Centrum voor Wiskunde en Informatica

**REPORT***RAPPORT*

Implicit induction techniques for the verification of PIM - a transformational toolkit for compilers

D. Naidich and T.B. Dinesh

Computer Science/Department of Software Technology

**CS-R9630 1996**

Report CS-R9630
ISSN 0169-118X

CWI
P.O. Box 94079
1090 GB  Amsterdam
The Netherlands

# Implicit Induction Techniques for the Verification of PIM – A Transformational Toolkit for Compilers

Dimitri Naidich T.B. Dinesh
*CWI*
*P.O. Box 94079, 1090 GB Amsterdam, The Netherlands*
e-mail: {dimitri,dinesh}@cwi.nl

**Abstract**

The development of the proof techniques presented in this paper was inspired by a proof problem for PIM – a transformational toolkit for compilers. PIM consists of the untyped lambda calculus extended with an algebraic rewriting system that characterizes the behavior of lazy stores and generalized conditionals. The first-order algebraic component of PIM has an $\omega$-complete conservative extension. Showing conservativeness of the extension requires proving that the additional equations of the extension are inductive consequences of the initial axioms. The complexity of the manual proofs motivated us to look into the current implicit induction procedures w.r.t. their applicability to this proof problem. However, the existing implicit induction methods turned out to be inadequate. In this paper we propose new implicit induction techniques adequate for solving the indicated proof problem.

*CR Subject Classification (1991):* I.2.3, I.1.3, I.2.2, D.3.4.
*Keywords & Phrases:* automated induction, code optimization, algebraic specifications, term rewriting.
*Note:* This work was supported by the Netherlands Organization for Scientific Research (NWO) in part under the *Generic Tools for Program Analysis and Optimization* project and in part by a *visiting research fellowship.*

## 1 Introduction

Algrebraic specifications are a useful means of prototyping programming languages [10]. Proofs by induction are important for the analysis of algebraic specifications [23], and the proof automation is necessary for solving practical problems. Advanced automated induction proof methods like [6, 7, 4], sometimes called *implicit induction*, have been developed for algebraic specifications. However, by the well-known incompleteness result for the induction proof methods [9], the development of new proof techniques is justifiable, in general.

The development of new proof techniques is highly stimulated by the case studies that cause problems to the existing proof methods. The development of the proof techniques presented in this paper was inspired by a proof problem for PIM – a transformational toolkit for compilers [11, 3]. PIM consists of the untyped lambda calculus extended with an algebraic rewriting system that characterizes the behavior of lazy stores and generalized conditionals. A major question left open in [11] was whether there existed a complete equational axiomatization of PIM's semantics. In [3], the affirmative answer to this question was given for $\text{PIM}_t$, the PIM's core algebraic component, under the assumption of certain reasonable restrictions on term formation. The complete $\text{PIM}_t$ logic was systematically derived as a two-step extension of the initial equational system $\text{PIM}_t^0$, the straightforward "interpreter" for closed $\text{PIM}_t$ terms. Since the intended semantics of $\text{PIM}_t^0$ in [11] was the final one, the first extension provided the initial algebra specification $\text{PIM}_t^+$ of the final algebra of $\text{PIM}_t^0$. The following extension $\text{PIM}_t^=$ provided the complete specification of the initial algebra of $\text{PIM}_t^+$. The proof of the completeness result involved proving that the additional equations of $\text{PIM}_t^=$ are inductive theorems of $\text{PIM}_t^+$. Some of these proofs turned out to be quite long with many cases which made the proof automation very desirable. This motivated us to look into the current implicit induction procedures w.r.t. their applicability to the above problem. However, due to the reasons discussed in section 3, the existing implicit induction methods turned out to be not quite adequate.

In this paper we propose new implicit induction techniques which enable highly automated proofs of the $\text{PIM}_t^=$ conjectures. The range of possible applications of the presented proof techniques is not confined to the solution of this particular problem. Application of induction reasoning for the analysis of programs in the full PIM is essential because of the absence of the complete axiomatization for the operational semantics of a universal programming language. Therefore, the presented proof techniques may constitute a part of a theorem proving environment for the full PIM.

## 2 The Complete Specification for $\text{PIM}_t$

The signature of the PIM's core algebraic component $\text{PIM}_t$ is given in Fig. 1. The equational system $\text{PIM}_t^+$ providing the initial algebra semantics for the $\text{PIM}_t$ interpreter is presented in Fig. 2. Finally, $\text{PIM}_t^=$ the complete extension of $\text{PIM}_t^+$ is presented in Fig. 3.

## 3 Preliminary Analysis

Many of the conjectures of $\text{PIM}_t^=$ are not problematic for the application of implicit induction. The main characteristic feature of the implicit induction approach is the way of using induction hypotheses. With the conventional induction methods the instances of induction hypotheses used in the proof should be fixed prior to the proof of the induction steps. Within implicit induction, *any* instance of the induction hypothesis can be used in the proof of the induction cases in a goal-oriented way. The correctness of applying the induction hypothesis is guarantied by the restriction imposed on the goal transformations: the transformations should decrease the complexity of the goal w.r.t. a well-founded ordering on propositons, so that the complexity of any applicable instance of the induction hypothesis is lower than that of the initial goal.

**sorts**

| | |
|---|---|
| $\mathcal{S}$ | (store structures) |
| $\mathcal{M}$ | (merge structures) |
| $\mathcal{A}$ | (addresses) |
| $\mathcal{B}$ | (booleans) |
| $\mathcal{V}$ | (base values) |

**functions**

| | | | |
|---|---|---|---|
| $\{\mathcal{A} \mapsto \mathcal{M}\}$ | $\rightarrow$ | $\mathcal{S}$ | (store cell) |
| $\mathcal{B} \rhd_s \mathcal{S}$ | $\rightarrow$ | $\mathcal{S}$ | (guarded store) |
| $\mathcal{S} \circ_s \mathcal{S}$ | $\rightarrow$ | $\mathcal{S}$ | (store composition) |
| $\emptyset_s$ | $\rightarrow$ | $\mathcal{S}$ | (null store) |
| $\mathcal{S} \mathbin{@} \mathcal{A}$ | $\rightarrow$ | $\mathcal{M}$ | (store dereference) |
| $[\mathcal{V}]$ | $\rightarrow$ | $\mathcal{M}$ | (merge cell) |
| $\mathcal{B} \rhd_m \mathcal{M}$ | $\rightarrow$ | $\mathcal{M}$ | (guarded merge) |
| $\mathcal{M} \circ_m \mathcal{M}$ | $\rightarrow$ | $\mathcal{M}$ | (merge composition) |
| $\emptyset_m$ | $\rightarrow$ | $\mathcal{M}$ | (null merge) |
| $\alpha_1, \alpha_2, \ldots$ | $\rightarrow$ | $\mathcal{A}$ | (address constants) |
| $\mathbf{T}, \mathbf{F}$ | $\rightarrow$ | $\mathcal{B}$ | (boolean constants) |
| $\mathcal{A} \asymp \mathcal{A}$ | $\rightarrow$ | $\mathcal{B}$ | (address comparison) |
| $\neg \mathcal{B}$ | $\rightarrow$ | $\mathcal{B}$ | (boolean negation) |
| $\mathcal{B} \wedge \mathcal{B}$ | $\rightarrow$ | $\mathcal{B}$ | (boolean conjunction) |
| $\mathcal{B} \vee \mathcal{B}$ | $\rightarrow$ | $\mathcal{B}$ | (boolean disjunction) |
| $\mathcal{M}!$ | $\rightarrow$ | $\mathcal{V}$ | (merge selection) |
| $c_1, c_2, \ldots$ | $\rightarrow$ | $\mathcal{V}$ | (base value constants) |
| $?$ | $\rightarrow$ | $\mathcal{V}$ | (unknown base value) |

Figure 1: Signature of $\textsc{Pim}_t$ Terms.

$$
\begin{array}{rcll}
\emptyset_\rho \circ_\rho l & = & l & \text{(L1)} \\
l \circ_\rho \emptyset_\rho & = & l & \text{(L2)} \\
l_1 \circ_\rho (l_2 \circ_\rho l_3) & = & (l_1 \circ_\rho l_2) \circ_\rho l_3 & \text{(L3)} \\
\mathbf{T} \triangleright_\rho l & = & l & \text{(L5)} \\
\mathbf{F} \triangleright_\rho l & = & \emptyset_\rho & \text{(L6)} \\
\\
\{a_1 \mapsto m\} \mathbin{@} a_2 & = & (a_1 \asymp a_2) \triangleright_m m & \text{(S1)} \\
\{a \mapsto \emptyset_m\} & = & \emptyset_s & \text{(S2)} \\
\emptyset_s \mathbin{@} a & = & \emptyset_m & \text{(S3)} \\
(s_1 \circ_s s_2) \mathbin{@} a & = & (s_1 \mathbin{@} a) \circ_m (s_2 \mathbin{@} a) & \text{(S4)} \\
\\
(\alpha_i \asymp \alpha_i) & = & \mathbf{T} \quad (i \geq 1) & \text{(A1)} \\
(\alpha_i \asymp \alpha_j) & = & \mathbf{F} \quad (i \neq j) & \text{(A2)} \\
\\
m \circ_m [v] & = & [v] & \text{(M2')} \\
[v]! & = & v & \text{(M3)} \\
\emptyset_m! & = & ? & \text{(M4)} \\
\\
\neg\mathbf{T} & = & \mathbf{F} & \text{(B1)} \\
\neg\mathbf{F} & = & \mathbf{T} & \text{(B2)} \\
\mathbf{T} \wedge p & = & p & \text{(B3)} \\
\mathbf{F} \wedge p & = & \mathbf{F} & \text{(B4)} \\
\mathbf{T} \vee p & = & \mathbf{T} & \text{(B5)} \\
\mathbf{F} \vee p & = & p & \text{(B6)} \\
\{a_1 \mapsto m_1\} \circ_s \{a_2 \mapsto m_2\} & = & (a_1 \asymp a_2) \triangleright_s \{a_1 \mapsto (m_1 \circ_m m_2)\} \circ_s & \\
 & & \neg(a_1 \asymp a_2) \triangleright_s (\{a_2 \mapsto m_2\} \circ_s \{a_1 \mapsto m_1\}) & \text{(S8)}
\end{array}
$$

Figure 2: Equations of $\text{PIM}_t^+$. The equations labeled (L$n$) are generic to merge or store structures, i.e., in each case '$\rho$' should be interpreted as one of either $s$ or $m$. Equations (A1) and (A2) are schemes for an infinite set of equations.

$$
\begin{array}{rcll}
p \rhd_\rho \emptyset_\rho &=& \emptyset_\rho & \text{(L4)}\\
p \rhd_\rho (l_1 \circ_\rho l_2) &=& (p \rhd_\rho l_1) \circ_\rho (p \rhd_\rho l_2) & \text{(L7)}\\
p_1 \rhd_\rho (p_2 \rhd_\rho l) &=& (p_1 \wedge p_2) \rhd_\rho l & \text{(L8)}\\
l \circ_\rho l_1 \circ_\rho l &=& l_1 \circ_\rho l & \text{(L9)}\\
(p \rhd_\rho l_1) \circ_{rho} (\neg p \rhd_\rho l_2) &=& (\neg p \rhd_\rho l_2) \circ_\rho (p \rhd_\rho l_1) & \text{(L10)}\\
(p_1 \rhd_\rho l) \circ_\rho (p_2 \rhd_\rho l) &=& (p_1 \vee p_2) \rhd_\rho l & \text{(L11)}\\
\\
(a \asymp a) &=& \mathbf{T} & \text{(A3)}\\
(a_1 \asymp a_2) &=& (a_2 \asymp a_1) & \text{(A4)}\\
(a_1 \asymp a_2) \wedge (a_1 \asymp a_3) &=& (a_1 \asymp a_2) \wedge (a_2 \asymp a_3) & \text{(A5)}\\
(a_1 \asymp a_2) \wedge \neg(a_1 \asymp a_3) &=& (a_1 \asymp a_2) \wedge \neg(a_2 \asymp a_3) & \text{(A6)}\\
\\
[m!] &=& [?] \circ_m m & \text{(M7)}\\
((p \rhd_m [?]) \circ_m m)! &=& m! & \text{(M8)}\\
\\
p \rhd_s \{a \mapsto m\} &=& \{a \mapsto (p \rhd_m m)\} & \text{(S5)}\\
(a_1 \asymp a_2) \rhd_s \{a_1 \mapsto m\} &=& (a_1 \asymp a_2) \rhd_s \{a_2 \mapsto m\} & \text{(S9)}\\
(a_1 \asymp a_2) \rhd_m (s \mathbin{@} a_1) &=& (a_1 \asymp a_2) \rhd_m (s \mathbin{@} a_2) & \text{(S10)}\\
(p \rhd_s s) \mathbin{@} a &=& p \rhd_m (s \mathbin{@} a) & \text{(S11)}\\
\{a \mapsto m\} \circ_s s &=& s \circ_s \{a \mapsto m \circ_m (s \mathbin{@} a)\} & \text{(S12)}\\
\\
\neg\neg p &=& p & \text{(B7)}\\
(p_1 \wedge p_2) \wedge p_3 &=& p_1 \wedge (p_2 \wedge p_3) & \text{(B8)}\\
p_1 \wedge p_2 &=& p_2 \wedge p_1 & \text{(B9)}\\
p \wedge p &=& p & \text{(B10)}\\
p \wedge \neg p &=& \mathbf{F} & \text{(B11)}\\
(p_1 \vee p_2) \vee p_3 &=& p_1 \vee (p_2 \vee p_3) & \text{(B12)}\\
p_1 \vee p_2 &=& p_2 \vee p_1 & \text{(B13)}\\
p \vee p &=& p & \text{(B14)}\\
p \vee \neg p &=& \mathbf{T} & \text{(B15)}\\
p_1 \wedge (p_2 \vee p_3) &=& (p_1 \wedge p_2) \vee (p_1 \wedge p_3) & \text{(B16)}\\
p_1 \vee (p_2 \wedge p_3) &=& (p_1 \vee p_2) \wedge (p_1 \vee p_3) & \text{(B17)}\\
\neg(p_1 \wedge p_2) &=& \neg p_1 \vee \neg p_2 & \text{(B18)}\\
\neg(p_1 \vee p_2) &=& \neg p_1 \wedge \neg p_2 & \text{(B19)}
\end{array}
$$

Figure 3: Additional Equations of $\textsc{Pim}_t^=$.

The most problematic conjecture for applying implicit induction is (L9) for $\rho = s$:

$$(s \circ_s s_1) \circ_s s = s_1 \circ_s s \tag{L9s}$$

In order to expose the problems of applying implicit induction techniques for proving (L9s) we present a characteristic part of the regular mathematical proof of (L9s).

**Example 3.1** We consider the case when $s = \{a \mapsto [v]\}$, $s_1 = s' \circ_s \{a_1 \mapsto [v_1]\}$:

$$(\{a \mapsto [v]\} \circ_s (s' \circ_s \{a_1 \mapsto [v_1]\})) \circ_s \{a \mapsto [v]\} \tag{1}$$

$$=$$

$$(s' \circ_s \{a_1 \mapsto [v_1]\}) \circ_s \{a \mapsto [v]\} \tag{2}$$

For (1)=(2), there are two further cases: $(a_1 \asymp a) = \mathbf{T}$, and $(a_1 \asymp a) = \mathbf{F}$.

**Case I.** $(a_1 \asymp a) = \mathbf{F}$.

$$
\begin{array}{ll}
\text{LHS :} & (1) \\
& =_{(\text{L3})} \\
& (\{a \mapsto [v]\} \circ_s s') \circ_s (\{a_1 \mapsto [v_1]\} \circ_s \{a \mapsto [v]\}) \\
& =_{(\text{S8,L6,L1,B2,L5})} \\
& (\{a \mapsto [v]\} \circ_s s') \circ_s (\{a \mapsto [v]\} \circ_s \{a_1 \mapsto [v_1]\}) \\
& =_{(\text{L3})} \\
& ((\{a \mapsto [v]\} \circ_s s') \circ_s \{a \mapsto [v]\}) \circ_s \{a_1 \mapsto [v_1]\} \\
& =_{(\text{L9s})} \\
& (s' \circ_s \{a \mapsto [v]\}) \circ_s \{a_1 \mapsto [v_1]\} \qquad (3) \\
\text{RHS :} & (2) \\
& =_{(\text{L3})} \\
& s' \circ_s (\{a_1 \mapsto [v_1]\} \circ_s \{a \mapsto [v]\}) \\
& =_{(\text{S8,L6,L1,B2,L5})} \\
& s' \circ_s (\{a \mapsto [v]\} \circ_s \{a_1 \mapsto [v_1]\})
\end{array}
$$

**Case II.** $(a_1 \asymp a) = \mathbf{T}$. In this case, we also use the following lemma:

$$(b \asymp b_1) = \mathbf{T} \Rightarrow b = b_1 \tag{LE}$$

$$
\begin{array}{ll}
\text{LHS :} & (1) \\
& =_{(\text{L3})} \\
& (\{a \mapsto [v]\} \circ_s s') \circ_s (\{a_1 \mapsto [v_1]\} \circ_s \{a \mapsto [v]\}) \\
& =_{(\text{S8,L5,B1,L6,L2})}
\end{array}
$$

$$(\{a \mapsto [v]\} \circ_s s') \circ_s \{a_1 \mapsto [v_1] \circ_m [v]\} \quad (4)$$

$$=_{(\mathrm{M2}')}$$

$$(\{a \mapsto [v]\} \circ_s s') \circ_s \{a_1 \mapsto [v]\}$$

$$=_{(\mathrm{LE})}$$

$$(\{a \mapsto [v]\} \circ_s s') \circ_s \{a \mapsto [v]\}$$

$$=_{(\mathrm{L9s})}$$

$$s' \circ_s \{a \mapsto [v]\} \quad (5)$$

RHS : $$(2)$$

$$=_{(\mathrm{L3})}$$

$$s' \circ_s (\{a_1 \mapsto [v_1]\} \circ_s \{a \mapsto [v]\})$$

$$=_{(\mathrm{S8,L5,B1,L6,L2})}$$

$$s' \circ_s \{a_1 \mapsto [v_1] \circ_m [v]\}$$

$$=_{(\mathrm{M2}')}$$

$$s' \circ_s \{a_1 \mapsto [v]\} \quad (6)$$

$$=_{(\mathrm{LE})}$$

$$s' \circ_s \{a \mapsto [v]\}$$

The application of the induction hypothesis (L9s) in the conventional mathematical proof above can be justified by considering induction on variable $s_1$ of (L9s).

□

The main problem with the implicit induction approach of proving (L9s) is to find a way to consider the term transformations above like a simplification process which allows for applying the inductive hypothesis like a universally quantified formula. The conventional simplification techniques of implicit induction for equational specifications [15, 6, 7, 2, 4] are based on term rewriting. However, neither of these techniques can be applied in the considered case for the following reasons:

(A) both orientations of (L3) are used in the term transformations above;

(B) although (S8) is used in an oriented way, its orientation does not comply with any well-founded term ordering;

(C) the replacement of $a$ with $a_1$ by the application of (LE) cannot be defined solely by matching the left-hand side $b$ of (LE); this replacement is non-orientable as well.

Problem (A) is easy to overcome by considering rewriting modulo associativity. Problem (B) requires more elaboration. (S8) is equivalent to the following two axioms:[1]

$$(a_1 \asymp a_2) = \mathbf{T} \Rightarrow \quad \{a_1 \mapsto m_1\} \circ_s \{a_2 \mapsto m_2\} \quad = \quad \{a_1 \mapsto (m_1 \circ_m m_2)\} \qquad (\mathrm{S6})$$
$$(a_1 \asymp a_2) = \mathbf{F} \Rightarrow \quad \{a_1 \mapsto m_1\} \circ_s \{a_2 \mapsto m_2\} \quad = \quad \{a_2 \mapsto m_2\} \circ_s \{a_1 \mapsto m_1\} \qquad (\mathrm{S7})$$

[1]This equivalence is a logical consequence of $(\mathrm{PIM}_t^+ \backslash \{(\mathrm{S8})\}) \cup \{p = \mathbf{T} \vee p = \mathbf{F}\}$.

(S6) and (S7) can replace (S8) in cases II and I resp. (S6) does not cause any orientation problems. (S7) is a kind of commutativity that is suggestive to consider like another modulo part of a rewrite relation. However, the conditional form of (S7) complicates the problem. We also need to provide an extended matching mechanism in order to use (LE), — problem (C). We present the solutions of problems (A)–(C) in section 5.3.

The other problems of applying implicit induction for the verification of $\textsc{Pim}_t^=$ are

(D) to generate induction cases like (1) = (2), and

(E) to generate further cases like I and II.

We solve problem (D) by using the algorithm for generating cover sets of terms for the theories with not-necessarily-free constructors [14] (see section 7). We address problem (E) by modifying the case-rewriting technique of [7] in order to provide the case generation by the combinations of conditional rewrite rules and conditional equivalences like (S6) and (S7) (see section 5.4).

Last but not least, we have to

(F) provide a proper quasi-order for the orientation of conditional equations of $\textsc{Pim}_t$ as rewrite rules and equivalences.

We solve problem (F) by using the ACRPO ordering [13] in section 8.

The proof techniques presented in the paper were implemented on the basis of the automated induction procedure [20]. In the appendix, we present the proof trace for (L9s) generated by our proof procedure.

The proof techniques developed in this paper are rather technical. So we present them as an instantiation of the generic implicit induction procedure [19]. This allows us to localize the technicalities and simplify their justification; it also makes the overall presentation clearer.

# 4 A Generic Implicit Induction Procedure

Having evolved from the completion-based proof method [18], the implicit induction methods are being considered as instances of induction on syntactic orderings [21, 8, 19]. This framework allows for abstract, compact and clear description of concrete implicit induction techniques. The obvious benefits of the abstract description are easiness of understanding concrete proof techniques and modifying them to suit particular needs. The generic implicit induction procedure used in this paper is taken from [19].

## 4.1 Basic Notions

A formula $\phi$ is a *semantic consequence* of axioms $Ax$, denoted $Ax \models \phi$, if $\phi$ is valid in every model of $Ax$. A formula $\phi$ is an *inductive consequence* of axioms $Ax$, denoted $Ax \models_{ind} \phi$, if $Ax \models \phi\gamma$ for any ground substitution $\gamma$. $\models_{ind}$ is the conventional notion of inductive consequence corresponding to the validity of a conjecture in all the term generated models of the axioms.

A *quasi-order* is a reflexive transitive relation. A *(partial) order* is a transitive but irreflexive relation. An *equivalence* is a reflexive, transitive, and symmetric relation. Given a quasi-order $\succeq$, the *strict part* of $\succeq$ is the (partial) order $\succ$ defined as follows: $a \succ b$ iff $a \succeq b$ and $b \not\succeq a$. The *equivalence part* of $\succeq$ is the equivalence $\sim$ defined as follows: $a \sim b$ iff $a \succeq b$ and $b \succeq a$. On the other hand, any partial order $\succ$ and equivalence $\sim$ such that $\succ \cap \sim = \emptyset$ define the quasi-order $\succeq \equiv \succ \cup \sim$. Given a quasi-order $\succeq$, we write $a \ltimes b$ if neither $a \succeq b$, nor $b \succeq a$. A quasi-order $\succeq$ is *well-founded* if there is no infinite strictly descending sequence $a_1 \succ a_2 \succ \ldots$ of elements. A relation $\gg$ is *stable* if $a \gg b$ implies $a\sigma \gg b\sigma$ for any substitution $\sigma$. A quasi-order $\succeq$ is *strongly stable* if $\succ$ and $\sim$ are stable.

We introduce the following concise notation. We consider possibly infinite conjunctions as implication premises. Given a possibly infinite set of formulas $\Phi$, we write $\wedge\Phi$ for the conjunction of the elements of $\Phi$. We write $Ax \models \bigwedge \Phi \Rightarrow \phi$ if there exists a finite subset $\Phi'$ of $\Phi$ such that $Ax \models \bigwedge \Phi' \Rightarrow \phi$. Given a binary relation on formulas $\ll$, a formula $\phi$, and a set of formulas $\Psi$, we write $\Psi_{\ll\phi}$ to denote the possibly infinite conjunction $\wedge\{\psi\sigma | \psi \in \Psi, \psi\sigma \ll \phi\}$.

Given axioms $Ax$ and substitutions $\sigma$, $\sigma'$, we write $Ax \models \sigma = \sigma'$ if $Ax \models x\sigma = x\sigma'$ for any variable $x$. Given a quasi-order on terms $\succeq_t$ and substitutions $\sigma$, $\sigma'$, we write $\sigma \succeq_t \sigma'$ if $x\sigma \succeq_t x\sigma'$ for any variable $x$.

### 4.2 Cover Substitutions

Many implicit induction procedures are instances of induction on propositional orderings [8]. The induction domain of such procedures consists of ground formulas, and induction orderings on the domain are generated by orderings on terms. The notion of cover set [21] reproduced below is used to obtain a finite representation of such induction domains. It forms the basis of the generic induction procedure used in the paper.

**Definition 4.1 (Cover substitutions [21] )** Let $Ax$ be axioms, and $\succeq_t$ a quasi-order on terms. A *cover set of substitutions* $CS(Ax, \succeq_t)$ is a set of substitutions $\{\sigma_i\}_i$ such that, for any ground substitution $\gamma$, there exists a substitution $\sigma_i$ and a ground substitution $\gamma'$ such that $\gamma \succeq_t \sigma_i\gamma'$ and $Ax \models \gamma = \sigma_i\gamma'$.

The necessary relation between ordering on terms and the generated induction ordering on formulas in the context of induction on propositional orderings is described by the following notion:

**Definition 4.2 (Ordering compatibility)** A quasi-order $\succeq$ on formulas is *compatible* with a quasi-order on terms $\succeq_t$ if $\sigma \succeq_t \sigma'$ implies $\phi\sigma \succeq \phi\sigma'$ for any formula $\phi$.

### 4.3 Inference System

Following the paradigm "Algorithms = Logic + Control" [16], the implicit induction procedures have been formalized by inference systems, e.g. [2, 21], specifying possible proof state transformations. The presented generic inference system $\mathcal{I}$, figure 4, is taken from [19]. Axioms $Ax$ and quasi-orders on formulas $\succeq$ and terms $\succeq_t$ are the parameters of $\mathcal{I}$. A proof state

**Generate**

$$\frac{(P \cup \{\phi\}, H)}{(P \cup P', H \cup \{\phi\})}$$

if, for any $\sigma \in CS(Ax, \succeq_t)$, $Ax \models (P \cup P' \cup H \cup \{\phi\})_{\prec \phi\sigma} \Rightarrow \phi\sigma$

**Simplify**

$$\frac{(P \cup \{\phi\}, H)}{(P \cup P', H)}$$

if $Ax \models (P \cup P' \cup H)_{\preceq \phi} \Rightarrow \phi$

Figure 4: Inference system $\mathcal{I}(Ax, \succeq, \succeq_t)$

of $\mathcal{I}(Ax, \succeq, \succeq_t)$ is a pair of sets of formulas $(P, H)$, where $P$ represents proof goals, and $H$ induction hypotheses.

The inference rules of $\mathcal{I}(Ax, \succeq, \succeq_t)$ are essentially descriptive. They specify the requirements on the proof state transformations to be satisfied by concrete proof procedures. The rule *Generate* specifies the generation of the induction cases of a proof goal, and their initial simplification. The rule *Simplify* specifies "regular" simplifications of proof goals which also may result in multiple subgoals, as well as in the goal elimination. The descriptive nature of $\mathcal{I}(Ax, \succeq, \succeq_t)$ facilitates the modular development of diverse simplification and cover set generation techniques, while providing us with a uniform framework for their justification.

A typical inference strategy of $\mathcal{I}(Ax, \succeq, \succeq_t)$ amounts to, *first*, the simplification of current goals by *Simplify* and, *second*, the generation of induction cases for the simplified goals by *Generate*. During the simplification, the goals either are eliminated, or form lemmas subjected to further induction cycles. The instances of the conjectures used in the simplification are determined "on the fly". The soundness of their usage is caused by the reduction of the goal complexity during the simplification. The proven conjectures are inductive theorems if all the goals are eventually eliminated.

The basic notion relating $\mathcal{I}(Ax, \succeq, \succeq_t)$ to proving inductive conjectures is that of successful derivation. We write $(P, H) \vdash (P', H')$ to denote that a proof state $(P', H')$ is obtained from a proof state $(P, H)$ by an application of an inference rule. Given a set of formulas $P$, a *successful derivation* is an inference sequence $(P, \emptyset) \vdash \ldots \vdash (\emptyset, H)$ for some $H$.

**Proposition 4.3 (Soundness of $\mathcal{I}(Ax, \succeq, \succeq_t)$ [19])** *Let $Ax$ be axioms, and $\succeq$ be a strongly stable well-founded quasi-order on formulas compatible with a quasi-order on terms $\succeq_t$. Given a set of propositions $P$, $Ax \models_{ind} P$ if there exists a successful derivation for $P$ by $\mathcal{I}(Ax, \succeq, \succeq_t)$.*

Instantiation of the generic induction procedure $\mathcal{I}(Ax, \succeq, \succeq_t)$ amounts to defining the following components:

- classes of axioms $Ax$ and goals $P$,
- induction orderings $\succeq$ and $\succeq_t$ on formulas and terms,

- a method of generating the cover sets w.r.t. $\succeq_t$, and
- simplification techniques corresponding to $\succeq$.

In the case of $\text{PIM}_t$, we will consider equational clauses as axioms and conjectures. We present the simplification techniques adequate to the verification of $\text{PIM}_t^=$ in the following section. In that section we also introduce an ordering on formulas $\succeq$ parameterized by an ordering on terms $\succeq_t$, and show the correctness of the presented simplification techniques w.r.t. $\succeq$. We introduce a concrete ordering on terms suitable for the verification of $\text{PIM}_t^=$ in section 8. We describe a proper method of generating cover substitutions in sections 7.

# 5 Simplification Techniques for $\text{PIM}_t$

As we pointed out in the introduction, in order to get an implicit induction procedure suitable for the verification of $\text{PIM}_t^=$, we intend to modify conventional simplification techniques. Within our generic framework, we are able to perform this modification in a very modular way. In [19], we introduced two kinds of concrete simplification techniques for equational clauses that cover a wide range of implicit induction procedures. Both kinds of techniques were based on the contextual term rewriting [4] as an advanced form of conditional rewriting. The techniques differed in the way the contextual rewriting was lifted to the level of clauses. These differences, in turn, were implied by the differences between the orderings on clauses $\succeq_c$ and $\succeq_{cw}$ used for the justification of these clause simplification techniques in the context of the generic induction procedure $\mathcal{I}(Ax, \succeq, \succeq_t)$. Finally, we got two concrete instantiations of the generic procedure, $\mathcal{J}(Ax, \succeq_t)$ and $\mathcal{K}(Ax, \succeq_t)$ for the instantiations of $\succeq$ in $\mathcal{I}$ with $\succeq_c$ and $\succeq_{cw}$ resp. (see figure 5).

We get an instantiation of $\mathcal{I}$ suitable for the verification of $\text{PIM}_t^=$ by modifying procedure $\mathcal{J}$ to a procedure Ł. The only substantial modification amounts to replacing the conventional rewrite relation with a more general rewriting modulo conditional equivalences that would allow us to solve problems (A)–(C). The other components defining Ł as an instance of $\mathcal{I}$, i.e. the ordering on clauses and clause rewriting as functions of term ordering, remain the same as for $\mathcal{J}$. As the generalized term rewriting inherits essential properties of the original rewrite relation, the justification of Ł as a instance of $\mathcal{I}$ remains essentially the same as the justification of $\mathcal{J}$.

## 5.1 Basic Notions

A *conditional equation* is a clause with a single positive atom. An expression $\wedge_i e_i \Rightarrow e$ denotes the clause $\vee_i \neg e_i \vee e$. As usual, we consider equations as multisets of terms, and clauses as multisets of atoms, i.e. we abstract from the order of terms in equations, and atoms in clauses. Given a clause $C$, $prem(C)$ denotes the set of the equations that form the negative atom of $C$. Given a quasi-order on terms $\succeq$, a conditional equation $\wedge_i a_i = b_i \Rightarrow a = b$ is a *conditional rewrite rule*, denoted $\wedge_i a_i = b_i \Rightarrow a \rightarrow b$, if

$$a \succ (b, a_1, b_1, \ldots, a_n, b_n).$$

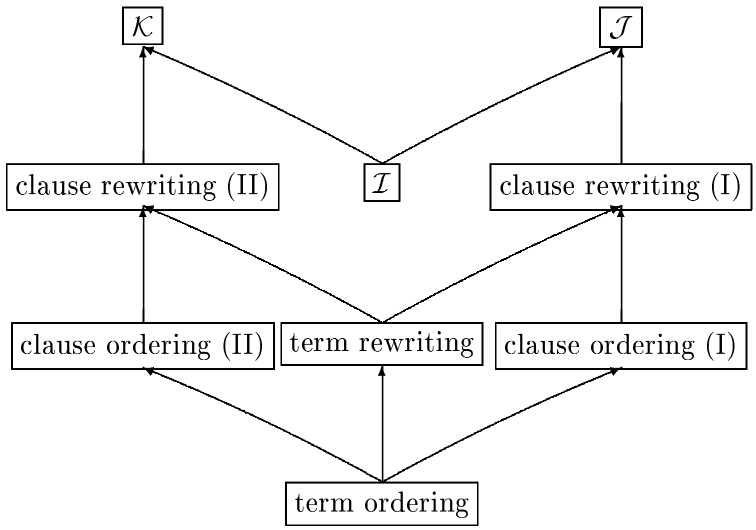


Figure 5: Modular Instantiations of $\mathcal{J}$

Given a binary relation $\gg$, $\overset{*}{\gg}$ denotes the reflexive and transitive closure of $\gg$, and $\overset{**}{\gg}$ denotes the reflexive, symmetric and transitive closure of $\gg$. Given an equivalence relation $\sim_t$ on terms, the *equivalence class* of a term $t$ is the set of terms $\{t' | t \sim_t t'\}$ denoted $[t]_{\sim_t}$. A relation on terms $\gg$ is *monotonic* if $s \gg t$ implies $f(\ldots s \ldots) \gg f(\ldots t \ldots)$. A *congruence* is a monotonic equivalence. Given a set of equations $E$, $\cong_E$ stands for the least congruence including $E$. A quasi-order on terms $\succeq_t$ is *strongly monotonic* if $\sim_t$ and $\succ_t$ are monotonic. A *reduction quasi-order* is a strongly stable and strongly monotonic well-founded quasi-order. We denote the *proper subterm relation* as $\triangleright$. A *reduction quasi-order* is a strongly stable and strongly monotonic well-founded quasi-order. A *decreasing* quasi-order is a reduction quasi-order $\succeq_t$ such that $t \succ_t s$ if $s$ is a proper subterm of $t$.

## 5.2 Simplification Relations On Terms

First, we modify the contextual rewriting [4] to address the problems (A)–(C) mentioned in the introduction.

### 5.2.1 Contextual Rewriting Modulo Conditional Equivalences

**Definition 5.1 (Conditional equivalence)** Let $\succeq_t$ be a quasi-order on terms. We call a conditional equation $\wedge_i(a_i = b_i) \Rightarrow a = b$ a *weak conditional equivalence* if $a \sim b$, and a *conditional equivalence*, denoted $\wedge_i(a_i = b_i) \Rightarrow a \sim b$ if $a \succ (a_1, b_1, \ldots, a_n, b_n)$ as well.

In section 8 we define a quasi-order which allows us to consider (L3s) and (S7) as conditional equivalences, and (LE) as a weak conditional equivalence.

The definition below is a modification of the contextual rewriting [4] for rewriting modulo conditional equivalences.

**Definition 5.2 (Contextual Rewriting Modulo Conditional Equivalences)** Let $\succeq_t$ be a quasi-order on terms, $R$ a sets of rewrite rules, $E$ a sets of weak conditional equivalences, and $\Delta$ a set of equations. Given terms $t, t'$, we write $\boxed{t \quad \rightarrow_{R/E,\Delta} \quad s}$ if

$t \quad \overset{**}{\leftrightharpoons}_{R/E,\Delta} \quad t'$ for some $t'$, and

**either** $t' \cong_\Delta s$ and $t' \succ_t s$

**or** there exists a rewrite rule $\rho \equiv \wedge_i(a_i = b_i) \Rightarrow a \rightarrow b$ in $R$ such that

1. $t'|_\omega = a\sigma$ and $s = t'[b\sigma]_\omega$ for some $\omega, \sigma$, and
2. for every $a_i, b_i$, there exist some $a'_i, b'_i$

$$a_i\sigma \quad \overset{*}{\rightarrow}_{R/E,\Delta} \quad a'_i \quad \cong_\Delta \quad b'_i \quad \overset{*}{\leftarrow}_{R/E,\Delta} \quad b_i\sigma,$$

where, for any terms $u, v$, $\boxed{u \quad \leftrightharpoons_{R/E,\Delta} \quad v}$ if

**either** $u \cong_\Delta v$ and $u \sim_t v$

**or** there exists a conditional equivalence $\wedge_i(c_i = d_i) \Rightarrow c \sim d$ in $E$ such that

1. $u \quad \cong_{\{c\sigma = d\sigma\}} \quad v$ for some $\sigma$, and
2. for every $c_i, d_i$, there exist some $c'_i, d'_i$,

$$c_i\sigma \quad \overset{*}{\rightarrow}_{R/E,\Delta} \quad c'_i \quad \cong_\Delta \quad d'_i \quad \overset{*}{\leftarrow}_{R/E,\Delta} \quad d_i\sigma.$$

**or** there exists a weak conditional equivalence $\wedge_i(c_i = d_i) \Rightarrow c = d$ in $E$ such that

1. $u \quad \cong_{\{c\sigma = d\sigma\}} \quad v$ for some $\sigma$, and
2. for any $c_i, d_i$, $c_i\sigma \quad \cong_\Delta \quad d_i\sigma$.

□

The distinction of occurrence $\omega$ in the definition of $\rightarrow_{R/E,\Delta}$ is necessary for its further refinement when lifting to the level of clauses (cf. section 5.3). We write $t \rightarrow^{\rho,\omega,\sigma}_{R/E,\Delta} s$ to distinguish the rewrite rule $\rho$ occurrence $\omega$ and substitution $\sigma$ in the definition above.

The following definition is necessary to describe the decidability aspect of the introduced rewrite relation.

**Definition 5.3 (Compact quasi-order)** We call a decreasing quasi-order $\succeq_t$ on terms *compact* if, for any term $t$, $[t]_{\sim_t}$ is finite.

**Proposition 5.1** *Let $\succeq_t$ be a decreasing quasi-order. Then*

*1.* $\rightarrow_{R/E,\Delta} \subseteq \succ_t$.

*2. If* $\succeq_t$ *is compact then* $\rightarrow_{R/E,\Delta}$ *is decidable.*

*Proof* The first property follows directly from the definition. The second property follows from the well-foundedness of $\succeq_t$, and the decidability of $\cong_\Delta$ [1], and $\overset{**}{\leftrightharpoons}$ for a compact quasi-order.

**Example 5.4** (Cf. example 3.1.)
Let $R = \{(\text{L9s})\}$, $E = \{(\text{L3}), (\text{S7})\}$, $\Delta = \{(a_1 \asymp a) = \mathbf{F}\}$.
Then (1) $\rightarrow_{R/E,\Delta}$ (3).

Let $R = \{(\text{S6}), (\text{M2}'), (\text{L9s})\}$, $E = \{(\text{L3}), (\text{LE})\}$, $\Delta = \{(a_1 \asymp a) = \mathbf{T}\}$.
Then (1) $\overset{*}{\rightarrow}_{R/E,\Delta}$ (5), (2) $\overset{*}{\rightarrow}_{R/E,\Delta}$ (6).

### 5.2.2 Identities Modulo Conditional Equivalences

We further introduce another elementary relation on terms, the identity modulo conditional equivalences used to determine some trivial logical consequences of the conditional equivalences.

**Definition 5.5 (Trivial equivalences)** Let $\succeq$ be a compact decreasing quasi-order, $E$ a set of weak conditional equivalences, and $\Delta$ a set of equations. Given terms $t, t'$, we write $\boxed{u \ \leftrightharpoons_{E,\Delta} \ v}$ for $u \ \overset{**}{\leftrightharpoons}_{\emptyset/E,\Delta} \ v$.

**Example 5.6** (Cf. example 3.1.)
Let $E = \{(\text{L3}), (\text{S7})\}$, $\Delta = \{(a_1 \asymp a) = \mathbf{F}\}$.
Then (3) $\leftrightharpoons_{E,\Delta}$ (2).

Let $E = \{(\text{LE})\}$, $\Delta = \{(a_1 \asymp a) = \mathbf{T}\}$.
Then (5) $\leftrightharpoons_{E,\Delta}$ (6).

## 5.3 Simplification relations on Clauses

### 5.3.1 Inductive Rewriting Modulo Conditional Equivalences

Having modified the rewrite relation on terms, we lift this modification to get the related simplification technique on the level of clauses to be justified in scope of the generic induction procedure. On this level, we have to distinguish the rewriting by axioms from the rewriting by induction hypotheses. This distinction amounts to splitting parameter $R$ of $\rightarrow_{R/E,\Delta}$ into two subsets, one for axioms, and another for hypotheses; we consider all the equivalences $E$ as axioms.

**Definition 5.7 (Inductive Rewriting Modulo Conditional Equivalences)** Let $P, P', P''$ be sets of clauses, and $C$ a clause. We write

$$C[t] \rightarrow_{P[P']/P''} C[s]$$

if $t \rightarrow^{\rho,\omega}_{R \cup R'/E, prem(C)} s$ for some subsets $R, R', E$ of $P, P', P''$ resp., and $\omega \neq \epsilon$ if $\rho \in R'$.

Note that the lifting of $t \rightarrow_{R/E,\Delta} s$ to the level of clauses defined above is analogous to that of the contextual rewrite relation [4] for procedure $\mathcal{J}$ in [19]. E.g., $\rightarrow_{P[P']/\emptyset}$ is a subset of the simplification relation on clauses employed in $\mathcal{J}$.[2]

**Example 5.8** (Cf. example 5.4.)
Let $R = \emptyset,\ R' = \{(\mathrm{L9s})\},\ E = \{(\mathrm{L3}), (\mathrm{S7})\}$.
Then

$$(a_1 \asymp a) = \mathbf{F} \Rightarrow (1) = (2) \quad \rightarrow_{R[R']/E} \quad (a_1 \asymp a) = \mathbf{F} \Rightarrow (3) = (2).$$

Let $R = \{(\mathrm{S6}), (\mathrm{M2}')\},\ R' = \{(\mathrm{L9s})\},\ E = \{(\mathrm{L3}), (\mathrm{LE})\}$.
Then

$$(a_1 \asymp a) = \mathbf{T} \Rightarrow (1) = (2) \quad \stackrel{*}{\rightarrow}_{R[R']/E} \quad (a_1 \asymp a) = \mathbf{T} \Rightarrow (5) = (6).$$

In order to put the simplification relations on clauses defined above in the context of the abstract induction procedure we use the induction ordering on clauses employed for the justification of $\mathcal{J}$:

**Definition 5.9 (Induction ordering $\succeq_c$ [19])** We write $(s = t)^+$ for the positive atom $s = t$, and $(s = t)^-$ for the negative atom $\neg s = t$. Given a quasi-order on terms $\succeq$, the complexity measure $\mu$ on atoms is defined by

$$\mu((s = t)^{\varepsilon}) = \{\{s, t\}\}$$

The relation on atoms $\succeq_a$ is defined by

$$A \succeq_a A' \text{ iff } \mu(A) \succeq^{mul} \mu(A').$$

The relation on clause witnesses $\succeq_c$ is defined as follows. For any clauses $C \equiv \vee_i A_i$, $C' \equiv \vee_j A'_j$ and substitutions $\sigma$, $\sigma'$,

$$\langle C, \sigma \rangle \succeq_c \langle C', \sigma' \rangle$$

if $\{\{A_i\sigma\}\}_i \succeq_a^{mul} \{\{A'_i\sigma'\}\}_j$, where $\{\{A'_i\sigma'\}\}_j$ stands for the multiset of elements $A_i$, and $\succeq_a^{mul}$ is the multiset ordering (on multisets of atoms) generated by $\succeq_a$.

The properties of $\succeq_c$ essential for implicit induction are formulated in the following proposition.

**Proposition 5.10 (Properties of $\succeq_c$ [19])** *If $\succeq_t$ is a decreasing quasi-order then $\succeq_c$ is a strongly stable well-founded quasi-order on clauses compatible with $\succeq_t$.*

The use of $\rightarrow_{P[P']/P''}$ as a simplification relation in the scope of the generic induction procedure is justified by the following proposition:

**Proposition 5.11** *Let $\succeq_t$ be a decreasing ordering, and $C \rightarrow_{P[P']/P''} C_1$. Then $P \cup P'' \models (\{C_1\} \cup P')_{\prec_c C} \Rightarrow C$.*

[2] We do not fully generalize the simplification relation of $\mathcal{J}$ here just to simplify the presentation.

*Proof* Let $C[t] \rightarrow_{P[P']/P''} C[s]$, and $t \rightarrow^{\rho,\omega,\sigma}_{R\cup R'/E,prem(C)} s$. Let $V$ be the set of instances of rewrite rules $\rho'\sigma'$ such that $\rho' \in P'$ and $\rho'$ is used in contextual rewriting of $t$ with matching substitution $\sigma'$. We show that

1. $P \models \wedge V \wedge C_1 \Rightarrow C$,
2. $\rho'\sigma' \prec_c C$ for any $\rho'\sigma' \in V$,
3. $C[s] \prec_c C[t]$.

1: This property follows directly from the definition of contextual rewriting.

2: By the definition of contextual rewriting for a decreasing quasi-order, for any $\rho'\sigma' \in V$ and any term $u$ occurring in $\rho'\sigma'$, $u \prec_t t$. Hence, $\rho'\sigma' \prec_c C$.

3: As $t \succ_t s$, $C[t] \succ_c C[s]$.

### 5.3.2 Tautologies Modulo Conditional Equivalences

We further lift the identity relation, Section 5.2.2, on the level of clauses.

**Definition 5.12** We say that $C$ is a $P$-*tautology* if $C \equiv (t = s \vee C')$, and $t \quad \leftrightharpoons_{E,prem(C)} \quad s$ for some subset $E$ of $P$.

**Example 5.13** (Cf. example 5.6.)
Let $E = \{(\mathrm{L3}), (\mathrm{S7})\}$.
Then $(a_1 \asymp a) = \mathbf{F} \Rightarrow (3) = (2)$ is an $E$-tautology.

Let $E = \{(\mathrm{L3}), (\mathrm{LE})\}$.
Then $(a_1 \asymp a) = \mathbf{T} \Rightarrow (5) = (6)$ is an $E$-tautology.

The use of $E$-tautologies in the scope of the generic induction procedure is justified by the following proposition:

**Proposition 5.14** *If $C$ is a $P$-tautology then $P \models C$.*

*Proof* Follows directly from the definition of $\leftrightharpoons_{E,prem(C)}$.

## 5.4 Inductive Case Rewriting Modulo Conditional Equivalences

In this section we address problem (E).

The definition below is a modification of the case rewriting [7] for rewriting modulo conditional equations.

**Definition 5.15 (Case Analysis)** Let $\succeq$ be a quasi-order on terms, $P, P', P''$ sets of clauses, and $C$ a clause. Consider those conditional rewrite rules in $P \cup P'$ whose left-hand sides match a term in $C$: let

$$\mathcal{E}_1 = \{E\sigma | E \Rightarrow a \to b \in P \cup P' \wedge C \equiv C[a\sigma]\}.$$

Consider those conditional equivalences in $P''$ whose sides match a term in $C$: let

$$\mathcal{E}_2 = \{E\sigma | E \Rightarrow a \sim b \in P'' \wedge (C \equiv C[a\sigma] \vee C \equiv C[b\sigma]\}.$$

Let $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2$ and, for any $E_\sigma \in \mathcal{E}$, $(E_\sigma \Rightarrow C) \to_{P[P']/P''} (E_\sigma \Rightarrow C'_\sigma)$ for some clause $C'_\sigma$. Then we write

$$C \quad \hookrightarrow_{P[P']/P"} \quad \{E_\sigma \Rightarrow C'_\sigma | E_\sigma \in \mathcal{E}\} \cup \{\vee_{E_\sigma \in \mathcal{E}} E_\sigma\}.$$

Note that the case patterns $\mathcal{E}$ are formed by both rewrite rules and conditional equivalences. Comparatively to using conditional equivalences in contextual rewriting, this is an even more non-standard feature of the presented simplification techniques.

Using the definition above, example 3.1 can be reconsidered as follows:

**Example 5.16**

Let

$$Ax_1 = \{(\mathrm{S6}),\ (\mathrm{M2}')\},\ H = \{(\mathrm{L9s})\},\ Ax_2 = \{(\mathrm{L3}),\ (\mathrm{S7}),\ (\mathrm{LE})\}.$$

Then

$$(1) = (2) \quad \hookrightarrow_{Ax_1[H]/Ax_2} \quad \begin{array}{lcl} \{(a_1 \asymp a) = \mathbf{F} & \Rightarrow & (3) = (2), \\ (a_1 \asymp a) = \mathbf{T} & \Rightarrow & (4) = (2), \qquad (*) \\ (a_1 \asymp a) = \mathbf{F} & \vee & (a_1 \asymp a) = \mathbf{T}\} \end{array},$$

and, further,

$$(*) \quad \xrightarrow{*}_{Ax_1[H]/Ax_2} \quad (a_1 \asymp a) = \mathbf{T} \Rightarrow (5) = (6)$$

Note the use of the induction hypothesis (L9s) in the case analysis of (1) = (2). After the initial transformation of (1) = (2) by the conditional equivalence (S7), the induction hypothesis (L9s) is applied via matching on a proper subterm of (1). It is an essential feature of the presented proof technique; this is the use of conditional equivalences for goal transformations that makes such applications of induction hypotheses possible.

The use of the case analysis as a simplification technique in the scope of the generic induction procedure is justified by the following proposition:

**Proposition 5.17** *Let $\succeq$ be a compact decreasing quasi-order, and $C \quad \hookrightarrow_{P[P']/P''} \quad P_1$. Then $P \cup P'' \models (P_1 \cup P')_{\prec_c C} \Rightarrow C$.*

*Proof* Consider the definition of $\hookrightarrow_{P[P']/P''}$. Let, for every $\sigma$, $(E_\sigma \Rightarrow C[t_\sigma]) \to_{P[P']/P''} (E_\sigma \Rightarrow C[s_\sigma])$ by $t_\sigma \to^{\rho_\sigma, \omega_\sigma, \theta_\sigma}_{R \cup R'/E, prem(C)} s_\sigma$. We show that

1. $P \cup P'' \models \wedge(P_1 \cup \{\rho_\sigma \theta_\sigma | \rho_\sigma \in P'\}) \Rightarrow C$,

2. $C_1 \prec_c C$ for any $C_1 \in P_1$,

3. If $\rho_\sigma \in P'$ then $\rho_\sigma\theta_\sigma \prec_c C$.

1: This property follows directly from the definition.

2,3: $\rho_\sigma = E_\sigma \Rightarrow (l \rightarrow r)_\sigma$. For any index $\sigma$, $t_\sigma \succ_t s_\sigma$, and $t_\sigma \succ_t t'$ for any $t' \in E_\sigma$. Also, $t_\sigma \succ_t l_\sigma$ if $\rho_\sigma \in P'$.

### 5.5 Clause Subsumption

A clause $C$ is *subsumed* by a clause $C'$ if there exists a substitution $\sigma$ such that, for every atom $a = b$ in $C'$, there exists a term $t$ such that $t[a\sigma] = t[b\sigma] \in C$ and, for any $\neg(a = b) \in C'$, $\neg(a\sigma = b\sigma) \in C$.

The simplification by subsumption is based on the subsumption of a clause by another clause.

**Definition 5.18 (Inductive subsumption)** Let $P$ and $P'$ be sets of clauses. Given a clause $C$, let $C'$ be a clause in $P \cup P'$ such that $C$ is subsumed by $C'$ with a matching substitution $\sigma$. Let also either $C' \in P$ or $C' \in P'$ and $C'\sigma \neq C$. Then we write $C \supset_{P[P']}$.

The simplification by subsumption facilitates the simplification by non-orientable clauses.

**Example 5.19** Consider the subgoal $(a_1 \asymp a) = \mathbf{F} \vee (a_1 \asymp a) = \mathbf{T}$ resulted from the case rewriting, example 5.16. Consider lemma

$$b = \mathbf{T} \vee b = \mathbf{F} \qquad \text{(OR)}$$

Then $(a_1 \asymp a) = \mathbf{F} \vee (a_1 \asymp a) = \mathbf{T}$ is subsumed by (OR).

**Proposition 5.20 (Properties of $\supset$)** *Let $\succeq_t$ be a decreasing quasi-order. Then $C \supset_{P[P']}$ implies $P \models P'_{\prec_c C} \Rightarrow C$.*

*Proof* Trivial.

## 6 An Implicit Induction Procedure

We have shown by examples that the simplification techniques introduced in the previous section are adequate for automating proving properties of $\text{PIM}_t$. Propositions 5.11,5.14,5.17,5.20 allow us to combine these simplification techniques to get the induction procedure $\mathrm{Ł}(Ax, \succeq_t)$, figure 6, as an instance of $\mathcal{I}(Ax, \succeq_c, \succeq_t)$.

**Proposition 6.1 (Soundness of $\mathrm{Ł}(Ax, \succeq_t)$)** *Let $Ax$ be axioms, and $\succeq_t$ be a compact decreasing quasi-order. Given a set of propositions $P$, $Ax \models_{ind} P$ if there exists a successful derivation for $P$ by $\mathrm{Ł}(Ax, \succeq_t)$.*

*Proof* The proposition is a direct consequence of propositions 4.3,5.10,5.11,5.14,5.17,5.20.

**Generate**

$$\frac{(P \cup \{C\}, H)}{(P \cup \cup_{\sigma \in CS(Ax, \succeq_t)} P^\sigma, H \cup \{C\})}$$

if, for every $\sigma \in CS(Ax, \succeq_t)$,
either $C\sigma \rightarrow_{Ax[P\cup\{C\}\cup H]/Ax} C^\sigma$, and $P^\sigma = \{C^\sigma\}$
or $C\sigma$ is an $Ax$-tautology and $P^\sigma = \emptyset$
or $C\sigma \hookrightarrow_{Ax[P\cup\{C\}\cup H]/Ax} P^\sigma$
or $C\sigma \supset_{Ax[P\cup\{C\}\cup H]}$

**Simplify**

$$\frac{(P \cup \{C\}, H)}{(P \cup P', H)}$$

if either $C \rightarrow_{Ax[P\cup\{C\}\cup H]/Ax} C_1$ and $P' = \{C_1\}$,
or $C\sigma$ is an $Ax$-tautology and $P' = \emptyset$,
or $C \hookrightarrow_{Ax[P\cup\{C\}\cup H]/Ax} P'$
or $C \supset_{Ax[P\cup\{C\}\cup H]}$

Figure 6: Proof procedure Ł$(Ax, \succeq_t)$

## 7 Cover Substitutions for $\text{PIM}_t$

In this section we address problem (D). Conventional methods of generating cover substitutions are based on term rewriting techniques [21, 14, 7]. However, none of the existing methods handle the rewriting modulo equations adequate to $\text{PIM}_t$ and, thus, cannot be applied directly. Instead, we use an indirect but easy way of solving the cover set problem. In general, any cover set generated w.r.t. any subsets $Ax'$, $\succeq'$ of given axioms $Ax$ and ordering $\succeq$ is a cover set w.r.t. $Ax$, $\succeq$. Cover substitutions for $\text{PIM}_t$ can be generated on the basis of conventional term rewriting by $\text{PIM}_t^0 \cup (M2')$, the terminating unconditional part of $\text{PIM}_t^+$. Fortunately, this subset of $\text{PIM}_t^+$ is representative enough to provide a useful cover set.

**Definition 7.1 (Cover terms [21]** ) Let $R$ be a set of rewrite rules. A *cover set of terms* $CT(R)$ is a set of substitutions $\{t_i\}_i$ such that, for any ground term $g$, there exists a term $t_i$ and a ground substitution $\gamma'$ such that $g \rightarrow_R^* t_i\gamma'$.

**Proposition 7.2 ([21])** *Let a rewrite system $R$ be oriented w.r.t. a quasi-order $\succeq_t$. Given a set of variables $V$, the set of all possible substitutions of $V$ with $CT(R)$ is a cover set of substitutions w.r.t. $R, \succeq_t$.*

The *reducibility tree* technique of [14] is an adequate technique to get the cover set of terms for $\text{PIM}_t^0 \cup (M2')$ because it is applicable to the specifications over non-free constructors. We are not going into the details here, since the technique is quite complicated and we just follow it. This procedure generates the terms

$$\emptyset_s,\ \emptyset_m,\ [v],\ \{a \mapsto [v]\},\ s_1 \circ_s \{a \mapsto [v]\}$$

as the cover terms for the sorts $\mathcal{S}$ and $\mathcal{M}$ w.r.t. axioms $(\Sigma_r, Ax_r)$ and rewrite relation $\xrightarrow{*}_{Ax_r}$, where the conventional rewrite relation $\rightarrow_R$ can be defined as $\rightarrow^{\emptyset}_{R,[\emptyset]/\emptyset}$. In section 8 we define the decreasing quasi-order $\succeq_{\mathrm{PIM}_t}$ such that $\xrightarrow{*}_{Ax_r} \subset \succeq_{\mathrm{PIM}_t}$. Hence, we can use the cover terms above to form the cover formulas w.r.t. quasi-order on formulas $(\succeq_{\mathrm{PIM}_t})_c$.

The only obstacle for applying the algorithm of [14] to the rewrite system $\mathrm{PIM}^0_t \cup (M2')$ is that due to the schemes of equations (A1) and (A2), $\mathrm{PIM}^0_t \cup (M2')$ is an infinite rewrite system. The most natural way to handle this problem would be to consider $\mathrm{PIM}^0_t$ like a parameterized specification over the sort parameters $\mathcal{A}$, $\mathcal{V}$ and function parameter $\asymp$, and to consider the necessary axioms about $\asymp$ like $\alpha \asymp \alpha = \mathbf{T}$ as the parameter constraints (cf. [5]). However, we would not like to complicate our presentation with the parameterized specification techniques. Instead, we modify the $\mathcal{A}$-, $\mathcal{V}$- and $\asymp$-related parts of $\mathrm{PIM}_t$ by considering the addresses and values generated by natural numbers. The modification of the proof techniques developed in this paper for the parameterized specification setting is straightforward.

We, therefore, introduce sort $\mathcal{N}$ of natural numbers with the constructors $0, \quad \mathcal{N}+1 \rightarrow \mathcal{N}$. We replace the address constants $\alpha_1, \alpha_2, \ldots$ with the constructor $\alpha_{\mathcal{N}} \rightarrow \mathcal{A}$, and the base value constants $c_1, c_2, \ldots$ with the constructor $c_{\mathcal{N}} \rightarrow \mathcal{V}$. We replace equation schemes (A1) and (A2) with the equations

$$\begin{aligned}
\alpha_0 \asymp \alpha_0 &= \mathbf{T} \\
\alpha_0 \asymp \alpha_{n+1} &= \mathbf{F} \\
\alpha_{n+1} \asymp \alpha_0 &= \mathbf{F} \\
\alpha_{n+1} \asymp \alpha_{k+1} &= \alpha_n \asymp \alpha_k
\end{aligned}$$

# 8 Orienting $\mathrm{PIM}^+_t$

In this section we address problem (F). To apply the induction procedure presented in section 4.3 (cf. propositions 4.3) we need to find a suitable decreasing quasi-ordering to orient the axioms and conjectures during the proof (cf. examples 3.1, 5.4, 5.6, 5.8, 5.16). Unfortunately, the straightforward orientation of (S8) from left to right does not comply with any decreasing quasi-ordering.

We can obtain better orientation results if we replace (S8) with (S6), (S7). (S7) still cannot be oriented w.r.t. any decreasing quasi-ordering, and our intention is to find a decreasing quasi-ordering $\succeq$ such that

$$\{a_1 \mapsto m_1\} \circ_s \{a_2 \mapsto m_2\} \sim \{a_2 \mapsto m_2\} \circ_s \{a_1 \mapsto m_1\}.$$

A decreasing quasi-ordering commutative on $\circ_s$ is an obvious choice. By the way, such a quasi-ordering makes impossible the strict orientation of (L3) when $\rho = s$:

$$\begin{aligned}
s_1 \circ_s (s_2 \circ_s s_3) &\succ (s_1 \circ_s s_2) \circ_s s_3 \sim \\
s_3 \circ_s (s_2 \circ_s s_1) &\succ (s_3 \circ_s s_2) \circ_s s_1 \sim \\
s_1 \circ_s (s_2 \circ_s s_3). &
\end{aligned}$$

However, the equivalence of the associativity is adequate to the $\text{PIM}_t^+$ analysis, because both orientations of (L3s) are used for term replacement there (cf. section 3).

So an adequate quasi-ordering should be a decreasing quasi-ordering AC on operator $\circ_s$, so that $\succeq$ permits the strict orientation of the axioms of $\text{PIM}_t^+$ other than (L3s) and (S7).

First we tried the AC-extensions of RPOs [17, 22], but the conditions imposed on the underlying precedence did not hold for $\text{PIM}_t^+$. Finally, the *ACRPO* ordering proposed in [13] turned out to be a suitable one. The only restriction imposed on the precedence in ACRPO is that equivalent operators must have the same status. We present the relevant notions below.

## 8.1 Basic Notions

We consider syntactic quasi-ordering on terms which depend on precedences and statuses of functional symbols (operators). A *precedence* is a quasi-ordering on operators. Each operator is assigned a status. We will consider *multiset*, *left-to-right*, *right-to-left* and *AC* statuses.

Due to the complexity of the definition of ACRPO we do not reproduce it here. Moreover, this definition is by itself irrelevant to our presentation. We rather present the relevant notions and properties of ACRPO.

ACRPO extends the RPOS [12] by comparing the flattened forms of terms.

**Definition 8.1** Let $\mathcal{F}_{AC}$ be a set of AC-operators. The *flattened form* $\overline{t}$ of a term $t$ is defined as follows:

1. $\overline{x} = x$ if $x$ is a variable,
2. $\overline{f(t_1, \ldots, t_n)} = f(\overline{t_1}, \ldots, \overline{t_n})$ if $f \notin \mathcal{F}_{AC}$,
3. $\overline{f(t_1, \ldots, t_n)} = f(T_1 \cup \ldots \cup T_n)$ if $f \in \mathcal{F}_{AC}$, where
   (a) $T_i = \{\{s_1, \ldots, s_m\}\}$ if $\overline{t_i} = f(s_1, \ldots, s_m)$,
   (b) $T_i = \{\{\overline{t_i}\}\}$, otherwise.

The main relevant property of ACRPO is the following.

**Proposition 8.2 ([13])** *Let $\succeq_p$ be a precedence over a (finite) set of operators with statuses so that the $\sim_p$-equivalent operators have the same status. Then the ACRPO ordering $\succeq_{ac}$ [13] based on this precedence is a decreasing quasi-order which is also AC-compatible i.e., for any AC-operator $\_ * \_$, $(x * y) * z \sim_{ac} x * (y * z)$ and $x * y \sim y * x$, where $x, y, z$ are variables. Also, $\succeq_{ac}$ is compact.*

Although the definition of ACRPO is quite complex, it differs from the definition of RPOS only in the case when the terms with equivalent top AC-operators are compared. In orienting $\text{PIM}_t^+$ this situation occurs only when comparing terms in (L3s) and (S7). Therefore, the rewrite rules of $\text{PIM}_t^+$can be determined by the following proposition.

**Proposition 8.3** *Let the AC-operators of $t$ do not occur in $s$. Then $t \succ_{ac} s$ iff $\bar{t} \succ_{rpos} \bar{s}$, where $\succeq_{rpos}$ denotes the recursive path ordering [12] over the same precedence and the status modified so that the AC operators become multiset.*

*Proof* Follows directly from the definitions of $\succeq_{ac}$ [13] and $\succeq_{rpos}$ [12].

This proposition also turns out to be sufficient for the strict orientation of the conjectures in the $\textsc{Pim}_t$ proofs like (L9s).

The equivalence part of ACRPO is easy to calculate using the following notion on flattened terms:

**Definition 8.4** Let $\succeq_p$ be a precedence. Flattened terms $f(t_1, \ldots, t_n)$, $g(s_1, \ldots, s_m)$ are *ac-equivalent* if $f \sim_g g$, $m = n$, and either

1. $f, g$ have left-to-right or right-to-left status, and $t_i$ is ac-equivalent to $s_i$ for each $i$, or
2. $f, g \in \mathcal{F}_{AC}$ or $f, g$ have multiset status, and there is a permutation $p$ of $(1, \ldots, n)$ such that $t_i$ is ac-equivalent to $s_{p(i)}$ for each $i$.

The proposition below can be applied for determining the equivalences (L3s) and (S7).

**Proposition 8.5 ([13])** *For any terms $t, s$, $t \sim_{ac} s$ iff $\bar{t}$ is ac-equivalent to $\bar{s}$.*

## 8.2 Precedence and Statuses for $\textsc{Pim}_t$

For the orientation of $\textsc{Pim}_t^+$, we use the following statuses:

- $\circ_s$ has AC status,
- $\circ_m$ has right-to-left status (for orienting (L3m)).
- the other operators have multiset status.

Also, the precedence used is as follows:[3]

$$
\begin{array}{rcll}
\rhd_\rho & \succ_p & \emptyset_\rho & \text{(for (L6))}, \\
@ & \succ_p & \rhd_m & \text{(for (S1))}, \\
@ & \succ_p & \asymp & \text{(for (S1))}, \\
\{\mapsto\} & \succ_p & \emptyset_s & \text{(for (S2))}, \\
@ & \succ_p & \emptyset_m & \text{(for (S3))}, \\
@ & \succ_p & \circ_m & \text{(for (S4))}, \\
\asymp & \succ_p & \mathbf{T} & \text{(for (A1))}, \\
\asymp & \succ_p & \mathbf{F} & \text{(for (A2))}, \\
! & \succ_p & ? & \text{(for (M4))}, \\
\neg & \succ_p & \mathbf{F} & \text{(for (B1))}, \\
\neg & \succ_p & \mathbf{T} & \text{(for (B2))}, \\
\circ_s & \succ_p & \{\mapsto\} & \text{(for (S6))}, \\
\circ_s & \succ_p & \circ_m & \text{(for (S6))}, \\
\circ_s & \succ_p & \asymp & \text{(for (S7))}.
\end{array}
$$

[3] We need to consider (S7) when defining the precedence in order to orient (S7) as a conditional equivalence.

### 8.3 Orienting (LE)

Since the sides of the conclusion of (LE) are different variables, they are incomparable w.r.t. $\succeq_{ac}$ and we still cannot justify the use of (LE) as weak conditional equivalence. Since the orientation of $\mathrm{PIM}_t$ equations described above does not depend on the address components of the compared terms it is suggestive to consider all the address terms equivalent. In order to get this effect we consider the substitution $\sigma_\alpha$ of all address variables with an address term, say $\alpha_0$, and finally define the term ordering $\succeq_{\mathrm{PIM}_t}$ as follows:

$$t \succ_{\mathrm{PIM}_t} s \iff t\sigma_\alpha \succ_{ac} s\sigma_\alpha$$
$$t \sim_{\mathrm{PIM}_t} s \iff t\sigma_\alpha \sim_{ac} s\sigma_\alpha$$

It is easy to see that the equations $\mathrm{PIM}_t^+ \setminus \{S8\} \cup \{S6\}$ are oriented w.r.t. $\succ_{\mathrm{PIM}_t}$, (L3s) and (S7) are conditional equivalence w.r.t. $\sim_{\mathrm{PIM}_t}$, and (LE) is a weak conditional equivalence w.r.t. $\sim_{\mathrm{PIM}_t}$.

The use of $\succeq_{\mathrm{PIM}_t}$ is justified by the following proposition.

**Proposition 8.6** $\succeq_{\mathrm{PIM}_t}$ *is a decreasing quasi-order.*

*Proof* Trivial.

Although $\sim_{\mathrm{PIM}_t}$ is not compact, the rewrite relation $\rightarrow_{R/E,\Delta}$ remains semi-decidable anyway (cf. proposition 5.1). After all, the decidability issue is irrelevant by itself to the soundness aspects of $\mathrm{PIM}_t$ proofs.

## 9 An Auxiliary Case Analysis Technique

As we mentioned, the conjecture (L9s) considered in detail through the paper was the most problematic for applying implicit induction techniques. Other conjectures were also proven with the presented techniques. The only minor problem we further encountered was related to the proofs of (S9) and (S10). Their mathematical proof amounts to a trivial case analysis on $(a_1 \asymp a_2)$. However, the case analysis techniques presented in section 5.4 did not allow to distinguish these cases, as there were no proper conditional equations.

This kind of case analysis is supported by the definition below.

**Definition 9.1 (Case analysis over a finite domain)** Let $Ax$ be axioms, $\succeq_t$ a decreasing quasi-order, $\{c_i\}_i$ a finite set of constants of a sort $s$, and $t$ a term of sort $s$ such that

1. $t \succ_t c_i$ for any $1 \leq i \leq k$, and
2. $Ax \models \vee_i x = c_i$, where $x$ is a variable.

Let $C[t'[t]]$ be conjecture such that $t$ is a strict subterm of $t'$. let $P \equiv \{t = c_i \Rightarrow C[t'[c_i]]\}_i$. Then we write $C \multimap_{Ax} P$.

**Example 9.2**

$$(\mathrm{S9}) \multimap \left\{ \begin{array}{l} (a_1 \asymp a_2) = \mathbf{T} \Rightarrow \mathbf{T} \rhd_s \{a_1 \mapsto m\} = \mathbf{T} \rhd_s \{a_2 \mapsto m\} \\ (a_1 \asymp a_2) = \mathbf{F} \Rightarrow \mathbf{F} \rhd_s \{a_1 \mapsto m\} = \mathbf{F} \rhd_s \{a_2 \mapsto m\} \end{array} \right\}$$

$$(\mathrm{S10}) \multimap \left\{ \begin{array}{l} (a_1 \asymp a_2) = \mathbf{T} \Rightarrow \mathbf{T} \rhd_s (s \mathbin{@} a_1) = \mathbf{T} \rhd_s (s \mathbin{@} a_2) \\ (a_1 \asymp a_2) = \mathbf{F} \Rightarrow \mathbf{F} \rhd_s (s \mathbin{@} a_1) = \mathbf{F} \rhd_s (s \mathbin{@} a_2) \end{array} \right\}$$

**Proposition 9.3** *Let $Ax$ be axioms and $\succeq_t$ a decreasing quasi-order. Then $C \multimap_{Ax} P$ implies $Ax \models P_{\prec_c C} \Rightarrow C$.*

*Proof.* First, $Ax \models P \Rightarrow C$. Also, $C \succ_c C'$ for any $C' \in P$.

## 10 Conclusion

In this paper we proposed new simplification techniques which, in combination with the known orientation and cover set generation methods, enable highly automated proofs of $\mathrm{PIM}_t^=$ conjectures. The orientation of conditional equations as rewrite rules and equivalences is the most complicated part of the proofs that required user guidance. However, this orientation is done only once.

As we mentioned in the introduction, the range of possible applications of the presented proof techniques is not confined to the solution of the considered proof problem. Application of induction reasoning for the analysis of programs in the full PIM is essential because of the absence of the complete axiomatization for the operational semantics of a universal programming language. Therefore, the presented proof techniques may constitute a part of a theorem proving environment for the full PIM.

**Acknowledgements** We thank Jan Heering for remarks on the paper.

## References

[1] W. Ackermanm. *Solvable Cases of the Decision Problem.* North-Holland, 1968.

[2] L. Bachmair. Proof by consistency in equational theories. In *3rd IEEE symposium on Logic in Computer science*, pages 228–233, 1988.

[3] J. A. Bergstra, T. B. Dinesh, J. Field, and J. Heering. A complete transformational toolkit for compilers. In *Proceedings European Symposium on Programming (ESOP '96)*, volume 1058 of *LNCS*, pages 92–107. Springer-Verlag, 1996. Full version: Technical Report RC 20342, IBM T. J. Watson Research Center, Yorktown Heights, and Technical Report CS-R9601, Centrum voor Wiskunde en Informatica (CWI), Amsterdam.

[4] E. Bevers and J. Lewi. Proof by consistency in conditional equational theories. In *Conditional and Typed Rewriting Systems, 2nd International Workshop*, volume 516 of *LNCS*, pages 194–205, 1990.

[5] A. Bouhoula. Sufficient completeness and parameterized proofs by induction. In *4th Int. Conf. on Algebraic and Logic Programming*, volume 850 of *LNCS*, pages 23–40, 1994.

[6] A. Bouhoula, E. Kounalis, and M. Rusinowitch. Automated mathematical induction. *Journal of Logic and Computation*, 5(5):631–668, 1995.

[7] A. Bouhoula and M. Rusinowitch. Implicit induction in conditional theories. *Journal of Automated Reasoning*, 14(2):189–235, 1995.

[8] F. Bronsard, U. S. Reddy, and R. W. Hasker. Induction using term orderings. In *12th International Conference on Automated Deduction*, volume 814 of *LNCS*, pages 102–117, 1994.

[9] M. Davis, Y. Matijasevic, and J. Robinson. Hilbert's tenth problem: Positive aspects of a negative solution. In F. E. Browder, editor, *Mathematical Developments Arising from Hilbert Problems*, AMS, pages 323–378. Providence, RI, 1976.

[10] A. v. Deursen, J. Heering, and P. Klint, editors. *Language Prototyping: An Algebraic Specification Approach*, volume 5 of *AMAST Series in Computing.* World Scientific Publishing Co., 1996.

[11] J. Field. A simple rewriting semantics for realistic imperative programs and its application to program analysis. In *Proceedings of the ACM SIGPLAN Workshop on Partial Evaluation and Sematics-Based Program Manipulation*, pages 98–107, 1992. Technical report: YALEU/DCS/RR-909.

[12] S. Kamin and J.-J. Levy. Attempts for generalizing the recursive path ordering. Unpublished Manuscript, 1980.

[13] D. Kapur, G. Sivakumar, and H. Zhang. A new method for proving termination of AC-rewrite systems. In *10th Conf. on Foundations of Software Technology and Theoretical Computer Science*, volume 472 of *LNCS*, pages 133–148, 1990.

[14] E. Kounalis. Testing for the ground (co-)reducibility property in term-rewriting systems. In *15th Colloquium on trees in algebra and programming*, volume 431 of *LNCS*, pages 221–238, 1990. See also J. Theoretical Computer Science, 106, 1992.

[15] E. Kounalis and M. Rusinowitch. Mechanizing inductive reasoning. In *Conf. of the American Association for Artificial Intelligence*, pages 240–245, 1990.

[16] R. A. Kowalski. Algorithm = logic + control. *Communications of the ACM*, 22(7):424–436, 1979.

[17] N. D. L. Bachmair. Commutation, transformation and termination. In *8th International Conference on Automated Deduction*, volume 230 of *LNCS*, pages 52–60, 1986.

[18] D. R. Musser. On proving inductive properties of abstract data types. In *7th ACM Symp. on Principles of Programming Languages*, pages 154–162, 1980.

[19] D. Naidich. On generic representation of implicit induction procedures. Techn. Report CS-R9620, 1996.

[20] D. Naidich and T. B. Dinesh. Specifying an automated induction proof procedure in ASF+SDF. In *ASF+SDF'95: Workshop on Generating Tools from Algebraic Specifications*, Techn. Rep. P9504, pages 233–254. University of Amsterdam, Programming Research Group, 1995.

[21] U. S. Reddy. Term rewriting induction. In *10th International Conference on Automated Deduction*, volume 449 of *LNCS*, pages 162–177, 1990.

[22] J. Steinbach. *Termination of Rewriting.* PhD thesis, Universität Kaiserslautern, 1994.

[23] C. Walther. Mathematical induction. In *Handbook on Logic in Artificial Intelligence and Logic Programming*, volume 2, pages 127–228. Clarendon Press, Oxford, 1994.

# A Proof Trace for (L9s)

The proof procedure Ł was specified in ASF+SDF (URL: `http://www.cwi.nl/~gipe/`) algebraic specification formalism. An input term for this executable specification consists of 8 parts:

- `sorts` introduces sort identifiers.
- `functions` introduces functional symbols in prefix notation.
- `variables` introduces variables used in axioms and clauses.
- `axioms` and `conjectures` introduces clauses, so that, given sequences of equations `E` and `E'`, `E => E'` denotes the clause $\wedge E \Rightarrow \vee E'$. The proof procedure proves the conjectures using the axioms.
- `ordering` determines the ordering procedure. Currently, the procedures for the recursive path ordering with status, and the sufficient conditions for ACRPO (propositions 8.2, 8.5) are implemented.
- `constructors` indicates, whether the constructors are free. If so, the test set generation procedure [6] is used to generate cover terms. The `cover terms` should be provided manually, otherwise.

The relation of the input signature to the $\mathrm{PIM}_t$ signature is given by the comments. The axioms are self-explanatory. To facilitate the proof, we consider easy lemmas (A0) and (OR) as extra axioms.

### Input $\mathrm{PIM}_t$ data

```
sorts
        [b, %%(booleans)
         v, %%(base values)
         a, %%(addresses)
         m, %%(merge structures)
         s  %%(store structures)
        ]
functions
        [
        tt   : []    -> b  {constructor}, %%(boolean)
        ff   : []    -> b  {constructor},

        0a   : []    -> a  {constructor}, %%(address)
        sa   : [a]   -> a  {constructor},

        0v   : []    -> v  {constructor}, %%(value)
        sv   : [a]   -> v  {constructor},

        0m   : []    -> m  {constructor}, %%(null merge)
        "[]" : [v]   -> m  {constructor}, %%(merge cell)

        0s   : []    -> s  {constructor}, %%(null store)
        "{}" : [a,m] -> s  {constructor}, %%(store cell)
```

```
        os   : [s,s] -> s  {constructor}, %%(store comp}

        om   : [m,m] -> m,    %%(merge composition}
        "="  : [a,a] -> b     %%(address comparison)
        ]
variables
        [ m_n: m, v_n: v, s_n: s, a_n: a, b_n: b]
axioms
        [
        L1m: => om( 0m(), m_1) = m_1,

        L2m: => om( m_1, 0m()) = m_1,

        L3m: => om( m_1, om( m_2, m_3)) =
                om( om( m_1, m_2), m_3),


        "M2'": => om( m_1, "[]"( v_1)) = "[]"( v_1),


        L1s: => os( 0s(), s_1) = s_1,

        L2s: => os(  s_1, 0s()) = s_1,

        L3s: => os( s_1, os( s_2, s_3)) =
                os( os( s_1, s_2), s_3),


        S2: => os( a_1, 0m()) = 0s(),

        S6: "="(a_1, a_2) = tt() =>
              os( "{}"( a_1, m_1), "{}"( a_2, m_2)) =
              "{}"( a_1, om( m_1, m_2)),

        S7: "="(a_1, a_2) = ff() =>
              os( "{}"( a_1, m_1), "{}"( a_2, m_2)) =
              os( "{}"( a_2, m_2), "{}"( a_1, m_1)),


        A1: => "="( 0a(), 0a()) = tt(),

        A2: => "="( 0a(), sa( a_1)) = ff(),

        A3: => "="( sa( a_1), 0a()) = ff(),

        A4: => "="( sa( a_1), sa( a_2)) =
               "="( a_1, a_2),
```

```
        %%Lemmas

        LE: "="( a_1, a_2) = tt() => a_1 = a_2,

        A0: => "="( a_1, a_1) = tt(),

        OR: => b_1 = tt(), b_1 = ff()

        ]
conjectures
        [ %%to be proven
        L9s: => os( os( s_1, s_2), s_1) = os( s_2, s_1)
        ]
precedence
        ["{}" > 0s, "=" > tt, "=" > ff,
         os > "{}", os > om, os > "="]
status
        [ %%see section 8.1
         tt : MULTISET, ff : MULTISET,
         0a : MULTISET, sa : MULTISET,
         0v : MULTISET, sv : MULTISET,
         0m : MULTISET, "[]" : MULTISET,
         0s : MULTISET, "{}" : MULTISET,
         "=" : MULTISET, os: AC,
         om: [1, 2]  %%left-to-right
        ]
ordering
        pim
free constructors?
        false
cover terms
        [0s(), "{}"(a_1,"[]"(v_1)),
         os( s_1, "{}"(a_1,"[]"(v_1))),
         0m(), "[]"(v_1)]
```

**Proof Trace** The proof trace below describes a successful derivation by L for the input proof state $(\{\text{L9s}\}, \emptyset)$. The keywords appearing in the trace are as follows:

- `"NORMALIZED"` indicates the normal form of a proof goal w.r.t. the application of rule *Simplify*.
- `"COVER CASES"` indicates the cover cases for a normalized goal produced by rule *Generate*.
- `"CASE"` indicates a case produced by the simplification of a goal by case rewriting.
- `"REWRITTEN BY"` indicates that a goal is simplified by rewriting.
- `"SUBSUMED BY"` indicates that a goal is simplified by subsumption.
- `"DELETED"` indicates that a current goal is simplified by deletion.

Note that the new goal numbers are assigned only when simplification of a goal results in multiple subgoals; the number of a modified goal remains the same, otherwise.

```
"NORMALIZED" L9s :  => os ( os ( s _ 1,
                                 s _ 2 ),
                          s _ 1 ) = os ( s _ 2,
                                         s _ 1 )
"COVER CASES"
[
L9s 1 :  => os ( os ( 0s (  ),
                          0s (  ) ),
                   0s (  ) ) = os ( 0s (  ),
                                    0s (  ) ),
L9s 2   :      =>
os ( os ( 0s (  ),
              "{}" ( a _ 1,
                      "[]" ( v _ 1 ) ) ),
       0s (  ) ) =
os ( "{}" ( a _ 1,
               "[]" ( v _ 1 ) ),
       0s (  ) ),
L9s 3   :      =>
os ( os ( 0s (  ),
              os ( s _ 4,
                     "{}" ( a _ 1,
                             "[]" ( v _ 1 ) ) ) ),
       0s (  ) ) =
os ( os ( s _ 4,
              "{}" ( a _ 1,
                      "[]" ( v _ 1 ) ) ),
       0s (  ) ),
L9s 4   :      =>
os ( os ( "{}" ( a _ 1,
                      "[]" ( v _ 1 ) ),
              0s (  ) ),
       "{}" ( a _ 1,
               "[]" ( v _ 1 ) ) ) = os ( 0s (  ),
                                            "{}" ( a _ 1,
                                                    "[]" ( v _ 1 ) ) ),
L9s 5   :      =>
os ( os ( "{}" ( a _ 2,
                      "[]" ( v _ 2 ) ),
              "{}" ( a _ 1,
                      "[]" ( v _ 1 ) ) ),
       "{}" ( a _ 2,
               "[]" ( v _ 2 ) ) ) =
os ( "{}" ( a _ 1,
               "[]" ( v _ 1 ) ),
       "{}" ( a _ 2,
               "[]" ( v _ 2 ) ) ),
L9s 6   :      =>
```

```
os ( os ( "{}" ( a _ 2,
                  "[]" ( v _ 2 ) ),
          os ( s _ 4,
                "{}" ( a _ 1,
                        "[]" ( v _ 1 ) ) ) ),
       "{}" ( a _ 2,
               "[]" ( v _ 2 ) ) ) =
os ( os ( s _ 4,
          "{}" ( a _ 1,
                  "[]" ( v _ 1 ) ) ),
       "{}" ( a _ 2,
               "[]" ( v _ 2 ) ) ),
L9s 7   :      =>
os ( os ( os ( s _ 4,
                "{}" ( a _ 1,
                        "[]" ( v _ 1 ) ) ),
          0s (  ) ),
       os ( s _ 4,
          "{}" ( a _ 1,
                  "[]" ( v _ 1 ) ) ) ) =
os ( 0s (  ),
       os ( s _ 4,
          "{}" ( a _ 1,
                  "[]" ( v _ 1 ) ) ) ),
L9s 8   :      =>
os ( os ( os ( s _ 4,
                "{}" ( a _ 2,
                        "[]" ( v _ 2 ) ) ),
          "{}" ( a _ 1,
                  "[]" ( v _ 1 ) ) ),
       os ( s _ 4,
          "{}" ( a _ 2,
                  "[]" ( v _ 2 ) ) ) ) =
os ( "{}" ( a _ 1,
               "[]" ( v _ 1 ) ),
       os ( s _ 4,
          "{}" ( a _ 2,
                  "[]" ( v _ 2 ) ) ) ),
L9s 9   :      =>
os ( os ( os ( s _ 3,
                "{}" ( a _ 2,
                        "[]" ( v _ 2 ) ) ),
          os ( s _ 4,
                "{}" ( a _ 1,
                        "[]" ( v _ 1 ) ) ) ),
       os ( s _ 3,
          "{}" ( a _ 2,
                  "[]" ( v _ 2 ) ) ) ) =
```

```
os ( os ( s _ 4,
              "{}" ( a _ 1,
                      "[]" ( v _ 1 ) ) ),
       os ( s _ 3,
              "{}" ( a _ 2,
                      "[]" ( v _ 2 ) ) ) ) ] ";"
 L9s 1 "SUBSUMED BY" L1s ";" L9s 2 "SUBSUMED BY" L1s ";"
L9s 3 "SUBSUMED BY" L1s ";"
L9s 4 "REWRITTEN" L2s ";"
"CASE"
L9s 5 1   ;   "=" ( a _ 2,
                    a _ 1 ) = tt (  )   =>
os ( os ( "{}" ( a _ 2,
                      "[]" ( v _ 2 ) ),
              "{}" ( a _ 1,
                      "[]" ( v _ 1 ) ) ),
       "{}" ( a _ 2,
               "[]" ( v _ 2 ) ) ) =
os ( "{}" ( a _ 1,
               "[]" ( v _ 1 ) ),
       "{}" ( a _ 2,
               "[]" ( v _ 2 ) ) ) ";" L9s 5 1 "REWRITTEN BY" S6 ";"
"CASE"
L9s 5 2   ;   "=" ( a _ 1,
                    a _ 2 ) = tt (  )   =>
os ( os ( "{}" ( a _ 2,
                      "[]" ( v _ 2 ) ),
              "{}" ( a _ 1,
                      "[]" ( v _ 1 ) ) ),
       "{}" ( a _ 2,
               "[]" ( v _ 2 ) ) ) =
os ( "{}" ( a _ 1,
               "[]" ( v _ 1 ) ),
       "{}" ( a _ 2,
               "[]" ( v _ 2 ) ) ) ";" L9s 5 2 "REWRITTEN BY" S6 ";"
"CASE"
L9s 5 3   ;   "=" ( a _ 1,
                    a _ 2 ) = ff (  )   =>
os ( os ( "{}" ( a _ 2,
                      "[]" ( v _ 2 ) ),
              "{}" ( a _ 1,
                      "[]" ( v _ 1 ) ) ),
       "{}" ( a _ 2,
               "[]" ( v _ 2 ) ) ) =
os ( "{}" ( a _ 1,
               "[]" ( v _ 1 ) ),
       "{}" ( a _ 2,
               "[]" ( v _ 2 ) ) ) ";" L9s 5 3 "REWRITTEN BY" L9s ";"
```

```
"CASE"
L9s 5 4   ;   "=" ( a _ 2,
                    a _ 1 ) = ff (  )   =>
os ( os ( "{}" ( a _ 2,
                   "[]" ( v _ 2 ) ),
              "{}" ( a _ 1,
                   "[]" ( v _ 1 ) ) ),
       "{}" ( a _ 2,
               "[]" ( v _ 2 ) ) ) =
os ( "{}" ( a _ 1,
               "[]" ( v _ 1 ) ),
       "{}" ( a _ 2,
               "[]" ( v _ 2 ) ) ) ";" L9s 5 4 "REWRITTEN BY" L9s ";"
"DISJUNCTION" L9s 5 5 ;  => "=" ( a _ 2,
                                  a _ 1 ) = tt (  ),
                            "=" ( a _ 1,
                                  a _ 2 ) = tt (  ),
                            "=" ( a _ 1,
                                  a _ 2 ) = ff (  ),
                            "=" ( a _ 2,
                                  a _ 1 ) = ff (  )
"CASE"
L9s 6 1   ;   "=" ( a _ 1,
                    a _ 2 ) = tt (  )   =>
os ( os ( "{}" ( a _ 2,
                   "[]" ( v _ 2 ) ),
              os ( s _ 4,
                     "{}" ( a _ 1,
                             "[]" ( v _ 1 ) ) ) ),
       "{}" ( a _ 2,
               "[]" ( v _ 2 ) ) ) =
os ( os ( s _ 4,
              "{}" ( a _ 1,
                   "[]" ( v _ 1 ) ) ),
       "{}" ( a _ 2,
               "[]" ( v _ 2 ) ) ) ";"
L9s 6 1 "REWRITTEN BY" L9s ";"
"CASE"
L9s 6 2   ;   "=" ( a _ 1,
                    a _ 2 ) = ff (  )   =>
os ( os ( "{}" ( a _ 2,
                   "[]" ( v _ 2 ) ),
              os ( s _ 4,
                     "{}" ( a _ 1,
                             "[]" ( v _ 1 ) ) ) ),
       "{}" ( a _ 2,
               "[]" ( v _ 2 ) ) ) =
os ( os ( s _ 4,
```

```
              "{}" ( a _ 1,
                      "[]" ( v _ 1 ) ) ),
       "{}" ( a _ 2,
               "[]" ( v _ 2 ) ) ) ";"
L9s 6 2 "REWRITTEN BY" L9s ";"
"CASE"
L9s 6 3   ;   "=" ( a _ 2,
                    a _ 1 ) = ff (  )   =>
os ( os ( "{}" ( a _ 2,
                      "[]" ( v _ 2 ) ),
              os ( s _ 4,
                     "{}" ( a _ 1,
                             "[]" ( v _ 1 ) ) ) ),
       "{}" ( a _ 2,
               "[]" ( v _ 2 ) ) ) =
os ( os ( s _ 4,
              "{}" ( a _ 1,
                      "[]" ( v _ 1 ) ) ),
       "{}" ( a _ 2,
               "[]" ( v _ 2 ) ) ) ";"
L9s 6 3 "REWRITTEN BY" L9s ";"
"DISJUNCTION" L9s 6 4 ;  => "=" ( a _ 1,
                                       a _ 2 ) = tt (  ),
                            "=" ( a _ 1,
                                       a _ 2 ) = ff (  ),
                            "=" ( a _ 2,
                                       a _ 1 ) = ff (  )
L9s 7 "REWRITTEN" L2s ";"
L9s 8 "REWRITTEN" L3s,L3s,L9s ";"
L9s 9 "REWRITTEN" L3s,L3s,L9s ";"
L9s 4 "REWRITTEN" S6 ";"
L9s 4 "REWRITTEN" "M2'" ";"
L9s 4 "SUBSUMED BY" L1s ";"
L9s 5 1 "SUBSUMED BY" "M2'" ";"
L9s 5 2 "SUBSUMED BY" "M2'" ";"
L9s 5 3 "DELETED" ";"
L9s 5 4 "DELETED" ";"
L9s 5 5 "SUBSUMED BY" OR ";"
L9s 6 1 "DELETED" ";"
L9s 6 2 "DELETED" ";"
L9s 6 3 "DELETED" ";"
L9s 6 4 "SUBSUMED BY" OR ";"
L9s 7 "REWRITTEN" L3s,L9s ";"
L9s 7 "REWRITTEN" L1s ";"
L9s 7 "SUBSUMED BY" L9s ";"
L9s 8 "SUBSUMED BY" L9s ";" L9s 9 "SUBSUMED BY" L9s ";" "Q.E.D."
```