Because a critical pair denotes a rewriting situation, the p of definition 55 is necessarily p_{red} as long as $\mathcal{A}$ is consistent with $\mathcal{IRR}(R)$.

Remark. We can now explain why consistency is important and what would happen if we allowed equations to be applied without regard for condition 4 of Definition 52. Take $R = \{f(a) \to w, g(f(b)) \to c\}$, $E = \{a = b\}$ and $\mathcal{A}_\circ = \{a \rightarrowtail q_a, b \rightarrowtail q_b, f(q_a) \rightarrowtail q_{fa}, g(q_{fa}) \rightarrowtail q_{gfa}\}$. We have

$$\begin{aligned}\mathcal{IRR}(R) = \{&a \rightarrowtail p_a, b \rightarrowtail p_b, c \rightarrowtail p_c, w \rightarrowtail p_c,\\ &f(p_a) \rightarrowtail p_{\text{red}}, f(p_b) \rightarrowtail p_{fb}, f(p_c) \rightarrowtail p_c,\\ &g(p_{fb}) \rightarrowtail p_{\text{red}}, g(p_c) \rightarrowtail p_c\}\end{aligned}$$

We omit the transitions whose left-hand side contains p_{red}: they always have p_{red} as a right-hand side.

Remark that $c \in R_{\text{in}}(g(f(b)))$. However, $g(f(b))$ is not recognised by $\mathcal{A}_\circ$ into any state and thus not by $\mathcal{A}_\circ \times \mathcal{IRR}(R)$ either, so we are not interested in its successors. But let us apply the equation $a = b$ without taking condition 4 of definition 52 into account: in particular, we add the transition $\langle q_b, p_b\rangle \rightarrowtail \langle q_a, p_a\rangle$. Therefore we now have $g(f(b)) \rightarrowtail g(f(\langle q_b, p_b\rangle)) \rightarrowtail g(f(\langle q_a, p_a\rangle)) \rightarrowtail g(\langle q_{fa}, p_{\text{red}}\rangle) \rightarrowtail \langle q_{gfa}, p_{\text{red}}\rangle$. We have $g(f(b)) \in \mathscr{L}(\mathcal{A}, \langle q_{gfa}, p_{\text{red}}\rangle)$ and are thus now interested in its successors: we would like to have some critical pair involving $g(f(b))$ and c. But we do not, because the only recognition path of $g(f(b))$, the one we just created, does not fulfil condition 3 of definition 55.



(a) Classical

(b) Innermost

Figure 2: Comparison of classical and innermost critical pairs

Definition 56 (Normalisation).

This is like in Definition 27, except we deal with pairs $\langle q, p\rangle$ instead of just q. In particular, when searching for a $\langle q', p'\rangle$ to set $d_{|\xi_i} \rightarrowtail \langle q', p'\rangle$, we take q' to be either the left component of some $d_{|\xi_i} \underset{\Delta/\mathfrak{E}}{\rightarrowtail} \langle q', p''\rangle$ or a new state, and p' to be the state such that $\Pi_2\left(d_{|\xi_i}\right) \overset{*}{\underset{\mathcal{IRR}(R)}{\rightarrowtail}} p'$. 56 ◀

Definition 57 (Completion of an innermost critical pair).

A critical pair $(\ell \to r, \sigma, \langle q, p\rangle)$ in automaton $\mathcal{A}$ is completed by first computing $N = Norm_{\mathcal{A}^{\mathfrak{R}}}(r\sigma, \langle q', p'\rangle)$ where q' is a new state and $\Pi_2(r\sigma) \overset{*}{\underset{\mathcal{IRR}(R)}{\rightarrowtail}} p'$, then adding to $\mathcal{A}$ the new states and the transitions appearing in N as well as the transition $\langle q', p'\rangle \overset{\mathfrak{R}}{\rightarrowtail} \langle q, p'\rangle$. If $r\sigma$ is a trivial configuration (ie. r is just a variable), only transition $r\sigma \overset{\mathfrak{R}}{\rightarrowtail} \langle q, \Pi_2(r\sigma)\rangle$ is added.

Afterwards, we execute the following supplementary operations. For any transition

$$f(\dots, \langle q, p_{\text{red}}\rangle, \dots) \rightarrowtail \langle q'', p''\rangle,$$

we add (if it is not present) a transition

$$f(\dots, \langle q, p'\rangle, \dots) \rightarrowtail \langle q'', p'''\rangle$$

with $f(\dots, p', \dots) \underset{\mathcal{IRR}(R)}{\rightarrowtail} p'''$. These new transitions are in turn recursively considered for the supplementary operations. 57 ◀

Definition 58 (Step of exact completion).

This is the same definition as Definition 29, but using Definition 57 instead of Definition 28 for each critical pair. 58 ◀

Example 59.

Let us look again at Example 30. We have $R_{\text{in}}(\mathscr{L}(\mathcal{A}_0, q_f)) = \{f(c(0,n))\}$. Note that $\mathcal{IRR}(R)$ has states $p_0, p_a, p_s, p_n, p_c, p_?, p_{\text{red}}$ and transitions

$$\begin{aligned}
&\{n \rightarrowtail p_n, 0 \rightarrowtail p_0, s(p_a) \rightarrowtail p_{\text{red}}, a(p_s) \rightarrowtail p_{\text{red}}\} \\
\cup &\bigcup_{p \neq p_a} \{s(p) \rightarrowtail p_s\} \\
\cup &\bigcup_{p \neq p_s} \{a(p) \rightarrowtail p_a\} \\
\cup &\bigcup_{p} \{c(p_0, p) \rightarrowtail p_c, c(p_s, p) \rightarrowtail p_c, c(p_a, p) \rightarrowtail p_c, c(p_n, p) \rightarrowtail p_?, c(p_?, p) \rightarrowtail p_?\} \\
\cup &\{f(p_n) \rightarrowtail p_{\text{red}}, f(p_c) \rightarrowtail p_{\text{red}}\} \\
\cup &\bigcup_{p \neq p_n, p_c} \{f(p) \rightarrowtail p_?\}.
\end{aligned}$$

The critical pair corresponding to CP_1 of Example 30, with rule $f(c(a(X), Y)) \to c(a(X), f(Y))$, the substitution $\sigma_1 = \{X \mapsto \langle q_s, p_s \rangle, Y \mapsto \langle q_n, p_n \rangle\}$ and the state $\left\langle q_f, p_{\text{red}} \right\rangle$, is not an innermost critical pair because the recognition path is

$$\begin{aligned}
f(c(a(\langle q_s, p_s \rangle), \langle q_n, p_n \rangle)) &\rightarrowtail f(c(\langle q_a, p_{\text{red}} \rangle, \langle q_n, p_n \rangle)) \\
&\rightarrowtail f(\langle q_c, p_{\text{red}} \rangle) \\
&\rightarrowtail \left\langle q_f, p_{\text{red}} \right\rangle,
\end{aligned}$$

and so there is a p_{red} at depth 1. 59 ◁

Lemma 60.

Let $\mathcal{A}$ be an automaton obtained from some $\mathcal{A}_\circ \times \mathcal{IRR}(R)$ after some steps of innermost completion. $\mathcal{A}$ is consistent with $\mathcal{IRR}(R)$. 60 ■

Proof.

$\mathcal{A}_\circ \times \mathcal{IRR}(R)$ is consistent with $\mathcal{IRR}(R)$ by construction. Lemma 54 shows that applying an equation preserves this. It remains to show that a step of exact completion does so as well.

The first steps of normalisation are preserving because the new $\left\langle q_i', p_i' \right\rangle$ are precisely chosen such that $\Pi_2\left(d_{|\xi_i}\right) \underset{\mathcal{IRR}(R)}{\overset{*}{\rightarrowtail}} p_i'$. The last step of normalisation is preserving too, as well as the remaining operations of the completion of a critical pair, because, again, we choose p' such that $r\sigma \underset{\mathcal{IRR}(R)}{\overset{*}{\rightarrowtail}} p'$.

The same goes for the supplementary operations. □

Lemma 61.

Let $\mathcal{A}$ be an automaton consistent with $\mathcal{IRR}(R)$, $(\ell \to r, \sigma, \langle q, p \rangle)$ a critical pair in $\mathcal{A}$, let $p_{r\sigma}$ be the state of $\mathcal{IRR}(R)$ such that $r\sigma \underset{\mathcal{IRR}(R)}{\overset{*}{\rightarrowtail}} p_{r\sigma}$ and $\mathcal{B}$ be the automaton resulting from the completion

of this critical pair. Let C be a context on $T(\Sigma)$ and $\langle q_1, p_1\rangle$ a state of $\mathcal{A}$ such that $C[\langle q,p\rangle] \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} \langle q_1, p_1\rangle$. Then there exists a state p_2 of $\mathcal{IRR}(R)$ such that $C[\langle q, p_{r\sigma}\rangle] \overset{*}{\underset{\mathcal{B}}{\rightarrowtail}} \langle q_1, p_2\rangle$. 61 ■

Proof.

Note that we have $p = p_1 = p_{\text{red}}$. We have to show that all the transitions used in the path $C[\langle q, p_{\text{red}}\rangle] \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} \langle q_1, p_{\text{red}}\rangle$ have some counterpart starting from $C[\langle q, p_{r\sigma}\rangle]$. First, observe that all transitions used to recognise subterms at positions of C that are parallel to the position of the hole can remain unchanged. Second, if some transition only comprises states the left component of which are in $\mathcal{A}_\circ$, then it has a counterpart for any choice of right components in $\mathcal{IRR}(R)$ because our automaton contains the whole product $\mathcal{A}_\circ \times \mathcal{IRR}(R)$. It remains to be shown that the transitions involving new states added by the normalisation during the completion of the considered critical pair also have their counterpart: they exist thanks to the supplementary operations of definition 57. □

Remark. The supplementary operations are necessary for this lemma. Indeed, take $R = \{g(f(b)) \to g(f(a)), f(a) \to c\}$, $\mathcal{A}_\circ = \{b \rightarrowtail q_b, f(q_b) \rightarrowtail q_{fb}, g(q_{fb}) \rightarrowtail q_{gfb}\}$. We have $\mathcal{IRR}(R) = \{a \rightarrowtail p_a, b \rightarrowtail p_b, c \rightarrowtail p_c, f(p_a) \rightarrowtail p_{\text{red}}, g(p_a) \rightarrowtail p_c, f(p_b) \rightarrowtail p_{fb}, g(p_{fb}) \rightarrowtail p_{\text{red}}, f(p_c) \rightarrowtail p_c, g(p_c) \rightarrowtail p_c\}$. There is a critical pair $PC_1 = (g(f(b)) \to g(f(a)), \varnothing, \left\langle q_{gfb}, p_{\text{red}}\right\rangle)$ in $\mathcal{A}_\circ \times \mathcal{IRR}(R)$, which is resolved by adding transitions $a \rightarrowtail \langle q_{N1}, p_a\rangle$, $f(\langle q_{N1}, p_a\rangle) \rightarrowtail \langle q_{N2}, p_{\text{red}}\rangle$, $g(\langle q_{N2}, p_{\text{red}}\rangle) \rightarrowtail \langle q_{N3}, p_{\text{red}}\rangle$ and $\langle q_{N3}, p_{\text{red}}\rangle \rightarrowtail \left\langle q_{gfb}, p_{\text{red}}\right\rangle$, thereby producing automaton $\mathcal{A}_1$. The supplementary operations do not create any new transition here.

There is a critical pair $PC_2 = (f(a) \to c, \varnothing, \langle q_{N2}, p_{\text{red}}\rangle)$ in $\mathcal{A}_1$, which is resolved by adding transitions $c \rightarrowtail \langle q_{N4}, p_c\rangle$ and $\langle q_{N2}, p_c\rangle$, thereby producing automaton $\mathcal{A}_2^\natural$. The supplementary operations are detailed further down and produce automaton $\mathcal{A}_2$.

Now consider that $g(c) \in R_{\text{in}}(g(f(a)))$ and $g(f(a)) \in \mathcal{L}(\mathcal{A}_2^\natural, \left\langle q_{gfb}, p_{\text{red}}\right\rangle)$ because we completed PC_1. But all what we have is $g(c) \underset{\mathcal{A}_2^\natural}{\rightarrowtail} g(\langle q_{N4}, p_c\rangle) \underset{\mathcal{A}_2^\natural}{\rightarrowtail} g(\langle q_{N2}, p_c\rangle)$, this last configuration being not productive. As a result, $g(c) \notin \mathcal{L}(\mathcal{A}_2^\natural, \left\langle q_{gfb}, p'\right\rangle)$ for any p'.

The supplementary operations are made after completion of PC_2. Since there is a transition $g(\langle q_{N2}, p_{\text{red}}\rangle) \rightarrowtail \langle q_{N3}, p_{\text{red}}\rangle$, we add a transition $g(\langle q_{N2}, p_c\rangle) \rightarrowtail \langle q_{N3}, p_c\rangle$. Then, since $q_{N3} \notin \mathcal{A}_\circ$ either, and there is a $\langle q_{N3}, p_{\text{red}}\rangle \rightarrowtail \left\langle q_{gfb}, p_{\text{red}}\right\rangle$, we add $\langle q_{N3}, p_c\rangle \rightarrowtail \left\langle q_{gfb}, p_c\right\rangle$. No further transition needs to be added. These transitions allow $g(c) \in \mathcal{L}(\mathcal{A}_2, \left\langle q_{gfb}, p_c\right\rangle)$.

6.4 Proof of correction

Definition 62 (Correct automaton).

An automaton $\mathcal{A}$ is correct wrt. R_{in} if for all state $\langle q, p_{\text{red}}\rangle$ of $\mathcal{A}$, for all $u \in \mathcal{L}(\mathcal{A}, \langle q, p_{\text{red}}\rangle)$ and for all $v \in R_{\text{in}}(u)$, either there is a state p of $\mathcal{IRR}(R)$ such that $v \in \mathcal{L}(\mathcal{A}, \langle q, p\rangle)$ or there is a critical pair $(\ell \to r, \sigma, \langle q_0, p_0\rangle)$ in $\mathcal{A}$ for some $\langle q_0, p_0\rangle$ and a context C on $T(\Sigma)$ such that $u \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} C[\ell\sigma] \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} C[\langle q_0, p_{\text{red}}\rangle] \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} \langle q, p_{\text{red}}\rangle$ and $v \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} C[r\sigma]$. 62 ◀

Lemma 63.

Any automaton produced by innermost completion starting from some $\mathcal{A}_\circ \times \mathcal{IRR}(R)$ is correct wrt. R_{in}. 63 ■

Proof.

Let $\mathcal{A}$ be such an automaton; it is consistent with $\mathcal{IRR}(R)$. Let $\langle q, p_{\text{red}}\rangle$ be a state of $\mathcal{A}$, $u \in \mathcal{L}(\mathcal{A}, \langle q, p_{\text{red}}\rangle)$ and $v \in R_{\text{in}}(u)$. By definition of innermost rewriting, there is a rule $\ell \to r$ of R, a substitution $\mu : \mathcal{X} \to T(\Sigma)$ and a context C such that $u = C[\ell\mu]$, $v = C[r\mu]$ and each strict subterm of u is a normal form. Let $u_0 = \ell\mu$ and $v_0 = r\mu$. There is a $\langle q_0, p_{\text{red}}\rangle$ such that $u_0 \in \mathcal{L}(\mathcal{A}, \langle q_0, p_{\text{red}}\rangle)$ and $C[\langle q_0, p_{\text{red}}\rangle] \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} \langle q, p_{\text{red}}\rangle$.

Since ℓ is linear, there is a $\sigma : \mathcal{X} \to Q_{\mathcal{A}}$ such that $\ell\mu \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} \ell\sigma \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} \langle q_0, p_{\text{red}}\rangle$ and $r\mu \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} r\sigma$. This entails that $u \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} C[\ell\sigma] \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} C[\langle q_0, p_{\text{red}}\rangle] \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} \langle q, p_{\text{red}}\rangle$ and $v \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} C[r\sigma]$.

Assume that there is no p_0 such that $v_0 \in \mathcal{L}(\mathcal{A}, \langle q_0, p_0\rangle)$ and show that $(\ell \to r, \sigma, \langle q_0, p_{\text{red}}\rangle)$ is a critical pair in $\mathcal{A}$. First, by assumption, there is no p such that $r\sigma \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} \langle q_0, p\rangle$. Conditions 1 and 2 of definition 55 are thus met. Suppose $\ell = f(\gamma_1, \ldots, \gamma_k)$ and show that condition 3 of definition 55 holds.[4] For each $i \in [\![1\,;k]\!]$, let $\langle q_i, p_i\rangle$ be the state of $\mathcal{A}$ such that $\gamma_i\mu \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} \gamma_i\sigma \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} \langle q_i, p_i\rangle$ in the path of recognition of $\ell\sigma$. Then, by consistency with $\mathcal{IRR}(R)$, for each $i \in [\![1\,;k]\!]$, $\gamma_i\mu \overset{*}{\underset{\mathcal{IRR}(R)}{\rightarrowtail}} p_i$. Since strict subterms of $\ell\mu$ are strict subterms of u as well, they are normal forms, thus $p_i \neq p_{\text{red}}$, which validates condition 3 of definition 55.

Assume now that $v_0 \in \mathcal{L}(\mathcal{A}, \langle q_0, p_0\rangle)$ and show that there is a p such that $v \in \mathcal{L}(\mathcal{A}, \langle q, p\rangle)$. This is obvious at the initial step $\mathcal{A}_\circ \times \mathcal{IRR}(R)$, and this property is conserved by completion as shown by lemma 61. □

Theorem 64 (Correctness).

Assuming R is left-linear, the innermost equational completion procedure defined above produces a correct result whenever it terminates and produces some fixpoint $\mathcal{A}_{\text{in}*}$:

$$\mathcal{L}(\mathcal{A}_{\text{in}*}) \supseteq R^*_{\text{in}}(\mathcal{L}(\mathcal{A}_\circ \times \mathcal{IRR}(R))). \qquad 64 \blacksquare$$

Proof.

Let $\mathcal{A}_{\text{in}*}$ be the calculated fixpoint automaton. By lemma 60, $\mathcal{A}_{\text{in}*}$ is consistent with $\mathcal{IRR}(R)$, and therefore, by lemma 63, $\mathcal{A}_{\text{in}*}$ is correct wrt. R_{in}. Since this automaton is a fixpoint, the case of definition 62 where there remains a critical pair cannot occur, and therefore, for all state $\langle q, p_{\text{red}}\rangle$ of $\mathcal{A}$, for all $u \in \mathcal{L}(\mathcal{A}_{\text{in}*}, \langle q, p_{\text{red}}\rangle)$ and for all $v \in R_{\text{in}}(u)$, there is a p' such that $v \in \mathcal{L}(\mathcal{A}_{\text{in}*}, \langle q, p\rangle)$. Thus, $\mathcal{A}_{\text{in}*}$ is R_{in}-closed: $R_{\text{in}}(\mathcal{L}(\mathcal{A}_{\text{in}*})) \subseteq \mathcal{L}(\mathcal{A}_{\text{in}*})$. Since the completion process only adds transitions, $\mathcal{L}(\mathcal{A}_{\text{in}*}) \supseteq \mathcal{L}(\mathcal{A}_\circ \times \mathcal{IRR}(R))$, which concludes the proof. □

6.5 Other results

Theorem 65.

If, given some left-linear R, some E and some $\mathcal{A}_\circ$, the classical equational completion terminates, producing fixpoint $\mathcal{A}_*$, as well as the innermost equational completion, producing $\mathcal{A}_{\text{in}*}$, then

$$\mathcal{L}(\mathcal{A}_{\text{in}*}) \subseteq \mathcal{L}(\mathcal{A}_* \times \mathcal{IRR}(R)). \qquad 65 \blacksquare$$

[4] If ℓ is a constant, then condition 3 is vacuously true.

Proof.

This because we complete less critical pairs and apply equations less often: to any innermost critical pair corresponds a classical one, and the same goes for situations of application. □

Innermost completion solves less critical pairs and thus adds fewer new states. In practice, it is thus likely to stop as often as standard completion. However, since it also performs less equational simplifications, we do not yet benefit of the theoretical guarantee of Theorem 49. This is mainly due to Definition 52 that requires that $p_1 = p_2$ when applying an equation to states $\langle q_1, p_1\rangle$ and $\langle q_2, p_2\rangle$. We can nevertheless state the following theorem that covers the 'reverse' example with the second set of contracting equations. The definition of a more general termination theorem for innermost completion needs further investigation and is left for future work.

Lemma 66.

Let R be a well-sorted, sufficiently complete TRS. Let $u, v \in T(\mathcal{C})^{\mathcal{S}}$ be two data-terms having the same sort. For any context C over $T(\Sigma)^{\mathcal{S}}$, $C[u]$ and $C[v]$ are either both reducible or both normal forms of R. 66 ■

Proof.

Let C be a context over $T(\Sigma)^{\mathcal{S}}$. Since u and v have the same sort, $C[u]$ is well-sorted if and only if $C[v]$ is well-sorted. If these terms are not well-sorted, they are both irreducible. Let us now suppose $C[u]$ and $C[v]$ are both well sorted. Let $\xi_\square$ be the position of the hole in C.

Suppose there is a position ξ prefix of $\xi_\square$ such that the symbol at position ξ is in $\mathcal{D}$. Let $u' = C[u]_{|\xi}$ and $v' = C[v]_{|\xi}$. Since $u', v' \in T(\Sigma)^{\mathcal{S}} \setminus T(\mathcal{C})^{\mathcal{S}}$ and R is sufficiently complete, they are both reducible, and thus so are $C[u]$ and $C[v]$.

If there is no ξ defined as above, then the potential redexes are at positions parallel to $\xi_\square$, and thus do not depend on u or v. □

Lemma 67.

There is an automaton $\mathcal{IRR}(R)$ verifying the properties of Theorem 25 that has exactly one state p_S for each sort S as well as two states $p_?$ and p_{red}. 67 ■

Proof.

By well-sortedness of R, any bad-sorted term is a normal form as long as it has no reducible strict subterm. All these terms can be recognised into state $p_?$. All the other states will recognise only well-sorted terms.

By sufficient completeness, any well-sorted term in which occurs a defined symbol is reducible, and will be recognised into p_{red}.

Let S be a sort and consider $u, v \in T(\mathcal{C})^{\mathcal{S}}$ of sort S. If there are two states p_u, p_v such that $u \in \mathcal{L}(\mathcal{IRR}(R), p_u)$ and $v \in \mathcal{L}(\mathcal{IRR}(R), p_v)$, then, by Lemma 66, p_u and p_v will play exactly the same role in $\mathcal{IRR}(R)$. We can therefore merge them. □

Lemma 68 (Condition 4 of Definition 52 is free).

Let $s = t$ be a sort-preserving equation over $T(\mathcal{C})^{\mathcal{S}}$ such that s and t have the same variables, $\mathcal{A}$ be an automaton consistent with an automaton $\mathcal{IRR}(R)$ as defined in Lemma 67, $\theta : \mathcal{X} \to Q_{\mathcal{A}}$ be a substitution and $\langle q_1, p_1\rangle$, $\langle q_2, p_2\rangle$ be states such that

1. $s\theta \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} \langle q_1, p_1\rangle$ and

2. $t\theta \overset{*}{\underset{\mathcal{A}}{\rightarrowtail}} \langle q_2, p_2 \rangle$.

Then $p_1 = p_2$. 68 ■

Proof.
By consistency of $\mathcal{A}$ with $\mathcal{IRR}(R)$, $\Pi_2(s\theta) \overset{*}{\underset{\mathcal{IRR}(R)}{\rightarrowtail}} p_1$ and $\Pi_2(t\theta) \overset{*}{\underset{\mathcal{IRR}(R)}{\rightarrowtail}} p_2$. If there is a variable X of s or t such that $\Pi_2(X\theta) \in \{p_{\text{red}}, p_?\}$, then p_1 and p_2 are both p_{red} or $p_?$.

Else, there is a substitution $\mu : \mathcal{X} \to T(\mathcal{C})^{\mathcal{S}}$ such that for all variable X of the domain of θ, $X\mu \overset{*}{\underset{\mathcal{IRR}(R)}{\rightarrowtail}} X\theta$. Since $s = t$ is sort-preserving, $s\mu$ and $t\mu$ are in $T(\mathcal{C})^{\mathcal{S}}$ and both have the same sort S. Thus, by construction of $\mathcal{IRR}(R)$, $p_1 = p_2 = p_S$. □

Theorem 69.
If R is left-linear, well-sorted and sufficiently complete, the equations of E are over $T(\mathcal{C})^{\mathcal{S}}$, sort preserving and between terms using the same variables, and classical equational completion terminates, then so does innermost equational completion. 69 ■

Proof.
Classical completion terminates on $\mathcal{A}_\circ \times \mathcal{IRR}(R)$ as well. Innermost completion considers and completes less critical pairs than classical completion (i.e. to each critical pair in the innermost process corresponds a critical pair in the classical completion of $\mathcal{A}_\circ \times \mathcal{IRR}(R)$), but, by Lemma 68, applies equations as often as classical completion. □

7 Related work

There is a recent and renewed interest for Data flow analysis of higher-order functional programs [OR11, KO11] that was initiated by [Jon87]. The latter was using tree grammars for the analysis and the former are using specific formalisms: pattern matching recursion schemes and ILTGs. In [Jon87], the approximation is hard-coded in the completion-like algorithm and it is defined by complex algorithms in [OR11, KO11]. This is not the case here where approximations are defined in a declarative and formal way using term equations. The technique we propose is able to cover all the examples of [OR11] with the same precision (and all the simple examples of [Jon87] with a better precision, as [OR11] do). On the opposite, since tree automata completion is based on regular languages, it cannot handle "non regular properties" covered by ILTGs [KO11]. However, we have shown that tree automata completion can take strategies into account. In all aforementioned work, evaluation strategies of functional programs are not taken into account. For instance, in Example 30 all those techniques will consider the term $c(a(s(0)), f(n))$ as reachable, though it is not with innermost strategy. As shown in Example 59, this is not the case with innermost completion.

Dealing with reachable terms and strategies was first addressed in [RV02] in the exact case for innermost and outermost strategies but only for some restricted classes of TRSs. As far as we know, the technique we propose is the first to over-approximate terms reachable by innermost rewriting for *any* left-linear TRSs. For instance, the examples of Section 5.1 and 5.2 are in the scope of innermost completion but are outside of the class of [RV02]. The first one because a right-hand side of a rule has two nested defined symbols and the second one because it is not right linear.

8 Conclusion

In this paper, we propose to use tree automata completion for the static analysis of higher-order functional programs. The first contribution is a proof that completion is guaranteed to terminate if the set E of approximation equations contains contraction equations for $T(\mathcal{C})$. The second contribution is a completion algorithm taking into account the rewriting strategy. The obtained algorithm minimizes the set of added transitions by completing the product automaton (between $\mathcal{A}_\circ$ and $\mathcal{IRR}(R)$), instead of computing this product after each completion step.

The tree automata completion framework seems to provide a simple and competitive alternative to other analysis techniques for functional programs. Firstly, and unlike ILTG, it uses basic algorithms (namely intersection and emptiness decision) which are known to be polynomial [CDG+08]. The efficiency of completion has been shown in practice with the approximation of $R^*(L_0)$ for huge TRSs (thousand of rewrite rules) [BBGL12]. Secondly, having a simple formalism makes certification of the computation possible. All the examples of this paper are certified by Coq, using the external completion checker [BGJ08]. Certification is important because efficient implementation of such analysers require low level optimisations that can threaten their correctness. Thirdly, defining approximations using equations is declarative and formal. Their declarative nature makes it possible to adapt approximations by tuning equations by hand or using an automatic refinement principle. Their formal nature makes it possible to precisely characterize their precision using Theorem 38 and, as shown in this paper, have a simple syntactical criterion ensuring termination of the analysis.

Besides, innermost completion should provide finer criteria for innermost termination proofs of TRSs. Approximations of sets of ancestors or descendants can improve existing termination techniques [Mid02, GHWZ05], such as the dependency pairs. In particular, one can prune edges in a dependency graph using such approximations. Dependency graphs can prove innermost termination [GTSKF06]. Thus, a precise approximation of innermost descendants should prune the innermost dependency graph, and provide a finer innermost termination criterion.

For further work, we want to study if the innermost completion covers the decidable classes of [RV02], like standard completion does for many decidable classes [FGVTT04]. Another objective is to define a completion for the outermost strategy and thus deal with the call-by-need evaluation strategy, used in Haskell[5]. We think that it is also possible to change the simplification algorithm so that finiteness of $T(\Sigma)/_{\equiv_E}$ (or $T(\mathcal{C})/_{\equiv_E}$) would be sufficient to have a terminating completion. In particular, this should make the definition of a termination theorem for innermost completion easier.

References

[ABB+05] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santos Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA Tool for the automated validation of internet security

[5] Note that in tree automata completion every computation is done only once, associated to a state and shared in the tree automaton, as in call-by-need.

protocols and applications. In *CAV'2005*, volume 3576 of *LNCS*, pages 281–285. Springer, 2005.

[BBGL12] Y. Boichut, B. Boyer, T. Genet, and A. Legay. Equational Abstraction Refinement for Certified Tree Regular Model Checking. In *ICFEM'12*, volume 7635 of *LNCS*. Springer, 2012.

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

[BCHK09] Y. Boichut, R. Courbis, P.-C. Héam, and O. Kouchnarenko. Handling non left-linear rules when completing tree automata. *IJFCS*, 20(5), 2009.

[BGJ08] B. Boyer, T. Genet, and T. Jensen. Certifying a Tree Automata Completion Checker. In *IJCAR'08*, volume 5195 of *LNCS*. Springer, 2008.

[BGJL07] Y. Boichut, T. Genet, T. Jensen, and L. Leroux. Rewriting Approximations for Fast Prototyping of Static Analyzers. In *RTA*, volume 4533 of *LNCS*, pages 48–62. Springer, 2007.

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

[CDE+09] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and José F. Quesada. Maude homepage, 2009. `http://maude.cs.uiuc.edu`.

[CDG+08] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, C. Löding, S. Tison, and M. Tommasi. Tree automata techniques and applications. `http://tata.gforge.inria.fr`, 2008.

[CR87] H. Comon and Jean-Luc Rémy. How to characterize the language of ground normal forms. Technical Report 676, INRIA-Lorraine, 1987.

[FGVTT04] G. Feuillade, T. Genet, and V. Viet Triem Tong. Reachability Analysis over Term Rewriting Systems. *Journal of Automated Reasonning*, 33 (3-4):341–383, 2004.

[Gen98] T. Genet. Decidable approximations of sets of descendants and sets of normal forms. In *Proc. 9th RTA Conf., Tsukuba (Japan)*, volume 1379 of *LNCS*, pages 151–165. Springer-Verlag, 1998.

[GHWZ05] A. Geser, D. Hofbauer, J. Waldmann, and H. Zantema. On tree automata that certify termination of left-linear term rewriting systems. In *RTA'05*, volume 3467 of *LNCS*, pages 353–367. Springer, 2005.

[GK00] T. Genet and F. Klay. Rewriting for Cryptographic Protocol Verification. In *Proc. 17th CADE Conf., Pittsburgh (Pen., USA)*, volume 1831 of *LNAI*. Springer-Verlag, 2000.

[GR10] T. Genet and R. Rusu. Equational tree automata completion. *Journal of Symbolic Computation*, 45:574–597, 2010.

[GTSKF06] J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *JAR*, 37(3):155–203, 2006.

[Jon87] N. D. Jones. Flow analysis of lazy higher-order functional programs. In S. Abramsky and C. Hankin, editors, *Abstract Interpretation of Declarative Languages*, pages 103–122. Ellis Horwood, Chichester, England, 1987.

[KO11] J. Kochems and L. Ong. Improved Functional Flow and Reachability Analyses Using Indexed Linear Tree Grammars. In *RTA'11*, volume 10 of *LIPIcs*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2011.

[LDG+12] X. Leroy, D. Doligez, J. Garrigue, D. Rémy, and J. Vouillon. The Objective Caml system release 3.12 – Documentation and user's manual. INRIA, 2012. `http://caml.inria.fr/ocaml/`.

[Lis12] A. Lisitsa. Finite Models vs Tree Automata in Safety Verification. In *RTA'12*, volume 15 of *LIPIcs*, pages 225–239, 2012.

[Mid02] A. Middeldorp. Approximations for strategies and termination. *ENTCS*, 70(6):1–20, 2002.

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

[RV02] P. Réty and J. Vuotto. Regular Sets of Descendants by some Rewrite Strategies. In *Proc. 13th RTA Conf., Copenhagen (Denmark)*, volume 2378 of *LNCS*. Springer-Verlag, 2002.

[Tak04] T. Takai. A Verification Technique Using Term Rewriting Systems and Abstract Interpretation. In *Proc. 15th RTA Conf., Aachen (Germany)*, volume 3091 of *LNCS*, pages 119–133. Springer, 2004.

[Tim12] Timbuk – reachability analysis and Tree Automata Calculations. IRISA / Université de Rennes 1, 2012. `http://www.irisa.fr/celtique/genet/timbuk/`.

[TKS00] T. Takai, Y. Kaji, and H. Seki. Right-linear finite-path overlapping term rewriting systems effectively preserve recognizability. In *Proc. 11th RTA Conf., Norwich (UK)*, volume 1833 of *LNCS*. Springer-Verlag, 2000.