Information and Computation **162**, 96–116 (2000)
doi:10.1006/inco.1999.2859, available online at http://www.idealibrary.com on IDEAL®

On Terminating Lemma Speculations[1]

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].

0890-5401/00 $35.00

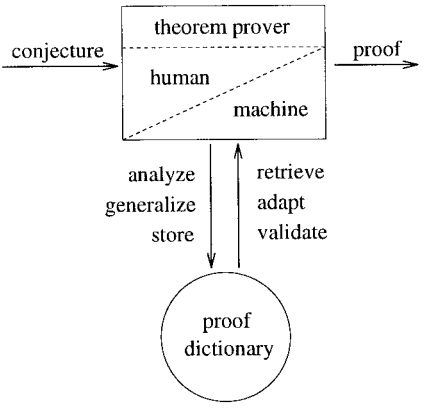


FIG. 1. Theorem proving with reuse.

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* $IH \to IC$ 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 $\mathrm{s}(x)$ (resp. empty and $\mathsf{add}(n, x)$) are the constructors of the sort number (resp. list):[3]

$$\begin{array}{lll}
\text{(plus-1,2)} & \mathsf{plus}(0, y) \equiv y & \mathsf{plus}(\mathrm{s}(x), y) \equiv 1\mathrm{s}(\mathsf{plus}(x, y)) \\
\text{(sum-1,2)} & \mathsf{sum}(\mathsf{empty}) \equiv 0 & \mathsf{sum}(\mathsf{add}(n, x)) \equiv \mathsf{plus}(n, \mathsf{sum}(x)) \\
\text{(app-1,2)} & \mathsf{app}(\mathsf{empty}, y) \equiv y & \mathsf{app}(\mathsf{add}(n, x), y) \equiv \mathsf{add}(n, \mathsf{app}(x, y))
\end{array}$$

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

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

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

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

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 φ_b and the step formula φ_s as

$$\varphi_b := \forall y\ \varphi[\mathsf{empty}, y]$$

$$\varphi_s := \forall n, x, y\ (\forall u\ \varphi[x, u]) \to \varphi[\mathsf{add}(n, x), y].$$

The following proof of the step formula φ_s is obtained by modifying the induction conclusion $\varphi[\mathsf{add}(n, x), y] =$

$$\mathsf{plus}(\mathsf{sum}(\mathsf{add}(n, x)), \mathsf{sum}(y)) \equiv \mathsf{sum}(\mathsf{app}(\mathsf{add}(n, x), y))\ \text{IC}$$

[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]

$$\mathsf{plus}(\mathsf{sum}(\mathsf{add}(n, x)), \mathsf{sum}(y)) \equiv \mathsf{sum}(\mathsf{app}(\mathsf{add}(n, x), y)) \qquad \text{IC}$$

$$\mathsf{plus}(\underline{\mathsf{plus}(n, \mathsf{sum}(x))}, \mathsf{sum}(y)) \equiv \mathsf{sum}(\mathsf{app}(\mathsf{add}(n, x), y)) \qquad (\mathsf{sum\text{-}2})$$

$$\mathsf{plus}(\mathsf{plus}(n, \mathsf{sum}(x)), \mathsf{sum}(y)) \equiv \mathsf{sum}(\underline{\mathsf{add}(n, \mathsf{app}(x, y))}) \qquad (\mathsf{app\text{-}2})$$

$$\mathsf{plus}(\mathsf{plus}(n, \mathsf{sum}(x)), \mathsf{sum}(y)) \equiv \underline{\mathsf{plus}(n, \mathsf{sum}(\mathsf{app}(x, y)))} \qquad (\mathsf{sum\text{-}2})$$

$$\mathsf{plus}(\mathsf{plus}(n, \mathsf{sum}(x)), \mathsf{sum}(y)) \equiv \mathsf{plus}(n, \underline{\mathsf{plus}(\mathsf{sum}(x), \mathsf{sum}(y))}) \qquad \text{IH}$$

$$\underline{\mathsf{plus}(n, \mathsf{plus}(\mathsf{sum}(x), \mathsf{sum}(y)))} \equiv \mathsf{plus}(n, \mathsf{plus}(\mathsf{sum}(x), \mathsf{sum}(y))) \qquad (\mathsf{lem\text{-}1})$$

$$\underline{\mathsf{true}} \qquad x \equiv x$$

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 (sum-2), (app-2), and (lem-1) as the catch. E.g., all we have to know about plus for proving φ_s 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* Φ_s as well as the *schematic catch* C_s (see Fig. 2).

If a new statement ψ shall be proved, a suitable induction axiom is selected by well-known automated methods, cf. [Wal94], from which a set of induction formulas I_ψ is computed for ψ. Then for proving an induction formula $\psi_i \in I_\psi$ by reuse, it is tested whether some proof shell [MPS] *applies for* ψ_i, i.e., whether ψ_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), ψ_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

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

[4] We omit a proof for the base formula φ_b as there are no particularities compared to the step case.

[5] Not to be confused with *generalization* of a formula φ as a preprocessing for proving φ by induction.

$$\Phi_s := \big(\forall u\ F(G(x), G(u)) \equiv G(H(x,u))\big) \rightarrow F(G(D(n,x)), G(y)) \equiv G(H(D(n,x), y))$$

$$C_s := \left\{ \begin{array}{lll} (1) & G(D(n,x)) & \equiv F(n, G(x)) \\ (2) & H(D(n,x), y) & \equiv D(n, H(x,y)) \\ (3) & F(F(x,y), z) & \equiv F(x, F(y,z)) \end{array} \right\}$$

FIG. 2. The proof shell $[MPS]_s$ for the proof of φ_s (simple analysis).

and times and prod are defined by the equations

(times-1,2) $\mathsf{times}(0, y) \equiv 0$, $\mathsf{times}(\mathrm{s}(x), y) \equiv \mathsf{plus}(y, \mathsf{times}(x, y))$

(prod-1,2) $\mathsf{prod}(\mathsf{empty}) \equiv \mathrm{s}(0)$, $\mathsf{prod}(\mathsf{add}(n, x)) \equiv \mathsf{times}(n, \mathsf{prod}(x))$.

The induction formulas computed for ψ are

$$\psi_b := \forall y\ \psi[\mathsf{empty}, y]$$

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

Obviously ψ is an instance of Φ and ψ_s is an instance of Φ_s w.r.t. the matcher $\pi := \{F/\mathsf{times}, G/\mathsf{prod}, H/\mathsf{app}, D/\mathsf{add}\}$. 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:

$$\pi(C_s) = \left\{ \begin{array}{lll} (4) & \mathsf{prod}(\mathsf{add}(n, x)) & \equiv \mathsf{times}(n, \mathsf{prod}(x)) \\ (5) & \mathsf{app}(\mathsf{add}(n, x), y) & \equiv \mathsf{add}(n, \mathsf{app}(x, y)) \\ (6) & \mathsf{times}(\mathsf{times}(x, y), z) & \equiv \mathsf{times}(x, \mathsf{times}(y, z)) \end{array} \right\}$$

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 times (6) and, if successful, ψ_s is proved. Compared to a direct proof of ψ_s we have saved the user interactions necessary to apply the right axioms in the right place (where the associativity of times must be verified in either case). Additionally, conjecture (6) has been *speculated as a lemma* which is required for proving conjecture ψ.

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) φ 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 $\varphi \rightarrow \varphi$.

Analyze [KW94]. The *simple proof analysis* which was illustrated in Section 2 analyzes a proof p of φ, yielding a *proof catch c*. Formally, the catch c is a finite subset of nonlogical axioms of AX such that c logically implies φ. For increasing the applicability of proof shells and the reusability of proofs, we have developed the

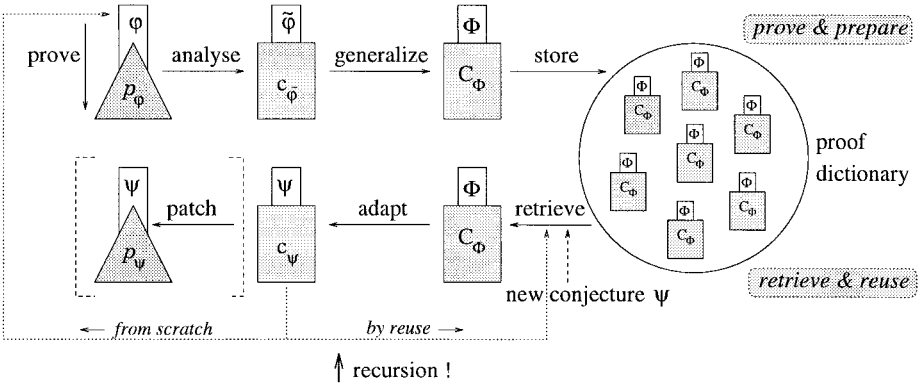


FIG. 3. The reuse process.

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 $\psi_2 := \mathsf{plus}(\mathsf{len}(x), \mathsf{len}(y)) \equiv \mathsf{len}(\mathsf{app}(x, y))$ 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 φ and c are *generalized* by replacing (different occurrences of) function *symbols* with (different) function *variables*. This yields a *schematic conjecture* Φ 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 Φ. 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 φ_s from Section 2, generalization yields $\Phi' := F^1(G^1(x), G^2(y)) \equiv G^3(H^1(x, y))$ 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 $\langle \Phi, C_1 \rangle, ..., \langle \Phi, C_n \rangle$ (sharing a common schematic goal formula Φ) are merged into a *proof volume* $PV := \langle \Phi, \{C_1, ..., C_n\} \rangle$ 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 ψ is to be proved, the proof dictionary is searched for a proof volume $PV := \langle \Phi, \{C_1, ..., C_n\} \rangle$ such that $\psi = \pi(\Phi)$ for

$$\Phi'_s := \big(\forall u\ F^1(G^1(x), G^2(u)) \equiv G^3(H^1(x,u))\big) \rightarrow F^1(G^1(D^1(n,x)), G^2(y)) \equiv G^3(H^1(D^1(n,x), y))$$

$$C'_s := \left\{ \begin{array}{lrcl} (7) & G^1(D^1(n,x)) & \equiv & F^2(n, G^1(x)) \\ (8) & H^1(D^1(n,x), y) & \equiv & D^4(n, H^1(x,y)) \\ (9) & G^3(D^4(n,x)) & \equiv & F^3(n, G^3(x)) \\ (10) & F^1(F^2(x,y), z) & \equiv & F^3(x, F^1(y,z)) \end{array} \right\}$$

FIG. 4. The proof shell PS'_s for the proof of φ_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 π. If successful, the schematic conjecture Φ and in turn also the proof volume *PV applies for* ψ (*via the matcher* π). Here some restrictions on the class of admissible matchers can be imposed to make the retrieval more efficient, cf. [KW95c]. E.g., $\pi_2 := \{F^1/\mathsf{plus}, G^{1,2,3}/\mathsf{len}, H^1/\mathsf{app}, D^{1,2}/\mathsf{add}\}$ is obtained by matching Φ'_s from Fig. 4 with ψ_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 ψ by reuse. For our example, the partially instantiated catch is obtained as

$$\pi_2(C'_s) = \left\{ \begin{array}{ll} (11) & \mathsf{len}(\mathsf{add}(n, x)) \equiv F^2(n, \mathsf{len}(x)) \\ (12) & \mathsf{app}(\mathsf{add}(n, x), y) \equiv D^4(n, \mathsf{app}(x, y)) \\ (13) & \mathsf{len}(D^4(n, x)) \equiv F^3(n, \mathsf{len}(x)) \\ (14) & \mathsf{plus}(F^2(x, y), z) \equiv F^3(x, \mathsf{plus}(y, z)) \end{array} \right\}.$$

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 Φ, these function variables have to be instantiated by known functions. Free function variables such as F^2, F^3, and D^4 in $\pi_2(C'_s)$ 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 ρ 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 ρ is called a *solution* (for the free function variables), and ψ is proved by reuse because semantical entailment is invariant w.r.t. (second-order) instantiation. Solution candidates ρ are computed by *second-order matching modulo symbolical evaluation*; cf. [KW95d]. For the example, the solution $\rho_2 := \{F^{2,3}/\mathsf{s}(w_2), D^4/\mathsf{add}\}$ is obtained which instantiates (11) to the axiom $\mathsf{len}(\mathsf{add}(n, x)) \equiv \mathsf{s}(\mathsf{len}(x))$.[6]

Patch [KW95b]. Often one is not only interested in the *provability* of ψ, but also in a *proof* of ψ 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 Φ (which is obtained by generalizing the proof p of φ) with the computed substitution $\tau := \rho \circ \pi$ because τ 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 ψ, 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 φ_i given in Table 1 below. For the sake of readability we use mathematical (infix) symbols for functions where appropriate, i.e., $\times$, $+$, $-$, $<>$, $|.|$, $\sum$, and $\prod$ denote times, plus, minus, app, len, sum, and prod, respectively. We use the

[6] The instantiations of F^1 and F^2 are different here, viz. plus and s, and this is why reuse fails for the simple analysis, usedcf. Fig. 2.

TABLE 1

Conjectures Proved and Lemmata Speculated by Reuse of φ_0

φ_0	$\sum k + \sum l \equiv \sum(k <> l)$	φ_{24}
Φ	$F^1(G^1(x), G^2(y)) \equiv G^3(H^1(x, y))$	
No.	Conjectures proved by reuse	Subgoals
φ_1	$\prod k \times \prod l \equiv \prod(k <> l)$	φ_{17}
φ_2	$\|k <> l\| \equiv \|l <> k\|$	φ_{12}
φ_3	$\|\mathsf{rev}(k)\| \equiv \|k\|$	φ_{13}
φ_4	$\mathsf{rev}(\mathsf{rev}(k)) \equiv k$	φ_{14}
φ_5	$\mathsf{rev}(l) <> \mathsf{rev}(k) \equiv \mathsf{rev}(k <> l)$	$\sigma_5(\varphi_{11})$
φ_6	$\mathsf{max}(\mathsf{maxl}(k), \mathsf{maxl}(l)) \equiv \mathsf{maxl}(k <> l)$	φ_{28}
φ_7	$\mathsf{min}(\mathsf{minl}(\mathsf{n}, k), \mathsf{minl}(\mathsf{n}, l)) \equiv \mathsf{minl}(\mathsf{n}, k <> l)$	φ_{29}
φ_8	$\mathsf{plusl}(\mathsf{m}, k) <> \mathsf{plusl}(\mathsf{m}, l) \equiv \mathsf{plusl}(\mathsf{m}, k <> l)$	—
φ_9	$\|k\| + \|l\| \equiv \|k <> l\|$	—
φ_{10}	$\mathsf{ncut}(m, \mathsf{ncut}(n, \mathsf{k})) \equiv \mathsf{ncut}(\mathsf{plus}(m, n), \mathsf{k})$	—
φ_{11}	$k <> (l <> \mathsf{p}) \equiv (k <> l) <> \mathsf{p}$	—
φ_{12}	$\|k <> n :: \mathsf{l}\| \equiv \mathsf{s}(\|k <> \mathsf{l}\|)$	—
φ_{13}	$\|k <> n :: \epsilon\| \equiv \mathsf{s}(\|k\|)$	—
φ_{14}	$\mathsf{rev}(k <> n :: \epsilon) \equiv n :: \mathsf{rev}(k)$	—
φ_{15}	$(\mathsf{i}^{\mathsf{n}})^m \equiv \mathsf{i}^{m \times \mathsf{n}}$	$[\varphi_{16}]$
φ_{16}	$\mathsf{i}^m \times \mathsf{i}^n \equiv \mathsf{i}^{m+n}$	φ_{17}
φ_{17}	$m \times (\mathsf{n} \times \mathsf{i}) \equiv (m \times \mathsf{n}) \times \mathsf{i}$	$[\varphi_{18}]$
φ_{18}	$m \times \mathsf{i} + n \times \mathsf{i} \equiv (m + n) \times \mathsf{i}$	φ_{24}
φ_{19}	$m \times \mathsf{i} + n \times \mathsf{i} \equiv \mathsf{i} \times (m + n)$	$[\varphi_{21}, \varphi_{24}]$
φ_{20}	$m \times \mathsf{n} \equiv \mathsf{n} \times m$	$[\varphi_{21}]$
φ_{21}	$m \times \mathsf{s}(\mathsf{n}) \equiv m + m \times \mathsf{n}$	$[\varphi_{22}]$
φ_{22}	$m + (\mathsf{i} + n) \equiv \mathsf{i} + (m + n)$	φ_{25}
φ_{23}	$m + n \equiv n + m$	φ_{25}
φ_{24}	$m + (n + \mathsf{i}) \equiv (m + n) + \mathsf{i}$	—
φ_{25}	$m + \mathsf{s}(n) \equiv \mathsf{s}(m + n)$	—
φ_{26}	$\mathsf{or}(\mathsf{mem}(\mathsf{m}, k), \mathsf{mem}(\mathsf{m}, l)) \equiv \mathsf{mem}(\mathsf{m}, k <> l)$	φ_{30}
φ_{27}	$\mathsf{r}(\mathsf{m}, k) <> \mathsf{rm}(\mathsf{m}, l) \equiv \mathsf{rm}(\mathsf{m}, k <> l)$	φ_{31}

Note. Conjectures φ_{28}, ..., φ_{31} cannot be proved by reusing the proof of φ_0. $\varphi_{28} := \mathsf{max}(m, \mathsf{max}(n, i)) \equiv \mathsf{max}(\mathsf{max}(m, n), i)$; $\varphi_{29} := \mathsf{min}(m, \mathsf{min}(n, i)) \equiv \mathsf{min}(\mathsf{min}(m, n), i)$; $\varphi_{30} := \mathsf{or}(\mathsf{or}(\mathsf{eq}(m, n), a), b) \equiv \mathsf{or}(\mathsf{eq}(m, n), \mathsf{or}(a, b))$; $\varphi_{31} := \mathsf{if}(\mathsf{eq}(m, n), k, n :: l) <> p \equiv \mathsf{if}(\mathsf{eq}(m, n), k <> p, n :: (l <> p))$.

convention w.r.t. variable names that i, j, n, m denote numbers, k, l, p, q denote lists, and x, y, z are variables of any sort.

Table 1 illustrates a typical session with the PLAGIATOR system: At the beginning of the session the human advisor submits statement φ_0 (in the first row) and a proof p of φ_0 to the system. Then the statements φ_1, φ_2, ... are presented to the PLAGIATOR, which proves the step formula for each statement only by reuse of p such that no user interactions are required. The third column shows the subgoals speculated by the system when proving a statement by reuse, i.e., the proof obligations which are returned for solving the schematic catch. Here "$-$" denotes that all proof obligations are simplified to tautologies by evaluation (i.e. the statement is

proved by reuse only), and "[...]" denotes that heuristics different from the heuristics given in [KW94, KW95b] are used.

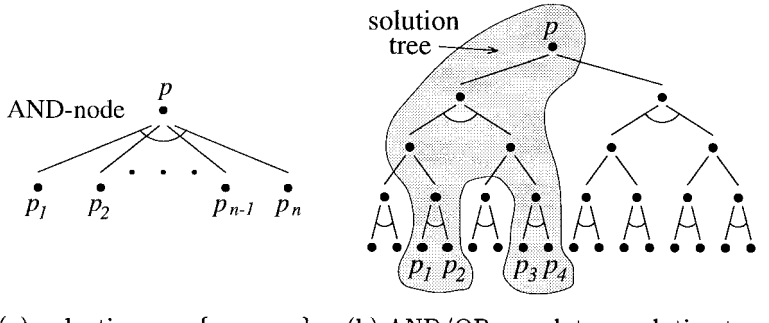
E.g., statement φ_{16} is speculated when verifying φ_{15}, which leads to speculating φ_{17}, which in turn entails speculation of conjecture φ_{18}, for which eventually φ_{24} is speculated. For φ_{11} an *instance* of conjecture φ_{21} is speculated, viz. the formula $\sigma_5(\varphi_{11})$ with $\sigma_5 = \{p/m :: \epsilon\}$.

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.

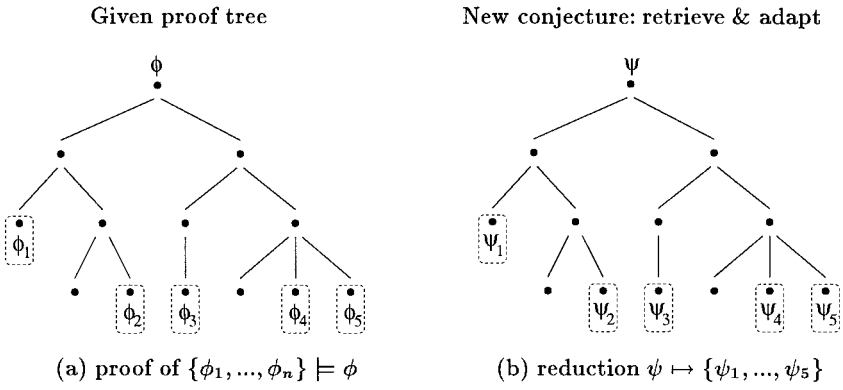


FIG. 6. Proof reuse as problem reduction.

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 simplified[7] 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 φ 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 σ. A conjecture is *irreducible* iff it is primitive or no reduction operator is applicable; i.e., no proof volume PV applies for φ. In the latter case, φ must be proved *directly* (by some human advisor or a machine), whereas in the first case φ is trivially "solvable."

In order to prevent infinite reuse sequences, we demand $\varphi >_F \varphi_i$ for each conjecture φ and each *reducible* member φ_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, $\varphi >_F \varphi_i$ is not required for guaranteeing termination if φ_i is irreducible. Thus, e.g., proving $\varphi_9 := |k| + |l| \equiv |k <> l|$ by reuse terminates vacuously as all formulas from the totally instantiated catch $\pi_2(\rho_2(C'_s))$ are instances of axioms, cf. Section 3. But when proving, e.g., φ_{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 $\psi := \mathsf{plus}(x, \mathsf{s}(x)) \equiv \mathsf{s}(\mathsf{plus}(x, x))$. The proof volume PV' containing the proof shell PS'_s from Fig. 4 applies for ψ via the second-order matcher[8] $\pi = \{F^1/\mathsf{plus}(w_1, \mathsf{s}(w_1)),\ G^1/w_1,\ G^3/\mathsf{s}(w_1),\ H^1/\mathsf{plus}(w_1, w_1), D^1/\mathsf{s}(w_2)\}$. Using the

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

[8] Special *argument variables* $w_1, w_2, ... \notin \mathcal{V}$ denote formal parameters in functional terms replacing function variables in second-order matchers, e.g., $D^1(n, x) = \mathsf{s}(x)$.

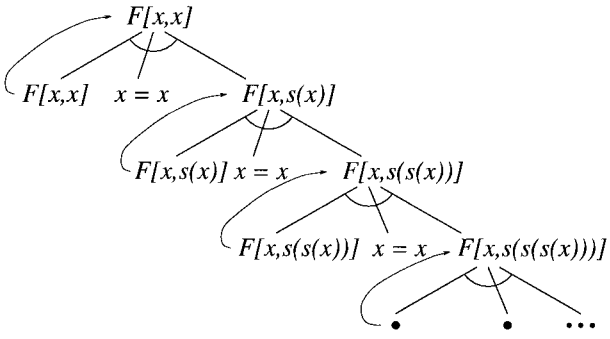


FIG. 7. A non-terminating reuse attempt.

solution $\rho = \{F^2/\mathsf{s}(w_2),\ D^4/\mathsf{s}(\mathsf{s}(w_2)),\ F^3/\mathsf{s}(\mathsf{s}(w_2))\}$ for the free function variables in the adaption step, the totally instantiated catch $\rho(\pi(C'_s))$ is computed as

$$(15) \qquad \mathsf{s}(x) \equiv \mathsf{s}(x)$$

$$(16) \qquad \mathsf{plus}(\mathsf{s}(x), \mathsf{s}(x)) \equiv \mathsf{s}(\mathsf{s}(\mathsf{plus}(x, x)))$$

$$(17) \qquad \mathsf{s}(\mathsf{s}(\mathsf{s}(x))) \equiv \mathsf{s}(\mathsf{s}(\mathsf{s}(x)))$$

$$(18) \qquad \mathsf{plus}(\mathsf{s}(x), \mathsf{s}(\mathsf{s}(x))) \equiv \mathsf{s}(\mathsf{s}(\mathsf{plus}(x, \mathsf{s}(x))))$$

Hence nonprimitive conjectures $\psi_1 := \mathsf{plus}(x, \mathsf{s}(x)) \equiv \mathsf{s}(\mathsf{plus}(x, x))$ and $\psi_2 := \mathsf{plus}(x, \mathsf{s}(\mathsf{s}(x))) \equiv \mathsf{s}(\mathsf{plus}(x, \mathsf{s}(x)))$ 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., cf. Fig. 7. Generally, PV' is applicable to all conjectures $\psi_{n+1} := \mathsf{plus}(x, \mathsf{s}^{n+1}(x)) \equiv \mathsf{s}(\mathsf{plus}(x, \mathsf{s}^n(x)))$ for $n \in \mathbb{N}$, and using ρ as the solution substitution then yields the nonprimitive conjectures ψ_{n+1} and ψ_{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 $>_F$ on the set of formulas:[9]

5.1. An Order on (Sets of) Symbols

We start by separating function symbols from the signature Σ into the set Σ^c of constructor function symbols, as 0, s, empty, add, etc., and the set Σ^d of defined function symbols, e.g., exp, prod, times, sum, plus, etc. Then the *defined-by relation* $>_{def}$ is a relation on Σ^d defined by:

[9] 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.

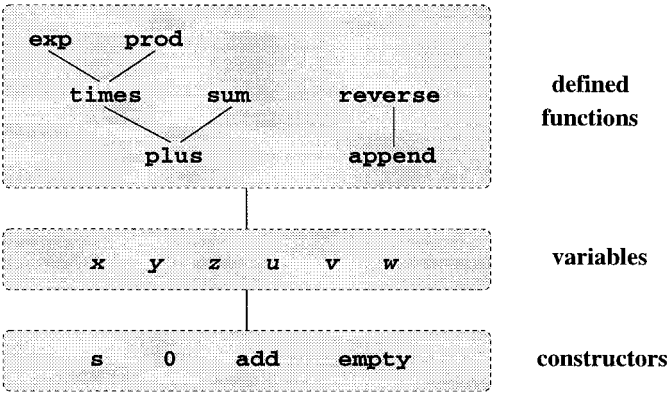


FIG. 8. Order on (sets of) symbols.

DEFINITION 1 (Defined-by relation $>_{def}$).

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

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

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

We extend $>_{def}$ to a *quasi-ordering* $\gtrsim$ on $2^{\Sigma \cup \mathscr{V}}$:

DEFINITION 2 (Multiset order $\gg_{def}$, quasi-ordering $\gtrsim$). Let $\gg_{def}$ be the strict multiset order imposed by $>_{def}$ on the multisets of Σ^d. Then for finite sets $S_1, S_2 \subseteq \Sigma \cup \mathscr{V}$, we define $S_1 \gtrsim S_2$ iff one of the following cases apply:[10]

(1) $S_1, S_2 \subseteq \Sigma^d$ and either $S_1 = S_2$ or $S_1 \gg_{def} S_2$

(2) $S_1 \subseteq \Sigma^d$, $S_2 \subseteq \mathscr{V}$

(3) $S_1, S_2 \subseteq \mathscr{V}$ and $|S_1| \geqslant |S_2|$

(4) $S_1 \subseteq \mathscr{V}$, $S_2 \subseteq \Sigma^c$

(5) $S_1, S_2 \subseteq \Sigma^c$

(6) $S_1 \gtrsim S' \gtrsim S_2$ for some $S' \subseteq \Sigma \cup \mathscr{V}$.

The strict part $\succ$ of $\gtrsim$ is defined as $\succ := \gtrsim \setminus \approx$, where $\approx$ is the equivalence relation $\gtrsim \cap \lesssim$ induced by $\gtrsim$ on $2^{\Sigma \cup \mathscr{V}}$.

$\gg_{def}$ is well-founded since $>_{def}$ is, cf. [DM79]. We find, e.g., {exp, prod, sum} $\succ$ {times, sum} $\succ$ {times, plus} $\succ$ {plus} $\succ \{x, y\} \approx \{u, v\} \succ \{z\} \succ$ {add, s}, and thus {exp, prod, sum} $\succ$ {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* $S_1, S_2 \subseteq \Sigma \cup \mathcal{V}$ *be finite. Then* $S_1 \approx S_2$ *iff either* $S_1, S_2 \subseteq \Sigma^d$ *and* $S_1 = S_2$, $S_1, S_2 \subseteq \mathcal{V}$ *and* $|S_1| = |S_2|$, *or* $S_1, S_2 \subseteq \Sigma^c$.

Proof. The *if* part is trivial and we verify the *only-if* part: Let $S_1 \gtrsim S_2 \gtrsim S_1$. Then by definition of $\gtrsim$, either $S_1, S_2 \subseteq \Sigma^d$, $S_1, S_2 \subseteq \mathcal{V}$, or $S_1, S_2 \subseteq \Sigma^c$. If $S_1, S_2 \subseteq \Sigma^d$ then $S_1 = S_2$, because $S_1 \gg_{def} S_2 \gg_{def} S_1$ would contradict the well-foundedness of $\gg_{def}$ otherwise. If $S_1, S_2 \subseteq \mathcal{V}$, then $|S_1| \geqslant |S_2| \geqslant |S_1|$, and therefore $|S_1| = |S_2|$. Otherwise $S_1, S_2 \subseteq \Sigma^c$. ∎

THEOREM 4 (Well-founded order on sets of symbols). $\succ$ *is well-founded on finite subsets of* $\Sigma \cup \mathcal{V}$.

Proof. Assume by way of contradiction that $\langle S_i \rangle_{i \in \mathbb{N}}$ is a sequence of finite sets with $S_i \succ S_{i+1}$. Then $S_i \not\subseteq \Sigma^c$ for all $i \in \mathbb{N}$, because all finite subsets of Σ^c are $\succ$-minimal, cf. Lemma 5.3. If $S_i \subseteq \mathcal{V}$ for some $i \in \mathbb{N}$, then $S_{i+j} \subseteq \mathcal{V}$ for all $j \in \mathbb{N}$ by definition of $\succ$ (and because of $S_i \not\subseteq \Sigma^c$ 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 $(\mathbb{N}, >)$. Consequently $S_i \subseteq \Sigma^d$ for all $i \in \mathbb{N}$ and $S_1 \gg_{def} S_2 \gg_{def} S_3 \gg_{def} \ldots$ contradicting the well-foundedness of $\gg_{def}$. Thus there is no infinite sequence of $\succ$-decreasing finite subsets of $\Sigma \cup \mathcal{V}$ and $\succ$ is well-founded. ∎

5.2. An Order on Formulas

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

For realizing this idea, we consider sets of (defined) *maximal sybols* (w.r.t. $>_{def}$), since their occurrences have a substantial influence on the difficulty of a proof:

DEFINITION 5 (Pure sets, maximal pure subset, $purify_{\succ}$). A finite subset $S \subseteq \Sigma \cup \mathcal{V}$ is called *pure* iff

(1) $S \subseteq \Sigma^d$ and $s_1 \not>_{def} s_2$ for all $s_1, s_2 \in S$,

(2) $S \subseteq \mathcal{V}$, or

(3) $S \subseteq \Sigma^c$.

We let $purify_{\succ} S$ denote the *maximal pure subset* of $S \subseteq \Sigma \cup \mathcal{V}$, i.e.

(1) $purify_{\succ} S$ is the set of $>_{def}$-maximal elements of $S \cap \Sigma^d$ if $S \cap \Sigma^d \neq \emptyset$,

(2) $purify_{\succ} S = S \cap \mathcal{V}$ if $S \cap \Sigma^d = \emptyset$ and $S \cap \mathcal{V} \neq \emptyset$,

(3) $purify_{\succ} S = S$ otherwise. ∎

For instance, $\{\mathsf{exp}, \mathsf{prod}, \mathsf{sum}\}$ and $\{x, y\}$ are pure whereas $\{\mathsf{exp}, \mathsf{times}, \mathsf{plus}\}$ as well as $\{\mathsf{prod}, \mathsf{sum}, x, \mathsf{s}\}$ are not. Furthermore, e.g., $purify_{\succ}\{x, y, \mathsf{s}\} = \{x, y\}$ and $purify_{\succ}\{\mathsf{exp}, \mathsf{times}, \mathsf{plus}, \mathsf{prod}, \mathsf{sum}, x, \mathsf{s}\} = \{\mathsf{exp}, \mathsf{prod}, \mathsf{sum}\}$.

We let $\mathcal{S}(\phi) \subseteq \Sigma \cup \mathcal{V}$ denote the set of all function and variable symbols in a (set of) term(s) or formula(s) ϕ. Thus if a formula φ contains at least one defined

function symbol, then $purify_{\succ}\,\mathscr{S}(\varphi)$ is the set of all maximal defined function symbols occurring in φ, with which the difficulty of (proving) φ is estimated.

Using $purify_{\succ}$ and $\succ$, a relation $>_F$ on formulas now can be defined, where we use the number of occurrences $\#_f(\phi) \in \mathbb{N}$ of a symbol $f \in \Sigma \cup \mathscr{V}$ in a (set of) term(s) or formula(s) ϕ:

DEFINITION 6 (Order $>_F$ on formulas).

$$\varphi >_F \psi \text{ iff (a) } purify_{\succ}\,\mathscr{S}(\varphi) \succ purify_{\succ}\,\mathscr{S}(\psi),$$

$$\text{or (b) } purify_{\succ}\,\mathscr{S}(\varphi) \approx purify_{\succ}\,\mathscr{S}(\psi) \text{ and}$$

$$\textstyle\sum_{f \in purify_{\succ}\mathscr{S}(\varphi)} \#_f(\varphi) > \sum_{f \in purify_{\succ}\mathscr{S}(\psi)} \#_f(\psi). \quad \blacksquare$$

$>_F$ 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 φ about some function f, quite inevitably also properties of functions g used for defining f (i.e., $f >_{def} g$) have to be considered, and this is *independent of whether g already occurs in φ 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 $\gg_{def}$ (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 φ containing one occurrence of maxl as well as rev and for ψ containing only two occurrences of maxl, criterion (a) succeeds with $purify_{\succ}\,\mathscr{S}(\varphi) = \{\mathsf{maxl},\ \mathsf{rev}\} \gg_{def} \{\mathsf{maxl}\} = purify_{\succ}\,\mathscr{S}(\psi)$ while the combined criterion would fail due to $\underline{\{}\mathsf{maxl}, \mathsf{rev}\underline{\}} \not\gg_{def} \underline{\{}\mathsf{maxl}, \mathsf{maxl}\underline{\}}$, where $\underline{\{}\ldots\underline{\}}$ denotes a multiset.

For an example of using $>_F$, reconsider Table 3. Proving φ_1 by reuse leads to speculating the lemmata φ_{17}, φ_{18}, and φ_{24} in turn. We find $purify_{\succ}\,\mathscr{S}(\varphi_1) = \{\mathsf{prod}, \mathsf{app}\} \succ \{\mathsf{times}\} = purify_{\succ}\,\mathscr{S}(\varphi_{17}) = purify_{\succ}\,\mathscr{S}(\varphi_{18}) \succ \{\mathsf{plus}\} = purify_{\succ}\,\mathscr{S}(\varphi_{24})$ and $\#_{\mathsf{times}}(\varphi_{17}) = 4 > 3 = \#_{\mathsf{times}}(\varphi_{18})$, and therefore $\varphi_1 >_F \varphi_{17} >_F \varphi_{18} >_F \varphi_{24}$. Also $\varphi_5 >_F \sigma_5(\varphi_{11})$ because $purify_{\succ}\,\mathscr{S}(\varphi_5) = \{\mathsf{rev}\} \succ \{\mathsf{app}\} = purify_{\succ}\,\mathscr{S}(\sigma_5(\varphi_{11}))$ for the instance $\sigma(\varphi_{11})$ of φ_{11} with $\sigma_5 = \{p/m :: \epsilon\}$ which is speculated when proving φ_5 by reuse.

5.3. The Refined Termination Order

However, the $>_F$-relation is still too weak for our purposes. Consider e.g., conjecture $\varphi_{15} := \mathsf{exp}(\mathsf{exp}(i, n), m) \equiv \mathsf{exp}(i, \mathsf{times}(m, n))$ for which the reuse procedure speculates lemma $\varphi_{16} := \mathsf{times}(\mathsf{exp}(i, m), \mathsf{exp}(i, n)) \equiv \mathsf{exp}(i, \mathsf{plus}(m, n))$, cf. Table 1. Since $purify_{\succ}\,\mathscr{S}(\varphi_{15}) = \{\mathsf{exp}\} = purify_{\succ}\,\mathscr{S}(\varphi_{16})$ and $\#_{\mathsf{exp}}(\varphi_{15}) = 3 = \#_{\mathsf{exp}}(\varphi_{16})$, $\varphi_{15} >_F \varphi_{16}$ does not hold. Also $\varphi_{23} \not>_F \varphi_{25}$ for the conjectures $\varphi_{23} := \mathsf{plus}(m, n) \equiv \mathsf{plus}(n, m)$ and $\varphi_{25} := \mathsf{plus}(m, \mathsf{s}(n)) \equiv \mathsf{s}(\mathsf{plus}(m, n))$ from Table 1, because $purify_{\succ}\,\mathscr{S}(\varphi_{23}) = \{\mathsf{plus}\} = purify_{\succ}\,\mathscr{S}(\varphi_{25})$ and $\#_{\mathsf{plus}}(\varphi_{23}) = 2 = \#_{\mathsf{plus}}(\varphi_{25})$.

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 $\mathsf{exp}(m, \mathsf{0}) \equiv \mathsf{s}(\mathsf{0})$ and $\mathsf{exp}(m, \mathsf{s}(n)) \equiv \mathsf{times}(m, \mathsf{exp}(m, n))$. We observe that the symbol times occurs in the second argument of exp in φ_{15}, while only the $>_{def}$-smaller function symbol plus and the variables m, n occur in the second arguments of exp in φ_{16}.

Based on this observation, we refine $>_F$ by an additional requirement which also considers the *arguments* of $>_{def}$-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* $R_f \subseteq \{x_1, ..., x_n\}$ with each term $f(x_1, ..., x_n)$, where $x_1, ..., x_n$ 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 $f(x_1, ..., x_n)$ is to be proved by induction. We let $\Pi_f \subseteq \{1, ..., n\}$ 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 $\mathsf{square}(x)$ defined as $\mathsf{times}(x, x)$ is excluded and therefore $\Pi_f \neq \emptyset$ if $f \in \Sigma^d$. Now the set $rst_{f,i}[t]$ of subterms of a term t which occupy the position of a recursion variable x_i of a function symbol f is computed as:

$$
\begin{aligned}
&(1) \quad rst_{f,i}[x] &&= \emptyset, && \text{for all} \quad x \in \mathscr{V},\\
&(2) \quad rst_{f,i}[f(t_1, ..., t_n)] &&= \{t_i\},^{11}\\
&(3) \quad rst_{f,i}[g(t_1, ..., t_m)] &&= rst_{f,i}[t_1] \cup \cdots \cup rst_{f,i}[t_m], && \text{for all} \quad g \neq f.
\end{aligned}
$$

$rst_{f,i}$ is extended to formulas φ by (3), where g is $\equiv$ or any connective symbol, e.g. $rst_{\mathsf{exp},2}[\varphi_{15}] = \{m, \mathsf{times}(m, n)\}$ and $rst_{\mathsf{exp},2}[\varphi_{16}] = \{m, n, \mathsf{plus}(m, n)\}$. If we compare the maximal symbols times resp. plus of the recursion arguments in this example, the formulas φ_{15} and φ_{16} now can be related, and the same holds for $rst_{\mathsf{plus},1}[\varphi_{23}] = \{m, n\} \succ \{m\} = rst_{\mathsf{plus},1}[\varphi_{25}]$. The latter comparison explains the treatment of (sets of) variables in our approach by the order $\succ$, 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 $purify_{\succ}\mathscr{S}(\varphi)$ 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 $\succcurlyeq$ imposed by $\succsim$ on multisets of finite subsets of $\Sigma \cup \mathscr{V}$, cf. [Der87], where e.g. $\underline{\{\{x, y\}, \{\mathsf{times}, \mathsf{sum}\}, \{x, y\}\}} \succcurlyeq \underline{\{\{u, v\}, \{\mathsf{plus}\}, \{x\}\}}$. For incorporation of recursion arguments, $>_F$ now is redefined:

DEFINITION 7 (Refined order $>_F$ on formulas). Let φ, ψ be formulas with $\mathscr{S}(\varphi) \cap \Sigma^d \neq \emptyset$ and $\mathscr{S}(\psi) \cap \Sigma^d \neq \emptyset$, let

[11] Note that only the *outermost* occurrences of (maximal symbols) f are considered, i.e., we do *not* stipulate $rst_{f,i}[f(t_1, ..., t_n)] = \{t_i\} \cup rst_{f,i}[t_1] \cup \cdots \cup rst_{f,i}[t_n]$.

(a) $\varphi \geqslant_1 \psi$ iff $purify_{\succ}\mathscr{S}(\varphi) \gtrsim purify_{\succ}\mathscr{S}(\psi)$,

(b) $\varphi \geqslant_2 \psi$ iff $\sum_{f \in purify_{\succ}\mathscr{S}(\varphi)} \#_f(\varphi) \geqslant \sum_{f \in purify_{\succ}\mathscr{S}(\psi)} \#_f(\psi)$,

(c) $\varphi \geqslant_3 \psi$ iff $\mathscr{M}_{rst}[\varphi] \gtrsim\!\!\!> \mathscr{M}_{rst}[\psi]$, where

$$\mathscr{M}_{rst}[\phi] := \underline{\{}\ purify_{\succ}\mathscr{S}(rst_{f,i}[\phi]) \mid f \in purify_{\succ}\mathscr{S}(\phi),\ i \in \Pi_f \underline{\}},$$

(d) $\varphi \geqslant_4 \psi$ iff $\varphi = \sigma(\psi)$ for some (first-order) substitution σ, let

$\equiv_i := \geqslant_i \cap \leqslant_i$

and let $>_i := \geqslant_i \backslash \equiv_i$ for each $i \in \{1, 2, 3, 4\}$.

Then $\varphi >_F \psi$ iff $\varphi \equiv_j \psi$ and $\varphi >_k \psi$ for some $k \in \{1, 2, 3, 4\}$ and all $j \in \{1, ..., k-1\}$.

Since Definition 5.7 demands that φ and ψ both contain one defined function symbol at least, we have $purify_{\succ}\mathscr{S}(\varphi) \subseteq \Sigma^d$. Therefore Π_f in (c) is defined and consequently $\geqslant_3$ is well-defined. Requirement (c) of Definition 5.7 incorporates the inspection of recursion arguments as demanded. By requirement (d), a pair of conjectures φ and ψ can also be related if ψ is *strictly more general* than φ. 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 $>_F$). *$>_F$ is well-founded.*

Proof. $>_1$ is well-founded by Theorem 5.4, and the well-foundedness of $>_2$ is obvious. $>_3$ is well-founded as the strict part $\gtrsim\!\!\!>$ of $\gtrsim$ is well-founded by Theorem 5.4 and [Der87]. $>_4$ is the strict subsumption order $\rhd$ on formulas which is also well-founded, cf. [DJ90]. Since $>_F$ is formed as a lexicographic combination of quasi-orderings whose strict parts are well-founded, $>_F$ is also well-founded. ▌

By Corollary 8 the reuse procedure terminates if we demand the *termination requirement for reuse*, viz. $\varphi >_F \varphi_i$ for each *reducible* member φ_i of a simplified totally instantiated catch which is computed when proving φ by reuse.

6. USEFULNESS OF $>_F$

The usefulness of $>_F$ is illustrated by Table 2. Here all pairs φ, φ' from Table 1 are compared by $>_F$, where φ' is speculated by the PLAGIATOR system as a lemma when the conjecture φ 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 $>_F$ 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 $>_F$ 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

TABLE 2

Termination of Reuse for the Speculated Lemmata of Table 1

Conjecture	Lemma	(a)	(b)	(c)
φ_1	φ_{17}	$\{\prod, <>\} \succ \{\times\}$		
φ_2	φ_{12}	$\{<>, \lvert.\rvert\} = \{<>, \lvert.\rvert\}$	$4 = 4$	$\{k, l\} \succ \{k\}$
φ_3	φ_{13}	$\{\mathsf{rev}, \lvert.\rvert\} \succ \{<>, \lvert.\rvert\}$		
φ_4	φ_{14}	$\{\mathsf{rev}\} = \{\mathsf{rev}\}$	$2 = 2$	$\{\mathsf{rev}\} \succ \{k\}$
φ_5	$\sigma_5(\varphi_{11})$	$\{\mathsf{rev}\} \succ \{<>\}$		
φ_6	φ_{28}	$\{\mathsf{maxl}\} \succ \{\mathsf{max}\}$		
φ_7	φ_{29}	$\{\mathsf{minl}\} \succ \{\mathsf{min}\}$		
φ_{15}	φ_{16}	$\{\mathsf{exp}\} = \{\mathsf{exp}\}$	$3 = 3$	$\{\times\} \succ \{+\}$
φ_{16}	φ_{17}	$\{\mathsf{exp}\} \succ \{\times\}$		
φ_{17}	φ_{16}	$\{\times\} = \{\times\}$	$4 > 3$	
φ_{18}	φ_{24}	$\{\times\} \succ \{+\}$		
φ_{19}	φ_{21}	$\{\times\} = \{\times\}$	$3 > 2$	
φ_{19}	φ_{24}	$\{\times\} \succ \{+\}$		
φ_{20}	φ_{21}	$\{\times\} = \{\times\}$	$2 = 2$	$\{m, n\} \succ \{m\}$
φ_{21}	φ_{22}	$\{\times\} \succ \{+\}$		
φ_{22}	φ_{25}	$\{+\} = \{+\}$	$4 > 2$	
φ_{23}	φ_{25}	$\{+\} = \{+\}$	$2 = 2$	$\{m, n\} \succ \{m\}$
φ_{26}	φ_{30}	$\{\mathsf{mem}, <>\} \succ \{\mathsf{or}, \mathsf{eq}\}$		
φ_{27}	φ_{31}	$\{\mathsf{rm}, <>\} \succ \{\mathsf{if}, \mathsf{eq}, <>\}$		

reveals that by the termination requirement unsuccessful reuse attempts can be avoided.

However, since our claim of the usefulness of $>_F$ 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 $>_F$ by examples for lemma speculation in induction theorem proving borrowed from [IB96]. There 50 theorems $T1, ..., T50$ are given which can be proved by 24 speculated lemmata $L1, ..., L24$ (and 12 generalizations).[12] The defined functions are

$$\mathsf{dbl}, \mathsf{half}, \mathsf{even}, \mathsf{len}, \mathsf{nth}, \mathsf{qrev}, \mathsf{cnt}, \mathsf{mem}, \mathsf{ordered}$$
$$\mathsf{rev} >_{def} \mathsf{app}, \mathsf{rotate} >_{def} \mathsf{app}, \mathsf{isort} >_{def} \mathsf{ins}$$

The theorem–lemma pairs are presented in Table 3, where theorems and lemmata are grouped together, e.g., $T8$ uses $L4$ and $L5$, while $T10$, $T17$, and $T19$ use $L8$. Column $>_F$ in Table 3 denotes the criterion of Definition 7 which is satisfied for the particular theorem–lemma pair, i.e., we obtain, e.g., $T8 >_F L4$ and $T8 >_F L5$ by criterion (b).[13]

For all examples presented in Table 3, each lemma φ' speculated in a proof of φ is $<_F$-smaller than φ. This observation gives additional evidence for the usefulness of

[12] Theorems $T36$–$T47$ do not use lemmata at all. Theorems $T27$–$T35$ are proved by *generalization* as lemma speculation and thus are not considered here; see below.

[13] For $T12$ only Lemma $L11$ is speculated, hence "$-$" in the last column.

TABLE 3

Theorems and Lemmata from [IB96]

No.	Theorem resp. Lemma	$>_F$
$T1$	$\mathsf{dbl}(X) \equiv X + X$	a
$T3$	$\mathsf{len}(X <> Y) \equiv \mathsf{len}(Y) + \mathsf{len}(X)$	a
$T7$	$\mathsf{len}(\mathsf{qrev}(X, Y)) \equiv \mathsf{len}(X) + \mathsf{len}(Y)$	a
$T13$	$\mathsf{half}(X + X) \equiv X$	a
$T16$	$\mathsf{even}(X + X)$	a
$L1$	$\mathsf{plus}(x, \mathsf{s}(y)) \equiv \mathsf{s}(\mathsf{plus}(x, y))$	
$T2$	$\mathsf{len}(\mathsf{app}(x, y)) \equiv \mathsf{len}(\mathsf{app}(y, x))$	c
$T4$	$\mathsf{len}(X <> X) \equiv \mathsf{dbl}(\mathsf{len}(X))$	a
$T6$	$\mathsf{len}(\mathsf{rev}(X <> Y)) \equiv \mathsf{len}(X) + \mathsf{len}(Y)$	a
$T20$	$\mathsf{even}(\mathsf{len}(X <> X))$	a
$L2$	$\mathsf{len}(\mathsf{app}(x, y :: z) \equiv \mathsf{s}(\mathsf{len}(\mathsf{app}(x, z)))$	
$T5$	$\mathsf{len}(\mathsf{rev}(X)) \equiv \mathsf{len}(X)$	a
$L3$	$\mathsf{len}(X <> Y :: \mathsf{empty}) \equiv \mathsf{s}(\mathsf{len}(X))$	
$T8$	$\mathsf{nth}(x, \mathsf{nth}(y, z)) \equiv \mathsf{nth}(y, \mathsf{nth}(x, z))$	c, c
$L4$	$\mathsf{nth}(\mathsf{s}(w), \mathsf{nth}(x, y :: z)) \equiv \mathsf{nth}(w, \mathsf{nth}(x, z))$	
$L5$	$\mathsf{nth}(\mathsf{s}(v), \mathsf{nth}(\mathsf{s}(w), x :: y :: z)) \equiv \mathsf{nth}(\mathsf{s}(v), \mathsf{nth}(w, x :: z))$	
$T9$	$\mathsf{nth}(w, \mathsf{nth}(x, \mathsf{nth}(y, z))) \equiv \mathsf{nth}(y, \mathsf{nth}(x, \mathsf{nth}(w, z)))$	c, c
$L6$	$\mathsf{nth}(\mathsf{s}(v), \mathsf{nth}(w, \mathsf{nth}(x, y :: z))) \equiv \mathsf{nth}(v, \mathsf{nth}(w, \mathsf{nth}(x, z)))$	
$L7$	$\mathsf{nth}(\mathsf{s}(u), \mathsf{nth}(v, \mathsf{nth}(\mathsf{s}(w), x :: y :: z))) \equiv \mathsf{nth}(\mathsf{s}(u), \mathsf{nth}(v, \mathsf{nth}(w, x :: z)))$	
$T10$	$\mathsf{rev}(\mathsf{rev}(x)) \equiv x$	c
$T17$	$\mathsf{rev}(\mathsf{rev}(\mathsf{app}(x, y))) \equiv \mathsf{app}(\mathsf{rev}(\mathsf{rev}(x)), \mathsf{rev}(\mathsf{rev}(y)))$	b
$T19$	$\mathsf{app}(\mathsf{rev}(\mathsf{rev}(x)), y) \equiv \mathsf{rev}(\mathsf{rev}(\mathsf{app}(x, y)))$	b
$L8$	$\mathsf{rev}(\mathsf{app}(x, y :: \mathsf{empty})) \equiv y :: \mathsf{rev}(x))$	
$T11$	$\mathsf{rev}(\mathsf{app}(\mathsf{rev}(x), \mathsf{rev}(y))) \equiv \mathsf{app}(y, x)$	b, b
$L9$	$\mathsf{rev}(\mathsf{app}(x, \mathsf{app}(y, z :: \mathsf{empty}))) \equiv z :: \mathsf{rev}(\mathsf{app}(x, y))$	
$L10$	$\mathsf{rev}(\mathsf{app}(\mathsf{app}(x, y :: \mathsf{empty}), \mathsf{empty})) \equiv y :: \mathsf{rev}(x, \mathsf{empty})$	
$T12$	$\mathsf{qrev}(X, Y) \equiv \mathsf{rev}(X) <> Y$	$a, -$
$T18$	$\mathsf{rev}(\mathsf{rev}(X) <> Y) \equiv \mathsf{rev}(Y) <> X$	a, a
$T21$	$\mathsf{rotate}(\mathsf{len}(X), X <> Y) \equiv Y <> X$	a, a
$L11$	$(X <> (Y :: \mathsf{empty})) <> Z \equiv X <> (Y :: Z)$	
$L13$	$(X <> Y) <> Z :: \mathsf{empty} \equiv X <> (Y <> Z :: \mathsf{empty})$	
$T14$	$\mathsf{ordered}(\mathsf{isort}(X))$	a
$L12$	$\mathsf{ordered}(Y) \rightarrow \mathsf{ordered}(\mathsf{ins}(X, Y))$	
$T22$	$\mathsf{even}(\mathsf{len}(\mathsf{app}(x, y))) \leftrightarrow \mathsf{even}(\mathsf{len}(\mathsf{app}(y, x)))$	c
$L14$	$\mathsf{even}(\mathsf{len}(\mathsf{app}(w, z))) \leftrightarrow \mathsf{even}(\mathsf{len}(\mathsf{app}(w, x :: y :: z)))$	
$T23$	$\mathsf{half}(\mathsf{len}(X <> Y)) \equiv \mathsf{half}(\mathsf{len}(Y <> X))$	a
$L15$	$\mathsf{len}(W <> X :: Y :: Z) \equiv \mathsf{s}(\mathsf{s}(\mathsf{len}(W <> Z)))$	
$T24$	$\mathsf{even}(\mathsf{plus}(x, y)) \leftrightarrow \mathsf{even}(\mathsf{plus}(y, x))$	c
$T25$	$\mathsf{even}(\mathsf{len}(X <> Y)) \leftrightarrow \mathsf{even}(\mathsf{len}(Y) + \mathsf{len}(X))$	a
$L16$	$\mathsf{even}(\mathsf{plus}(x, y)) \leftrightarrow \mathsf{even}(\mathsf{plus}(x, \mathsf{s}(\mathsf{s}(y))))$	
$T26$	$\mathsf{half}(X + Y) \equiv \mathsf{half}(Y + X)$	a
$L17$	$X + \mathsf{s}(\mathsf{s}(Y)) \equiv \mathsf{s}(\mathsf{s}(X + Y))$	
$T48$	$\mathsf{len}(\mathsf{isort}(X)) \equiv \mathsf{len}(X)$	a
$L18$	$\mathsf{len}(\mathsf{ins}(X, Y)) \equiv \mathsf{s}(\mathsf{len}(Y))$	
$T49$	$X \in \mathsf{isort}(Y) \rightarrow X \in Y$	a
$L19$	$X \neq Y \rightarrow (X \in \mathsf{ins}(Y, Z) \rightarrow X \in Z)$	
$T50$	$\mathsf{cnt}(X, \mathsf{isort}(Y)) \equiv \mathsf{cnt}(X, Y)$	a, a
$L20$	$\mathsf{cnt}(X, \mathsf{ins}(X, Y)) \equiv \mathsf{s}(\mathsf{cnt}(X, Y))$	
$L21$	$X \neq Y \rightarrow \mathsf{cnt}(X, \mathsf{ins}(Y, Z)) \equiv \mathsf{cnt}(X, Z)$	

TABLE 4

Generalization by Lemma Speculation

No.	Theorem resp. Lemma	$>_F$
$T15$	$\mathsf{plus}(x, \mathsf{s}(x)) \equiv \mathsf{s}(\mathsf{plus}(x, x))$	d
$L1$	$\mathsf{plus}(x, \mathsf{s}(y)) \equiv \mathsf{s}(\mathsf{plus}(x, y))$	

$>_F$ because all lemmata are $<_F$-smaller than the conjecture under consideration, independent *of how* the used lemma was speculated.

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 $\sqsupset$ such that $\varphi \sqsupset \psi$ for *each* sound generalization ψ of a conjecture φ, because there are non-well-founded generalizations such as $\psi := \varphi \wedge \varphi'$ for some φ', 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

[BKR92] Bouhoula, A., Kounalis, E., and Rusinowitch, M. (1992), SPIKE: An automatic theorem prover, *in* "Proceedings of the Conference on Logic Programming and Automated Reasoning (LPAR-92), St. Petersburg, Russia," Springer-Verlag, Berlin.

[BM79] Boyer, R. S., and Moore, J. Strother (1979), "A Computational Logic," ACM Monograph Series, Academic Press, San Diego.

[Bra94] Brauburger, J. (1994), "PLAGIATOR—Design and Implementation of a Learning Theorem Prover," Diploma Thesis, TH Darmstadt. [in German]

[Der87] Dershowitz, N. (1987), Termination of rewriting, *J. Symbolic Comput.* **3**, 69–115.

[DJ90] Dershowitz, N., and Jouannaud, J.-P. (1990), "Rewrite Systems," Handbook of Theoretical Computer Science: Formal Models and Semantics (Jan van Leeuwen, Ed.), Vol. B, chap. 6, pp. 243–320, Elsevier Science, Amsterdam.

[DM79] Dershowitz, N., and Manna, Z. (1979), Proving termination with multiset orderings, *Commun. Assoc. Comput. Mach.* **22**, 465–476.

[Ell89] Ellman, T. (1989), Explanation-based learning: A survey of programs and perspectives, *ACM Comput. Surv.* **21**, 163–221.

[GW92] Giunchiglia, F., and Walsh, T. (1992), A theory of abstraction, *Artificial Intelligence* **57**, 323–389.

[Hal89] Hall, R. P. (1989), Computational approaches to analogical reasoning: A comparative analysis, *Artificial Intelligence* **39**, 39–120.

[HS96] Hutter, D., and Sengler, C. (1996), INKA: The next generation, *in* "Proceedings of the 13th International Conference on Automated Deduction (CADE-96), New Brunswick, NJ" (M. A. McRobbie and J. K. Slaney, Eds.), Lecture Notes in Artificial Intelligence, Vol. 1104, pp. 288–292, Springer-Verlag, Berlin.

[IB96] Ireland, A., and Bundy, A. (1996), Productive use of failure in inductive proof, *J. Automat. Reasoning*. [Special Issue on Automation of Proofs by Mathematical Induction].

[KB97] Kolbe, T., and Brauburger, J. (1997), PLAGIATOR—A learning prover, *in* "Proceedings of the 14th International Conference on Automated Deduction (CADE-97), Townsville, Australia," Lecture Notes in Artificial Intelligence, Vol. 1249, Springer-Verlag, New York.

[KW94] Kolbe, T., and Walther, C. (1994), Reusing proofs, *in* "Proceedings of the 11th European Conference on Artificial Intelligence (ECAI-94), Amsterdam" (A. Cohn, Ed.), pp. 80–84, Wiley, New York.

[KW95a] Kolbe, T., and Walther, C. (1995), Adaptation of proofs for reuse, *in* "Adaptation of Knowledge for Reuse. Papers from the 1995 AAAI Fall Symposium, Cambridge, MA" (D. W. Aha and A. Ram, Eds.), pp. 61–67, AAAI Press.

[KW95b] Kolbe, T., and Walther, C. (1995), Patching proofs for reuse, *in* "Proceedings of the Euromean Conference on Machine Learning (ECML-95), Heraklion, Greece" (N. Lavrac and S. Wrobel, Eds.), pp. 303–306, Lecture Notes in Artificial Intelligence, Vol. 912.

[KW95c] Kolbe, T., and Walther, C. (1995), Proof management and retrieval, *in* "Proceeddings of the IJCAI'95 Workshop on Formal Approaches to the Reuse of Plans, Proofs, and Programs," pp. 16–20.

[KW95d] Kolbe, T., and Walther, C. (1995), Second-order matching modulo evaluation—A technique for reusing proofs, *in* "Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95), Montreal, Canada," pp. 190–195.

[KW96a] Kolbe, T., and Walther, C. (1996), "On the Benefits of Reusing Past Problem Solutions," Technical Report IBN 96/35, TH Darmstadt.

[KW96b] Kolbe, T., and Walther, C. (1996), Proving theorems by mimicking a human's skill, *in* "Acquisition, Learning & Demonstration: Automating Tasks for Users. Papers from the 1996 AAAI Spring Symposium, Menlo Park, CA" (Y. Gil, Ed.), pp. 50–56, AAAI Press.

[KW96c] Kolbe, T., and Walther, C. (1996), Termination of theorem proving by reuse, *in* "Proceedings of the 13th International Conference on Automated Deduction (CADE-96), New Brunswick, NJ" (M. A. McRobbie and J. K. Slaney, Eds.), Lecture Notes in Artificial Intelligence, Vol. 1104, pp. 106–120, Springer-Verlag, Berlin.

[KZ88] Kapur, D., and Zhang, H. (1988), RRL: A rewrite rule laborarory, *in* "Proceedings of the 9th International Conference on Automated Deduction (CADE-88), Argonne" (E. Lusk and R. Overbeek, Eds.), Lecture Notes in Computer Science, Vol. 310, pp. 768–769, Springer-Verlag, Berlin.

[Nil71] Nilsson, N. J. (1971), "Problem Solving Methods in Artificial Intelligence," McGraw–Hill, New York.

[Wal94] Walther, C. (1999), Mathematical induction, *in* "Handbook of Logic in Artificial Intelligence and Logic Programming" (D. M. Gabbay, C. J. Hogger, and J. A. Robinson, Eds.), Vol. 2, pp. 127–227, Oxford Univ. Press, Oxford.