Redundancy of Arguments Reduced to Induction*

María Alpuente[1] Rachid Echahed[2] Santiago Escobar[1**] Salvador Lucas[1]

[1] DSIC, UPV, Camino de Vera s/n, 46022 Valencia, Spain.
{alpuente,sescobar,slucas}@dsic.upv.es
[2] Leibniz-IMAG, 46, Av. Felix Viallet, 38031 Grenoble, France.
Rachid.Echahed@imag.fr

Abstract. We demonstrate that the problem of identifying redundant arguments of function symbols, i.e. parameters which can be replaced by any expression without changing the associated semantics, boils down to proving the validity of a particular class of inductive theorems in the equational theory of confluent, sufficiently complete term rewrite systems (TRSs). Hence, existing results for proving inductive theorems can be exploited, which solve the problem in many interesting cases where previously developed methods fail to recognize and remove redundancies. In particular, this novel formulation directly yields a new decidability result for the redundancy problem which is based on the so-called *standard theories*. As an additional result which stems from the inductive encoding of the redundancy problem, we finally propose two different techniques for the analysis of redundant arguments, which are respectively based on *inductionless induction* and *abstract rewriting* (a technique for approximating normal forms in sufficiently complete, left linear, canonical TRSs).

1 Introduction

The application of automatic transformation processes during the formal development and optimization of programs can introduce encumbrances in the generated code that programmers usually (or presumably) do not write. Examples are redundant arguments in the functions defined in the program [1, 2, 7, 13, 15, 18, 20, 21, 24, 27].

Example 1. Consider the following program, which can be used for adding and substracting natural numbers in Peano's notation:

```
minus(x,0) → x                    plus(0,y) → y
minus(0,s(y)) → 0                 plus(s(x),y) → s(plus(x,y))
minus(s(x),s(y)) → minus(x,y)
```

* Work partially supported by CICYT TIC2001-2705-C03-01, Acciones Integradas HI2000-0161, HA2001-0059, HU2001-0019, and Generalitat Valenciana GV01-424.

** S. Escobar was supported by grant 4342 of *Universidad Politécnica de Valencia* during a stay at Leibniz-IMAG Lab.

If we specialize this program for the call `minusplus(x,y) = minus(plus(x,y),y)`, which adds `x` to `y` and then removes `y` from the sum, thus returning the original `x`, the optimized program which can be obtained by using an automatic specializer of functional programs such as [3] is:

(r_1) `minusplus(0,0)` $\rightarrow$ `0`
(r_2) `minusplus(s(x),0)` $\rightarrow$ `s(plus0(x))`
(r_3) `minusplus(0,s(y))` $\rightarrow$ `minus1(y)`
(r_4) `minusplus(s(x),s(y))` $\rightarrow$ `s(minusplus(x,y))`
(r_5) `plus0(0)` $\rightarrow$ `0`
(r_6) `plus0(s(x))` $\rightarrow$ `s(plus0(x))`
(r_7) `minus1(0)` $\rightarrow$ `0`
(r_8) `minus1(s(y))` $\rightarrow$ `minus1(y)`

Note that the second argument of the function `minusplus` is redundant for the semantics of computed values and would not typically be written by a programmer who writes this program by hand. Known procedures for removing dead code such as [7, 18, 21] as well as standard (post-specialization) renaming/compression procedures (see e.g. [3]) cannot remove the redundant argument either. Moreover, redundant argument filtering procedures for logic programs such as the one included in the partial deduction system ECCE [19] do not recognize the redundancy of this parameter either.

In this paper, we provide a characterization of the problem of redundancy of arguments in terms of inductive theorems (see [2] for a detailed comparison of the problem of redundancy of arguments w.r.t. existing techniques). This complements some previous results presented in [2], where the decidability of the redundancy for the class of right-ground TRSs was proven. Now, by exploiting induction we are able to prove decidability in a different and incomparable class of programs, namely the standard theories of [23]. In [2] we also proposed two different criteria for recognizing redundancy which requires either that the redundant arguments occur at a variable position in every lhs or a joinability condition on the rhs's of the rules, which prevented our methods from coping with many interesting examples, such as the program of Example 1. In this paper we provide two different, novel criteria for recognizing redundancy, which are based on *inductionless induction* [10] and abstract rewriting [8], respectively. The combination of these methods catches redundancy in many new practical cases, including Example 1. Of course, in exchange for the conditions of [2], different extra conditions are required, which are also discussed in the paper.

1.1 Plan of the paper

After some preliminaries in Section 2, in Section 3 we recall from [2] the notion of redundancy of arguments of function symbols and we show how the redundancy of arguments is reduced to the validity of inductive theorems. Then, undecidability and decidability results as well as different methods for recognizing inductive theorems can be exploited. In Section 4, we recall the inductive proof technique called *inductionless induction* [10], and show how it can be applied for detecting redundant arguments in many cases which were not solved by previous methods such as [2]. We also identify a class of rewrite systems for which the problem of detection of redundant arguments is decidable. This class is different from the one identified in [2] and hence also the decidability results in this paper and those in [2] are incomparable and complement each other. In Section 5, we show how the abstract rewriting technique of [8], for approximating normal forms of terms, can be used to prove inductive theorems, and thus to detect new redundancies. Section 6 concludes the paper.

2 Preliminaries

Let us first introduce the main notations used in the paper. For full or missing definitions about term rewriting, we refer to [14]; and for theorem proving in automated reasoning, we refer to [6]. Let $\rightarrow \subseteq A \times A$ be a binary relation on a set A. We denote the inverse of $\rightarrow$ by $\leftarrow$, the symmetric closure by $\leftrightarrow$, the transitive closure by $\rightarrow^+$, the reflexive and transitive closure by $\rightarrow^*$, and the reflexive, symmetric and transitive closure by $\leftrightarrow^*$. We say that $\rightarrow$ is *confluent* if, for every $a, b, c \in A$, whenever $a \rightarrow^* b$ and $a \rightarrow^* c$, there exists $d \in A$ such that $b \rightarrow^* d$ and $c \rightarrow^* d$. We say that $\rightarrow$ is *terminating* (or *well-founded*) iff there is no infinite sequence $a_1 \;\rightarrow\; a_2 \;\rightarrow\; a_3 \;\cdots$.

Throughout the paper, $\mathcal{X}$ denotes a countable set of variables and Σ denotes a finite set of function symbols $\{\mathtt{f}, \mathtt{g}, \ldots\}$, each one having a fixed arity given by a function $ar : \Sigma \rightarrow \mathbb{N}$. By $\mathcal{T}(\Sigma, \mathcal{X})$ we denote the set of terms; $\mathcal{T}(\Sigma)$ is the set of ground terms or Herbrand domain, i.e., terms without variable occurrences. A term is said to be linear if it has no multiple occurrences of a single variable. A k-tuple $t_1, \ldots, t_k$ of terms is written $\overline{t}$. The number k of elements of the tuple $\overline{t}$ will be clarified by the context. $\mathcal{V}ar(t)$ is the set of variables in t. A *substitution* is a mapping $\sigma : \mathcal{X} \rightarrow \mathcal{T}(\Sigma, \mathcal{X})$ which homomorphically extends to a mapping $\sigma : \mathcal{T}(\Sigma, \mathcal{X}) \rightarrow \mathcal{T}(\Sigma, \mathcal{X})$. Let $Subst(\Sigma, \mathcal{X})$ denote the set of substitutions and $Subst(\Sigma)$ be the set of ground substitutions, i.e., substitutions on $\mathcal{T}(\Sigma)$. If $\sigma(t)$ is a ground term, we call σ a *grounding substitution* for t. A *unifier* of two terms t, s is a substitution σ with $\sigma(t) = \sigma(s)$. A *most general unifier* (*mgu*) of t, s is a unifier σ such that for each unifier σ' of t, s there exists θ such that $\sigma' = \theta \circ \sigma$.

Terms are viewed as labelled trees in the usual way. Positions $p, q, \ldots$ are represented by chains of positive natural numbers used to address subterms of t. By Λ, we denote the empty chain. The subterm at position p of t is denoted as $t|_p$ and $t[s]_p$ is the term t with the subterm at position p replaced by s. By $\mathcal{P}os(t)$ we denote the set of positions of a term t. The symbol labeling the root of t is denoted as $root(t)$. A context is a term C with zero or more 'holes', $\Box$ (a fresh constant symbol). We usually write simply $C[\,]$ to denote arbitrary context, clarifying the number and location of holes 'in situ'. If C is a context and t a term, $C[t]$ denotes the result of replacing the hole in C by t.

A rewrite rule is an ordered pair (l, r), written $l \rightarrow r$, with $l, r \in \mathcal{T}(\Sigma, \mathcal{X})$, $l \notin \mathcal{X}$ and $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$. The left-hand side (*lhs*) of the rule is l and r is the right-hand side (*rhs*). A TRS is a pair $\mathcal{R} = (\Sigma, R)$ where R is a set of rewrite rules and Σ is called the signature. An instance $\sigma(l)$ of the lhs of a rule $l \rightarrow r$

is a redex. A term t without redexes is said a normal form. By $\mathsf{NF}_{\mathcal{R}}$ we denote the set of finite normal forms of $\mathcal{R}$. Given $\mathcal{R} = (\Sigma, R)$, we consider Σ as the disjoint union $\Sigma = \mathcal{C} \uplus \mathcal{F}$ of symbols $c \in \mathcal{C}$, called *constructors*, and symbols $f \in \mathcal{F}$, called *defined functions*, where $\mathcal{F} = \{f \mid f(\bar{l}) \to r \in R\}$ and $\mathcal{C} = \Sigma - \mathcal{F}$. Then, $\mathcal{T}(\mathcal{C}, \mathcal{X})$ is the set of constructor terms. A pattern is a term $f(l_1, \ldots, l_n)$ such that $f \in \mathcal{F}$ and $l_1, \ldots, l_n \in \mathcal{T}(\mathcal{C}, \mathcal{X})$. A term t rewrites to s (at position p), written $t \to_{\mathcal{R}} s$ (or just $t \to s$), if $t|_p = \sigma(l)$ and $s = t[\sigma(r)]_p$, for some rule $l \to r \in R$, $p \in \mathcal{P}os(t)$ and substitution σ. A TRS $\mathcal{R}$ is left linear if all its lhs's are linear terms. A constructor system (*CS*) is a TRS whose lhs's are patterns. A TRS $\mathcal{R}$ is *terminating* (resp. *confluent*) if the relation $\to_{\mathcal{R}}$ is terminating (resp. confluent). A TRS $\mathcal{R}$ is *canonical* or *convergent* if the relation $\to_{\mathcal{R}}$ is terminating and confluent. If the TRS $\mathcal{R}$ is canonical, the normal form of a term $t \in \mathcal{T}(\Sigma)$ exists, it is unique, and it will be denoted by $t{\downarrow_{\mathcal{R}}} \in \mathsf{NF}_{\mathcal{R}}$. A TRS $\mathcal{R}$ is *sufficiently complete* if $\forall t \in \mathcal{T}(\Sigma)$, $\exists t' \in \mathcal{T}(\mathcal{C})$ such that $t \leftrightarrow^*_{\mathcal{R}} t'$. Two terms t, s are *joinable*, denoted by $t \downarrow s$, if there exists a term u such that $t \to^* u$ and $s \to^* u$.

To avoid confusion, in the sequel syntactic equality of terms is represented by $\equiv$. An equation is a formula of the form $r = s$ (or $s = r$) where $r, s \in \mathcal{T}(\Sigma, \mathcal{X})$. An *equational system* is a set of equations. If E is a set of equations between terms of $\mathcal{T}(\Sigma, \mathcal{X})$, $\leftrightarrow^*_E$ is the smallest congruence on $\mathcal{T}(\Sigma, \mathcal{X})$ such that $\sigma(s) \leftrightarrow^*_E \sigma(t)$ for all equations $s = t \in E$ and for all substitutions σ. Given a set of equations (or rewrite rules) E, $s = t$ is a *logical consequence* of E, denoted by $E \vdash s = t$, if $s \leftrightarrow^*_E t$. The *equational theory* of E is the set of equations that are logical consequences of E. The *minimal Herbrand model* (often called minimal model) $\mathcal{I}_E$ of a set of equations E is the quotient algebra $\mathcal{T}(\Sigma)/_{\leftrightarrow^*_E}$. We say that a first-order equation $s = t$ is an inductive consequence of a set of equations (or rewrite rules) E iff $\mathcal{I}_E \models s = t$, i.e. $\sigma(s) \leftrightarrow^*_E \sigma(t)$ for all grounding substitution σ for t and s. The set of all inductive consequences of E is called the *inductive theory of* E. Inductive consequences of E will also be called *inductive theorems* in what follows.

3 Redundant arguments

The redundancy of an argument of a function f in a TRS $\mathcal{R}$ depends on the semantic properties of $\mathcal{R}$ that we are interested in observing. The semantics considered in this paper, which is the most commonly considered in functional programming, is the set of values (ground constructor terms) that $\mathcal{R}$ is able to produce in a finite number of rewriting steps from a ground term ($\mathsf{eval}_{\mathcal{R}}(t) = \{s \in \mathcal{T}(\mathcal{C}) \mid t \to^*_{\mathcal{R}} s\}$). We often omit the subindex $\mathcal{R}$ when it is clear from the context. Other semantics which are relevant to the redundancy problem are discussed in [1, 2, 22].

Roughly speaking, a redundant argument of a function f is an argument t_i which we do not need to consider in order to compute the semantics of any call containing a subterm $f(t_1, \ldots, t_k)$.

Definition 1 (Redundancy of an argument). *[2] Let $\mathcal{R} = (\Sigma, R)$ be a TRS, $f \in \Sigma$, and $i \in \{1, \ldots, ar(f)\}$. The i-th argument of f is* redundant *(w.r.t. $\mathsf{eval}_{\mathcal{R}}$) if, for all context $C[\ ]$ and for all $t, s \in \mathcal{T}(\Sigma)$ such that $root(t) = f$, $\mathsf{eval}_{\mathcal{R}}(C[t]) = \mathsf{eval}_{\mathcal{R}}(C[t[s]_i])$.*

We denote by $rarg_{\mathsf{eval}_{\mathcal{R}}}(f)$ the set of redundant arguments of a symbol $f \in \Sigma$ w.r.t. the semantics $\mathsf{eval}_{\mathcal{R}}$ for Σ.

When analyzing a property of a function f in $\mathcal{R}$, it is useful to get rid of the contexts and perform easier, local analyses which allow us to center the attention on the syntactic structure of the rewriting rules. This motivates the following.

Definition 2 (Local redundancy of an argument). *[2] Let $\mathcal{R} = (\Sigma, R)$ be a TRS, $f \in \Sigma$, and $i \in \{1, \ldots, ar(f)\}$. The i-th argument of f is* locally redundant *(w.r.t. $\mathsf{eval}_{\mathcal{R}}$) if, for all $t, s \in \mathcal{T}(\Sigma)$ such that $root(t) = f$, $\mathsf{eval}_{\mathcal{R}}(t) = \mathsf{eval}_{\mathcal{R}}(t[s]_i)$.*

We denote by $lrarg_{\mathsf{eval}_{\mathcal{R}}}(f)$ the set of locally redundant arguments of a symbol f w.r.t. $\mathsf{eval}_{\mathcal{R}}$.

Redundancy of an argument w.r.t. the semantics eval implies local redundancy w.r.t. eval, i.e., $rarg_{\mathsf{eval}}(f) \subseteq lrarg_{\mathsf{eval}}(f)$. Unfortunately, the converse statement is not generally true. The following result in [2] ensures that local redundancy implies redundancy when the TRS is ground confluent and sufficiently complete.

Theorem 1. *Let $\mathcal{R}$ be a ground confluent and sufficiently complete TRS. Then, for all $f \in \Sigma$, $lrarg_{\mathsf{eval}}(f) = rarg_{\mathsf{eval}}(f)$.*

Now the question of how to single out locally redundant arguments arises. In order to tackle this problem, we formalize the redundancy problem in terms of the inductive theory of the program.

3.1 Inductive theorems expressing redundancy of arguments

The following results formalize the relation between inductive theorems and redundancy of arguments in confluent and sufficiently complete TRSs.

Proposition 1. *Let $\mathcal{R}$ be a confluent TRS, $f \in \Sigma$, and $i \in \{1, \ldots, ar(f)\}$. The i-th argument of f is* locally redundant *(w.r.t. eval) if the equation $t = t[y]_i$ is an inductive theorem of $\mathcal{R}$, where $t = f(x_1, \ldots, x_{ar(f)})$ and $x_1, \ldots, x_{ar(f)}, y$ are distinct variables.*

Theorem 2. *Let $\mathcal{R}$ be a confluent and sufficiently complete TRS, $f \in \Sigma$, and $i \in \{1, \ldots, ar(f)\}$. The i-th argument of f is* redundant *(w.r.t. eval) iff the equation $t = t[y]_i$ is an inductive theorem of $\mathcal{R}$; where $t = f(x_1, \ldots, x_{ar(f)})$ and $x_1, \ldots, x_{ar(f)}, y$ are distinct variables.*

Hence, for confluent, sufficiently complete TRSs, the redundancy problem is reduced to the problem of checking validity of a particular class of inductive theorems. The problem of identifying the inductive theory of a TRS is in general undecidable, as shown by [11] even for a very restricted class of TRSs: finite, canonical, left- and right-linear, and right monadic (right hand sides have depth at most 1) CSs. However, several methods for (semi)-automatically proving validity of inductive theorems have been developed, such as the *cover set* method [28], *test set* method [9], *rewriting induction* method [25], and *inductionless induction* method [10, 11], which generalizes the former ones. Also, the *abstract rewriting* method of [8] can be used for proving inductive theorems.

In the following, we show how, both the inductionless induction as well as abstract rewriting methods can be successfully applied for detecting redundancy of arguments, where the methods discussed in [2] fail.

4 Inductionless induction

We briefly recall the inductionless induction method for proving validity of inductive theorems (see [10, 11] for details). Inductionless induction tackles how to (semi)-automatically prove a set of equations $\mathcal{C}$ in the minimal Herbrand model of a set of equations E without making use of induction schemes (induction rules). It uses a (first-order) axiomatization $\mathcal{A}$ of the minimal model of E, $\mathcal{I}_E$, such that $\mathcal{C} \cup \mathcal{A} \cup E$ is consistent if and only if $\mathcal{C}$ is valid in $\mathcal{I}_E$, i.e. $\mathcal{I}_E \models \mathcal{C}$. A *normal axiomatization* $\mathcal{A}$ of $\mathcal{I}_E$ is a finite recursive set of purely universal formulas such that $\mathcal{I}_E \models \mathcal{A}$, $\mathcal{I}_E$ is the only Herbrand model of $E \cup \mathcal{A}$ up to isomorphism, and for all ground terms s, t representative of its congruence class of $\mathcal{I}_E$, $s \not\equiv t \Rightarrow \mathcal{A} \models s \neq t$. The method relies on *saturation techniques* [5, 6] for performing the proof by consistency of $\mathcal{C} \cup \mathcal{A} \cup E$, thus any saturation-based general-purpose first-order theorem prover can be used for inductive validity. The (in)consistency proofs are performed in two stages: first deductions on $\mathcal{C} \cup E$ are computed by saturation, yielding new consequences; then, these new consequences are checked for inconsistency w.r.t. $\mathcal{A}$.

Deductions are performed by *superposition* defined by the following inference rules between the set of equations E and an equation $c \in \mathcal{C}$. We assume below that $\succeq$ is a reduction ordering [14] which is total on ground terms, i.e. a relation $\succ$ which is irreflexive, transitive, well-founded, total, monotonic, and stable under substitutions. A well-known reduction ordering is the *recursive path ordering*, based on a total ordering $\succ_\Sigma$ (called *precedence*) on Σ.

$$\text{Superposition} \quad \frac{l = r \qquad c[s]}{\sigma(c[r])} \quad \begin{array}{l}\text{if } \sigma = mgu(l, s), s \text{ is not a variable,}\\ \quad \sigma(r) \not\succeq \sigma(l), l = r \in E, c[s] \in \mathcal{C}.\end{array}$$

$$\text{Equality resolution} \quad \frac{C \vee s \neq t}{\sigma(C)} \quad \text{if } \sigma = mgu(s, t), C \vee s \neq t \in \mathcal{C}.$$

Given a ground equation c, $\mathcal{C}^{\prec}$ is the set of ground instances of equations in $\mathcal{C}$ that are strictly smaller than c in this ordering. A ground equation (conjecture) c

is *entailed* by a set of equations (conjectures) $\mathcal{C}$ if $E \cup \mathcal{A} \cup \mathcal{C}^{\prec} \vdash c$. A non-ground equation is entailed if all its ground instances are. An inference is redundant if one of its premises or its conclusion are entailed by $\mathcal{C}$. A set of equations is *saturated* if all inferences are redundant. A *derivation sequence* is a sequence $\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_n, \ldots$ such that each $\mathcal{C}_{i+1}$ is obtained from $\mathcal{C}_i$ by adding some logical consequences or by removing some entailed equations. A derivation sequence is *fair* if every equation which can be persistently derived is eventually derived.

The application of inductionless induction to the redundancy problem is illustrated in the following.

Example 2. Consider the optimized TRS of Example 1, which is saturated and can be oriented using the recursive path ordering with the precedence $\mathtt{minusplus} > \mathtt{minus1} > \mathtt{plus0} > \mathtt{s} > \mathtt{0}$, and the axiomatization $\{\forall \mathtt{x}, \mathtt{y}, \mathtt{s(x)} \neq \mathtt{0} \wedge \mathtt{s(x)} = \mathtt{s(y)} \Rightarrow \mathtt{x} = \mathtt{y}\}$.
We can prove the redundancy of the second argument of `minusplus` by proving the validity of the equation (c_1) $\mathtt{minusplus(x,y)} = \mathtt{minusplus(x,w)}$. We have four possible inferences by saturation, the other ones are renamings:

$$
\begin{array}{lrcl}
(c_{1,1}) & \mathtt{0} & = & \mathtt{minusplus(0,w)} \\
(c_{1,2}) & \mathtt{s(plus0(x))} & = & \mathtt{minusplus(s(x),w)} \\
(c_{1,3}) & \mathtt{minus1(y)} & = & \mathtt{minusplus(0,w)} \\
(c_{1,4}) & \mathtt{s(minusplus(x,y))} & = & \mathtt{minusplus(s(x),w)}
\end{array}
$$

Here, no equation is entailed. After superposing again, we obtain (up to renaming):

$$
\begin{array}{lrcl}
(c_{1,5}) & \mathtt{0} & = & \mathtt{0} \\
(c_{1,6}) & \mathtt{0} & = & \mathtt{minus1(w)} \\
(c_{1,7}) & \mathtt{s(0)} & = & \mathtt{minusplus(s(0),w)} \\
(c_{1,8}) & \mathtt{s(s(plus0(x)))} & = & \mathtt{minusplus(s(s(x)),w)} \\
(c_{1,9}) & \mathtt{s(plus0(x))} & = & \mathtt{s(plus0(x))} \\
(c_{1,10}) & \mathtt{s(plus0(x))} & = & \mathtt{s(minusplus(x,w))} \\
(c_{1,11}) & \mathtt{minus1(y)} & = & \mathtt{minus1(w)} \\
(c_{1,12}) & \mathtt{s(minus1(y))} & = & \mathtt{minusplus(s(0),w)} \\
(c_{1,13}) & \mathtt{s(s(minusplus(x,y)))} & = & \mathtt{minusplus(s(s(x)),w)} \\
(c_{1,14}) & \mathtt{s(minusplus(x,y))} & = & \mathtt{s(minusplus(x,w))}
\end{array}
$$

Here, all equations are entailed: $c_{1,5}$ and $c_{1,9}$ are trivially entailed, and $c_{1,1} \cup c_{1,3} \vdash c_{1,6}$, $c_{1,2} \cup r_5 \vdash c_{1,7}$, $c_{1,2} \cup r_6 \vdash c_{1,8}$, $c_{1,2} \cup c_{1,4} \vdash c_{1,10}$, $c_{1,3} \vdash c_{1,11}$, $c_{1,3} \cup c_{1,4} \vdash c_{1,12}$, $c_{1,4} \vdash c_{1,13}$, and $c_1 \vdash c_{1,14}$.

Then, the set $\mathcal{R} \cup \{c_1, c_{1,1}, c_{1,2}, c_{1,3}, c_{1,4}\}$ is saturated and it is immediate to check its consistency w.r.t. the axiomatization. Therefore, the theorem is proved and hence the second argument of `minusplus` is redundant. We have checked this automatically by using the theorem prover Spike [9] which implements a particular implicit induction technique, namely the one which is based on *test sets*.

Unfortunately, in many interesting cases, existing methods for inductive validity may run forever without proving validity of inductive theorems as in the following example.

Example 3. Let $\mathcal{R}$ be the following saturated TRS oriented using the recursive path ordering with the precedence $\mathtt{f} > \mathtt{id} > \mathtt{s} > \mathtt{0}$ and status left to right for `f`:

$$
\begin{array}{ll}
\mathtt{f(0,0)} \to \mathtt{0} & \mathtt{id(0)} \to \mathtt{0} \\
\mathtt{f(0,s(y))} \to \mathtt{f(0,y)} & \mathtt{id(s(x))} \to \mathtt{s(id(x))} \\
\mathtt{f(s(x),0)} \to \mathtt{f(x,0)} & \\
\mathtt{f(s(x),s(y))} \to \mathtt{f(x,s(id(y)))} &
\end{array}
$$

Consider the axiomatization of Example 2. The first and second arguments of `f` are redundant whereas it cannot be detected by previous redundancy results in [2]. In order to prove the redundancy of the first argument of `f`, we consider the conjecture: (c_1) $\mathtt{f(x,y)} = \mathtt{f(z,y)}$. Superposing c_1 with $\mathcal{R}$, we get the following (the other equations are renamings):

$$
\begin{array}{lrcl}
(c_{1,1}) & \mathtt{0} & = & \mathtt{f(z,0)} \\
(c_{1,2}) & \mathtt{f(0,y)} & = & \mathtt{f(z,s(y))} \\
(c_{1,3}) & \mathtt{f(x,0)} & = & \mathtt{f(z,0)} \\
(c_{1,4}) & \mathtt{f(x,s(id(y)))} & = & \mathtt{f(z,s(y))}
\end{array}
$$

Here, equation $c_{1,3}$ is entailed. Superposing $c_{1,1}$, $c_{1,2}$ and $c_{1,4}$ with $\mathcal{R}$, we obtain (up to renaming):

$$
\begin{array}{lrcl}
(c_{1,4}) & \mathtt{0} & = & \mathtt{0} \\
(c_{1,5}) & \mathtt{0} & = & \mathtt{f(z,0)} \\
(c_{1,6}) & \mathtt{f(0,y)} & = & \mathtt{f(0,y)} \\
(c_{1,7}) & \mathtt{f(0,y)} & = & \mathtt{f(z,s(id(y)))} \\
(c_{1,8}) & \mathtt{0} & = & \mathtt{f(z,s(0))} \\
(c_{1,9}) & \mathtt{f(0,y)} & = & \mathtt{f(z,s(s(y)))} \\
(c_{1,10}) & \mathtt{f(x,s(id(y)))} & = & \mathtt{f(z,s(id(y)))} \\
(c_{1,11}) & \mathtt{f(x,s(0))} & = & \mathtt{f(z,s(0))} \\
(c_{1,12}) & \mathtt{f(x,s(s(id(y))))} & = & \mathtt{f(z,s(s(y)))}
\end{array}
$$

Here, all equations except $c_{1,12}$ are entailed. Thus, we can obtain infinitely many equations
$\mathtt{f(x,s}^n\mathtt{(id(y)))} = \mathtt{f(z,s}^n\mathtt{(y))}$ and the process may run forever unless an extra lemma $\mathtt{id(x)} = \mathtt{x}$ is manually provided.

Several techniques to improve termination of the inductive validity process have been developed such as deduction of lemmas which might help to prove an inductive theorem [26, 17, 16]. On the other hand, different criteria can be used to stop the saturation process, such as the homeomorphic embedding (which is commonly used in program transformation for avoiding infinite sequences [3]). Unfortunately, important properties, such as refutationally completeness or finite saturation under common conditions, get lost.

Therefore, it is interesting to consider decidable classes of TRSs where the inductionless induction method terminates, and thus, redundancy of arguments can be decided. In the next section we present a result for the decidability of redundancy based on inductionless induction, which is complementary to the decidability result of [2]. We postpone to Section 5 the use of finite approximations based on abstract interpretation, such as the abstract rewriting of [8], to formalize static analyses of redundancy.

4.1 Standard Theories

In this section we consider the standard theories of [23], a class of TRSs where the saturation process is finite, thus the validity of an inductive theorem is decidable.

Standard theories are particular sets of equations which are finitely closed by superposition. We need the following: the *depth* d of a subterm $s = t|_p$ is the length of the position p: $d = |p|$; and a variable is *shallow* in a term if it occurs only at depth 0 or 1 in the term.

Definition 3. *[23] A* standard *signature* Σ *is a signature where every function symbol* f *in* Σ *has an associated set of shallow positions* $sh(f)$ *and a set of linear positions* $lin(f)$, *such that* $lin(f) \cap sh(f) = \varnothing$ *and* $sh(f) \cup lin(f) = \{1, \ldots, ar(f)\}$.

Definition 4. *[23] A term* s *is a* standard *term iff it is a variable or a term of the form* $f(s_1, \ldots, s_n)$ *where if* $i \in sh(f)$ *then* s_i *is a variable or a ground term and if* $i \in lin(f)$ *then all variables in* s_i *are linear in* s.

Note that, according to the previous definition, all ground terms are standard, whereas not every linear term is, because no term with variables occurring at depth ≥ 1 are allowed at shallow positions. Furthermore, the only non-linear variables of a standard term are shallow variables occurring at shallow positions.

Definition 5. *[23] An equation* $s = t$ *is* standard *iff*

1. *s is linear and t is ground or*
2. *s is a standard term $f(\ldots, g(t), \ldots)$ and t is a variable or*
3. *s and t are standard terms sharing only shallow variables and no variable x is both a shallow position argument and a linear position argument in $s = t$.*

A standard presentation *is a set of* standard equations *and a* standard theory *is a theory axiomatizable by a standard presentation.*

Theorem 3. *[23] Every standard presentation E can be finitely closed under superposition.*

Hence, for standard theories, the saturation process is finite and then the inductionless induction method (hence, the redundancy of arguments) is decidable. Note that we naturally specialize the notion of standard presentations (as originally defined in [23]) to "standard TRSs".

Theorem 4. *Let $\mathcal{R}$ be a standard confluent TRS. Let the equation $t = s$ be standard (within $\mathcal{R}$). It is decidable if $t = s$ is an inductive theorem of $\mathcal{R}$.*

Corollary 1. *Let $\mathcal{R}$ be a standard, confluent, and sufficiently complete TRS, $f \in \Sigma$, and $i \in \{1, \ldots, ar(f)\}$. Let the equation $t = t[y]_i$ be standard (within $\mathcal{R}$) such that $t = f(x_1, \ldots, x_{ar(f)})$ and $x_1, \ldots, x_{ar(f)}, y$ are distinct variables. It is decidable if the i-th argument of f is* redundant *(w.r.t.* eval*).*

Even if the class of standard theories is somehow restrictive, it still allows to detect redundancy of arguments in significant examples.

Example 4. Consider the following TRS, where extra variables are allowed in right hand sides.

```
f(0,y) → y                 g(0,y) → y
f(s(x),y) → g(u,y)         g(s(x),y) → f(u,y)
```

Here, we can automatically prove that the first argument of `f` (and `g`) is redundant. Note that this example cannot be dealt by previous results in [2].

We consider the problem of identifying new decidable classes of TRSs (w.r.t. the particular class of inductive theorems which express redundancy), as well as developing new decision algorithms for these programs, as an interesting line of work which we plan to pursue as future work. In the following section, however, instead of focusing in deeper decidability matters, we investigate finite approximations of the validity problem which lead us to formalize more practical static redundancy analyses. As an application of the analysis, we revisit Example 3, which is shown to be correctly analyzed by applying the new methodology based on abstract rewriting, whereas it is not coped by (unoptimized) inductionless induction.

5 Abstract rewriting

Abstract interpretation is a theory to extract relevant information from programs without considering all details given by the standard semantics [12]. In [8], Bert and Echahed proposed a framework called *abstract rewriting* which is based on an abstract interpretation of (conditional) term rewriting systems for approximating the normal form $t{\downarrow_{\mathcal{R}}}$ of a term t in a canonical CS $\mathcal{R}$. They make use of the notion of an abstract domain of terms. In this section, we use the technique of abstract rewriting in order to prove inductive theorems, and thus to detect redundant arguments, in the setting of canonical and sufficiently complete CS's. We first recall the abstract rewriting methodology of [8].

Definition 6 (Abstract Domain). [8] *Let Σ be a signature. The abstract specification of Σ is $\mathcal{A}(\Sigma) = \Sigma \cup \{\top, \bot, \sqcup, \sqcap\}$.*

Intuitively, an abstract term t approximates the set of its ground instances, where the symbols $\bot$ and $\top$ stand for the empty set and the set of all constructor terms, respectively. Similarly, the symbols $\sqcup$ and $\sqcap$ correspond to set union and set intersection operators, respectively.

Let $\_^a : \Sigma \to \mathcal{A}(\Sigma)$ be the obvious identity signature morphism between the concrete and the abstract signatures. The signature morphism $\_^a$ is extended to a translation function on terms $\_^a : \mathcal{T}(\Sigma, \mathcal{X}) \to \mathcal{T}(\mathcal{A}(\Sigma))$ such that $x^a = \top\ \forall x \in \mathcal{X}$ and $(f(t_1, \ldots, t_n))^a = f^a(t_1^a, \ldots, t_n^a)\ \forall f \in \Sigma$. Besides, given an abstract term t, the concrete set $\gamma(t)$ is the largest set of ground terms such that $\gamma(\top) = \mathcal{T}(\Sigma)$, $\gamma(\bot) = \varnothing$, $\gamma(f^a(t_1, \ldots, t_{ar(f)})) = \{f(s_1, \ldots, s_{ar(f)}) \mid \forall 1 \leq i \leq ar(f), s_i \in \gamma(t_i)\}$, $\gamma(t_1 \sqcup t_2) = \gamma(t_1) \cup \gamma(t_2)$, and $\gamma(t_1 \sqcap t_2) = \gamma(t_1) \cap \gamma(t_2)$.

In order to approximate normal forms of terms, it is defined a partial order $\leq$ on abstract terms such that $t \leq t'$ iff $\gamma(t) \subseteq \gamma(t')$. Given a term $s \in \mathcal{T}(\Sigma, \mathcal{X})$, $t \in \mathcal{T}(\mathcal{A}(\Sigma))$ is an approximation of s iff $\forall s' \in \mathcal{T}(\Sigma)$ such that $(s')^a \leq s^a$, $(s'{\downarrow_\mathcal{R}})^a \leq t$, or equivalently $\{\sigma(s){\Downarrow_\mathcal{R}} \mid \sigma(s) \in \mathcal{T}(\Sigma)\} \subseteq \gamma(t)$.

The set of abstract terms is larger than the set of concrete terms. In order to compute approximations of normal forms of concrete terms, finite subsets of the set of abstract terms are introduced by the so-called finite upper closures $up : \mathcal{T}(\mathcal{A}(\Sigma)) \to \mathcal{T}(\mathcal{A}(\Sigma))$ such that up is *monotonic* $(\forall t, t' \in \mathcal{T}(\mathcal{A}(\Sigma)), t \leq t' \Rightarrow up(t) \leq up(t'))$, *extensive* $(\forall t \in \mathcal{T}(\mathcal{A}(\Sigma)), t \leq up(t))$, and *idempotent* $(up \circ up = up)$. The main objective of a finite upper closure is to restrict $\mathcal{T}(\mathcal{A}(\mathcal{C}))$ to a finite set $\mathcal{T}^{up}(\mathcal{A}(\mathcal{C}))$.

The notions of *abstract rewriting system* and *abstract rewriting calculus* are defined. An abstract rewriting system is associated to a TRS $\mathcal{R}$ and a finite upper closure up in order to approximate the normal forms of ground instances of concrete terms with variables. In concrete, a "computed" abstract TRS efficiently determines an approximation for any concrete term. Due to the properties of abstract terms and the ordering $\leq$, the classical definition of rewriting is extended to the abstract rewriting calculus (see [8] for details).

Now, we can exploit abstract rewriting for proving inductive theorems, which demonstrates redundancy of arguments. Let us first introduce some auxiliary results.

Definition 7. *We define the set of up-minimal substitutions as:* $Subst_{up}(\mathcal{C}, \mathcal{X}) = \{\sigma \in Subst(\mathcal{C}, \mathcal{X}) \mid \forall \sigma' \in Subst(\mathcal{C}, \mathcal{X}) \wedge \forall x \in \mathcal{X}, \sigma'(x)^a \leq \sigma(x)^a \Rightarrow up(\sigma'(x)^a) \equiv \sigma(x)^a\}$

Theorem 5. *Let $\mathcal{R}$ be a left linear, canonical, sufficiently complete CS, and* up *be a finite upper closure. Let $\mathcal{R}_c^{up}$ be the "computed" abstract TRS associated to $\mathcal{R}$ and up. The equation $s = t$ is an inductive theorem of $\mathcal{R}$ if for all $\sigma \in Subst_{up}(\mathcal{C}, \mathcal{X})$, $up(\sigma(s)^a){\Downarrow_{\mathcal{R}_c^{up}}} \equiv up(\sigma(t)^a){\Downarrow_{\mathcal{R}_c^{up}}} \equiv \delta$ such that $\delta \in \mathcal{T}(\mathcal{A}(\mathcal{C}) - \{\top, \bot, \sqcup, \sqcap\})$.*

The following result is the key for the detection of redundant arguments.

Corollary 2. *Let $\mathcal{R}$ be a left linear, canonical, sufficiently complete CS, and* up *be a finite upper closure. Let $\mathcal{R}_c^{up}$ be the "computed" abstract TRS associated to $\mathcal{R}$ and up. Let $f \in \Sigma$ and $i \in \{1, \ldots, ar(f)\}$. The i-th argument of f is* redundant *(w.r.t.* eval*) iff for all $\sigma \in Subst_{up}(\mathcal{C}, \mathcal{X})$, $up(\sigma(t)^a){\Downarrow_{\mathcal{R}_c^{up}}} \equiv up(\sigma(t[y]_i)^a){\Downarrow_{\mathcal{R}_c^{up}}} \equiv \delta$ such that $\delta \in \mathcal{T}(\mathcal{A}(\mathcal{C}) - \{\top, \bot, \sqcup, \sqcap\})$, where $t = f(x_1, \ldots, x_{ar(f)})$ and $x_1, \ldots, x_{ar(f)}, y$ are distinct variables.*

Clearly, the analysis depends on the chosen upper closure up. A standard upper closure is defined by taking the maximum depth of all constructor terms of left-hand sides of a TRS.

Example 5. Consider the TRS of Example 3 and the finite upper closure *head* defined by $head(\mathtt{0}^a) = \mathtt{0}^a$, $head(\mathtt{s}(t)) = \mathtt{s}(\top)$ if $t \in \mathcal{T}(\mathcal{C})$, and $head(f(t_1, \ldots, t_n)) = f(head(t_1), \ldots, head(t_n))$ otherwise. We can prove by abstract rewriting that both arguments of f are redundant. The "computed" abstract TRS for $\mathcal{R}$ and *head* is:

$$
\begin{array}{ll}
\mathtt{f}^a(\mathtt{0}^a,\mathtt{0}^a) \to \mathtt{0}^a & \mathtt{id}^a(\mathtt{0}^a) \to \mathtt{0}^a \\
\mathtt{f}^a(\mathtt{0}^a,\mathtt{s}^a(\top)) \to \mathtt{0}^a & \mathtt{id}^a(\mathtt{s}^a(\top)) \to \mathtt{s}^a(\top) \\
\mathtt{f}^a(\mathtt{s}^a(\top),\mathtt{0}^a) \to \mathtt{0}^a & \\
\mathtt{f}^a(\mathtt{s}^a(\top),\mathtt{s}^a(\top)) \to \mathtt{0}^a &
\end{array}
$$

Then, the first and second arguments of f are redundant since for the equations $\mathtt{f}(x, y) = \mathtt{f}(x', y)$ and $\mathtt{f}(x, y) = \mathtt{f}(x, y')$, and every substitution $\sigma \in Subst_{head}(\mathcal{C}, \mathcal{X})$, $\sigma(\mathtt{f}(x, y))^a\!\downarrow_{\mathcal{R}_c^{up}} \equiv \sigma(\mathtt{f}(x', y))^a\!\downarrow_{\mathcal{R}_c^{up}} \equiv \mathtt{0}^a$ and $\sigma(\mathtt{f}(x, y))^a\!\downarrow_{\mathcal{R}_c^{up}} \equiv \sigma(\mathtt{f}(x, y'))^a\!\downarrow_{\mathcal{R}_c^{up}} \equiv \mathtt{0}^a$. Note that this example can not be handled by the inductionless induction method nor by previously discussed methods such as [1, 2].

For the sake of clarity, let us finally show, by means of one example, that there exist still interesting cases where the method in [2] succeeds whereas none of the results in this paper apply. This demonstrates that the different methods are incomparable and hence could be fruitfully combined to develop a practical tool for the detection of redundant arguments.

Example 6. Consider the following TRS $\mathcal{R}$ which is a slight modification of Example 3.26 of [4]:

```
h(0,0) → s(0)
h(0,s(y)) → s(0)
h(s(0),y) → s(s(0))
h(s(s(x)),y) → s(h(h(x,y),y))
```

The inductionless induction method can not automatically prove that the second argument of h is redundant since there is no automatizable reduction ordering for the TRS which could orient the equations (see [4]). On the other hand, abstract rewriting can not prove the redundancy of the second argument of h since for any finite upper closure there exists a value for the second argument which returns a term containing $\top$, e.g. using the upper closure *head* of Example 5, $\mathtt{h}(\mathtt{s}(\top), \mathtt{0})\!\downarrow_{\mathcal{R}_c^{up}} \equiv \mathtt{s}(\top)$. Nevertheless, the method in [2] succeeds in proving that the second argument of h is redundant since all variables of the second argument appear in positions of redundant arguments of the rhs of the corresponding rule and $\mathtt{s(0)} \downarrow \mathtt{s(0)}$.

6 Conclusion

We have shown how the problem of detecting redundant arguments reduces to that of validity of inductive theorems in confluent, sufficiently complete TRSs. As the set of inductive theorems is not recursively enumerable in general, we identify a class of rewrite systems in which detection of redundant arguments is decidable.

We have also shown how "inductionless induction" as well as "abstract rewriting" techniques can be applied to detect redundant arguments and particularly in some examples that cannot be handled by previously developed methods.

However, the natural question whether it is possible to specialize methods for inductive validity to the concrete problem of redundancy arises. In future work, we plan to deepen on this point as well as to integrate the methods described in this paper into the prototype tool presented in [2].

References

1. M. Alpuente, S. Escobar, and S. Lucas. Redundant Arguments in Term Rewriting. In *9th Int'l Workshop on Functional and Logic Programming (WFLP2000)*, SPUPV 2000.2039, pages 309–323, 2000.
2. M. Alpuente, S. Escobar, and S. Lucas. Removing Redundant Arguments of Functions. In *9th International Conference on Algebraic Methodology And Software Technology, AMAST 2002*, LNCS to appear. Springer-Verlag, 2002.
3. M. Alpuente, M. Falaschi, and G. Vidal. Partial Evaluation of Functional Logic Programs. *ACM Toplas*, 20(4):768–844, 1998.
4. T. Arts and J. Giesl. A collection of examples for termination of term rewriting using dependency pairs. Technical Report AIB-2001-09, RWTH Aachen, Germany, 2001.
5. L. Bachmair and H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):217–247, June 1994.
6. L. Bachmair and H. Ganzinger. Resolution theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume 1, chapter 2, pages 19–99. Elsevier Science, 2001.
7. S. Berardi, M. Coppo, F. Damiani, and P. Giannini. Type-based useless-code elimination for functional programs. In *Proceedings of SAIG 2000*, volume 1924 of *LNCS*, pages 172–189. Springer-Verlag, 2000.
8. D. Bert and R. Echahed. Abstraction of Conditional Term Rewriting Systems. In J. Lloyd, editor, *International Symposium on Logic Programming, ILPS-95, Portland, Oregon*, pages 162–176. MIT Press, 1995.
9. A. Bouhoula and M. Rusinowitch. Implicit induction in conditional theories. *Journal of Automated Reasoning*, 14:189–235, 1995.
10. H. Comon. Inductionless induction. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume 1, chapter 14, pages 913–962. Elsevier Science, 2001.
11. H. Comon and R. Nieuwenhuis. Induction = I-Axiomatization + First-Order Consistency. *Information and Computation*, 159(1/2):151–186, 2000.
12. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, NY.
13. P. Cousot and R. Cousot. Higher-order abstract interpretation (and application to comportment analysis generalizing strictness, termination, projection and PER analysis of functional languages), invited paper. In *Proceedings of the 1994 International Conference on Computer Languages, ICCL'94*, pages 95–112, Toulouse, France, 16–19 May 1994. IEEE Computer Society Press, Los Alamitos, California.

14. N. Dershowitz and D. Plaisted. Rewriting. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume 1, chapter 9, pages 535–610. Elsevier Science, 2001.
15. J. Hughes. Backwards Analysis of Functional Programs. In D. Bjørner, A.P. Ershov, and N.D. Jones, editors, *IFIP Workshop on Partial Evaluation and Mixed Computation*, pages 187–208, 1988.
16. A. Ireland and A. Bundy. Using failure to guide inductive proofs. *Automated Reasoning*, 16:38–35, 1996.
17. D. Kapur and M. Subramaniam. Lemma discovery in automating induction. In *Proc. of CADE'96*, LNCS 914, pages 403–407. Springer-Verlag, Berlin, 1996.
18. N. Kobayashi. Type-based useless variable elimination. In *Proceedings of PEPM-00*, pages 84–93. ACM Press, 2000.
19. M. Leuschel. The ECCE partial deduction system and the DPPD library of benchmarks. Accessible via `http://www.cs.kuleuven.ac.be/~lpai.`, 1998.
20. M. Leuschel and M. H. Sørensen. Redundant Argument Filtering of Logic Programs. In John Gallager, editor, *Proceedings of the 6th International Workshop on Logic Program Synthesis and Transformation (LOPSTR'96)*, volume 1207 of *Lecture Notes in Computer Science*, pages 83–103, Stockholm, Sweden, August 1996. Springer-Verlag, Berlin.
21. Y. A. Liu and S. D. Stoller. Eliminating dead code on recursive data. *Science of Computer Programming*, 2002. Preliminary version in *Proc. of SAS'99*, LNCS 1694:211–231. Springer-Verlag, Berlin, 1999.
22. S. Lucas. Transfinite rewriting semantics for term rewriting systems. In *Proc. of 12th International Conference on Rewriting Techniques and Applications, RTA'01*, LNCS 2051, pages 216–230. Springer-Verlag, Berlin, 2001.
23. R. Nieuwenhuis. Basic paramodulation and decidable theories. In *Proceedings of the Eleventh Annual IEEE Symposium On Logic In Computer Science (LICS'96)*, pages 473–483, New York, USA, 1996. IEEE Computer Society Press.
24. A. Pettorossi and M. Proietti. Transformation of logic programs: Foundations and techniques. *Journal of Logic Programming*, pages 261–32, 1994.
25. U.S. Reddy. Term rewriting induction. In M. E. Stickel, editor, *Proceedings of the 10th International Conference on Automated Deduction*, volume 449 of *Lecture Notes in Computer Science*, pages 162–177. Springer-Verlag, 1990.
26. T. Walsh. A divergence critic for inductive proofs. *Artificial Intelligence Research*, 4:209–235, 1996.
27. M. Wand and I. Siveroni. Constraint systems for useless variable elimination. In *Proceedings of POPL'99*, LNCS, pages 291–302. Springer-Verlag, 1999.
28. H. Zhang. *Reduction, Superposition, and Induction: Automated Reasoning in an Equation Logic.* PhD thesis, Renselaer Polytechnic Institute, 1988.