Flexible heuristics for simplification with conditional lemmas by marking formulas as forbidden, mandatory, obligatory, and generous

Tobias Schmidt-Samoa

FB Informatik
Technische Universität Kaiserslautern
Postfach 3049
67653 Kaiserslautern (Germany)
schmidt@informatik.uni-kl.de

ABSTRACT. *Due to its practical importance, context-dependent simplification of goals with* conditional *lemmas has been studied for three decades, mostly under the label of "contextual rewriting". We present a flexible framework for controlling the recursive relief of conditions by marking formulas in goals and lemmas. Within this framework, by marking goal formulas as* forbidden, *we can simulate and improve the well-known approaches of* contextual rewriting *and* case rewriting. *Furthermore, we develop novel heuristics which may mark goal formulas as* mandatory *and lemma formulas as* obligatory *or* generous. *Our case studies in the field of rewrite-based inductive theorem proving are encouraging.*

KEYWORDS: *conditional lemmas, case rewriting, contextual rewriting, heuristics, markings.*

1. Introduction

In this paper, we present a flexible framework for guiding proof search with markings. Because of the novelty of our approach, we present many motivating examples and skip some of the technical details. We concentrate on one special application domain, namely, equality-based inductive theorem proving in clausal first-order logic with implicitly universally quantified variables using atoms over the following predefined predicate symbols: *equality atoms* (symbol $=$), *definedness atoms* (`def`) to establish the domain of partially defined operators, and *order atoms* ($<$) to explicitly represent order constraints in a fixed wellfounded order. We present the examples within our inductive theorem prover QUODLIBET [AVE 03]. Nevertheless, our approach is in principle widely applicable. We will comment on this topic in Section 5.

When performing (mutual) inductive proofs for lemmas $\varphi_1, \ldots, \varphi_n$ with a rewrite-based theorem prover, there are at least three important tasks:

1) finding appropriate inductive case splits;

2) speculating appropriate auxiliary lemmas;

3) simplifying the goals from Task 1 to valid formulas using the lemmas from Task 2 and possibly smaller instances of $\varphi_1, \ldots, \varphi_n$ as induction hypotheses.

Task 3 provides the best chances for automation. Since the simplification process may be very time-consuming, automation has to be done carefully. Most work during the simplification process is caused by the application of (conditional) lemmas. In our case studies, they cause at least 50% of all proof steps. The process for applying a lemma can be divided into two steps: choosing a lemma; and checking the lemma for applicability and relieving its conditions. The first step can be supported by rippling techniques [BUN 93]. The *relief test* during the second step has to be done by recursively calling the simplification process. We will present a novel *extensive* but *efficient* relief test. By "extensive", we mean that the test should not fail too often if the lemma application may contribute to the proof (cf. Section 3).

1.1. *Simplification with conditional lemmas*

To perform proofs of lemmas, we use a sequent calculus, where the sequents are just lists of literals, i.e. just *clauses*. We apply the inference rules reductively: Each inference rule reduces a goal (*conclusion*) to a (possibly empty) list of subgoals (*premises*). Roughly speaking, a goal consists of a clause. Consider the application of a lemma to rewrite a subterm of a goal literal by replacing the left-hand side of an equation in the lemma by its right-hand side. More precisely, a clause $\{l_1, \ldots, l_n\}$ can be interpreted as an implication $\overline{l_1} \wedge \cdots \wedge \overline{l_{n-1}} \Rightarrow l_n$, where $\overline{l}$ is the conjugate (classical negation) of l. A lemma is called *conditional* if $n{>}1$. As in [ZHA 95], we fix one literal in the lemma clause by calling it the *head literal*; the conjugates of the other literals are called *condition literals*. For each inference step, we also fix one literal in the goal clause, called *focus literal*; the conjugates of the other literals are called *context literals*. The head literal of a lemma may be applied for proving a goal if the condition literals can be proved valid in the "context". According to [BOY 88], we have to relieve the conditions.

More precisely, *rewriting* of a goal clause with a lemma clause instantiated by a substitution σ is only possible if the head literal of the lemma is an equation $s = t$ (or $t = s$) and $s\sigma$ is equal to a subterm of the focus literal of the goal clause. The subterm is then replaced with $t\sigma$ resulting in a *rewrite subgoal*. To relieve the conditions, for each instantiated condition literal, a *condition subgoal* is created that essentially extends the original goal by the instantiated condition literal.[1] If an instantiated con-

1. If we use partially defined operators or apply a lemma as induction hypothesis, additional *definedness* and *order subgoals* have to be created.

dition literal is equal to a context literal[2] we say that it is *directly fulfilled* in the goal and the context literal is called a *cut-off literal* as it cuts off the subgoal that otherwise would have to be created for the condition literal. A lemma is *directly applicable* to a goal if all condition literals are directly fulfilled in the goal. Following [GEN 35], literals of a goal clause are called *principal* in an application of an inference rule if their presence (in the conclusion) is required for the applicability of this inference rule. In a rewrite step, the principal literals are the focus literal and the cut-off literals. A lemma may also be applied for *subsumption* provided that the head literal matches the focus literal. The application results in the same subgoals without the rewrite subgoal.

In general, the relief test for a condition subgoal is performed by a recursive call of the simplification process. Thus, the extent and efficiency of the test depend on the simplification process. Various simplification processes differ e.g. in the way they use equality information. `NQTHM` [BOY 88] and `ACL2` [KAU 00] use the cross fertilization technique while `RRL` [ZHA 95] uses a constant congruence closure algorithm. In `RDL` [ARM 03], decision procedures can be used by the simplification process.

1.2. *Flexible control with markings*

The elements in the goals that can be used during the relief test have to be restricted since the condition subgoals contain the original goal. Thus, without restrictions the relief test may result in an infinite process: the lemma can be applied to the condition subgoals over and over again. We concentrate on the question *which* elements can be used during the relief test.

Practically, there are two major ways to restrict proof steps that may be used during the relief test by marking elements in the goals:

1) In previous approaches known from the literature such as Contextual Rewriting in [ZHA 95] and Case Rewriting in [BOU 93], elements are excluded from certain condition subgoals. In the original approaches, excluded elements are completely eliminated from the subgoals resulting in *unsafe* applications, i.e. we may derive invalid goals by applying valid lemmas to valid goals. We may model these approaches in a safe way by just *marking* excluded elements as *forbidden*, instead of eliminating them from the subgoals. The meaning is that a forbidden element in a goal must not be principal in the application of an inference rule.

2) We propose a novel, alternative approach by marking elements in goals as *mandatory*: If we apply an inference rule to a goal, one of the mandatory elements must be principal. With a mandatory marking we may favor those proof steps that *locally* contribute to the proof.

By marking elements as mandatory instead of forbidden, we overcome some difficulties of previous approaches [ZHA 95, BOU 93]: As we can use every element dur-

2. Instead of using only pure syntactic equality, we first perform some additional normalizing transformations on the literals.

ing the relief test provided that there is also one mandatory element involved, we can achieve a more extensive relief test. Furthermore, we develop techniques to restrict the relief test in a user-defined way with *obligatory* and *generous* markings in the lemmas to achieve the right balance between efficiency and extent. Our flexible framework allows us to combine the different markings in an arbitrary way.

For our heuristics, we assume that a goal can be divided into three parts w.r.t. the inference rule applied as presented for rewriting in Section 1.1: An inference rule is applicable to a goal if it contains the *principal* part for the application. The other elements form the *context*.[3] Whereas the principal part may be modified in an arbitrary way—new elements may be added, old elements may be changed or removed—the context is passively inherited to the new subgoals. Within the principal part we may identify some elements as *cut-off* elements. These elements are passively inherited just like the context but they may cut-off some of the subgoals that otherwise would have to be created. Therefore, the number of subgoals resulting from the application depends on the cut-off part. Instead of confusing the reader with formal definitions of these concepts, we will exemplify this partitioning of goals in Section 2.1 with further inference rules.

The influence of the markings in the goals on proof search can be defined in two steps:

1) we restrict the proof steps I that can be applied to a goal G according to the markings and the partitioning of goal G into principal part, cut-off part and context w.r.t. I;

2) we define the markings for the new subgoals.

Whereas we fix Step 1, the inheritance procedure in Step 2 may be realized in different ways. Step 2 may also be influenced manually. Therefore, we get a flexible mechanism to restrict proof search.

1.3. *Organization of the paper*

In Section 2, we exemplify the partitioning of goals w.r.t. the inference rule applied and present a simple example illustrating the advantages of our novel heuristics based on a mandatory marking in comparison to previous approaches based on a forbidden marking. In Section 3, we motivate, describe and illustrate our heuristics based on markings. For each heuristics, we identify proof patterns that cannot be handled with this heuristics. For our novel heuristics based on a mandatory marking, we can solve these problems at the expense of additional auxiliary lemmas or by using a generous marking which extends proof search. We compare the different heuristics in Section 4. There, we also provide evidence for the adequacy of our modeling of Contextual Rewriting with a forbidden marking. We conclude in Section 5.

3. Note that the partitioning of goals into principal part and context here slightly differs from the classification into focus and context literals in Section 1.1.

2. A simple example

Originally, we have developed our heuristics for the inductive theorem prover QUODLIBET [AVE 03]. Therefore, we illustrate our approach with QUODLIBET. Nevertheless, it can be easily applied to other sequent based theorem provers as well.

QUODLIBET admits *partial* definitions of operators over *free constructors* using (possibly *non-terminating*) *positive/negative*-conditional equations as well as *constructor*, *destructor*, and *mutual* recursion. *Inductive validity* is defined as validity in the class of so-called *data models*, the models that do not equalize any different constructor ground terms.

2.1. *Partitioning of goals into principal part, cut-off part, and context*

We illustrate the partitioning of goals into principal part, cut-off part, and context with those inference rules of QUODLIBET that are used in the examples within this paper. A formal treatment of these concepts in connection with pruning proof trees and reusing proofs can be found in [SCH 06]. We identify the principal and cut-off part of the goals. The context is given as those elements in the goal that are not principal. We do not present a formal definition of the inference rules but only give some intuition about their typical usage and semantics.

`compl-lit` can be applied to a goal G if G contains a literal l and its conjugate $\overline{l}$. In this case, no new subgoals are created as the goal is valid due to complementary literals. The complementary literals l and $\overline{l}$ are principal for the application. For this inference rule, there exist no cut-off literals.

$=$`-decomp` can be applied to a goal G if G contains an equation $s = t$ such that the toplevel symbols of s and t are identical. Therefore, the equation $s = t$ is principal for the application. Let $s_1, \ldots, s_n$ (resp. $t_1, \ldots, t_n$) be the subterms of s (resp. t) at the minimal positions where s and t differ.[4] To prove $s = t$, it suffices to prove $s_i = t_i$ for each $i \in \{1, \ldots, n\}$. Therefore, we apply the following for each $i \in \{1, \ldots, n\}$: If $s_i \neq t_i$ is present in G, then $s_i \neq t_i$ is a cut-off literal for the application as it prevents the creation of the new subgoal for $s_i = t_i$ (due to complementary literals). Otherwise, we generate one new subgoal adding $s_i = t_i$ to the original goal.

If s and t are identical, no new subgoals will be created. This is the typical usage of the inference rule.

$\neq$`-unif` can be applied to a goal G if G contains a negated equation $s \neq t$ and s and t are unifiable constructor terms. Therefore, the equation $s \neq t$ is principal for the application. For this inference rule, there exist no cut-off literals. Let

4. We may restrict the depth of the difference positions considered with a parameter of the inference rule.

σ be the most general unifier of s and t. The inference rule generates one new subgoal removing the principal literal $s \neq t$ from G and instantiating all other literals with substitution σ. The validity of the new subgoal entails the validity of the original goal G because for the instances that are not covered by σ, the negated equation $s \neq t$ holds true.

Note that we only classify $s \neq t$ as principal for the application. All other literals are modified in a uniform way with substitution σ but the applicability of the inference rule does not depend on these literals.

2.2. *An example proof*

The following example presents a proof pattern that can be handled by our novel heuristics but cannot be proved by previous approaches such as Contextual and Case Rewriting summarized in Section 3.3. It is taken from our case study that the greatest common divisor (`gcd`) of two natural numbers is idempotent, commutative and associative (at least if the numbers are not zero).

EXAMPLE 1. — Let the specification consist of two sorts: `Bool` for the boolean values with constructors `true` and `false`; `Nat` representing the natural numbers with constructors `0` for zero and `s` for the successor function. We consider the defined operators `+`, `*`, `-`, `div`, `gcd`, `leq` and `div-p` that represent the corresponding arithmetic operations on natural numbers, a less-or-equal and a divisibility predicate on natural numbers. We consider the formal specification of `gcd` only, given by Axioms (1) to (4). The `gcd` of two natural numbers is defined if at least one of its arguments is not zero. If exactly one of the arguments is not zero this argument is the result of the operation. Otherwise, we recursively call `gcd` with the smaller argument and the difference of greater and smaller argument which ensures that the definition is terminating.

(1) $\{\texttt{gcd}(x, y) = x,\ y \neq \texttt{0},\ x = \texttt{0}\ \}$
(2) $\{\texttt{gcd}(x, y) = y,\ x \neq \texttt{0},\ y = \texttt{0}\ \}$
(3) $\{\texttt{gcd}(x, y) = \texttt{gcd}(x, \texttt{-}(y, x)),\ \texttt{leq}(x, y) \neq \texttt{true},\ x = \texttt{0},\ y = \texttt{0}\ \}$
(4) $\{\texttt{gcd}(x, y) = \texttt{gcd}(\texttt{-}(x, y), y),\ \texttt{leq}(x, y) = \texttt{true},\ \neg\texttt{def leq}(x, y),\ x = \texttt{0},\ y = \texttt{0}\ \}$

As auxiliary lemma for the associativity of `gcd`, we want to prove the following lemma on divisibility:

(5) $\{\texttt{div-p}(\texttt{gcd}(x, y), z) = \texttt{true},\ \texttt{div-p}(x, z) \neq \texttt{true},\ x = \texttt{0}\ \}$

We assume that the following lemmas are activated for automatic applications:

(6) $\{\texttt{def gcd}(x, y),\ x = \texttt{0}\ \}$
(7) $\{\texttt{div-p}(\texttt{gcd}(x, y), x) = \texttt{true},\ y = \texttt{0}\ \}$
(8) $\{\texttt{div-p}(x, z) = \texttt{true},\ \texttt{div-p}(x, y) \neq \texttt{true},\ \texttt{div-p}(y, z) \neq \texttt{true}\ \}$

For each of the axioms and lemmas, we choose the first literal as head literal for the following reasons:

– The axioms define operator gcd using the first literal as rewrite rule from left to right.

– Lemma (6) contains a definedness atom as first literal. Due to our monotonic semantics based on data models, we cannot prove negated definedness literals as focus literals. Therefore, the definedness atom should be present in the goal the lemma is applied to. Such a lemma is called a *domain* lemma as it establishes the domain of a (partial) operator.

– In Lemma (7), the left-hand side of the first literal is the only term that binds all variables of the lemma.

– In Lemma (8), the first literal is the positive literal in a Horn clause.

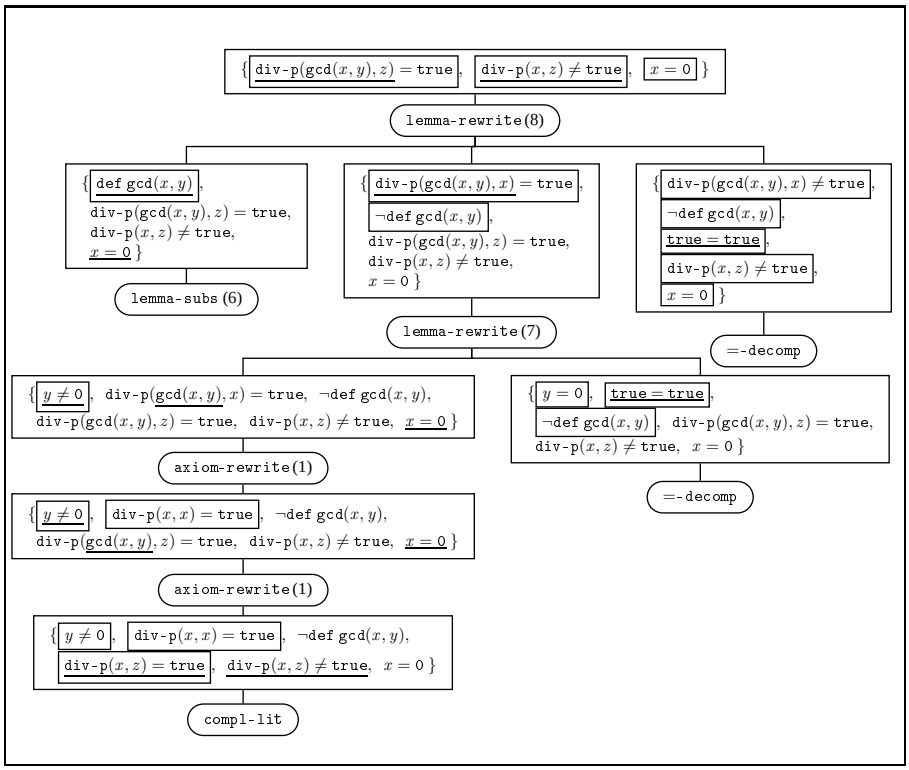

Figure 1. *Proof state tree for Goal* (5) *of Example 1*

Figure 1 contains the whole proof state tree for Goal (5) as it is created by our novel heuristics. A proof state tree consists of goal and inference nodes representing the application of inference rules. The root goal node consists of the conjecture to be proved and is displayed at the top of the proof state tree. The root goal node is rewritten by the conditional Lemma (8) using the substitution $[x \leftarrow \mathtt{gcd}(x, y), z \leftarrow z, y \leftarrow x]$. The substitution can be determined by using the first literal of the root goal as focus literal and matching the head literal to the focus literal. The uninstantiated *extra* vari-

able y can be bound by matching the third lemma literal to the second goal literal (cf. Section 3.6). Then, the third lemma literal is *directly fulfilled* by the second goal literal which itself is a *cut-off* literal. Thus, the first two goal literals are *principal* for the application. In Figure 1, principal literals are underlined and mandatory literals are framed. Our novel heuristics applies an inference rule automatically only if one of the principal literals is also mandatory, i.e. if one of the underlined literals is also framed. The application results in three new subgoals (from left to right):[5] one *definedness subgoal* (since the substitution binds a constructor variable to a non-constructor term), one *condition subgoal* and one *rewrite subgoal*. As there is a condition subgoal, the lemma is not *directly applicable*. The definedness subgoal is proved by a direct application of Lemma (6) for subsumption. Rewriting the condition subgoal with Lemma (7) leads to another condition subgoal. For its proof we rewrite the second and fourth literal with Axiom (1). Note that these literals have been the focus literals of the previous lemma applications that have generated the considered condition subgoal. Thus, these applications are possible only with our novel heuristics (cf. Section 3). Altogether, we get a *closed* proof state tree, i.e. a proof state tree whose leaves are inference nodes. Therefore, Lemma (5) is inductively valid provided that this holds true for the applied lemmas. We call such a closed proof state tree just a *proof*. □

3. Controlling the application of conditional lemmas

Lemmas are provided to guide the proof process. On the one hand, they should be applied automatically as far as possible[6] to free the user from routine work. On the other hand, heuristics have to control the applications to guarantee the termination of the process within a reasonable amount of time. Thus, we have to find the right balance between extent and efficiency.

We essentially restrict proof search with markings in goals and lemmas (cf. Sections 3.2 to 3.5). Our novel heuristics based on a mandatory marking is inspired by the contribution of proof steps which we define in Section 3.1. To get a practicable method, we adapt additional heuristics known from the literature—in particular those presented in [BOY 88]. We summarize these heuristics in Section 3.6.

Note that, in general, in our domain neither confluence nor termination properties can be assumed for rewriting with lemmas (cf. Example 21). Therefore, heuristics based on wellfounded orders are not always applicable. If applicable, these heuristics may be combined with our marking techniques.

5. This order is relevant insofar as we maximally downfold the definedness literals, the condition literals, and the rewrite literal $s\sigma \neq t\sigma$ to the right, i.e. enhance the subgoals to the right with the negation of these literals to the left.

6. at least if they may contribute to the proof (cf. Section 3.1)

3.1. *Contributing proof steps and elements*

Analyzing a performed proof, a *proof step* (i.e. the application of an inference rule) may *contribute* to a proof for a goal in two ways: Firstly, no new subgoals are created at all; thus, the goal is proved. Secondly, each subgoal contains new information in the form of *new* (i.e. added or changed) literals that are needed for the proof (i.e. become principal in a further contributing proof step). Otherwise, the proof step is *non-contributing* and can be eliminated: If one subgoal can be proved without using one of the new literals, this proof can also be used for the original goal.

DEFINITION 2 (CONTRIBUTING PROOF STEPS / ELEMENTS). — *A proof step I of a proof P for goal G* contributes *to P if every direct subgoal SG created by I contains a new element that contributes to the proof for SG in P. An element of a goal G* contributes *to proof P for G if it becomes principal for one contributing proof step I of P.*

In Example 1, the only non-contributing proof step is the first application of Axiom (1) to rewrite the second literal. Indeed, this literal—the only new one—does not contribute to the proof for its subgoal.

The notion of contribution can be used for pruning proof trees by eliminating non-contributing proof steps. Thereby, we can

– get simpler proofs;

– determine superfluous literals in a goal that do not contribute to the proof;

– enhance the reusability of a proof by focusing on contributing literals.

For a detailed discussion on these topics and a more involved bottom-up definition of a logically stronger notion of contribution, we refer to [SCH 06].

The notion of contribution captures what we want but cannot be directly exploited for proof search: As contribution of a proof step depends on the proof performed, it can be *checked* only after the proof has been completed. But we can easily *ensure* that we perform only contributing proof steps by using one of the new elements as principal element in the *next* proof step.

DEFINITION 3 (LOCALLY CONTRIBUTING PROOF STEPS / ELEMENTS). — *A proof step I in a proof P for a goal G* locally contributes *to P if every direct subgoal SG created by I contains a new element that becomes principal in the proof step performed for SG in P.*

LEMMA 4. — *If every proof step in a proof P locally contributes to P, then every proof step contributes to P.*

PROOF. — This is proved immediately by structural induction on proof trees. ■

Note that, in general, a proof step does not have to be contributing even if it is locally contributing, because the new elements may become principal only in non-contributing proof steps.

As we will see, this strict usage of local contribution is too restrictive for guiding proof search. It excludes too many proofs in which all proof steps are contributing but some of them do not contribute locally.

3.2. *Novel heuristics based on mandatory markings in goals*

We aim at avoiding non-contributing proof steps. To be able to define local restrictions on proof steps in a flexible way, we introduce a mandatory marking in goals.

RESTRICTION 5 (CAUSED BY MANDATORY MARKING). — An inference rule may be applied to a goal G with a *mandatory marking* only if one of the mandatory elements is principal in the proof step applied to G. □

If we mark only new elements in a subgoal as mandatory in a proof, it is ensured that all proof steps (locally) contribute to that proof. But then, the proof search is too restricted. It will find only "linear" proofs: We can apply only those inference rules that also use new elements introduced by the previous proof step. For a successful proof, however, it may be necessary to apply inference rules "in parallel" that are not linearizable. Such proofs are impossible with this strict usage of a mandatory marking as the following example illustrates.

EXAMPLE 6. — Given three boolean valued constants $\mathtt{p1}, \mathtt{p2}, \mathtt{p3}$, we assume the activation of Lemmas (9) and (10) and want to prove Goal (11)

(9) $\{\mathtt{p1} = \mathtt{p3}\}$ (10) $\{\mathtt{p2} = \mathtt{p3}\}$ (11) $\{\mathtt{p1} = \mathtt{true},\ \mathtt{p2} \neq \mathtt{true}\}$

To prevent trivial loops we use equations for rewriting just in one direction. We present our examples in such a way that equations are always applied for rewriting from left to right. Therefore, the only way to prove Goal (11) is to rewrite p1 and p2 to p3. Then the resulting subgoal is tautological as it contains complementary literals. But if we mark only new elements as mandatory, this proof is prohibited since the second rewrite step does not use a new element. □

Alternatively, if all elements of subgoals are marked as mandatory, the marking has no effect and the search space contains too many proof steps that do not contribute to the proof. Our compromise results in the following default heuristics which can be fine-tuned with a *generous* marking in the lemmas as explained in Section 3.5:

HEURISTICS 7 (FOR MARKING ELEMENTS AS MANDATORY). — At the beginning of a proof attempt for a lemma every element in the clause is marked as mandatory. Thus, there are no restrictions for performing proof steps.

For *applicability subgoals*—i.e. definedness or condition subgoals of applicative inference rules—the marking of the parent goal is not inherited to the subgoal, but a new set of mandatory elements is introduced that consists exactly of the new elements of the subgoal. With this *strict* marking heuristics, it is guaranteed that one of the new definedness or condition literals is used in the next proof step.

For order subgoals, we mark only the single new order atom as mandatory. In this case, the proof has to proceed by treating the order atom.

For all other subgoals—i.e. rewrite subgoals or subgoals created by other inference rules—the mandatory elements of the parent goal stay mandatory in the subgoal (unless they are deleted) and are supplemented with all new elements of the subgoal. Thus, we use a *relaxed* marking heuristics. We can perform rewrite steps even if they do not contribute to the proof. This is helpful for the speculation of auxiliary lemmas. □

EXAMPLE 8 (1 CONTINUED). — In Figure 1, mandatory literals are framed, principal literals are underlined. Thus, we can apply an inference rule only if at least one of the underlined literals is also framed.

The proof starts at the root goal node with all literals marked as mandatory. After applying Lemma (8), the resulting definedness subgoal has one mandatory literal—the first one—that is handled by the following subsumption with Lemma (6). The mandatory literals of the condition subgoal—the second subgoal—are the first two literals. Note that the repeated application of Lemma (8) is prevented as none of its principal literals is mandatory anymore. Instead, the first literal is handled by the following rewrite step with Lemma (7), that introduces the first literal as the only mandatory literal for the new condition subgoal. This single mandatory literal is used in the rewrite step with Axiom (1). As this inference rule modifies the second literal of the resulting rewrite subgoal, it is added to the set of mandatory literals. Analogously, literal four is added to this set after the next rewrite step with Axiom (1). Finally, the inference rule `compl-lit` can be applied to the rewritten subgoal although not both literals are mandatory. It suffices that one mandatory literal is principal for the application. Note that all literals in the rewrite subgoal of the application of Lemma (8) are mandatory since the goal is not an applicability subgoal (cf. Heuristics 7). This is justified by the fact that an infinite loop of the same lemma application is already avoided because the original goal is not contained in the new subgoal. The relaxed mandatory markings heuristics for rewrite subgoals is, for instance, required for the second application of Axiom (1): Otherwise, $y \neq \mathtt{0}$ would not stay mandatory after the first rewrite step with Axiom (1) and the second application would not obey the restrictions caused by the mandatory marking. □

EXAMPLE 9. — As another example, we consider the defined operators `less` and `+`, given by the following axioms:

(12) $\{\ \mathtt{less}(\mathtt{0}, \mathtt{s}(y)) = \mathtt{true}\ \}$
(13) $\{\ \mathtt{less}(x, \mathtt{0}) = \mathtt{false}\ \}$
(14) $\{\ \mathtt{less}(\mathtt{s}(x), \mathtt{s}(y)) = \mathtt{less}(x, y)\ \}$
(15) $\{\ \mathtt{+}(x, \mathtt{0}) = x\ \}$
(16) $\{\ \mathtt{+}(x, \mathtt{s}(y)) = \mathtt{s}(\mathtt{+}(x, y))\ \}$

Given the additional lemmas

(17) $\{\mathtt{def}\ \mathtt{+}(x, y)\ \}$
(18) $\{\mathtt{less}(x, \mathtt{+}(x, y)) = \mathtt{true},\ y = \mathtt{0}\ \}$
(19) $\{\mathtt{less}(x, z) = \mathtt{true},\ \mathtt{less}(x, y) \neq \mathtt{true},\ \mathtt{less}(y, z) \neq \mathtt{true}\ \}$

we want to prove the inductive validity of the following goal by simplification:

(20) $\{\, \mathtt{less}(x, \mathtt{+}(y, z)) = \mathtt{true},\ \mathtt{less}(x, y) \neq \mathtt{true} \,\}$

For each of the axioms and lemmas, we choose the first literal as head literal for the following reasons: The axioms define operators `less` and + using the first literal as rewrite rule from left to right. In Lemma (18), the left-hand side of the first literal is the only term that binds all variables of the lemma. In Lemma (19), the first literal is the positive literal in a Horn clause.

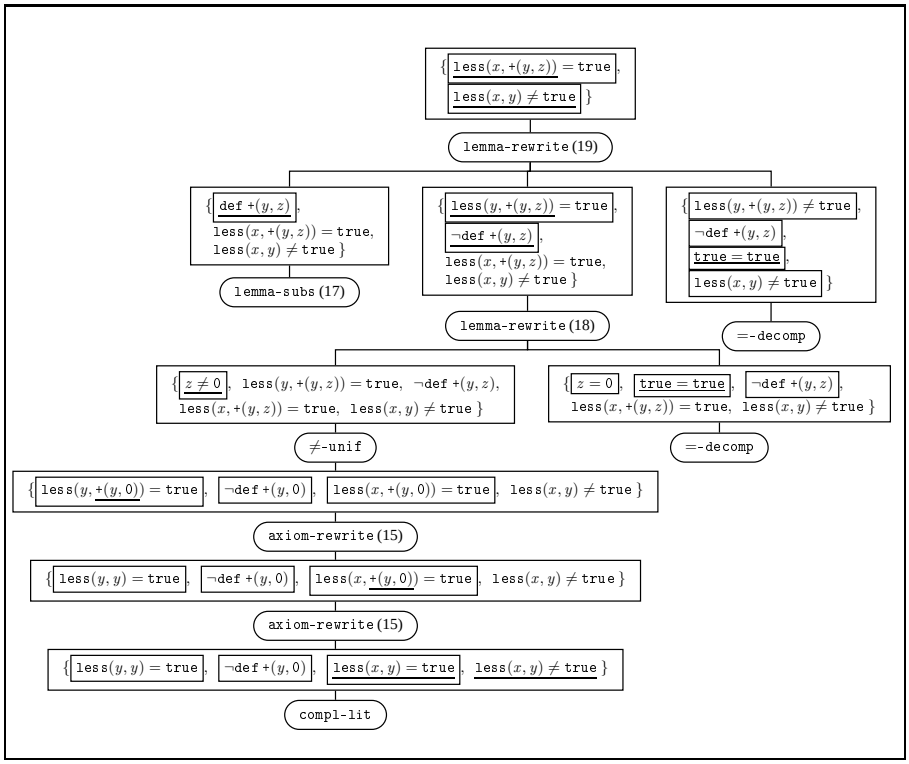

Figure 2. *Proof state tree for Goal* (20) *of Example 9*

Figure 2 contains the whole proof state tree for Goal (20) as it is created by our novel heuristics with a mandatory marking. Again, mandatory literals are framed, principal literals are underlined.

The proof starts at the root goal node with all literals marked as mandatory. The root goal node is rewritten by the conditional Lemma (19) using the substitution $[z \leftarrow \mathtt{+}(y, z)]$. The substitution can be determined by using the first literal of the root goal as focus literal and matching the head literal to the focus literal. The second lemma literal is directly fulfilled by the second goal literal (binding the extra variable y to itself). The application results in three new subgoals (from left to right): one definedness subgoal, one condition subgoal and one rewrite subgoal. The definedness

subgoal has one mandatory literal—the first one—that is handled by the following subsumption with Lemma (17). The mandatory literals of the condition subgoal are the first two literals. Note that these mandatory literals prevent the repeated application of Lemma (19). Instead, the first literal is handled by the following rewrite step with Lemma (18), that introduces the first literal as the only mandatory literal for the new condition subgoal. This single mandatory literal is used by the inference rule $\neq$`-unif`. As this inference rule modifies the first three literals of the resulting subgoal, they become the mandatory literals. The following rewrite steps do not alter the sets of mandatory literals as we do not start new sets for rewrite goals. This results in one rewrite step that does not contribute to the proof. Finally, the inference rule `compl-lit` can be applied to the rewritten subgoal although not both literals are mandatory. It suffices that one mandatory literal is principal for the application. □

Examples 8 and 9 contain a basic proof pattern that cannot be proved with Contextual Rewriting (cf. Section 3.3.1) but with our novel heuristics. This proof pattern is illustrated in Example 10 in an abstract way. We use it for comparing our novel heuristics (cf. Figure 3a described in Example 10) with Contextual Rewriting (cf. Figure 3b described in Example 15) and Case Rewriting according to [BOU 93] (cf. Figure 3c described in Example 17).

EXAMPLE 10 ([ZHA 95], SIMPLIFIED). — Given three boolean valued constants $\mathtt{q1}, \mathtt{q2}, \mathtt{q3}$, we assume the activation of the following lemmas

(21) $\{\mathtt{q1} = \mathtt{true},\ \mathtt{q2} \neq \mathtt{true}\}$ (22) $\{\mathtt{q1} = \mathtt{true},\ \mathtt{q3} \neq \mathtt{true}\}$ (23) $\{\mathtt{q2} = \mathtt{true},\ \mathtt{q3} = \mathtt{true}\}$

and want to prove the goal

(24) $\{\mathtt{q1} = \mathtt{true}\}$

As Lemmas (21) and (22) are Horn clauses we use the first literal as head literal. Lemma (23) does not suggest a head literal itself. We may use an arbitrary one or both literals. Due to efficiency considerations and as the lemmas are symmetric in `q2` and `q3`, we decide to choose just the first one.

Using a mandatory marking, the proof is found automatically (cf. Figure 3a). In the condition subgoal after applying Lemma (23), literal $\mathtt{q1} = \mathtt{true}$ can be used as focus literal to rewrite `q1` to `true` although this literal is not mandatory. This can be done since the condition literal of the applied lemma is mandatory. □

For an extensive relief test, the following property would be useful: If a goal can be proved by simplification without any restrictions on the elements that can be used then it can also be proved obeying the restrictions caused by a mandatory marking. Unfortunately, this strong property does not hold as will be shown in Example 11, a simple generalization of Example 10.

The interaction of head literals in lemma clauses and mandatory literals in goal clauses restricts the search space of the simplification process very much: In most cases, a lemma will be applied to a goal only if the head literal of the lemma is mandatory in the goal. The proof step then transfers the mandatory marking from the head

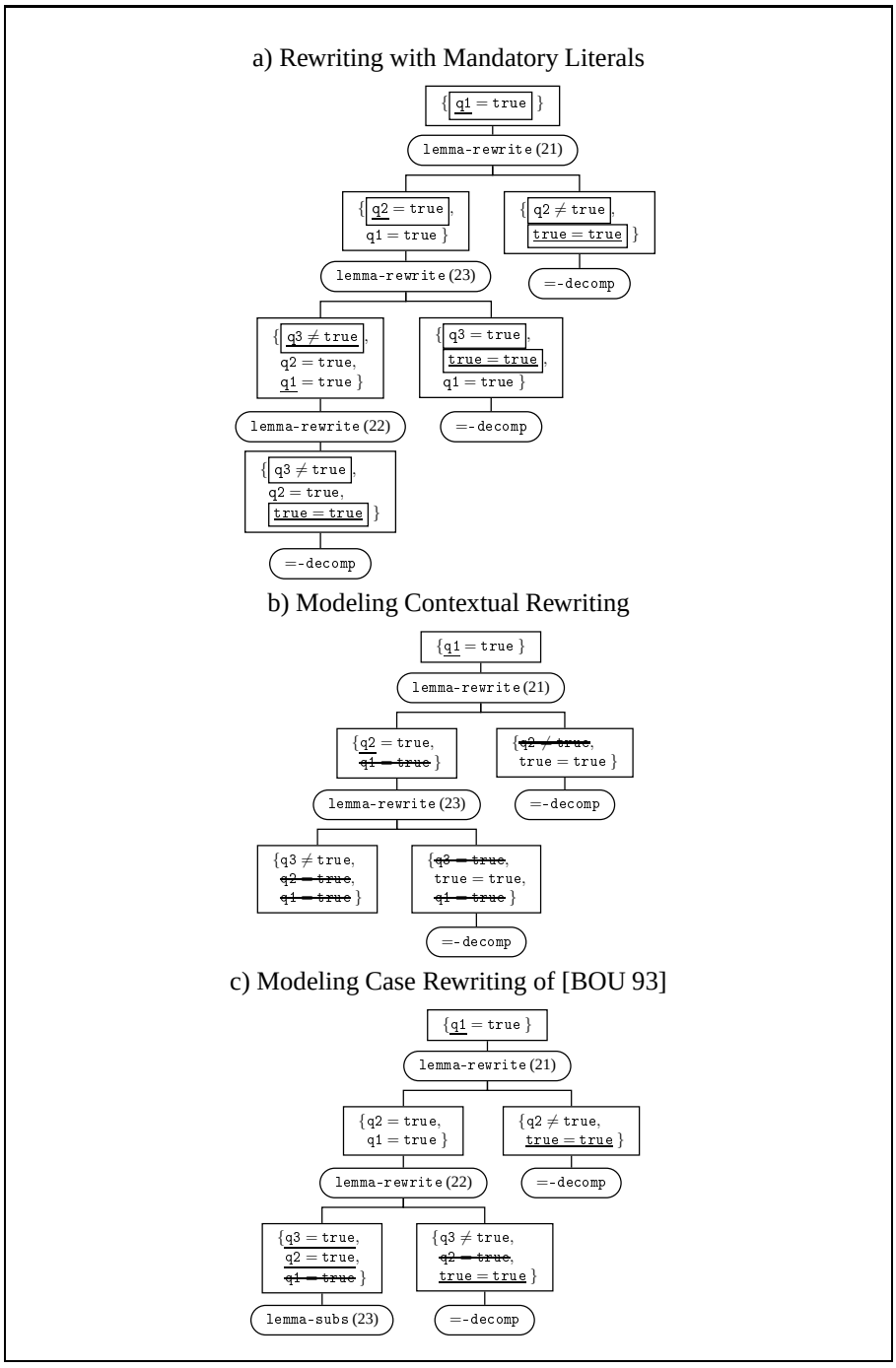

Figure 3. *Proof state trees for Example 10*

literal to its condition literals. This direction can be inverted automatically only if the gap between the head literal and one of the condition literals can be closed in one step within the goal clause, i.e. if one condition literal is a mandatory literal of the goal clause as in Example 10. Otherwise, we have to use auxiliary lemmas to bridge the gap.

EXAMPLE 11. — Given five boolean valued constants $\mathtt{r1}, \ldots, \mathtt{r5}$, we assume the activation of the following lemmas

(25) $\{\mathtt{r1} = \mathtt{true},\ \mathtt{r2} \neq \mathtt{true}\}$

(26) $\{\mathtt{r1} = \mathtt{true},\ \mathtt{r3} \neq \mathtt{true}\}$

(27) $\{\mathtt{r2} = \mathtt{true},\ \mathtt{r4} \neq \mathtt{true}\}$

(28) $\{\mathtt{r3} = \mathtt{true},\ \mathtt{r5} \neq \mathtt{true}\}$

(29) $\{\mathtt{r4} = \mathtt{true},\ \mathtt{r5} = \mathtt{true}\}$

and want to prove the goal

(30) $\{\mathtt{r1} = \mathtt{true}\}$

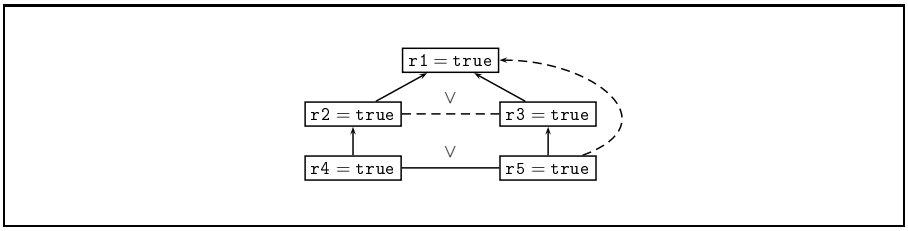

Figure 4. *Illustration of Example 11*

The specification is illustrated in Figure 4 by solid lines. We assume that the first literal is used as head literal for each lemma. There is a gap of two steps e.g. between $\mathtt{r1} = \mathtt{true}$ and $\mathtt{r5} = \mathtt{true}$ that cannot be closed automatically. Using the mandatory markings heuristics, our automatic proof control performs two proof attempts which are illustrated in Figure 5. None of the lemmas can be applied to one of the two open goals without violating the restrictions caused by the mandatory marking. In the open goal of the first proof attempt, for instance, only literal $\mathtt{r5} = \mathtt{true}$ is marked as mandatory. Therefore, the only way to obey the restriction caused by the mandatory marking would be to apply Lemma (29). But as r4 is not present in the goal, we cannot apply the lemma for rewriting r4 as required by the activation. The same argument holds true for the open goal of the second proof attempt, Lemma (28) and operator r3 which is not present in the goal.

We can overcome this situation by introducing e.g. one of the following auxiliary lemmas (illustrated in Figure 4 by dashed lines):

(31) $\{\ \mathtt{r2} = \mathtt{true},\ \mathtt{r3} = \mathtt{true}\ \}$ (32) $\{\ \mathtt{r1} = \mathtt{true},\ \mathtt{r5} \neq \mathtt{true}\ \}$

Each of these lemmas as well as Goal (30) (after activating one of (31), (32)) can be proved automatically. Goal (30), for instance, can be proved analogously to Goal (24) in Example 10 if we activate Lemma (31) (cf. Figure 3a replacing $\mathtt{q}i$ with $\mathtt{r}i$). If

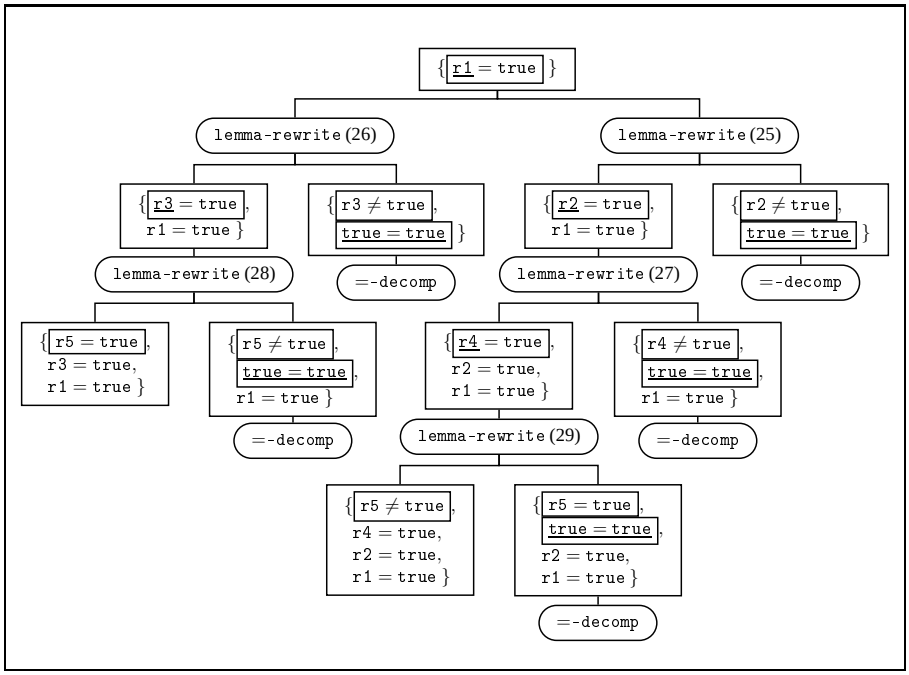

Figure 5. *Two failed proof attempts for Goal* (30) *of Example 11*

we activate Lemma (32) we can prove the open goal of the second proof attempt in Figure 5. Thus, we can bridge the gap. □

Theorem 12 states that we can always close gaps with auxiliary lemmas.

THEOREM 12. — *If a goal can be proved by simplification without any restrictions on the elements that can be used then it can also be proved with a mandatory marking with the help of some auxiliary lemmas which themselves can be proved with a mandatory marking. More precisely, if a proof violates the restrictions at n goal nodes then we need at most n auxiliary lemmas.*

PROOF. — A proof step that creates a subgoal that does not possess any new literal cannot contribute to a proof. Hence, it can be eliminated from the proof. Thus, we can assume that a proof contains at least one new literal in each subgoal. As we mark at least one of the new literals as mandatory, each subgoal contains at least one mandatory literal. If a goal in the proof violates the restrictions caused by a mandatory marking we can introduce a new lemma consisting of this goal clause. As each proof attempt for a new lemma starts with all literals marked as mandatory, the proof of the lemma succeeds with a mandatory marking. Whatever head literal is chosen for this lemma, it can be applied to prove the goal that formerly violated the restrictions caused by a mandatory marking. The lemma is applicable because at least one literal in the goal is mandatory. ■

Unfortunately, the required auxiliary lemmas cannot be calculated automatically. In fact, the automatic generation of lemmas according to the last proof would counteract the mandatory marking because proof search would continue for the auxiliary lemmas with all elements marked as mandatory again. Nevertheless, the auxiliary lemmas may be manually extracted from failed proof attempts. In contrast to this, Contextual Rewriting may not even be able to make use of auxiliary lemmas simply because one cannot build a bridge when a bank is forbidden.

3.3. *Simulating approaches from the literature with forbidden markings in goals*

Other approaches known from the literature perform the relief test in such a way that certain elements are excluded from the generated subgoals. In these approaches, excluded elements are completely eliminated from the subgoals, resulting in unsafe lemma applications. Within our flexible framework, we model these approaches in a safe way by introducing a forbidden marking in goals.

RESTRICTION 13 (CAUSED BY FORBIDDEN MARKING). — An inference rule may be applied to a goal G with a *forbidden marking* only if all forbidden elements that are principal in this proof step are cut-off elements. □

We account for the elimination of forbidden elements in the approaches known from the literature in the following way: Once an element is marked as forbidden in a goal G, it remains forbidden in the whole proof attempt for G. Our modeling with a forbidden marking improves the versions in the literature insofar as forbidden elements may serve as cut-off elements. Let G be a goal with a forbidden marking and G' be the goal derived from G by eliminating all forbidden elements. Roughly speaking, since forbidden elements must not be principal (unless they are cut-off elements), the same inference rules are applicable to G and G'. The applications result basically in the same subgoals. Therefore, proof search is essentially the same in both cases except that the approaches that eliminate forbidden elements have to prove

– additional subgoals that are cut off by the forbidden elements;

– stronger goals since conditions in form of forbidden elements are missing. This may even cause a failure of a proof attempt because of a subgoal which is, in fact, trivial if we do not eliminate forbidden elements.

Thus, we consider the use of a forbidden marking as an adequate and safe alternative for modeling previous approaches from the literature. The additional cut-off elements do not change the search space, but relax the success criterion of our proof search. In Section 4, we perform case studies to validate the adequacy of our modeling of Contextual Rewriting and to demonstrate the additional benefits of using forbidden elements as cut-off elements.

3.3.1. *Contextual rewriting*

Contextual Rewriting in the narrower sense is used e.g. in `NQTHM` [BOY 88], `ACL2` [KAU 00], `RRL` [ZHA 95], and more recently in `RDL` [ARM 03]. These approaches

vastly differ in their simplification process e.g. in the way they use equality information. NQTHM [BOY 88] and ACL2 [KAU 00] use the cross fertilization technique while RRL [ZHA 95] uses a constant congruence closure algorithm. In RDL [ARM 03], decision procedures can be used by the simplification process. Nevertheless, they use the same literals to perform the relief test: The focus literal in the applicability subgoals as well as all downfolded literals are marked as forbidden. On the one hand, Contextual Rewriting is not very restrictive because it admits non-contributing proof steps. On the other hand, it is often too restrictive as can be seen in our examples:

EXAMPLE 14 (8 CONTINUED). — The second application of Axiom (1) in Figure 1 rewrites a literal initially used as focus literal. Thus, Contextual Rewriting fails. □

EXAMPLE 15 (10 CONTINUED). — Two lemmas have to be applied to the same goal literal to perform a successful proof. But after applying one lemma, the focus literal is forbidden for the rest of the proof attempt. This situation is depicted for one proof attempt in Figure 3b where forbidden literals are marked by crossing them out. The proof attempt fails at the left-most leaf as $\mathtt{q1} = \mathtt{true}$ cannot be used anymore. It is not possible to overcome this situation with auxiliary lemmas. □

Not surprisingly, Example 11—a generalization of Example 10—cannot be proved with Contextual Rewriting either. Nevertheless, a slight modification changes the example in such a way that it can be proved with Contextual Rewriting but not with our novel heuristics with a mandatory marking (in the simple form presented in Section 3.2 and without auxiliary lemmas).

EXAMPLE 16. — Given six boolean valued constants $\mathtt{s1}, \ldots, \mathtt{s6}$, we assume the activation of the following lemmas

(33) $\{\mathtt{s1} = \mathtt{true},\ \mathtt{s3} \neq \mathtt{true}\}$

(34) $\{\mathtt{s2} = \mathtt{true},\ \mathtt{s4} \neq \mathtt{true}\}$

(35) $\{\mathtt{s3} = \mathtt{true},\ \mathtt{s5} \neq \mathtt{true}\}$

(36) $\{\mathtt{s4} = \mathtt{true},\ \mathtt{s6} \neq \mathtt{true}\}$

(37) $\{\mathtt{s5} = \mathtt{true},\ \mathtt{s6} = \mathtt{true}\}$

and want to prove the goal

(38) $\{\mathtt{s1} = \mathtt{true},\ \mathtt{s2} = \mathtt{true}\}$

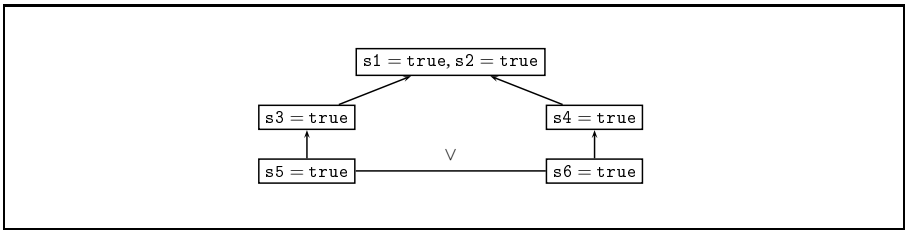

Figure 6. *Illustration of Example 16*

The specification is illustrated in Figure 6. We assume that the first literal is used as head literal for each lemma. Whereas Contextual Rewriting can apply Lemmas (33)

to (37) one after the other, our heuristics based on mandatory markings cannot close the gap without auxiliary lemmas or generous literals (cf. Sections 3.2 and 3.5). □

3.3.2. *Case rewriting*

Case Rewriting tries to overcome the difficulties of Contextual Rewriting by a special treatment of lemmas that can be applied to rewrite the same redex alternatively, such as e.g. Lemmas (21) and (22) in Example 10. In this sense, our approach with a mandatory marking is a novel form of Case Rewriting. There are at least two further approaches known from the literature [BOU 93, KOU 90].

The approach for Case Rewriting proposed in [KOU 90] restricts the relief test by order constraints which we cannot use in general for our application domain as we allow non-terminating operator definitions.

In the approach of Case Rewriting in [BOU 93], a term is rewritten by a set of n lemmas resulting in $n + 1$ new subgoals: For each lemma one rewrite subgoal is created; additionally one *well-coveredness* subgoal is produced. This last subgoal is to guarantee the completeness of the case split w.r.t. the given lemmas.

EXAMPLE 17 (15 CONTINUED). — For the specification of Example 10, this Case Rewriting approach can be modeled by applying the two lemmas in succession as depicted in Figure 3c. In the well-coveredness goal—i.e. the left-most goal—only the condition literals may be used. As the case split for Lemmas (21) and (22) is complete according to Lemma (23), the proof can be completed. □

The approach of [BOU 93] seems to have the following limitations: The set of lemmas applied depends only on the focus literal but not on the context. A single well-coveredness goal is sufficient only if all lemmas differ only in a single condition literal. The well-coveredness goal cannot be proved if the case split is incomplete.

EXAMPLE 18 (14 CONTINUED). — The applications of Lemma (8) and Axiom (1) in Figure 1 do not form a complete case split. Actually, they even do not rewrite the same redex. Thus, Case Rewriting according to [BOU 93] cannot be applied. Moreover, in contrast to [BOU 93], our approach with a mandatory marking can make use of other goal literals to prove the well-coveredness subgoal. □

3.4. *Enhancing the efficiency with obligatory markings in lemmas*

On the one hand, the use of a mandatory marking as explained in Section 3.2 results in an *extensive* relief test. But as we call the simplification process recursively for any condition subgoal whose condition literal is not directly fulfilled in the goal, it may be very time-consuming. On the other hand, using only directly applicable lemmas is a very *efficient* but not extensive relief test because it checks only syntactic equality. As a compromise, we introduce an obligatory marking in lemmas that restricts the relief test for obligatory elements to the efficient syntactic test. This guides the proof search in a user-defined way, manually controlling the degree of extent and efficiency for each lemma separately.

RESTRICTION 19 (CAUSED BY OBLIGATORY MARKING). — A lemma with an *obligatory marking* may be applied to a goal G only if all obligatory elements are directly fulfilled in the goal. □

Thus, a lemma with an obligatory marking is only applicable if the instantiated obligatory elements are present in the goal. Therefore, we may interpret the obligatory marking as a user-defined means to extend the principal part in the goal w.r.t. the applied lemma.

By marking elements as obligatory, we restrict proof search. In doing so, we may prevent the automatic derivation of proofs. Thus, elements are *automatically* marked as obligatory only if the head literal of the lemma is an equation that is used as rewrite rule with a *general term* as left-hand side, i.e. a term of the form $f(x_1, \ldots, x_n)$ in which the x_i are pairwise different variables. Without obligatory elements, the relief test would be invoked too often in this case, most of the time unsuccessfully.

HEURISTICS 20 (FOR MARKING ELEMENTS AS OBLIGATORY). — Our automatic default heuristics chooses one obligatory element if the lemma is used as rewrite rule with a general term as left-hand side. □

EXAMPLE 21. — If we use the first literal of the trichotomy of less

(39) $\{\ \mathtt{less}(x, y) = \mathtt{true},\ \ \mathtt{less}(y, x) = \mathtt{true},\ \ x = y\ \}$

as head literal then it is activated as rewrite rule for operator `less` with a general term as left-hand side. When applying this lemma, the relief test reduces the question whether x is less than y to the question whether y is not less than x and whether they are unequal. But this relief test is in most cases not simpler than the original problem. In Example 9, the activation of Lemma (39) would increase the number of inference steps from 9 to 12. But if the context says that neither x is less than y nor y is less than x then we can derive that x is equal to y by Lemma (39). Thus, we may prove the remaining literals of the clause in the possibly very useful context that x is equal to y. Therefore, the automatic application of the lemma should be restricted by marking the second lemma literal as obligatory. □

EXAMPLE 22. — Another example is given by the axioms of a division operator:

(40) $\{\mathtt{div1}(x, y, u, v) = u,\ \ x \neq v\ \}$

(41) $\{\mathtt{div1}(x, y, u, v) = \mathtt{div1}(x, y, \mathtt{s}(u), \mathtt{+}(v, y)),\ \ x = v\ \}$

Without marking literals as obligatory the relief test illustrated in Figure 7 does not terminate. During the relief test for Axiom (40), Axiom (41) can be applied for rewriting because the mandatory literal directly fulfills the condition literal of Axiom (41). Since the rewriting changes the second goal literal it becomes mandatory and thus can be used for further rewrite steps with the axioms. If the conditions of the axioms are marked as obligatory, already the first relief test is prevented. □

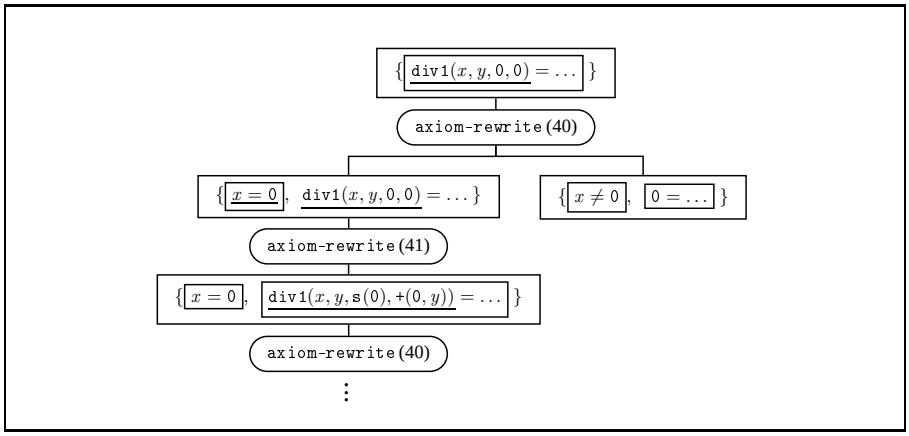

Figure 7. *Non-terminating relief test in Example 22*

3.5. *Enhancing the extent with generous markings in lemmas*

The efficiency of the relief test can be improved and manually controlled with obligatory markings. To enhance the extent of the relief test based on a mandatory marking in a user-defined way, we now introduce generous markings in lemmas.

The idea of our mandatory markings is to prefer (locally) contributing proof steps, and, therefore, to prevent non-contributing proof steps. But as explained in Example 6, applying only those proof steps that locally contribute to a proof, restricts proof search too much and prevents too many proofs where all proof steps are contributing but some of them do not locally contribute to the proof. Therefore, our default Heuristics 7 for marking mandatory elements in subgoals applies *strict* or *relaxed* mandatory marking heuristics depending on the type of the subgoal, i.e. whether it is an applicability subgoal, an order subgoal or another subgoal. But even with this default Heuristics 7, proof search is restricted in such a way that we may require auxiliary lemmas just to compensate for the restrictions caused by our mandatory markings (cf. Example 11 and Theorem 12). Auxiliary lemmas may be required for two reasons:

1) to find a proof at all (cf. Example 11); or

2) to improve the efficiency of proof search by introducing shortcuts in the search space. With auxiliary lemmas, proof search may be guided on a fine-grained level. But this burdens the user with having to pick suitable auxiliary lemmas.

Instead of introducing auxiliary lemmas that just compensate for the restrictions caused by mandatory markings, we may vary the mandatory markings heuristics. On the one hand, if we use only the strict mandatory markings heuristics, all proof steps locally contribute to the proof. On the other hand, if we use only the relaxed mandatory markings heuristics, we do not pose any restrictions on proof search at all. As a compromise, we introduce generous markings. With generous elements the default behavior

for marking mandatory elements in the subgoals as described in Heuristics 7 can be changed in a flexible way.

RESTRICTION 23 (CAUSED BY GENEROUS MARKING). — If a lemma with a *generous marking* is applied to a goal G, it causes the following restriction on the mandatory marking of a condition or rewrite subgoal SG:

If SG is generated from a generous element, then the mandatory marking of SG is inherited from G and supplemented with all the new elements of SG. Thus, the marking is generated with the relaxed marking heuristics used for rewrite subgoals in Heuristics 7.

If the corresponding element is not generous, then a new set of mandatory elements is introduced for SG consisting exactly of the new elements in SG. In this case, the marking is generated with the strict marking heuristics used for condition subgoals in Heuristics 7. □

HEURISTICS 24 (FOR MARKING ELEMENTS AS GENEROUS). — Our automatic default heuristics marks exactly the head literal of every rewrite lemma as generous. □

Note that, with Heuristics 24, the mandatory marking Heuristics 7 in Section 3.2 now works exactly as without a generous marking before.

Generous elements relax the restrictions caused by a mandatory marking. Therefore, we may avoid some auxiliary lemmas.

EXAMPLE 25 (11 CONTINUED). — If all literals in Lemmas (25) and (26) are generous, we may first apply these two lemmas. This results in the following clause:

(42) $\{\ \texttt{r2} = \texttt{true},\ \texttt{r3} = \texttt{true},\ \texttt{r1} = \texttt{true}\ \}$

Due to the generous marking, all literals in this clause are mandatory. Therefore, we can prove this clause in the same way as Lemma (31). □

Using the relaxed mandatory markings heuristics for generous elements clearly extends the search space. Therefore, one would expect that we get a less efficient relief test. Often, this holds true but, in general, it is not that easy to analyze the effects of generous elements on proof search. Generous elements enable more proof steps that do not locally contribute to the proof. For these proof steps, we do not know whether they contribute to the proof. Often, they are non-contributing. But, sometimes, they may enable additional proof steps that introduce shortcuts during proof search. They may avoid many failed proof attempts resulting in improved efficiency. Thus, generous elements may have similar effects on proof search as auxiliary lemmas: They may enable a proof at all and they may increase the efficiency of proof search. But they also extend the search space—just as auxiliary lemmas do—which may decrease the efficiency of proof search as well.

In general, the introduction of auxiliary lemmas allows us to guide proof search on a more fine-grained level than this can be done by marking elements as generous. In the former case, we may introduce just the lemma instance required for closing the

proof state tree, whereas, in the latter case, many lemmas may become applicable—most of them resulting in unsuccessful proof attempts. Therefore, auxiliary lemmas do not extend the search space as much as generous markings do. Thus, for efficiency reasons it is advantageous to use auxiliary lemmas. But generous markings relieve the user of the burden of picking these auxiliary lemmas. Thus, we recommend their use in a limited way for complicated proofs.

Often, the coarse-grained extension of the search space caused by generous elements introduces too many non-contributing proof steps which contain unnecessary proof obligations in terms of open goal nodes. These additional proof obligations countervail the benefits of the generous markings in such a way that, actually, the efficiency of proof search decreases when using generous markings on their own. As mentioned in Section 3.1, we may analyze and prune a performed proof by eliminating non-contributing proof steps with hindsight. The combination of generous markings with this pruning technique allows us

1) to search for a proof performing proof steps that do not locally contribute; and

2) to eliminate proof obligations of non-contributing proof steps.

Only in this combination, generous markings can display their full power. As the pruning techniques are outside the scope of this paper, we do not consider generous markings in more detail. Instead, we refer to [SCH 06].

Even in combination with pruning, generous markings should be used with caution. We recommend their use if a lemma (or a literal in a goal) is expected to be essential for the proof, i.e. a proof cannot be found without using the lemma (or the literal in the goal), and the proof itself is expected to be easy but not necessarily linear. Usually, we assume that these properties hold true for goals containing definedness atoms which are proved by applying corresponding domain lemmas. Therefore, we usually mark as generous:

– negated definedness atoms in all lemmas because they generate definedness atoms in the corresponding condition subgoals; and

– all literals in domain lemmas.

Furthermore, if an operator f is defined in terms of other function symbols using defining rules which are not recursive, we may decide to reduce terms containing operator f with its defining rules in any case. Then, the literals in the defining rules of operator f should be marked as generous.

3.6. *Further heuristics for guiding proof search*

To increase the performance of our proof control we apply further heuristics (cf. [BOY 88]):

– We prevent repeated applications of the same inference rule with the same principal literals within one proof attempt (disregarding cut-off literals): For the application

of an inference rule I to a goal G, we inspect all the applications on the branch of the proof state tree from the root goal to G.

In particular, when using generous markings for condition literals in lemmas, this mechanism is required for avoiding trivial rewrite loops: As the condition subgoal generated from a generous condition literal inherits the mandatory marking from its parent goal, the same inference step is applicable again.

– We do not apply lemmas that apparently do not support the proof of the goal. There may be two reasons for this: Firstly, the conditions of the lemma and the context of the goal are inconsistent, i.e. the context contains the negation of one condition. Secondly, the focus literal of the goal is rewritten to an obviously unsatisfiable literal as e.g. $t \neq t$.

– During an automatic proof attempt, we do not want to guess any instantiations of lemma variables. Thus, *extra* variables—i.e. variables that are not bound by matching the head literal to the focus literal—must be instantiated by matching condition literals to context literals.

– *Permutative* lemmas as e.g. the commutativity of + are applied only w.r.t. a fixed wellfounded total term order. By this, we hope to prevent infinite rewrite chains with permutative lemmas.

– To prevent infinite loops when proving applicability subgoals, the maximal recursion depth can be restricted.

4. Case studies

In this paper, we have presented novel heuristics to restrict the relief test for conditional lemmas. To validate our novel heuristics on some real case studies, we have to integrate them into an inductive proof process. We want to study solely the effects on proof search caused by the different heuristics based on markings. Therefore, we compare the different heuristics with as few differences as possible. Instead of comparing different systems that implement the various heuristics, we use the same proof process as well as the same specifications within a single system. Thus, we realize Contextual Rewriting as explained in Section 3.3.1. At the end of this section we will also point out how the results obtained by our simulations carry over to other provers.

We choose our inductive theorem prover QUODLIBET [AVE 03] to perform our simulations. It provides the following advantages: QUODLIBET strictly separates the automatic proof control implemented by tactics from the logic engine given by an inference system. The flexibility of the inference rules enables the simulation of the different heuristics easily. Note that systems based on Contextual Rewriting eliminate the focus literal from the condition subgoals. Thus, their underlying inference systems would have to be changed to simulate our novel heuristics. Last but not least, QUODLIBET provides various statistics to compare the different heuristics.

We summarize our inductive proof process that is influenced by [BOY 88]: The whole proof process is controlled by a so-called *database*. It stores information about

the *analysis* of defined operators and the *activated lemmas*. The analysis of a defined operator is used for performing an inductive case split automatically; instead of generating induction hypotheses at the beginning of the proof as in explicit induction, we may apply lemmas as induction hypotheses. These applications create an additional order subgoal to guarantee the wellfoundedness of the induction scheme. Only activated lemmas may be used within the simplification process. During the activation of a lemma the user may provide the head and the obligatory and generous literals of the lemma. Otherwise, they are determined by some heuristics (cf. Section 3.4 for obligatory, Section 3.5 for generous, and [SCH 04] for head literals).

The simplification process is divided into phases: The first phase proves simple tautologies, the second removes redundant literals, the third applies directly applicable lemmas, the fourth decomposes literals and applies lemmas even if they are not directly applicable, the fifth uses equalities for *cross-fertilization* [BOY 88]. During each phase the tactics use each literal *successively* as focus literal. This is justified since all inference rules add literals only to the *front* of the goal. Lemmas are tested in reverse activation order, which may be changed to influence the proof search. If a head literal is an equation (whose left-hand side is not a variable), it is used for *rewriting*; otherwise, for *subsumption*. Subsumption is checked for first; then the subterms of the focus literal are tested for rewriting, using an innermost left-to-right strategy. If a lemma can be applied and all its applicability subgoals can be proved, its application will not be deleted anymore. Thus, no alternative proof attempts for successful applications will be tried out during this tactic execution. Contrariwise, a lemma application—together with all proof attempts of the applicability subgoals—is deleted if the relief test fails. This results in a backtracking step. Further details can be found in [SCH 04].

For a fair evaluation of forbidden literals, we have slightly modified the automatic application of axioms during our simplification process when using forbidden literals: If axioms can be applied to the same redex alternatively (such as the axioms of the `gcd` in Example 1) a Cut with the condition literal(s) will be performed automatically. This then enables the application of all axioms. Otherwise, already the second application would be prevented because the rewrite literal becomes forbidden after the first application. This simulates the *operator unfolding* operation in systems like `NQTHM` where all axioms are given in one operator definition. Together with the additional admission of forbidden literals as cut-off literals, this results in a modeling where Contextual Rewriting can display its full power.

We compare the different heuristics in the following case studies whose details can be found at [SCH 05a]:

`sortalgos` This case study contains a collection of sorting algorithms such as bubblesort, insertionsort, mergesort and quicksort. We prove that the sorting algorithms return an ordered list which is a permutation of the input list.

Table 1. *Complexity of the case studies*

Example	Lemmas	Man. Interact.	Autom. Appl.	Del.	Fin. P.	Runtime
`sortalgos`	111	1 + 0	2233	40	2193	5.95
`gcd`	85	8 + 2	1114	13	1101	2.92
`sqrt` (H)	51	11 + 1	1062	14	1048	5.91
`sqrt` (E)	38	2 + 0	529	2	527	1.46
`exp-exhelp`	27	0 + 6	1278	116	1162	7.22
`Lpo`	154	5 + 67	5458	404	5054	36.89

`gcd` In this case study, we prove that the greatest common divisor of two natural numbers is associative, commutative and idempotent. For the proofs, we exploit dependencies between divisibility and order relations.

`sqrt` We prove that $\sqrt{2}$ is irrational, based on the ideas of Hippasus of Metapontum (H) and Euclid of Alexandria (E), respectively. In [WIE 03], the proof of the irrationality of $\sqrt{2}$ is used as a challenging problem for comparing 15 different theorem provers w.r.t. their ability to formalize and prove mathematics.

`exp-exhelp` This case study is taken from [KAP 96]. It states the equivalence of call-by-value and call-by-name evaluations for simple arithmetic expressions containing function calls.

`Lpo` In this case study, we prove that the lexicographic path order [KAM 80] is a simplification order [DER 87]. Furthermore, we prove the equivalence of different implementations of the `Lpo` [LÖC 04].

The last two examples are challenging as they contain mutually recursive operators. Table 1 illustrates the complexity of the examples. It contains in column

Lemmas the number of lemmas (constant for all heuristics); and

for our novel heuristics with mandatory and obligatory literals, in column

Man. Interact. the number of manual interactions (manually applied inference rules + manually chosen induction order);

Autom. Appl. the number i of automatically applied inference rules (including the later deleted ones);

Del. the number d of deleted inference rules due to a failed relief test;

Fin. P. the number of inference rules in the final proof, i.e. $i - d$; and

Table 2. *Comparison of the different heuristics*

Heur.	Open Lemmas	Autom. Appl.	Del.	Fin. P.	Runtime
Example `sortalgos`					
∅	2	2348	143	2205	6.77
{o}	0	2143	19	2124	5.56
{m}	0	2072	107	1965	5.53
{m,o}	0	2007	40	1967	5.09
{f}	0	2354	234	2120	6.31
{f,o}	0	2168	60	2108	5.13
Example `gcd`					
∅	—	—	—	—	—
{o}	0	911	5	906	2.13
{m}	—	—	—	—	—
{m,o}	0	893	9	884	2.14
{f}	—	—	—	—	—
{f,o}	**9**	974	18	956	2.07
Example `sqrt` (H)					
∅	1	2008	969	1039	16.11
{o}	0	1059	26	1033	5.98
{m}	0	1064	31	1033	6.22
{m,o}	0	1046	13	1033	5.85
{f}	0	1021	60	961	5.40
{f,o}	0	980	25	955	5.28
Example `sqrt` (E)					
∅	0	477	4	473	1.24
{o}	0	471	0	471	1.18
{m}	0	477	4	473	1.24
{m,o}	0	471	0	471	1.25
{f}	3	483	16	467	1.27
{f,o}	**3**	467	0	467	1.26
Example `exp-exhelp`					
∅	0	3290	9	3281	299.28
{o}	0	3290	9	3281	298.90
{m}	0	1278	116	1162	7.26
{m,o}	0	1278	116	1162	7.22
{f}	0	1342	204	1138	7.85
{f,o}	0	1342	204	1138	7.77
Example `Lpo`					
∅	1	11125	1089	10036	291.37
{o}	0	7621	848	6773	153.85
{m}	0	5746	971	4775	46.44
{m,o}	0	4988	330	4658	31.28
{f}	1	32946	26667	6279	370.79
{f,o}	**1**	8050	2135	5915	50.89

Runtime the runtime in seconds measured by a CMU COMMON LISP system on a machine with a 1 GHz Intel III processor and 4 GB RAM.

Table 2 contains for each example and each heuristics based on a combination of **o**bligatory, **m**andatory and **f**orbidden markings the following statistics: in column "Open Lemmas", the number of proof state trees that cannot be closed with this heuristics; the entries in the other columns take into account only those proof state trees that are closed with *all* heuristics. We do not count applications and deletions of inference steps of open proof state trees because failed proof attempts tend to create large proof state trees, tampering our results.

From the statistics in Table 2, we draw the following conclusions:

– Since the specification of `gcd` contains non-terminating rewrite rules, it can be performed only with an *obligatory* marking. For the other examples, obligatory markings restrict the search space without influencing the resulting proofs very much: The number of inference steps in the final proof is nearly the same regardless of the usage of obligatory markings.

– The *best heuristics w.r.t. efficiency* (as underlined in the table) use a combination of mandatory/obligatory **{m,o}** and forbidden/obligatory **{f,o}** markings, respectively. As the same simplification process is used, these two heuristics can differ only if a proof step cannot be applied due to the restrictions caused by mandatory or forbidden markings. Only for the `Lpo` example, a major advantage in efficiency can be determined, favoring our novel **{m,o}** heuristics.

– As printed in bold in the table, of 466 lemmas in total *13 lemmas cannot be proved* with **{f,o}**, but our novel **{m,o}** heuristics proves all of them.

Note that in our modeling of Contextual Rewriting with **{f,o}**, 13528 subgoals are created, but 1296 definedness and 648 condition subgoals—as well as the proof trees rooted in them—are cut-off by using forbidden literals as cut-off literals.

Finally, to answer the question about the adequacy of our simulation of Contextual Rewriting, we converted one of our case studies into a proof script for a prover based on Contextual Rewriting. We chose the `gcd` example as it contains most failed proof attempts with a forbidden marking. As prover we used `NQTHM` [BOY 88] because we did not want to use the decision procedures for linear arithmetics integrated in `ACL2`.[7] Instead, we used the *shell* principle to define our own type for natural numbers. We applied the following transformations to the original proof script: The specification style is changed from constructor to destructor recursion. Partial definitions are simulated using `F` as undefined value. As `NQTHM` is untyped, we explicitly restrict all lemmas to natural numbers only. These transformations were quite easy. Additionally, we added one operator definition just to provide a suitable induction scheme for the proof of one lemma as well as four auxiliary lemmas to enable the proof of two

7. Our marking techniques are also well suited for the integration of decision procedures [SCH 05b]. But in this paper, we focus on the application of conditional lemmas. Therefore, we have performed our case studies without considering decision procedures.

lemmas—namely Lemma (7) of Example 1 and a similar lemma that are proved in QUODLIBET by mutual induction. These are two of the nine lemmas that failed with our simulation of **{f,o}** in QUODLIBET. From the remaining seven lemmas, only two are not proved automatically. Note that this is not a weakness of our simulation of Contextual Rewriting. Instead, the difference is caused by different induction principles: NQTHM uses explicit induction. Thus, it does not apply lemmas inductively but splits, at the beginning of a proof, the induction steps of conditional lemmas in different cases and immediately adds a promising induction hypothesis. In contrast to this, we use descente infinie (cf. [WIR 04]). Therefore, we have to use the relief test more often in QUODLIBET. Beside the failed proofs in the statistics, two lemmas proved by our novel heuristics with simplification are proved by induction in NQTHM. Thus, these proofs are more complex in NQTHM.

5. Conclusion

Rewriting with conditional lemmas is at the heart of many (inductive) theorem provers. Especially for interactive theorem provers, it is essential not only to prove as many lemmas automatically as possible but also to restrict proof search in a suitable way such that the proof process stops within a reasonable amount of time.

In this paper, we have developed a framework that allows us to restrict proof search in a flexible way using heuristics based on markings in goals and lemmas. Within our framework we can simulate Case Rewriting and Contextual Rewriting with a forbidden marking in goals. The adequacy of our simulation of Contextual Rewriting is demonstrated by carrying over a case study to NQTHM. Furthermore, we have developed a novel heuristics **{m,o}** based on the orthogonal concepts of a mandatory marking in the goals and an obligatory marking in the lemmas. For the comparison of the heuristics we chose the well established application domain of rewrite-based simplification in inductive theorem proving. Our simulation of Contextual Rewriting **{f,o}** is competitive with our novel heuristics **{m,o}** regarding efficiency but not regarding extent. Nevertheless, the benefits of our novel heuristics are slightly decreased in theorem provers using explicit induction because they do not perform a relief test for induction hypotheses.

Neither Case Rewriting nor Contextual Rewriting nor our novel heuristics are perfect. For all of them, we have identified proof patterns that cannot be handled with the basic version of these heuristics. For our novel heuristics, we can always overcome these difficulties using auxiliary lemmas or a generous marking in the lemmas relaxing the restrictions caused by a mandatory marking. This is not possible e.g. for Contextual Rewriting. Furthermore, our framework allows us to choose between the different heuristics and to combine them easily. With obligatory and generous markings in lemmas we can fine-tune the degree of extent and efficiency of the proof search manually.

Our framework depends only on the partitioning of goals into principal part, cut-off part and context according to the inference rule applied. The basic distinction

between principal part and context was already introduced in Gentzen's seminal work on sequent calculi [GEN 35]. Therefore, this partitioning (and also the refinement with cut-off formulas) should pose no problems for inference systems based on sequent calculi. Indeed, a similar form of lemma application occurs in all practice-oriented mathematical assistance systems and the concepts behind our marking as mandatory, forbidden, obligatory, and generous are all in great demand and applicable, provided that we extend the inheritance procedures to the new inference rules in a meaningful way. As explained in Section 4, systems based on Contextual Rewriting eliminate the focus literal from the condition subgoals. Thus, their underlying inference systems have to be changed as a prerequisite for the integration of our marking techniques. This may require significant technical effort. Then, the marking techniques may be realized using wrapper functions for the inference rules as it is done in QUODLIBET.

The partitioning of a goal into a principal part, a cut-off part and a context can be further exploited to determine the contribution of performed proofs. We elaborate on this topic in [SCH 06]. The information about the contribution of a proof can be used for extracting a pruned proof, determining superfluous literals in goals, and enhancing the reusability of subproofs. In the future, we will investigate the combination of the different marking heuristics and the pruning methods in more detail. The combination should lead to a more extensive relief test. In this way, we hope to reduce the number of auxiliary lemmas and backtracking steps required. The pruning techniques made available by the contribution of proofs enhance efficiency by eliminating superfluous proof steps.

Acknowledgements

First of all, I would like to thank Claus-Peter Wirth for encouraging me to write this paper, for his patience and the effort he made by proof-reading and improving numerous versions of this paper. I owe more to Claus-Peter than I can express here. Furthermore, I would like to thank Jürgen Avenhaus for improving the readability of the paper with his suggestions, Bernd Löchner and the anonymous referees for helpful comments on earlier drafts of this paper, and my wife Katja for pointing this interesting special issue out to me.

6. References

[ARM 03] ARMANDO A., RANISE S., "Constraint contextual rewriting", *J. Symb. Comput.*, vol. 36, num. 1-2, 2003, p. 193–216.

[AVE 03] AVENHAUS J., KÜHLER U., SCHMIDT-SAMOA T., WIRTH C.-P., "How to Prove Inductive Theorems? QUODLIBET!", BAADER F., Ed., *CADE*, vol. 2741 of *LNCS*, Springer, 2003, p. 328–333.

[BOU 93] BOUHOULA A., RUSINOWITCH M., "Automatic Case Analysis in Proof by Induction", *IJCAI*, 1993, p. 88–94.

[BOY 88] BOYER R. S., MOORE J. S., *A Computational Logic Handbook*, Academic Press Professional, Inc., 1988.

[BUN 93] BUNDY A., STEVENS A., VAN HARMELEN F., IRELAND A., SMAILL A., "Rippling: A Heuristic for Guiding Inductive Proofs", *Artif. Intell.*, vol. 62, num. 2, 1993, p. 185–253.

[DER 87] DERSHOWITZ N., "Termination of Rewriting", *J. Symb. Comput.*, vol. 3, num. 1/2, 1987, p. 69–116.

[GEN 35] GENTZEN G., "Untersuchungen über das logische Schließen", *Mathematische Zeitschrift*, vol. 39, 1934/35, p. 176–210 and 405–431.

[KAM 80] KAMIN S., LEVI J.-J., "Two generalizations of the recursive path ordering", report , 1980, Dep. of Computer Science, University of Illinois, Urbana, IL, Unpublished note.

[KAP 96] KAPUR D., SUBRAMANIAM M., "Automating Induction over Mutually Recursive Functions", WIRSING M., NIVAT M., Eds., *AMAST*, vol. 1101 of *LNCS*, Springer, 1996, p. 117–131.

[KAU 00] KAUFMANN M., MANOLIOS P., MOORE J. S., *Computer-Aided Reasoning: An Approach*, Kluwer Academic Publishers, 2000.

[KOU 90] KOUNALIS E., RUSINOWITCH M., "Mechanizing Inductive Reasoning", *AAAI*, 1990, p. 240–245.

[LÖC 04] LÖCHNER B., "Things to know when implementing LPO", SCHULZ S., SUTCLIFFE G., TAMMET T., Eds., *Proceedings of the 1st Workshop on Empirically Successful First Order Reasoning (ESFOR '04)*, ENTCS, Elsevier, 2004, Extended version to appear in *International Journal on Artificial Intelligence Tools*.

[SCH 04] SCHMIDT-SAMOA T., *The New Standard Tactics of the Inductive Theorem Prover* QUODLIBET, SEKI-Report SR–2004–01 (ISSN 1437–4447), SEKI, Saarland Univ., 2004, `http://www.ags.uni-sb.de/~cp/p/sr200401/welcome.html`.

[SCH 05a] SCHMIDT-SAMOA T., "How to Prove Inductive Theorems? QUODLIBET!", `www-avenhaus.informatik.uni-kl.de/quodlibet/`, 1999–2005, Homepage of the inductive theorem prover QUODLIBET.

[SCH 05b] SCHMIDT-SAMOA T., "An Even Closer Integration of Linear Arithmetic into Inductive Theorem Proving", CARETTE J., FARMER W. M., Eds., *Calculemus*, ENTCS, 2005, To appear.

[SCH 06] SCHMIDT-SAMOA T., "Flexible Heuristic Control for Combining Automation and User-Interaction in Inductive Theorem Proving", PhD thesis, Tech. Univ. Kaiserslautern, 2006, submitted.

[WIE 03] WIEDIJK F., "Comparing Mathematical Provers", ASPERTI A., BUCHBERGER B., DAVENPORT J. H., Eds., *MKM*, vol. 2594 of *LNCS*, Springer, 2003, p. 188–202.

[WIR 04] WIRTH C.-P., "Descente Infinie + Deduction", *Logic Journal of the IGPL*, vol. 12, num. 1, 2004, p. 1–96, Oxford University Press, `http://www.ags.uni-sb.de/~cp/p/d/welcome.html`.

[ZHA 95] ZHANG H., "Contextual Rewriting in Automated Reasoning", *Fundam. Inform.*, vol. 24, num. 1/2, 1995, p. 107–123.