On Terminating Lemma Speculations

Christoph Walther and Thomas Kolbe

Fachbereich Informatik, Technische Hochschule Darmstadt,
Alexanderstr. 10, D-64283 Darmstadt, Germany
E-mail: chr.walther@informatik.th-darmstadt.de

The improvement of theorem provers by reusing previously computed proofs is investigated. A method for reusing proofs is formulated as an instance of the problem reduction paradigm such that lemmata are speculated as proof obligations, being subject for subsequent reuse attempts. We motivate and develop a termination requirement, prove its soundness, and show that the reusability of proofs is not spoiled by the termination requirement imposed on the reuse procedure. Additional evidence for the general usefulness of the proposed termination order is given for lemma speculation in induction theorem proving.

1. INTRODUCTION

We investigate the improvement of theorem provers by reusing previously computed proofs, cf. [KW94, KW95b, KW96b] and Fig. 1. Our work has similarities with the methodologies of explanation-based learning [Ell89], analogical reasoning [Hal89], and abstraction [GW92], cf. [KW95b] for a more detailed comparison.

Consider the following general architecture: Some problem solver PS is augmented with a facility for storing and retrieving solutions of problems solved during the system's lifetime. The problem solver can be either some machine, a machine supported interactively by a human advisor, or a human only. One can think of several benefits by providing some memory for making a problem solver cognizant of previous work:

(1) the quality of the solution process is improved (i.e. less resources are required as compared to problem solving from scratch);
(2) the performance of the problem solver is improved (i.e., more problems are solvable as compared to problem solving from scratch);
(3) the quality of solutions is improved (e.g., a better plan, if PS is a planner).

1 This work was supported under Grants Wa652/4-1,2,3 by the Deutsche Forschungsgemeinschaft as part of the focus program “Deduktion.” A preliminary version of this work was presented at CADE-13 [KW96c].
The presence and the degree of these benefits strongly depend on the quality of the problem solver and the domain it is operating on, cf. [KW96a]. Here we consider a domain where problems are conjectures to be proved. We have developed and implemented the Plagiator system [Bra94, KB97] which proves theorems by mathematical induction\(^2\) in the spirit of the problem reduction paradigm [Nil71]: If a conjecture is submitted to the system, it tries to find a proof by inspecting its memory (called a proof dictionary) for reusing proofs of previously verified conjectures. If successful, the retrieval results in a set of conjectures, the truth of which is sufficient for the truth of the given conjecture. Then for each of these retrieved conjectures, the proof dictionary is searched again for reusable proofs and so on, until eventually a retrieved conjecture is either obviously true or the retrieval fails. In the latter case, a human advisor is called for providing a hand crafted proof for such a conjecture, which subsequently—after some (automated) preparation steps—is stored in the proof dictionary to be in stock for future reasoning problems.

In this way the system shall exhibit an intelligent behavior, although it is unable to find an original proof on its own, thus motivating the system’s name, viz. the German word for plagiarist. Our approach has two benefits, as revealed by several experiments with the Plagiator system [KW95d]: (1) Human labor is saved, because the number of required user interactions is decreased. (2) The performance of the overall system is improved, because the system is able to speculate lemmata, which are helpful to prove a given conjecture. The latter feature is particularly important, because it is retained if the human advisor is substituted by a machine, i.e. an automated induction theorem prover, cf. [BKR92, BM79, HS96, IB96, KZ88, Wal94]: Many domains, such as induction theorem proving or planning, do not have complete problem solvers, i.e., problem solvers which solve each solvable problem. Then the speculation of useful subgoals yields a relevant improvement of the system’s problem solving performance.

Here we formulate our method for reusing proofs as an instance of the problem reduction paradigm and then develop a termination requirement for proof reuse.

\(^2\) Throughout this paper *induction* stands for *mathematical induction* and should not be confused with induction in the sense of machine learning.
We prove the soundness of our proposal and show that reusability of proofs is not spoiled by the termination requirement imposed on the reuse procedure. We also give evidence for the general usefulness of our termination requirement for lemma speculation in induction theorem proving.

2. REUSING PROOFS—AN EXAMPLE

Let us briefly sketch our method for reusing proofs (see [KW94] for more details): An induction formula $IH \rightarrow IC$ is either a step formula or a base formula in which case $IH$ equals true. Induction formulas are proved by modifying the induction conclusion $IC$ using given axioms until the induction hypothesis $IH$ is applicable.

For instance, let the functions plus, sum, and app be defined by the following equations where $0$ and $s(x)$ (resp. empty and add$(n, x)$) are the constructors of the sort number (resp. list):³

\[
\begin{align*}
    (\text{plus-1,2}) & \quad \text{plus}(0, y) \equiv y \\
    (\text{sum-1,2}) & \quad \text{sum}($\text{empty}$) \equiv 0 \\
    (\text{app-1,2}) & \quad \text{app}($\text{empty}$, y) \equiv y
\end{align*}
\]

These defining equations form a theory which may be extended by lemmata, i.e., statements which were (inductively) inferred from the defining equations and other already proved statements. For instance

\[
(\text{lem-1}) \quad \text{plus}(\text{plus}(x, y), z) \equiv \text{plus}(x, \text{plus}(y, z))
\]

can be easily proved and therefore may be used like any defining equation in subsequent deductions. We aim to optimize proving such conjectures as (lem-1) by reusing previously computed proofs of other conjectures. For instance consider the statement

\[
\varphi[x, y] := \text{plus}($\text{sum}(x)$, $\text{sum}(y)$) \equiv \text{sum}($\text{app}(x, y)$).
\]

We prove the conjecture $\forall x, y \ \varphi[x, y]$ by induction upon the list-variable $x$ and obtain two induction formulas, viz. the base formula $\varphi_b$ and the step formula $\varphi_s$ as

\[
\begin{align*}
    & \varphi_b := \forall y \ \varphi[$\text{empty}$, y] \\
    & \varphi_s := \forall n, x, y (\forall u \ \varphi[x, u]) \rightarrow \varphi[$\text{add}(n, x)$, $y$].
\end{align*}
\]

The following proof of the step formula $\varphi_s$ is obtained by modifying the induction conclusion $\varphi[$\text{add}(n, x)$, $y$] = $ plus($\text{sum}($\text{add}(n, x)$)$, $\text{sum}(y)$) $\equiv \text{sum}($\text{app}($\text{add}(n, x)$, $y$)$) IC$

³ We usually omit universal quantifiers at the top level of formulas as well as the sort information for variables.
in a backward chaining style, i.e., each statement is implied by the statement in the line below, where terms are underlined if they have been changed in the corresponding proof step.\(^4\)

\[
\begin{align*}
\text{plus}(\text{sum}(\text{add}(n, x)), \text{sum}(y)) &= \text{sum}(\text{app}(\text{add}(n, x), y)) & \text{IC} \\
\text{plus}(\text{plus}(n, \text{sum}(x)), \text{sum}(y)) &= \text{sum}(\text{app}(\text{add}(n, x), y)) & (\text{sum-2}) \\
\text{plus}(\text{plus}(n, \text{sum}(x)), \text{sum}(y)) &= \text{sum}(\text{add}(n, \text{app}(x, y))) & (\text{app-2}) \\
\text{plus}(\text{plus}(n, \text{sum}(x)), \text{sum}(y)) &= \text{plus}(n, \text{sum}(\text{app}(x, y))) & (\text{sum-2}) \\
\text{plus}(\text{plus}(n, \text{sum}(x)), \text{sum}(y)) &= \text{plus}(n, \text{plus}(\text{sum}(x), \text{sum}(y))) & \text{IH} \\
\text{plus}(n, \text{plus}(\text{sum}(x), \text{sum}(y))) &= \text{plus}(n, \text{plus}(\text{sum}(x), \text{sum}(y))) & (\text{lem-1}) \\
\text{true} & x \equiv x
\end{align*}
\]

Given such a proof, it is analyzed to distinguish its relevant features from its irrelevant parts. Relevant features are specific to the proof and are collected in a proof catch because similar requirements must be satisfied if this proof is to be reused later on. We consider features like the positions where equations are applied, induction conclusions and hypotheses, and general laws such as \(x \equiv x\), etc. as irrelevant because they can always be satisfied. So the catch of a proof is a subset of the set of leaves of the corresponding proof tree.

Analysis of the above proof yields \((\text{sum-2}), (\text{app-2}), and (\text{lem-1})\) as the catch. E.g., all we have to know about plus for proving \(\varphi_i\), is its associativity, but not its semantics or how plus is computed. We then generalize\(^5\) the conjecture, the induction formula and the catch for obtaining a so-called proof shell. This is achieved by replacing function symbols by function variables denoted by capital letters \(F, G, H\), etc., yielding the schematic conjecture \(\Phi := F(G(x), G(y)) \equiv G(H(x, y))\) with the corresponding schematic induction formula \(\Phi_i\), as well as the schematic catch \(C_i\) (see Fig. 2).

If a new statement \(\psi\) shall be proved, a suitable induction axiom is selected by well-known automated methods, cf. [Wal94], from which a set of induction formulas \(I_\psi\) is computed for \(\psi\). Then for proving an induction formula \(\psi_i \in I_\psi\) by reuse, it is tested whether some proof shell \([MPS]\) applies for \(\psi_i\), i.e., whether \(\psi_i\) is a (second-order) instance of the schematic induction formula of \([MPS]\). If the test succeeds, the obtained (second-order) matcher is applied to the schematic catch of \([MPS]\), and if all formulas of the instantiated schematic catch can be proved (which may necessitate further proof reuses), \(\psi_i\) is verified by reuse since the truth of an instantiated schematic catch implies the truth of its instantiated schematic induction formula.

E.g., assume that the new conjecture \(\forall x, y \\psi[x, y]\) shall be proved, where

\[\psi[x, y] := \text{times}(\text{prod}(x), \text{prod}(y)) \equiv \text{prod}(\text{app}(x, y))\]

\(^4\) We omit a proof for the base formula \(\varphi_i\) as there are no particularities compared to the step case.

\(^5\) Not to be confused with generalization of a formula \(\varphi\) as a preprocessing for proving \(\varphi\) by induction.
\[
\Phi_s := \{ 
    \forall u \ F(G(x), G(u)) \equiv G(H(x, u)) \}
\]

\[
F(G(D(n, x)), G(y)) \equiv G(H(D(n, x), y))
\]

\[
C_s := \{ 
    (1) \quad G(D(n, x)) \equiv F(n, G(x)) \\
    (2) \quad H(D(n, x), y) \equiv D(n, H(x, y)) \\
    (3) \quad F(F(x, y), z) \equiv F(x, F(y, z))
\}
\]

**FIG. 2.** The proof shell [MPS], for the proof of \( \phi \), (simple analysis).

and \( \text{times} \) and \( \text{prod} \) are defined by the equations

\[
\text{times}(0, y) \equiv 0, \quad \text{times}(s(x), y) \equiv \text{plus}(y, \text{times}(x, y))
\]

\[
\text{prod}(\text{empty}) \equiv s(0), \quad \text{prod}(\text{add}(n, x)) \equiv \text{times}(n, \text{prod}(x)).
\]

The induction formulas computed for \( \psi \) are

\[
\psi_s := \forall y \ \psi[\emptyset, y] \\
\psi_s := \forall n, x, y \ \forall y \ \psi[x, u] \rightarrow \psi[\text{add}(n, x), y].
\]

Obviously \( \psi \) is an instance of \( \Phi \) and \( \psi_s \) is an instance of \( \Phi_s \) w.r.t. the matcher \( \pi := \{F/\text{times}, G/\text{prod}, H/\text{app}, D/\text{add}\} \). Hence (only considering the step case) we may reuse the given proof by instantiating the schematic catch \( C_s \) and subsequent verification of the resulting proof obligations:

\[
\pi(C_s) = \{ 
    (4) \quad \text{prod}(\text{add}(n, x)) \equiv \text{times}(n, \text{prod}(x)) \\
    (5) \quad \text{app}(\text{add}(n, x), y) \equiv \text{add}(n, \text{app}(x, y)) \\
    (6) \quad \text{times}(\text{times}(x, y), z) \equiv \text{times}(x, \text{times}(y, z))
\}
\]

Features (4) and (5) are axioms, viz. (prod-2) and (app-2), and therefore are obviously true. So it only remains to prove the associativity of \( \text{times} \) (6) and, if successful, \( \psi \) is proved. Compared to a direct proof of \( \psi_s \), we have saved the user interactions necessary to apply the right axioms in the right place (where the associativity of \( \text{times} \) must be verified in either case). Additionally, conjecture (6) has been speculated as a lemma which is required for proving conjecture \( \psi \).

3. THE PHASES OF THE REUSE PROCEDURE

Our approach for reusing proofs is organized into the steps illustrated in Fig. 3.

**Prove** [cf. Sections 1, 2]. If required, a direct proof \( p \) for (an induction formula) \( \phi \) from a set of axioms \( AX \) is given by the human advisor or an automated induction theorem prover. The set of axioms \( AX \) consists of defining equations, previously proved lemmata, and logical axioms such as \( x \equiv x \), and \( \phi \rightarrow \phi \).

**Analyze** [KW94]. The simple proof analysis which was illustrated in Section 2 analyzes a proof \( p \) of \( \phi \), yielding a proof catch \( c \). Formally, the catch \( c \) is a finite subset of nonlogical axioms of \( AX \) such that \( c \) logically implies \( \phi \). For increasing the applicability of proof shells and the reusability of proofs, we have developed the
refined proof analysis which also distinguishes different occurrences of function symbols in the conjecture and in the catch of a proof. For instance the (step formula of) statement $\psi_2 := \text{plus}(\text{len}(x), \text{len}(y)) = \text{len}(\text{app}(x, y))$ cannot be proved by reusing the proof shell from Fig. 2, because one formula of the instantiated catch does not hold, cf. [KW94]. However, the reuse succeeds if refined analysis is applied (see below).

**Generalize** [KW94]. Both $\phi$ and $c$ are generalized by replacing (different occurrences of) function symbols with (different) function variables. This yields a schematic conjecture $\Phi$ and a schematic catch $C$, where the latter is a set of schematic formulas which—if considered as a set of first-order hypotheses—logically implies the schematic conjecture $\Phi$. Such a pair $PS := \langle \Phi, C \rangle$ is called a proof shell and serves as the data structure for reusing the proof $p$. E.g., after the refined analysis of the proof of $\psi_s$ from Section 2, generalization yields $\Phi := F^1(\text{G}^1(x), \text{G}^2(y)) \equiv \text{G}^3(\text{H}^1(x, y))$ and the proof shell of Fig. 4. Here, e.g., the function variables $F^1, F^2, F^3$ correspond to different occurrences of the function symbol plus; e.g., the schematic equation (10) stems from generalizing (lem-1).

**Store** [KW95c]. Proofs shells $\langle \Phi, C_1 \rangle, \ldots, \langle \Phi, C_n \rangle$ (sharing a common schematic goal formula $\Phi$) are merged into a proof volume $PV := \langle \Phi, \{ C_1, \ldots, C_n \} \rangle$ which then is stored in the proof dictionary $PD$, i.e., a library of proof ideas organized as a set of proof volumes.

**Retrieve** [KW95c]. If a new conjecture $\psi$ is to be proved, the proof dictionary is searched for a proof volume $PV := \langle \Phi, \{ C_1, \ldots, C_n \} \rangle$ such that $\psi = \pi(\Phi)$ for

\[
\Phi_s := \{ \forall u \quad F^1(\text{G}^1(x), \text{G}^2(u)) \equiv \text{G}^3(\text{H}^1(x, u)) \} \rightarrow \\
F^4(\text{G}^1(\text{D}^1(n, x)), \text{G}^2(\text{y})) \equiv \text{G}^3(\text{H}^1(\text{D}^1(n, x), \text{y}))
\]

\[
C_s := \{ \begin{align*}
(7) & \quad H^1(\text{D}^1(n, x), \text{y}) \equiv \text{D}^1(n, H^1(x, y)) \\
(8) & \quad G^1(\text{D}^1(n, x)) \equiv F^2(n, G^1(x)) \\
(9) & \quad G^3(\text{D}^1(n, x)) \equiv F^3(n, G^3(x)) \\
(10) & \quad F^1(F^3(x, y), z) \equiv F^3(x, F^1(y, z))
\end{align*}
\}
\]

FIG. 4. The proof shell $PS_s$ for the proof of $\phi_s$ (refined analysis). Note that corresponding function variables in the induction hypothesis (resp. the induction conclusion) have been identified during the analysis phase.
some second-order matcher $\pi$. If successful, the schematic conjecture $\Phi$ and in turn also the proof volume $PV$ applies for $\psi$ (via the matcher $\pi$). Here some restrictions on the class of admissible matchers can be imposed to make the retrieval more efficient, cf. [KW95c]. E.g., $\pi_2 := \{ F^3/\text{plus}, G^{1,2,3}/\text{len}, H^4/\text{app}, D^{1,2}/\text{add} \}$ is obtained by matching $\Phi'_i$ from Fig. 4 with $\psi_2$ above. Then a catch $C_i$ is selected by heuristic support from the proof volume $PV$ and the partially instantiated catch $\pi(C_i)$ serves as a candidate for proving $\psi$ by reuse. For our example, the partially instantiated catch is obtained as

$$\pi_2(C_i) = \{ (11) \quad \text{len}(\text{add}(n, x)) = F^2(n, \text{len}(x)) \}
\{ (12) \quad \text{app}(\text{add}(n, x), y) = D^4(n, \text{app}(x, y)) \}
\{ (13) \quad \text{len}(D^4(n, x)) = F^3(n, \text{len}(x)) \}
\{ (14) \quad \text{plus}(F^3(x, y), z) = F^3(x, \text{plus}(y, z)) \}.$$  

Adapt [KW95d, KW95a]. Since a partially instantiated catch $\pi(C_i)$ may contain free function variables, i.e., function variables which occur in $C_i$ but not in $\Phi$, these function variables have to be instantiated by known functions. Free function variables such as $F^2$, $F^3$, and $D^4$ in $\pi_2(C_i)$ result from the refined analysis and provide an increased flexibility of the approach, because different instantiations correspond to different proofs. Hence a further second-order substitution $\rho$ is required for replacing these function variables so that the resulting proof obligations, i.e., all formulas in the totally instantiated catch $\rho(\pi(C))$, are provable from $AX$. Such a second-order substitution $\rho$ is called a solution (for the free function variables), and $\psi$ is proved by reuse because semantical entailment is invariant w.r.t. (second-order) instantiation. Solution candidates $\rho$ are computed by second-order matching modulo symbolical evaluation; cf. [KW95d]. For the example, the solution $\rho_2 := \{ F^2, G^3/\text{plus}, D^4/\text{add} \}$ is obtained which instantiates (11) to the axiom $\text{len}(\text{add}(n, x)) \equiv s(\text{len}(x))$.

Patch [KW95b]. Often one is not only interested in the provability of $\psi$, but also in a proof of $\psi$ which can be presented to a human or can be processed subsequently. In this case it is not sufficient just to instantiate the schematic proof $P$ of $\Phi$ (which is obtained by generalizing the proof $p$ of $\varphi$) with the computed substitution $\tau := \rho \cdot \pi$ because $\tau$ might destroy the structure of $P$. Therefore the instantiated proof $\tau(P)$ is patched (which always succeeds) by removing void (resp. inserting additional) inference steps for obtaining a proof $p'$ of $\psi$, cf. [KW95b].

Apart from initial proofs provided by the human advisor in the Prove step, none of these steps necessitates human support. Thus the proof shell from Fig. 4 can be automatically reused for proving the step formulas of the apparently different conjectures $\varphi_i$ given in Table 1 below. For the sake of readability we use mathematical (infix) symbols for functions where appropriate, i.e., $\times$, $+$, $\cdot$, $\prec$, $\{\}$, $\Sigma$, and $\Pi$ denote times, plus, minus, app, len, sum, and prod, respectively. We use the

\footnote{The instantiations of $F^3$ and $F^2$ are different here, viz. plus and $s$, and this is why reuse fails for the simple analysis, used cf. Fig. 2.}
TABLE 1
Conjectures Proved and Lemmata Speculated by Reuse of $\varphi_8$

<table>
<thead>
<tr>
<th>No.</th>
<th>Conjectures proved by reuse</th>
<th>Subgoals</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varphi_1$</td>
<td>$\prod[k \times l = 1] (k &lt; l)$</td>
<td>$\varphi_{27}$</td>
</tr>
<tr>
<td>$\varphi_2$</td>
<td>$</td>
<td>k &lt; l</td>
</tr>
<tr>
<td>$\varphi_3$</td>
<td>$</td>
<td>\text{rev}(k)</td>
</tr>
<tr>
<td>$\varphi_4$</td>
<td>$\text{rev}(\text{rev}(k)) = k$</td>
<td>$\varphi_{14}$</td>
</tr>
<tr>
<td>$\varphi_5$</td>
<td>$\text{rev}(l) &lt; \text{rev}(k) = \text{rev}(k &lt; l)$</td>
<td>$\sigma_5(\varphi_{11})$</td>
</tr>
<tr>
<td>$\varphi_6$</td>
<td>$\max(\text{max}(k), \text{max}(l)) = \max(k &lt; l)$</td>
<td>$\varphi_{28}$</td>
</tr>
<tr>
<td>$\varphi_7$</td>
<td>$\text{min}(\text{min}(n, k), \text{min}(n, l)) = \text{min}(n, k &lt; l)$</td>
<td>$\varphi_{29}$</td>
</tr>
<tr>
<td>$\varphi_8$</td>
<td>$\text{plus}(m, k) &lt; \text{plus}(m, l) = \text{plus}(m, k &lt; l)$</td>
<td>$\varphi_{30}$</td>
</tr>
<tr>
<td>$\varphi_9$</td>
<td>$</td>
<td>k</td>
</tr>
<tr>
<td>$\varphi_{10}$</td>
<td>$\text{nclut}(m, \text{nclut}(n, k)) = \text{nclut}(\text{nclut}(m, n), k)$</td>
<td>$\varphi_{32}$</td>
</tr>
<tr>
<td>$\varphi_{11}$</td>
<td>$k &lt; l (l &lt; p) = (k &lt; l &lt; p)$</td>
<td>$\varphi_{33}$</td>
</tr>
<tr>
<td>$\varphi_{12}$</td>
<td>$</td>
<td>k &lt; n : l</td>
</tr>
<tr>
<td>$\varphi_{13}$</td>
<td>$</td>
<td>k &lt; n : e</td>
</tr>
<tr>
<td>$\varphi_{14}$</td>
<td>$\text{rev}(k &lt; n : e) = n : \text{rev}(k)$</td>
<td>$\varphi_{36}$</td>
</tr>
<tr>
<td>$\varphi_{15}$</td>
<td>$(p^n)^m = \text{move}(n, m)$</td>
<td>$\varphi_{18}$</td>
</tr>
<tr>
<td>$\varphi_{16}$</td>
<td>$p^n \times p^m = \text{move}(n, m)$</td>
<td>$\varphi_{17}$</td>
</tr>
<tr>
<td>$\varphi_{17}$</td>
<td>$m \times (n \times l) = (m \times n) \times l$</td>
<td>$\varphi_{19}$</td>
</tr>
<tr>
<td>$\varphi_{18}$</td>
<td>$m \times i + n \times i = (m + n) \times i$</td>
<td>$\varphi_{20}$</td>
</tr>
<tr>
<td>$\varphi_{19}$</td>
<td>$m \times i + i \times i = i \times (m + n)$</td>
<td>$\varphi_{21}$</td>
</tr>
<tr>
<td>$\varphi_{20}$</td>
<td>$m \times n = n \times m$</td>
<td>$\varphi_{22}$</td>
</tr>
<tr>
<td>$\varphi_{21}$</td>
<td>$m \times (n \times m) = m \times n \times m$</td>
<td>$\varphi_{23}$</td>
</tr>
<tr>
<td>$\varphi_{22}$</td>
<td>$m + (i + n) = i + (m + n)$</td>
<td>$\varphi_{24}$</td>
</tr>
<tr>
<td>$\varphi_{23}$</td>
<td>$m + n = n + m$</td>
<td>$\varphi_{25}$</td>
</tr>
<tr>
<td>$\varphi_{24}$</td>
<td>$m + (n + i) = (m + n) + i$</td>
<td>$\varphi_{26}$</td>
</tr>
<tr>
<td>$\varphi_{25}$</td>
<td>$m + n = n + m$</td>
<td>$\varphi_{27}$</td>
</tr>
<tr>
<td>$\varphi_{26}$</td>
<td>$\text{or} (\text{mem}(m, k), \text{mem}(m, l)) = \text{mem}(m, k &lt; l)$</td>
<td>$\varphi_{28}$</td>
</tr>
<tr>
<td>$\varphi_{27}$</td>
<td>$r(m, k) &lt; &gt; r(m, l) = r(m, k &lt; l)$</td>
<td>$\varphi_{29}$</td>
</tr>
</tbody>
</table>

Note: Conjectures $\varphi_{28}, \ldots, \varphi_{41}$ cannot be proved by reusing the proof of $\varphi_0$. $\varphi_{28} := \max(m, \text{max}(m, n)) \leq \max(m, \text{max}(n, m)), \varphi_{29} := \min(m, \text{min}(n, l)) = \min(l \leq n, m), i \times \varphi_{30} := \text{or} (\text{eq}(m, n), a, b) \equiv \text{or} (\text{eq}(m, n), \text{or}(a, b)); \varphi_{31} := (\text{eq}(m, n), k, n < l) < > p \equiv (\text{eq}(m, n), k < l)$.
proved by reuse only), and “[…]” denotes that heuristics different from the heuristics given in [KW94, KW95b] are used.

E.g., statement $\varphi_{16}$ is speculated when verifying $\varphi_{15}$, which leads to speculating $\varphi_{17}$, which in turn entails speculation of conjecture $\varphi_{18}$, for which eventually $\varphi_{24}$ is speculated. For $\varphi_{11}$ an instance of conjecture $\varphi_{21}$ is speculated, viz. the formula $\sigma_{5}(\varphi_{11})$ with $\sigma_{5} = \{p/m \vdash \varepsilon\}$.

4. REUSING PROOFS AS PROBLEM REDUCTION

Our method for reusing proofs can be viewed as an instance of the problem reduction paradigm, where a problem $p$ is mapped to a finite set of subproblems $\{p_1, ..., p_n\}$ by some (problem-)reduction operators, and each of the subproblems $p_i$ is mapped to a finite set of subproblems in turn, etc.; cf. [Nil71] and Fig. 5a. The reduction process stops successfully if each subproblem eventually is reduced to a primitive problem $p'$ where primitiveness is a syntactical notion depending on the particular problem solving domain. The only requirement is that primitive problems are trivially solvable indeed and that a solution is obvious. Since it is demanded in addition that each reduction operator only yields a set $P$ of subproblems for a given problem $p$ such that the solvability of all subproblems in $P$ implies the solvability of $p$, successful termination of the reduction process entails the solvability of the original problem.

Problem solving within this paradigm creates a search space which is organized as an AND/OR-tree: Several reduction operators may be applicable for a problem, which creates an OR-branch in the search tree. On solving all subproblems obtained by the application of one reduction operator, an AND-branch is created, cf. Fig.5b.

However, problem reduction needs not stop successfully on a given problem; i.e., there may be problems which are infinitely reduced by the reduction operators such that at least one nonprimitive subproblem always remains. We therefore demand for each reduction step $p \mapsto \{p_1, ..., p_n\}$ that $p > p_i$ for all $i \in \{1, ..., n\}$, where $>$ is a well-founded relation on the set of problems, and it is obvious that problem reduction always terminates (either unsuccessfully with a set of some nonprimitive problems or successfully) if this requirement is satisfied. The well-founded relation $>$ also depends on the domain and (considered as a set) should be as large as possible w.r.t. $\subseteq$. Here we consider the termination of the reuse process:

(a) reduction $p \mapsto \{p_1, ..., p_n\}$  
(b) AND/OR search tree, solution tree

FIG. 5. The problem reduction paradigm.
When reusing proofs, problems are conjectures to be proved. The reduction operators are implicitly given with the proof volumes, where the selection of a particular catch (among all catches of a proof dictionary) corresponds to the selection of a particular reduction operator (among other applicable operators), cf. the retrieval step in Section 3. After the computation of a solution substitution in the adaption step, a finite set of simplified7 conjectures is obtained from the totally instantiated catch, which can be considered the result of applying a reduction operator to a conjecture, cf. Fig. 6. A conjecture is “primitive” in our framework iff it is an instance of an axiom, i.e., \( \varphi = \sigma(\psi) \) for some \( \psi \in AX \) and some first-order matcher \( \sigma \). A conjecture is irreducible iff it is primitive or no reduction operator is applicable; i.e., no proof volume \( PV \) applies for \( \varphi \). In the latter case, \( \varphi \) must be proved directly (by some human advisor or a machine), whereas in the first case \( \varphi \) is trivially “solvable.”

In order to prevent infinite reuse sequences, we demand \( \varphi > F \varphi \), for each conjecture \( \varphi \) and each reducible member \( \varphi_i \) of a simplified totally instantiated catch, where \( > F \) is a well-founded relation on formulas. Since proof reuse never is attempted for an irreducible conjecture, \( \varphi > F \varphi \), is not required for guaranteeing termination if \( \varphi_i \) is irreducible. Thus, e.g., proving \( \varphi_9 := |k| + |l| = |k < > l| \) by reuse terminates vacuously as all formulas from the totally instantiated catch \( \pi_1(\rho_2(C_1)) \) are instances of axioms, cf. Section 3. But when proving, e.g., \( \varphi_{15} \) by reuse, \( \varphi_{15} > F \varphi_{16} \) is required, cf. Table 1.

5. TERMINATION OF THE REUSE PROCEDURE

As an example of a never ending attempt for proving a statement by reuse, consider conjecture \( \psi := \text{plus}(x, s(x)) \equiv s(\text{plus}(x, x)) \). The proof volume \( PV' \) containing the proof shell \( PS' \) from Fig. 4 applies for \( \psi \) via the second-order matcher\(^8\) \( \pi = \{ F^1/\text{plus}(w_1, s(w_1)), G^1/w_1, G^2/s(w_1), H^1/\text{plus}(w_1, w_1), D^1/s(w_2) \} \). Using the

7 Simplified conjectures are obtained by symbolic evaluation [Wal94]. E.g. \( s(t_1) = s(t_2) \) is simplified to \( t_1 = t_2 \) and \( \text{plus}(s(t_1), t_2) \) is simplified to \( s(\text{plus}(t_1, t_2)) \).

8 Special argument variables \( w_1, w_2, \ldots \notin F \) denote formal parameters in functional terms replacing function variables in second-order matchers, e.g., \( D^1(n, x) = s(x) \).
solution $\rho = \{F^2(s(w_2)), D^4(s(s(w_2))))\}$ for the free function variables in the adaption step, the totally instantiated catch $\rho(\pi(C_2))$ is computed as

\begin{align*}
(15) & \quad s(x) \equiv s(x) \\
(16) & \quad \text{plus}(s(x), s(x)) \equiv s(\text{plus}(x, x)) \\
(17) & \quad s(s(s(x))) \equiv s(s(s(x))) \\
(18) & \quad \text{plus}(s(x), s(s(x))) \equiv s(\text{plus}(x, s(x)))
\end{align*}

Hence nonprimitive conjectures $\psi_1 := \text{plus}(x, s(x)) \equiv s(\text{plus}(x, x))$ and $\psi_2 := \text{plus}(x, s(s(x))) \equiv s(\text{plus}(x, s(x)))$ are obtained by simplification from (16) and (18). With $\psi = \psi_1$, the proof volume $PV''$ can be applied again giving rise to an infinite reuse sequence.\footnote{We cannot use orderings developed in the area of term rewriting systems such as recursive path orderings, cf., e.g., [Der87, DJ90], because the requirements of stability and monotonicity (which reduction orderings must satisfy) are too strong to be useful in our domain.} and using $\rho$ as the solution substitution then yields the nonprimitive conjectures $\psi_{n+1}$ and $\psi_{n+2}$ as proof obligations. Thus $\langle \psi_{n+1} \rangle_{n \in \mathbb{N}}$ is an infinite reuse sequence.

For preventing such infinite reuse sequences, we impose a termination requirement on the reuse procedure. Based on experiments with the Plagiator system, we develop a well-founded relation $\succ_f$ on the set of formulas: 9.

5.1. An Order on (Sets of) Symbols

We start by separating function symbols from the signature $\Sigma$ into the set $\Sigma^c$ of constructor function symbols, as $\emptyset$, $\mathfrak{s}$, empty, add, etc., and the set $\Sigma^d$ of defined function symbols, e.g., exp, prod, times, sum, plus, etc. Then the defined-by relation $\succ_{df}$ is a relation on $\Sigma^d$ defined by: 9.
Definition 1 (Defined-by relation >\text{def}).

\[ f >\text{def} g \ \text{iff} \ (1) \ g \text{ occurs in one of the defining equations for } f \text{ and } f \neq g \]

or \( (2) \ f >\text{def} h >\text{def} g \) for some \( h \in \Sigma^d \).

Obviously, >\text{def} is transitive and by the requirements for the introduction of function symbols which in particular exclude mutual recursion, >\text{def} is well-founded. We have, for instance, \( \text{exp} >\text{def} \text{ times} >\text{def} \text{ plus} \) and \( \text{prod} >\text{def} \text{ times} >\text{def} \text{ plus} \) as well as \( \text{sum} >\text{def} \text{ plus} \), cf. also Fig. 8.

We extend >\text{def} to a quasi-ordering \( \succeq \) on \( 2^\Sigma^d \):

\text{Definition 2 (Multiset order} >\text{def}, \text{ quasi-ordering} \succeq \text{). Let} >\text{def} \text{ be the strict multiset order imposed by} >\text{def} \text{ on the multisets of} \Sigma^d \text{. Then for finite sets} S_1, S_2 \subseteq \Sigma^d \text{, we define} S_1 \succeq S_2 \text{ iff one of the following cases apply:}^{10}

\begin{align*}
(1) & \quad S_1, S_2 \subseteq \Sigma^d \quad \text{and either } S_1 = S_2 \text{ or } S_1 >\text{def} S_2 \\
(2) & \quad S_1 \subseteq \Sigma^d, S_2 \subseteq \varphi^d \\
(3) & \quad S_1, S_2 \subseteq \varphi^d \quad \text{and } |S_1| \geq |S_2| \\
(4) & \quad S_1 \subseteq \varphi^d, S_2 \subseteq \Sigma^d \\
(5) & \quad S_1, S_2 \subseteq \varphi^d \\
(6) & \quad S_1 \succeq S'_1 \succeq S_2 \quad \text{for some } S'_1 \subseteq \Sigma^d \cup \varphi^d.
\end{align*}

The strict part > of \( \succeq \) is defined as > := \( \succeq \setminus \approx \), where \( \approx \) is the equivalence relation \( \simeq \cap \approx \) induced by \( \simeq \) on \( 2^\Sigma^d \).

>\text{def} is well-founded since >\text{def} is, cf. [DM79]. We find, e.g., \{ exp, prod, sum \} > \{ times, sum \} > \{ times, plus \} > \{ plus \} > \{ x, y \} \approx \{ u, v \} > \{ z \} > \{ add, s \}, and thus \{ exp, prod, sum \} > \{ add, s \}, cf. Fig. 8.

\( ^{10} \) As each finite set also is a multiset, finite sets can be compared by the multiset order.
LEMMA 3 (Equivalent sets of symbols). Let $S_1, S_2 \subseteq \Sigma \cup \tau^c$ be finite. Then $S_1 \approx S_2$ iff either $S_1, S_2 \subseteq \Sigma^d$ and $S_1 = S_2$, $S_1, S_2 \subseteq \tau^c$ and $|S_1| = |S_2|$, or $S_1, S_2 \subseteq \Sigma^e$.

Proof. The if part is trivial and we verify the only-if part: Let $S_1 \supseteq S_2 \supseteq S_1$. Then by definition of $\approx$, either $S_1, S_2 \subseteq \Sigma^d$, $S_1, S_2 \subseteq \tau^c$, or $S_1, S_2 \subseteq \Sigma^e$. If $S_1, S_2 \subseteq \Sigma^d$ then $S_1 = S_2$, because $S_1 \gg \defeq S_2 \gg \defeq S_1$ would contradict the well-foundedness of $\gg \defeq$ otherwise. If $S_1, S_2 \subseteq \tau^c$, then $|S_1| \geq |S_2| \geq |S_1|$, and therefore $|S_1| = |S_2|$. Otherwise $S_1, S_2 \subseteq \Sigma^e$.

THEOREM 4 (Well-founded order on sets of symbols). $\gg$ is well-founded on finite subsets of $\Sigma \cup \tau^c$.

Proof. Assume by way of contradiction that $\langle S_i \rangle_{i \in \mathbb{N}}$ is a sequence of finite sets with $S_i \gg S_{i+1}$. Then $S_i \not\in \Sigma^e$ for all $i \in \mathbb{N}$, because all finite subsets of $\Sigma^e$ are $\gg$-minimal, cf. Lemma 5.3. If $S_i \not\in \tau^c$ for some $i \in \mathbb{N}$, then $S_{i+1} \not\in \tau^c$ for all $j \in \mathbb{N}$ by definition of $\gg$ (and because of $S_i \not\in \Sigma^e$ for all $i \in \mathbb{N}$). Hence with Lemma 5.3, $|S_i| > |S_{i+1}| > |S_{i+2}| > \ldots$ contradicting the well-foundedness of $\langle \mathbb{N}, \gg \rangle$. Consequently $S_i \subseteq \Sigma^d$ for all $i \in \mathbb{N}$ and $S_i \gg \defeq S_{i+1} \gg \defeq S_{i+2} \gg \defeq \ldots$ contradicting the well-foundedness of $\gg \defeq$. Thus there is no infinite sequence of $\gg$-decreasing finite subsets of $\Sigma \cup \tau^c$ and $\gg$ is well-founded.

5.2. An Order on Formulas

We use the well-founded order $\gg$ on sets of symbols for defining an order $\gg_p$ on formulas (which is later refined to the desired termination order, cf. Section 5.3). The idea underlying the development of $\gg_p$ is to model the difficulty of a proof; i.e., $\varphi \gg_p \psi$ should hold if $\varphi$ is (expected to be) harder provable than $\psi$.

For realizing this idea, we consider sets of (defined) maximal symbols (w.r.t. $\gg \defeq$), since their occurrences have a substantial influence on the difficulty of a proof:

DEFINITION 5 (Pure sets, maximal pure subset, $\text{purify}_\gg$). A finite subset $S \subseteq \Sigma \cup \tau^c$ is called pure iff

1. $S \subseteq \Sigma^d$ and $s_1 \gg \defeq s_2$ for all $s_1, s_2 \in S$,
2. $S \subseteq \tau^c$, or
3. $S \subseteq \Sigma^e$.

We let $\text{purify}_\gg S$ denote the maximal pure subset of $S \subseteq \Sigma \cup \tau^c$, i.e.

1. $\text{purify}_\gg S$ is the set of $\gg \defeq$-maximal elements of $S \cap \Sigma^d$ if $S \cap \Sigma^d \neq \emptyset$,
2. $\text{purify}_\gg S = S \cap \tau^c$ if $S \cap \Sigma^d = \emptyset$ and $S \cap \tau^c \neq \emptyset$,
3. $\text{purify}_\gg S = S$ otherwise.

For instance, $\{\exp, \prod, \sum\}$ and $\{x, y\}$ are pure whereas $\{\exp, \times, \text{plus}\}$ as well as $\{\prod, \sum, x, s\}$ are not. Furthermore, e.g., $\text{purify}_\gg \{x, y, s\} = \{x, y\}$ and $\text{purify}_\gg \{\exp, \times, \plus, \prod, \sum, x, s\} = \{\exp, \prod, \sum\}$.

We let $\mathcal{S}(\phi) \subseteq \Sigma \cup \tau^c$ denote the set of all function and variable symbols in a (set of) term(s) or formula(s) $\phi$. Thus if a formula $\varphi$ contains at least one defined
function symbol, then \( \text{purify}_\varphi \mathcal{F}( \varphi ) \) is the set of all maximal defined function symbols occurring in \( \varphi \), with which the difficulty of (proving) \( \varphi \) is estimated.

Using \( \text{purify}_\varphi \) and \( \succ \), a relation \( \succ_\varphi \) on formulas now can be defined, where we use the number of occurrences \( \#_f( \varphi ) \in \mathbb{N} \) of a symbol \( f \in \Sigma \cup \mathcal{F} \) in a (set of) term(s) or formula(s) \( \varphi \).

**Definition 6 (Order \( \succ_\varphi \) on formulas).**

\[
\varphi \succ_\varphi \psi \text{ iff } (a) \text{ purify}_\varphi \mathcal{F}( \varphi ) \succ \text{ purify}_\varphi \mathcal{F}( \psi ),
\]

or (b) \( \text{ purify}_\varphi \mathcal{F}( \varphi ) \approx \text{ purify}_\varphi \mathcal{F}( \psi ) \) and

\[
\sum_{f \in \text{purify}_\varphi \mathcal{F}( \varphi )} \#_f( \varphi ) > \sum_{f \in \text{purify}_\varphi \mathcal{F}( \psi )} \#_f( \psi ).
\]

\( \succ_\varphi \) is well-founded, because it is formed as a lexicographic combination of well-founded relations, cf. Theorem 5.4. The restriction to maximal defined functions models the observation that for proving a statement \( \varphi \) about some function \( f \), quite inevitably also properties of functions \( g \) used for defining \( f \) (i.e., \( f \succ_\varphi^\mathcal{F} g \)) have to be considered, and this is independent of whether \( g \) already occurs in \( \varphi \) or not.

Criterion (b) is a simple refinement regarding the number of occurrences of maximal symbols. Note that although (a) compares sets of maximal symbols with the multiset-order \( \succ^\mathcal{F} \) (cf. case (1) in the definition of \( \succ \)) and (b) compares the number of occurrences of maximal symbols, we do not merge these criteria such that the multisets of occurrences of maximal symbols would be compared.

This is because, e.g., for \( \varphi \) containing one occurrence of \( \text{maxl} \) as well as \( \text{rev} \) and for \( \psi \) containing only two occurrences of \( \text{maxl} \), criterion (a) succeeds with \( \text{purify}_\varphi \mathcal{F}( \varphi ) = \{ \text{maxl}, \text{rev} \} \succ^\mathcal{F} \{ \text{maxl} \} = \text{purify}_\varphi \mathcal{F}( \psi ) \) while the combined criterion would fail due to \( \{ \text{maxl}, \text{rev} \} \succ^\mathcal{F} \{ \text{maxl}, \text{maxl} \} \), where \( \{ \ldots \} \) denotes a multiset.

For an example of using \( \succ_\varphi \), reconsider Table 3. Proving \( \varphi_1 \) by reuse leads to speculating the lemmata \( \varphi_{17}, \varphi_{18}, \) and \( \varphi_{24} \) in turn. We find \( \text{purify}_\varphi \mathcal{F}( \varphi_1 ) = \{ \text{prod}, \text{app} \} \succ^\mathcal{F} \{ \text{times} \} = \text{purify}_\varphi \mathcal{F}( \varphi_{17} ) = \text{purify}_\varphi \mathcal{F}( \varphi_{18} ) \succ^\mathcal{F} \{ \text{plus} \} = \text{purify}_\varphi \mathcal{F}( \varphi_{24} ) \) and \( \#_{\text{times}}( \varphi_{17} ) = 4 \succ 3 = \#_{\text{times}}( \varphi_{18} ) \), and therefore \( \varphi_1 \succ_\varphi \varphi_{17} \succ_\varphi \varphi_{18} \succ_\varphi \varphi_{24} \). Also \( \varphi_5 \not\succ_\varphi \sigma_5(\varphi_{11}) \) because \( \text{purify}_\varphi \mathcal{F}( \varphi_5 ) = \{ \text{rev} \} \succ^\mathcal{F} \{ \text{app} \} = \text{purify}_\varphi \mathcal{F}( \sigma_5(\varphi_{11}) ) \) for the instance \( \sigma(\varphi_{11}) \) of \( \varphi_{11} \) with \( \sigma_5 = \{ p/m :: \varepsilon \} \) which is speculated when proving \( \varphi_5 \) by reuse.

### 5.3. The Refined Termination Order

However, the \( \succ_\varphi \) relation is still too weak for our purposes. Consider e.g., conjecture \( \varphi_{15} :\equiv \text{exp}(\text{exp}(i,n),m) \equiv \text{exp}(i,\text{times}(m,n)) \) for which the reuse procedure speculates lemma \( \varphi_{16} :\equiv \text{times}(\text{exp}(i,m),\text{exp}(i,n)) \equiv \text{exp}(i,\text{plus}(m,n)) \), cf. Table 1. Since \( \text{purify}_\varphi \mathcal{F}( \varphi_{15} ) = \{ \text{exp} \} = \text{purify}_\varphi \mathcal{F}( \varphi_{16} ) \) and \( \#_{\text{exp}}( \varphi_{15} ) = 3 = \#_{\text{exp}}( \varphi_{16} ), \) \( \varphi_{15} \not\succ_\varphi \varphi_{16} \) does not hold. Also \( \varphi_{23} \succ_\varphi \varphi_{25} \) for the conjectures \( \varphi_{23} :\equiv \text{plus}(m,n) \equiv \text{plus}(m,s(n)) \) and \( \varphi_{25} :\equiv \text{plus}(m,\text{switch}(m)) \equiv \text{switch}(\text{plus}(m,n)) \) from Table 1, because \( \text{purify}_\varphi \mathcal{F}( \varphi_{23} ) = \{ \text{plus} \} = \text{purify}_\varphi \mathcal{F}( \varphi_{25} ) \) and \( \#_{\text{plus}}( \varphi_{23} ) = 2 = \#_{\text{plus}}( \varphi_{25} ). \)

As a remedy, we also consider the arguments in an application of a maximal function symbol in a conjecture. Since induction theorem proving strongly depends...
on the recursive definition of functions, we focus on their recursion arguments like the second argument of \( \exp \) which is defined by \( \exp(m, 0) = s(0) \) and \( \exp(m, s(n)) = \times(m, \exp(m, n)) \). We observe that the symbol \( \times \) occurs in the second argument of \( \exp \) in \( \varphi_{15} \), while only the \( \defsymbol \)-smaller function symbol plus and the variables \( m, n \) occur in the second arguments of \( \exp \) in \( \varphi_{16} \).

Based on this observation, we refine \( >_F \) by an additional requirement which also considers the arguments of \( \defsymbol \)-maximal function symbols in a formula: Since all defined function symbols \( f \in \Sigma^d \) are introduced by algorithmic specifications (from which the defining equations are uniformly obtained), we may identify non-empty sets of so-called recursion variables \( R_f \equiv \{x_1, \ldots, x_n\} \) with each term \( f(x_1, \ldots, x_n) \), where \( x_1, \ldots, x_n \) are distinct variables, if \( f \) is recursively defined, cf. [Wal94] and the notion of “measured subsets” in [BM79]. Each such set \( R_f \) stipulates the variables to be induced upon when a statement containing a term \( f(x_1, \ldots, x_n) \) is to be proved by induction. We let \( \Pi_f \equiv \{1, \ldots, n\} \) denote the set of recursion positions with \( i \in \Pi_f \) iff \( x_i \in R_f \) for some \( R_f \). For the sake of simplicity we only consider here recursively defined functions, i.e., a function such as square defined as \( \times(m, \exp(m, n)) \) is excluded and therefore \( \Pi_f \not\subseteq \emptyset \) if \( f \in \Sigma^d \). Now the set \( \text{rst}_{f, i}([t]) \) of subterms of a term \( t \) which occupy the position of a recursion variable \( x_i \) of a function symbol \( f \) is computed as:

\[
\begin{align*}
1) \quad \text{rst}_{f, i}([x]) &= \emptyset, &\text{for all } x \in \mathcal{V}, \\
2) \quad \text{rst}_{f, i}([f(t_1, \ldots, t_m)]) &= \{t_i\},^{11} \\
3) \quad \text{rst}_{f, i}([g(t_1, \ldots, t_m)]) &= \text{rst}_{f, i}([t_1]) \cup \cdots \cup \text{rst}_{f, i}([t_m]), &\text{for all } g \neq f.
\end{align*}
\]

\( \text{rst}_{f, i} \) is extended to formulas \( \varphi \) by (3), where \( g \) is \( \equiv \) or any connective symbol, e.g.

\[ \text{rst}_{\exp, 2}([\varphi_{15}]) = \{m, \times(m, n)\}, \quad \text{and } \text{rst}_{\exp, 2}([\varphi_{16}]) = \{m, n, \text{plus}(m, n)\}. \]

If we compare the maximal symbols \( \times \) resp. plus of the recursion arguments in this example, the formulas \( \varphi_{15} \) and \( \varphi_{16} \) now can be related, and the same holds for \( \text{rst}_{\text{plus}, 1}([\varphi_{23}]) = \{m, n\} \supset \{m\} = \text{rst}_{\text{plus}, 1}([\varphi_{25}]) \). The latter comparison explains the treatment of (sets of) variables in our approach by the order \( >_F \), since the proof of a statement with different variables at recursion positions of maximal symbols is usually more difficult than the proof of a statement with only one variable, cf. [Wal94].

As \( \text{purify}_{>_F} \) \( \varphi \) may contain several defined functions each of which may have several recursion arguments, we have to compare several sets each containing some maximal symbols. We merge these comparisons into one by using the nonstrict multiset order \( \supseteq \) imposed by \( \supseteq \) on multisets of finite subsets of \( \Sigma \cup \mathcal{V} \), cf. [Der87], where e.g. \{\{x, y\}, \{\times, \text{sum}\}, \{x, y\}\} \supseteq \{\{u, v\}, \{\text{plus}\}, \{x\}\}.

For incorporation of recursion arguments, \( >_F \) now is redefined:

**Definition 7 (Refined order \( >_F \) on formulas).** Let \( \varphi, \psi \) be formulas with \( \varphi \in \Sigma^d \not\subseteq \emptyset \) and \( \psi \in \Sigma^d \not\subseteq \emptyset \), let

\( ^{11} \) Note that only the outermost occurrences of (maximal symbols) \( f \) are considered, i.e., we do not stipulate \( \text{rst}_{f, i}([f(t_1, \ldots, t_n)]) = \{t_i\} \cup \text{rst}_{f, i}([t_i]) \cup \cdots \cup \text{rst}_{f, i}([t_n]) \).
(a) \( \varphi \geq_1 \psi \) iff \( \text{purify} \succ \mathcal{G}(\varphi) \geq \text{purify} \succ \mathcal{G}(\psi) \),

(b) \( \varphi \geq_2 \psi \) iff \( \sum_{f \in \text{purify} \succ \mathcal{G}(\varphi)} \# f(\varphi) \geq \sum_{f \in \text{purify} \succ \mathcal{G}(\psi)} \# f(\psi) \),

(c) \( \varphi \geq_3 \psi \) iff \( \mathcal{M}_{\text{rest}}[\varphi] \geq \mathcal{M}_{\text{rest}}[\psi] \), where
\[
\mathcal{M}_{\text{rest}}[\psi] := \{ \text{purify} \succ \mathcal{G}(\text{rxt}_{i_j}[\psi]) | f \in \text{purify} \succ \mathcal{G}(\psi), i \in \Pi \},
\]

(d) \( \varphi \geq_4 \psi \) iff \( \varphi = \sigma(\psi) \) for some (first-order) substitution \( \sigma \), let
\[
\equiv_i := \geq_i \cap \leq_i,
\]

and let \( \succ := \geq \setminus \equiv \), for each \( i \in \{ 1, 2, 3, 4 \} \).

Then \( \varphi \geq_2 \psi \) iff \( \equiv \psi \) and \( \varphi \geq_k \psi \) for some \( k \in \{ 1, 2, 3, 4 \} \) and all \( j \in \{ 1, \ldots, k-1 \} \).

Since Definition 5.7 demands that \( \varphi \) and \( \psi \) both contain one defined function symbol at least, we have \( \text{purify} \succ \mathcal{G}(\varphi) \subseteq \Sigma^d \). Therefore \( \Pi_f \) in (c) is defined and consequently \( \geq_3 \) is well-defined. Requirement (c) of Definition 5.7 incorporates the inspection of recursion arguments as demanded. By requirement (d), a pair of conjectures \( \varphi \) and \( \psi \) can also be related if \( \psi \) is strictly more general than \( \varphi \). This feature is useful in particular if a speculated lemma can be obtained as a generalization by inverted substitution [Wal94]; see Section 6.

**Corollary 8** (Well-foundedness of refined \( \geq_F \)). \( \geq_F \) is well-founded.

**Proof.** \( \geq_1 \) is well-founded by Theorem 5.4, and the well-foundedness of \( \geq_2 \) is obvious. \( \geq_3 \) is well-founded as the strict part \( \gg \) of \( \geq_3 \) is well-founded by Theorem 5.4 and [Der87]. \( \geq_4 \) is the strict subsumption order \( \gg \) on formulas which is also well-founded, cf. [DJ90]. Since \( \geq_F \) is formed as a lexicographic combination of quasi-orderings whose strict parts are well-founded, \( \geq_F \) is also well-founded.

By Corollary 8 the reuse procedure terminates if we demand the termination requirement for reuse, viz. \( \varphi \geq_F \varphi' \), for each reducible member \( \varphi \) of a simplified totally instantiated catch which is computed when proving \( \varphi \) by reuse.

### 6. USEFULNESS OF \( \geq_F \)

The usefulness of \( \geq_F \) is illustrated by Table 2. Here all pairs \( \varphi, \varphi' \) from Table 1 are compared by \( \geq_F \), where \( \varphi' \) is speculated by the **Plagiator** system as a lemma when the conjecture \( \varphi \) is to be proved by reuse. Columns (a), (b), and (c) compare conjectures and lemmata by criteria (a), (b), and (c) from Definition 7.

Note that many other proof obligations are generated by reuse which do not have to be related by \( \geq_F \) as they are (variants of) axioms and therefore irreducible. So far, we were not faced with a conjecture which can be proved by reuse without the termination requirement, but cannot if the termination requirement is obeyed. This supports our claim that the well-founded relation \( \geq_F \) indeed is useful for guaranteeing the termination of the reuse procedure without spoiling the system’s performance. The example from the beginning of Section 5 does not contradict this claim, because reuse is not successful there. So quite on the contrary, this example
reveals that by the termination requirement unsuccessful reuse attempts can be avoided.

However, since our claim of the usefulness of \( \text{>}_F \) is based only on experiments with the Plagiator system, we also analyzed lemma speculation in induction theorem proving in general. Table 3 illustrates the usefulness of \( \text{>}_F \) by examples for lemma speculation in induction theorem proving borrowed from [IB96]. There theorems \( T_1, \ldots, T_{50} \) are given which can be proved by 24 speculated lemmata \( L_1, \ldots, L_{24} \) (and 12 generalizations). The defined functions are

\[
\begin{align*}
\text{dbl}, \text{half}, \text{even}, \text{len}, \text{nth}, \text{qrev}, \text{cnt}, \text{mem}, \text{ordered}
\end{align*}
\]

\[
\begin{align*}
\text{rev} & \text{def} \ \text{app}, \\
\text{rotate} & \text{def} \ \text{app}, \\
\text{isort} & \text{def} \ \text{ins}
\end{align*}
\]

The theorem-lemma pairs are presented in Table 3, where theorems and lemmata are grouped together, e.g., \( T_8 \) uses \( L_4 \) and \( L_5 \), while \( T_{10}, T_{17}, \) and \( T_{19} \) use \( L_8 \). Column \( \text{>}_F \) in Table 3 denotes the criterion of Definition 7 which is satisfied for the particular theorem-lemma pair, i.e., we obtain, e.g., \( T_8 >_F L_4 \) and \( T_8 >_F L_5 \) by criterion (b).13

For all examples presented in Table 3, each lemma \( \phi' \) speculated in a proof of \( \phi \) is \( \text{<}_F \)-smaller than \( \phi \). This observation gives additional evidence for the usefulness of

12 Theorems \( T_36 \)–\( T_{47} \) do not use lemmata at all. Theorems \( T_{27} \)–\( T_{35} \) are proved by \textit{generalization} as lemma speculation and thus are not considered here; see below.

13 For \( T_{12} \) only Lemma \( L_{11} \) is speculated, hence “–” in the last column.
<table>
<thead>
<tr>
<th>No.</th>
<th>Theorem resp. Lemma</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>(\text{dbl}(X) = X + X)</td>
</tr>
<tr>
<td>T3</td>
<td>(\text{len}(X &lt; &gt; Y) = \text{len}(Y) + \text{len}(X))</td>
</tr>
<tr>
<td>T7</td>
<td>(\text{len}(\text{rev}(X, Y)) = \text{len}(X) + \text{len}(Y))</td>
</tr>
<tr>
<td>T13</td>
<td>(\text{half}(X + X) = X)</td>
</tr>
<tr>
<td>T16</td>
<td>(\text{even}(X + X))</td>
</tr>
<tr>
<td>L1</td>
<td>(\text{plus}(x, s(y)) = s(\text{plus}(x, y)))</td>
</tr>
<tr>
<td>T2</td>
<td>(\text{len}(\text{app}(x, y)) = \text{len}(\text{app}(y, x)))</td>
</tr>
<tr>
<td>T4</td>
<td>(\text{len}(X &lt; &gt; X) = \text{dbl}(\text{len}(X)))</td>
</tr>
<tr>
<td>T6</td>
<td>(\text{len}(\text{rev}(X &lt; &gt; Y)) = \text{len}(X) + \text{len}(Y))</td>
</tr>
<tr>
<td>T20</td>
<td>(\text{even}(\text{len}(X &lt; &gt; X)))</td>
</tr>
<tr>
<td>L2</td>
<td>(\text{len}(\text{app}(x, y :: z) = s(\text{len}(\text{app}(x, z))))</td>
</tr>
<tr>
<td>T5</td>
<td>(\text{len}(\text{rev}(X)) = \text{len}(X))</td>
</tr>
<tr>
<td>L3</td>
<td>(\text{len}(X &lt; &gt; Y :: \text{empty}) = s(\text{len}(X)))</td>
</tr>
<tr>
<td>T8</td>
<td>(\text{nth}(x, \text{nth}(x :: z)) = \text{nth}(y, \text{nth}(x, z)))</td>
</tr>
<tr>
<td>L4</td>
<td>(\text{nth}(!x, \text{nth}(x :: z)) = \text{nth}(w, \text{nth}(x, z)))</td>
</tr>
<tr>
<td>L5</td>
<td>(\text{nth}(s(w), \text{nth}(s(x :: z))) = \text{nth}(w, \text{nth}(s(x :: z))))</td>
</tr>
<tr>
<td>T9</td>
<td>(\text{nth}(w, \text{nth}(x, y :: z)) = \text{nth}(w, \text{nth}(y :: z)))</td>
</tr>
<tr>
<td>L6</td>
<td>(\text{nth}(s(w), \text{nth}(w, x :: z)) = \text{nth}(s(x, x :: z)))</td>
</tr>
<tr>
<td>L7</td>
<td>(\text{nth}(s(s(w), x :: z)) = \text{nth}(x, s(w, x :: z)))</td>
</tr>
<tr>
<td>T10</td>
<td>(\text{rev}(\text{rev}(x)) = x)</td>
</tr>
<tr>
<td>T17</td>
<td>(\text{rev}(\text{rev}(\text{app}(x, y))) = \text{app}(\text{rev}(\text{rev}(x)), \text{rev}(\text{rev}(y))))</td>
</tr>
<tr>
<td>T19</td>
<td>(\text{app}(\text{rev}(\text{rev}(x), y)) = \text{rev}(\text{rev}(\text{app}(x, y))))</td>
</tr>
<tr>
<td>L8</td>
<td>(\text{rev}(\text{app}(x, y :: \text{empty})) = y :: \text{rev}(x))</td>
</tr>
<tr>
<td>T11</td>
<td>(\text{rev}(\text{app}(x, y)) = \text{app}(y, x))</td>
</tr>
<tr>
<td>L9</td>
<td>(\text{rev}(\text{app}(x, y :: \text{empty})) = z :: \text{rev}(\text{app}(x, y)))</td>
</tr>
<tr>
<td>T10</td>
<td>(\text{rev}(\text{app}(x, y :: \text{empty}, \text{empty})) = y :: \text{rev}(x, \text{empty}))</td>
</tr>
<tr>
<td>T12</td>
<td>(\text{rev}(X, Y) = \text{rev}(X) &lt; &gt; Y)</td>
</tr>
<tr>
<td>T18</td>
<td>(\text{rev}(\text{rev}(X) &lt; &gt; Y) = \text{rev}(Y) &lt; &gt; X)</td>
</tr>
<tr>
<td>T21</td>
<td>(\text{rotate}(\text{len}(X), X &lt; &gt; Y) = Y &lt; &gt; X)</td>
</tr>
<tr>
<td>L11</td>
<td>((X &lt; &gt; (Y :: \text{empty})) &lt; &gt; Z = X &lt; &gt; (Y :: Z))</td>
</tr>
<tr>
<td>L13</td>
<td>((X &lt; &gt; Y) &lt; &gt; Z :: \text{empty} = X &lt; &gt; (Y &lt; &gt; Z :: \text{empty}))</td>
</tr>
<tr>
<td>T14</td>
<td>(\text{ordered}(\text{sort}(X)))</td>
</tr>
<tr>
<td>L12</td>
<td>(\text{ordered}(Y) \rightarrow \text{ordered}(\text{ins}(X, Y)))</td>
</tr>
<tr>
<td>T22</td>
<td>(\text{even}(\text{len}(\text{app}(x, y))) \Rightarrow \text{even}(\text{len}(\text{app}(y, x))))</td>
</tr>
<tr>
<td>L14</td>
<td>(\text{even}(\text{len}(\text{app}(w, z))) \Rightarrow \text{even}(\text{len}(\text{app}(w, x :: y :: z))))</td>
</tr>
<tr>
<td>T23</td>
<td>(\text{half}(\text{len}(X &lt; &gt; Y)) = \text{half}(\text{len}(Y &lt; &gt; X)))</td>
</tr>
<tr>
<td>L15</td>
<td>(\text{len}(W &lt; &gt; X :: Y :: Z) = s(\text{len}(W &lt; &gt; Z)))</td>
</tr>
<tr>
<td>T24</td>
<td>(\text{even}(\text{plus}(x, y)) \Rightarrow \text{even}(\text{plus}(y, x)))</td>
</tr>
<tr>
<td>T25</td>
<td>(\text{even}(\text{len}(X &lt; &gt; Y)) \Rightarrow \text{even}(\text{len}(Y) + \text{len}(X)))</td>
</tr>
<tr>
<td>L16</td>
<td>(\text{even}(\text{plus}(x, y)) \Rightarrow \text{even}(\text{plus}(x, s(y))))</td>
</tr>
<tr>
<td>T26</td>
<td>(\text{half}(X + Y) = \text{half}(Y + X))</td>
</tr>
<tr>
<td>L17</td>
<td>(X + s(X) = s(X + Y))</td>
</tr>
<tr>
<td>T48</td>
<td>(\text{len}(\text{sort}(X)) = \text{len}(X))</td>
</tr>
<tr>
<td>T49</td>
<td>(\text{len}(\text{ins}(X, Y)) = s(\text{len}(Y)))</td>
</tr>
<tr>
<td>T19</td>
<td>(X \neq Y \rightarrow (X \in \text{ins}(Y, Z) \rightarrow X \in Z))</td>
</tr>
<tr>
<td>T30</td>
<td>(\text{cnt}(X, \text{sort}(Y)) = \text{cnt}(X, Y))</td>
</tr>
<tr>
<td>L20</td>
<td>(\text{cnt}(X, \text{ins}(X, Y)) = s(\text{cnt}(X, Y)))</td>
</tr>
<tr>
<td>L21</td>
<td>(X \neq Y \rightarrow \text{cnt}(X, Y :: \text{ins}(Y, Z)) = \text{cnt}(X, Z))</td>
</tr>
</tbody>
</table>
For dealing with the only remaining theorem-lemma pair from [IB96], viz. $T15 >_F L1$ in Table 4, criterion $(d)$ of Definition 7 is used. This is because Lemma $L1$ which is speculated for proving Theorem $T15$ can also be obtained as a generalization (by inverted substitution, cf. [Wal94]) of $T15$. Our order $>_F$ is appropriate only for this kind of generalizations and an extension of the termination requirement for incorporating other generalizations is a subject for future research. Note that there is no well-founded relation $\sqsubseteq$ such that $\varphi \sqsubseteq \psi$ for each sound generalization $\psi$ of a conjecture $\varphi$, because there are non-well-founded generalizations such as $\psi := \varphi \land \varphi'$ for some $\varphi'$, cf. [Wal94]. But as sophisticated heuristics are used for deciding when and which generalization is performed, one might find a well-founded relation sufficient for dealing with practical examples.

7. CONCLUSION

We have developed a termination requirement for our method of reusing proofs which is based on a partial, well-founded ordering on formulas. We have proved the soundness of our proposal and gave evidence that only unsuccessful reuse attempts are prevented by the termination requirement imposed on the reuse procedure.

We also considered the termination of lemma speculation for induction theorem proving in general. The analysis of problem sets in this domain [IB96] gives additional evidence for the usefulness of our termination requirement since also here no successful lemma speculations are prevented. Future work might investigate the treatment of generalizations within this framework.

ACKNOWLEDGMENTS

We are grateful to Thomas Arts and Jürgen Giesl for helpful discussions and comments on this paper.

Received July 27, 1998; final manuscript received October 5, 1998; published online September 6, 2000

REFERENCES

ON TERMINATING LEMMA SPECULATIONS


