Mandatory versus Forbidden Literals in Simplification with Conditional Lemmas

Tobias Schmidt-Samoa

FB Informatik, Tech. Univ. Kaiserslautern, Germany
`schmidt@informatik.uni-kl.de`

Abstract. Due to its practical importance, context-dependent simplification with conditional lemmas has been studied for three decades, mostly under the label "contextual rewriting". We develop novel heuristics restricting the relief of conditions with mandatory literals in the goals and obligatory literals in the lemmas. Our case studies in the field of rewrite-based inductive theorem proving are encouraging.

1 Introduction

We are concerned with 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.

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 successful 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 [5]. 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 (see Section 3).

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 [11], 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.*

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.* Furthermore, the instantiated condition literals have to be fulfilled in the "context": 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 condition 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 [6], 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.

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` [4] and `ACL2` [8] use the cross fertilization technique while `RRL` [11] uses a constant congruence closure algorithm. In `RDL`[1], decision procedures can be used by the simplification process.

The literals 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 will concentrate on the question *which* literals can be used during the relief test.

Practically, there are two major ways to restrict the literals in the relief test by marking literals in the goal clause:

---

[1] If we use partially defined operators or apply a lemma as induction hypothesis, additional *definedness* and *order subgoals* have to be created. A lemma may also be applied for *subsumption* if the head literal matches the focus literal, resulting in the same subgoals without the rewrite subgoal.

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

1. In previous approaches known from the literature such as Contextual Rewriting in [11] and Case Rewriting in [3], literals are excluded from certain condition subgoals. This can be modeled by marking these literals as *forbidden*. The meaning is that a forbidden literal of a goal must not be principal in the application of an inference rule (unless it is a cut-off literal).
2. We, however, will mark subgoal literals as *mandatory*: If we apply an inference rule to a goal, one of the mandatory literals must be principal.

By marking literals as mandatory instead of forbidden, we overcome some difficulties of previous approaches [11, 3]: As we can use every literal during the relief test provided that there is also one mandatory literal 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 literals* to gain efficiency.

2 A Simple Example

The following example presents a proof pattern that can be handled by our novel heuristics but cannot be proved by previous approaches summarized in Section 3.2. 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 unequal to zero). We present our examples in the style of our inductive theorem prover QuodLibet [2]. It 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.

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. Due to lack of space we only consider the formal specification of `gcd` given by Axioms (1) to (4). The `gcd` of two natural numbers is defined iff one of its arguments is unequal to zero. If exactly one of the arguments is unequal to 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) $\{\ \mathtt{gcd}(x,y) = x,\ y \neq \mathtt{0},\ x = \mathtt{0}\ \}$
(2) $\{\ \mathtt{gcd}(x,y) = y,\ x \neq \mathtt{0},\ y = \mathtt{0}\ \}$
(3) $\{\ \mathtt{gcd}(x,y) = \mathtt{gcd}(x, \mathtt{-}(y,x)),\ \mathtt{leq}(x,y) \neq \mathtt{true},\ x = \mathtt{0},\ y = \mathtt{0}\ \}$
(4) $\{\ \mathtt{gcd}(x,y) = \mathtt{gcd}(\mathtt{-}(x,y), y),\ \mathtt{leq}(x,y) = \mathtt{true},\ \neg\mathtt{def}\ \mathtt{leq}(x,y),\ x = \mathtt{0},\ y = \mathtt{0}\ \}$

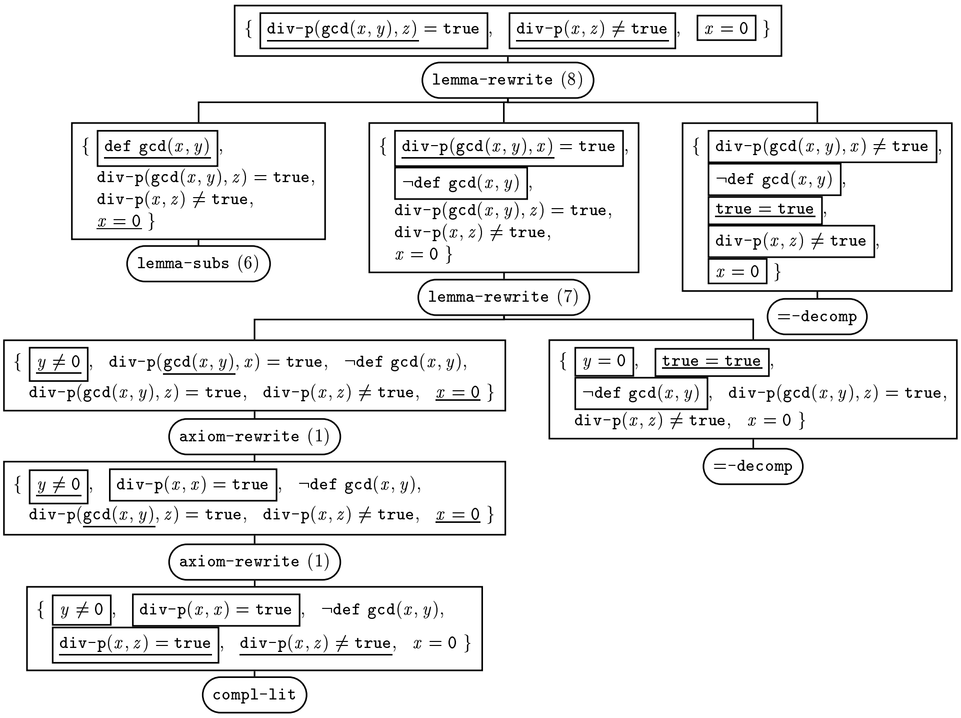


Fig. 1. Proof State Tree for Goal (5) of Example 1

As auxiliary lemma for the associativity of gcd, we want to prove:

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

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

(6) $\{ \texttt{def gcd}(x,y),\ x = 0 \}$
(7) $\{ \texttt{div-p}(\texttt{gcd}(x,y),x) = \texttt{true},\ y = 0 \}$
(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. 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 atom in a Horn clause.

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 $[x \leftarrow \texttt{gcd}(x,y), z \leftarrow z, y \leftarrow x]$. 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* variable y can be bound by matching the third lemma literal to the second goal literal (see Section 3.3). 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):[3] one *definedness subgoal* (since the substitution binds a 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 only possible with our novel heuristics (see 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. □

3 Controlling the Application of Conditional Lemmas

Lemmas are provided to guide the proof process. On the one hand, they should be checked for their contribution to a proof intensively 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. Note that in our domain neither confluence nor termination properties can be assumed for rewriting with lemmas.

3.1 The Mandatory Literals Heuristics

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. We aim at avoiding non-contributing proof steps.

[3] 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.

Definition 1 (Contribution of a Proof Step / Literal).
A proof step S of proof P for goal G *contributes* to P if every direct subgoal G' created by S contains a new literal that contributes to the proof for G' in P. A literal of a goal G *contributes* to proof P for G if it becomes principal for one contributing proof step S 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 captures what we want but is too inefficient: As contribution of a proof step depends on the proof performed, it can only be *checked* after the proof has been completed. But we can easily *ensure* that we perform only contributing proof steps by using one of the new literals as principal literal in the *next* proof step. To be able to define local restrictions on proof steps in a flexible way, we introduce a mandatory marking on goal literals.

Restriction 1 (Caused by Mandatory Literals)
An inference rule may only be applied to a goal G with *mandatory literals* if one of the mandatory literals is principal in this proof step.

If we mark only new literals in a subgoal as mandatory in a proof, it is ensured that all proof steps contribute to that proof. But then, the proof search is too restrictive. It will only find "linear" proofs: We can only apply those inference rules that also use new literals 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 mandatory literals as the following example illustrates.

Example 2. Given three boolean valued constants $\mathtt{p1}, \mathtt{p2}, \mathtt{p3}$, we assume the activation of the following lemmas

(9) $\{\ \mathtt{p1} = \mathtt{p3}\ \}$ (10) $\{\ \mathtt{p2} = \mathtt{p3}\ \}$

and want to prove goal

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

The only way to prove Goal (11) is to rewrite $\mathtt{p1}$ and $\mathtt{p2}$ to $\mathtt{p3}$. Then the resulting subgoal is tautological as it contains complementary literals. But if we mark only new literals as mandatory, this proof is prohibited since the second rewrite step does not use a new literal. □

Alternatively, if all literals are mandatory, the search space contains too many proof steps that do not contribute to the proof. Our compromise is the following:

Heuristics 1 (Marking Mandatory Literals)
At the beginning of a proof attempt for a lemma every literal 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—the literals of the parent goal are not marked in the subgoal, but a new set of mandatory literals is introduced that consists exactly of the new literals of the subgoal. For order subgoals, we mark only the single new order atom as mandatory. For all other subgoals—i.e. rewrite subgoals or subgoals created by other inference rules—however, the mandatory literals of the parent goal stay mandatory in the subgoal and are supplemented with all new literals of the subgoal.

With this heuristics we can perform rewrite steps even if they do not contribute to the proof. This is also helpful for the speculation of auxiliary lemmas.

Example 1 (continued). In Figure 1, mandatory literals are framed, principal literals are underlined. Thus, we can only apply an inference rule 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 (see Heuristics 1). 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. □

Example 3 extracts the proof pattern from Example 1 that cannot be proved with Contextual Rewriting (see Section 3.2) but with our novel heuristics.

Example 3 ([11], simplified). Given three boolean valued constants $\mathtt{q1}, \mathtt{q2}, \mathtt{q3}$, we assume the activation of the following lemmas

(12) $\{\ \mathtt{q1} = \mathtt{true},\ \mathtt{q2} \neq \mathtt{true}\ \}$

(13) $\{\ \mathtt{q1} = \mathtt{true},\ \mathtt{q3} \neq \mathtt{true}\ \}$

(14) $\{\ \mathtt{q2} = \mathtt{true},\ \mathtt{q3} = \mathtt{true}\ \}$

and want to prove goal

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

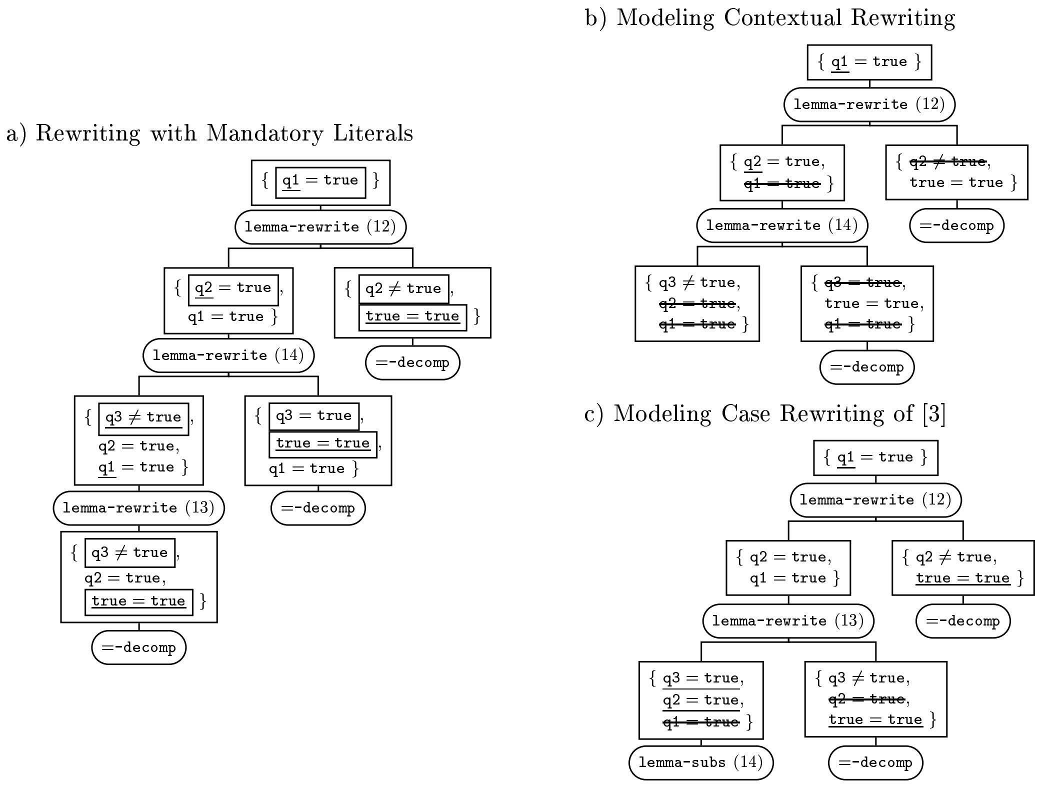


Fig. 2. Proof State Trees for Example 3

As Lemmas (12) and (13) are Horn clauses we use the first literal as head literal. Lemma (14) 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 activate just the first one.

Using mandatory literals, the proof is found automatically (see Figure 2a). In the condition subgoal after applying Lemma (14), literal $\texttt{q1} = \texttt{true}$ 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 literals then it can also be proved obeying the restrictions caused by mandatory literals. Unfortunately, this strong property does not hold as will be shown in Example 4, a simple generalization of Example 3.

The interaction of head literals in lemma clauses and mandatory literals in goal clauses restricts the search space of the simplification process drastically: In most cases, a lemma will only be applied to a goal if the head literal of the lemma is mandatory in the goal. The proof step then transfers the mandatory marking from the head literal to its condition literals. This direction can only be inverted automatically 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 3. Otherwise, we have to use auxiliary lemmas to bridge the gap.

Example 4. Given five boolean valued constants $\mathtt{r1}, \ldots, \mathtt{r5}$, we assume the activation of the following lemmas

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

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

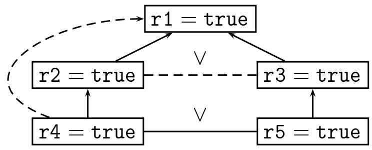
(18) $\{\ \mathtt{r2} = \mathtt{true},\ \mathtt{r4} \neq \mathtt{true}\ \}$

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

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

and want to prove the goal

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



The specification is illustrated in the diagram 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 $\mathtt{r4} = \mathtt{true}$ that cannot be closed automatically. We can overcome this situation by introducing e.g. one of the following auxiliary lemmas (illustrated in the diagram by dashed lines):

(22) $\{\ \mathtt{r2} = \mathtt{true},\ \ \mathtt{r3} = \mathtt{true}\ \}$

(23) $\{\ \mathtt{r1} = \mathtt{true},\ \ \mathtt{r4} \neq \mathtt{true}\ \}$

Each of these lemmas as well as Lemma (21) (after activating one of (22), (23)) can be proved automatically. Thus, we can bridge the gap. □

Theorem 1 states that we can always close gaps with auxiliary lemmas.

Theorem 1. *If a goal can be proved by simplification without any restrictions on literals then it can also be proved with mandatory literals with the help of some auxiliary lemmas which themselves can be proved with mandatory literals. 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 literals 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 every new literal as mandatory (unless the goal is an order subgoal), each subgoal contains at least one mandatory literal. If a goal in the proof violates the restrictions caused by mandatory literals 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 mandatory literals. Whatever head literal is chosen for this lemma, it can be applied to prove the goal that formerly violated the restrictions caused by mandatory literals. □

Although the required auxiliary lemmas cannot be calculated automatically, they 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 you cannot build a bridge when a bank is forbidden.

3.2 Alternative Approaches in the Literature

In our setting, it is possible to model alternative approaches known from the literature to perform the relief test by marking literals as forbidden.

Restriction 2 (Caused by Forbidden Literals)
An inference rule may only be applied to goal G with *forbidden literals* if all forbidden literals that are principal in this proof step are cut-off literals.

Once a literal is marked as forbidden in a goal G, it remains forbidden in the whole proof attempt for G. In fact, the approaches in the literature do not even create these literals in the corresponding subgoals. Thus our modeling with forbidden literals improves the versions in the literature insofar as forbidden literals may serve as cut-off literals.

Contextual Rewriting in the narrower sense is used e.g. in `NQTHM` [4], `ACL2` [8], `RRL` [11], and more recently in `RDL` [1]. As explained in Section 1, these approaches vastly differ in their simplification process. Nevertheless, they use the same literals to do 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 3 (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 2b 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 any more. It is not possible to overcome this situation with auxiliary lemmas. □

Example 1 (continued). The second application of Axiom (1) rewrites a literal initially used as focus literal. Thus, Contextual Rewriting fails. □

Case Rewriting tries to overcome the difficulties of Contextual Rewriting by a special treatment of lemmas that can be applied in parallel to rewrite a term, such as e.g. Lemmas (12) and (13) in Example 3. In this sense, our approach with mandatory literals is a novel form of Case Rewriting. There are at least two further approaches known from the literature [3, 9].

The approach for Case Rewriting proposed in [9] restricts the relief test by order constraints which we cannot use for our application domain as we allow non-terminating operator definitions.

In the approach of Case Rewriting in [3], 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 3 (continued). For the specification of Example 3, this Case Rewriting approach can be modeled by applying the two lemmas in succession as depicted in Figure 2c. In the well-coveredness goal—i.e. the left-most goal—only the condition literals may be used. As the case split for Lemmas (12) and (13) is complete according to Lemma (14), the proof can be completed. □

The approach of [3] 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 only sufficient if all lemmas differ only in a single condition literal. The well-coveredness goal cannot be proved if the case split is incomplete.

Example 1 (continued). The applications of the lemmas and axioms do not form a complete case split. Actually, they even do not rewrite the same subterm. Thus, the proof fails. In contrast to this approach, our approach with mandatory literals can make use of other goal literals to prove the well-coveredness subgoal. □

3.3 Getting Fit for Practice: Obligatory Literals and Other Pretests

On the one hand, the use of mandatory literals as explained in Section 3.1 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* relief test because it is only syntactical. As a compromise, we introduce an obligatory marking on lemma literals that restricts the relief test for obligatory literals 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 3 (Caused by Obligatory Literals)
A lemma with *obligatory literals* may only be applied to a goal G if all obligatory literals are directly fulfilled in the goal clause.

Using obligatory literals may prevent the automatic derivation of proofs. Thus, literals 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 literals, the relief test would be invoked too often in this case, most of the time unsuccessfully.

Heuristics 2 (Marking Obligatory Literals)
Our automatic default heuristics chooses one obligatory literal if the lemma is used as rewrite rule with a general term as left-hand side.

Example 5. If we use the first literal of the trichotomy of less

(24) $\{\ \texttt{less}(x, y) = \texttt{true},\ \ \texttt{less}(y, x) = \texttt{true},\ \ x = y\ \}$

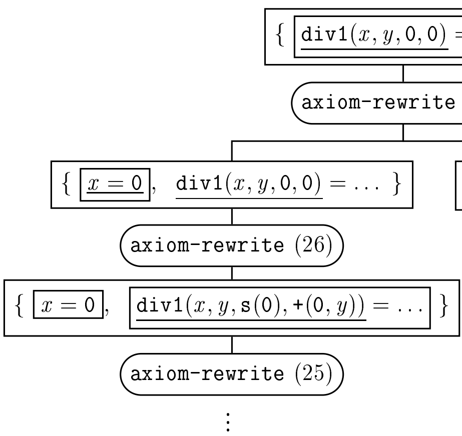
as head literal then it is activated as rewrite rule for operator less with a general term as left-hand side. When applying this lemma, the relief test reduces the question whether x is less than y to the question whether y is not less than x and whether they are unequal. But this relief test is in most cases not simpler than the original problem. This usually increases the number of useless inference steps. On the other hand, 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 (24). 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 6. Another example is given by the axioms of a division operator:

(25) $\{\ \texttt{div1}(x, y, u, v) = u,\ \ x \neq v\ \}$

(26) $\{\ \texttt{div1}(x, y, u, v) = \texttt{div1}(x, y, \texttt{s}(u), \texttt{+}(v, y)),\ \ x = v\ \}$

Without using obligatory literals the following relief test does not terminate:



During the relief test for Axiom (25), Axiom (26) can be applied for rewriting because the mandatory literal directly fulfills the condition literal of Axiom (26). 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. □

To increase the performance of our proof control we use further pretests:

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

– 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 only applied 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. Instead of comparing different systems that implement the various heuristics, we isolate their effects by using the same proof process as well as the same specifications within a single system. Thus, we realize Contextual Rewriting as explained in Section 3.2. 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 [2] 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 allows 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 system has 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 [4]: 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 obligatory literals of the lemma. Otherwise, they are determined by some heuristics (see Section 3.3 for obligatory and [10] 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* [4]. 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 [10].

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 in parallel (such as 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 [13]: a bunch of sorting algorithms (`sortalgos`); properties of the `gcd` such as associativity; two proofs that $\sqrt{2}$ is irrational, based on the ideas of Hippasos of Metapont (H) and Euclid of Alexandria (E), respectively; a proof that the lexicographic path order `Lpo` is a simplification order; an example

Example	Lemmas	Man. Interact.	Autom. Appl.	Deletions	Runtime
`sortalgos`	111	1 + 0	2213	57	4.65
`gcd`	85	8 + 2	1118	18	2.26
$\sqrt{2}$ (H)	51	11 + 1	1004	27	5.05
$\sqrt{2}$ (E)	38	2 + 0	535	11	1.13
`Lpo`	147	5 + 67	5950	1087	37.47
`exp-exhelp`	27	0 + 6	1368	276	10.81

Table 1. Complexity of the Case Studies

`exp-exhelp` taken from [7] stating the equivalence of call-by-value and call-by-name evaluations for simple arithmetic expressions containing function calls. The last two examples contain mutually recursive operators. Table 1 illustrates the complexity of the examples. It contains the number of lemmas (constant for all heuristics), and, for our novel heuristics with mandatory and obligatory literals, the number of manual interactions (manually applied inference rules + manually chosen induction order), the number of automatically applied inference rules (including the later deleted ones), the number of deleted inference rules due to a failed relief test and the runtime in seconds measured by a CMU Common Lisp system on a machine with a 1330 MHz AMD processor and 512 MB RAM.

Example	Heur.	Open Lemmas	Autom. Appl.	Deletions	Fin. Proof	Runtime
`sortalgos`	∅	2	2425	48	2377	5.66
	{o}	0	2222	28	2194	4.59
	{m}	0	2106	137	<u>1969</u>	4.43
	{m,o}	0	<u>2031</u>	57	1974	<u>4.10</u>
	{f}	0	2398	208	2190	4.99
	{f,o}	0	2238	55	2183	4.25
`gcd`	∅	—	—	—	—	—
	{o}	0	914	8	906	1.68
	{m}	—	—	—	—	—
	{m,o}	0	<u>902</u>	14	<u>888</u>	<u>1.63</u>
	{f}	—	—	—	—	—
	{f,o}	**9**	961	18	943	<u>1.63</u>
$\sqrt{2}$ (H)	∅	1	2291	1038	1253	19.75
	{o}	0	1294	47	1247	7.46
	{m}	0	1014	52	962	5.80
	{m,o}	0	988	26	962	5.00
	{f}	0	1026	62	964	4.80
	{f,o}	0	<u>985</u>	27	<u>958</u>	<u>4.48</u>
$\sqrt{2}$ (E)	∅	0	521	18	503	1.08
	{o}	0	501	0	501	1.00
	{m}	0	497	27	470	1.05
	{m,o}	0	<u>477</u>	9	<u>468</u>	0.97
	{f}	3	496	16	480	0.97
	{f,o}	**3**	480	0	480	<u>0.91</u>
`Lpo`	∅	5	22835	10879	11956	472.37
	{o}	2	10931	3395	7536	184.78
	{m}	2	5907	1430	4477	44.78
	{m,o}	0	<u>5263</u>	924	<u>4339</u>	<u>30.25</u>
	{f}	2	10750	3668	7082	64.59
	{f,o}	**1**	7484	876	6608	39.03
`exp-exhelp`	∅	1	2620	9	2611	84.49
	{o}	1	2620	9	2611	85.24
	{m}	0	1122	232	890	8.31
	{m,o}	0	1122	232	890	8.32
	{f}	0	<u>1051</u>	174	<u>877</u>	<u>5.87</u>
	{f,o}	0	<u>1051</u>	174	<u>877</u>	<u>5.87</u>

Table 2. Comparison of the Different Heuristics

Table 2 contains for each example and each heuristics based on a combination of **o**bligatory, **m**andatory and **f**orbidden literals the following statistics: in column "Open Lemmas", the number p of proof state trees that cannot be closed with this heuristics; and for all proof state trees that are closed with *all* heuristics:

i**:** in column "Autom. Appl.", the number i of inference steps applied automatically;
d**:** in column "Deletions", the number d of deleted inference steps;
f**:** in column "Fin. Proof", the number f of inference steps in the final proof, i.e. $i - d$;
r**:** and in column "Runtime", the runtime r in seconds.

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

The *best heuristics w.r.t. i and r* (as underlined in the table) use a combination of mandatory/obligatory **{m,o}** and forbidden/obligatory **{f,o}** literals, 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 literals. The restrictions caused by forbidden literals can be checked slightly more efficiently than that caused by mandatory literals: for forbidden literals, we do not have to consider inference rules whose focus literal is forbidden; but for mandatory literals, we cannot exclude inference rules whose focus literal is not mandatory since there may be other principal literals which are mandatory. Therefore, a few more inference rules can be applied and deleted in the some runtime for heuristics **{f,o}** in comparison to heuristics **{m,o}** in the `gcd` example. But 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 459 lemmas in total *13 lemmas cannot be proved* with **{f,o}**, but our novel **{m,o}** heuristics is the only one that proves all of them.

In our modeling of Contextual Rewriting with **{f,o}**, 12761 subgoals are created, but 1165 definedness and 263 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 forbidden literals. As prover we used `NQTHM` [4] because we did not want to use the decision procedures for linear arithmetics integrated in ACL2. 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 lemmas—namely Lemma (7) and a similar lemma that are proved in QuodLibet by mutual induction. These are two of the lemmas that failed with our simulation in QuodLibet. From the remaining seven lemmas that failed with our simulation in QuodLibet 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 does not apply lemmas inductively but splits, at the beginning of a proof, the induction steps of conditional lemmas in different cases for each hypothesis and the conclusion of the lemma. 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 with induction in NQTHM. Thus, these proofs are more complicated in NQTHM.

5 Conclusion

We have developed a novel heuristics **{m,o}** for the relief test of conditional lemmas, based on the orthogonal concepts of mandatory literals in the goals and obligatory literals in the lemmas. We have identified patterns that can be proved with our novel heuristics only. For 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. Our simulation of Contextual Rewriting seems to be adequate as demonstrated by carrying over a case study to NQTHM. 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. In the future, we will investigate more flexible, manually controlled heuristics for marking mandatory literals. In this way, we hope to reduce the number of auxiliary lemmas and backtracking steps required.

References

1. A. Armando and S. Ranise. *Constraint contextual rewriting.* J. of Symb. Comp., 36(1-2): 193–216, 2003.

2. J. Avenhaus, U. Kühler, T. Schmidt-Samoa, and C.-P. Wirth. *How to prove inductive theorems? QuodLibet!* 19^{th} CADE, LNAI 2741, pp. 328–333. Springer, 2003.
3. A. Bouhoula and M. Rusinowitch. *Automatic Case Analysis in Proof by Induction.* 13^{th} IJCAI, volume 1, pp. 88–94, August 1993. Morgan Kaufmann.
4. R. S. Boyer and J S. Moore. *A Computational Logic Handbook.* Academic Press, 1988.
5. A. Bundy, A. Stevens, F. van Harmelen, A. Ireland, and A. Smaill. *Rippling: A heuristic for guiding inductive proofs.* Artificial Intelligence, 62(2):185–253, 1993.
6. G. Gentzen. *Untersuchungen über das logische Schließen.* Mathematische Zeitschrift, 39:176–210, 405–431, 1934f.
7. D. Kapur and M. Subramaniam. *Automating induction over mutually recursive functions.* 5^{th} AMAST, LNCS 1101, pp. 117–131, 1996.
8. M. Kaufmann, and P. Manolios, J S. Moore. *Computer-Aided Reasoning: An Approach.* Kluwer Academic Publishers, 2000.
9. E. Kounalis and M. Rusinowitch. *Mechanizing inductive reasoning.* 8^{th} AAAI, pp. 240–245, 1990.
10. T. Schmidt-Samoa. *The new standard tactics of the inductive theorem prover QuodLibet.* SEKI Report SR-2004-01, Universität des Saarlandes, 2004. `www.ags.uni-sb.de/~veire/SEKI/2004/SR-2004-01/`
11. H. Zhang. *Contextual rewriting in automated reasoning.* Fundamenta Informaticae, 24(1/2):107–123, 1995.
12. H. Zhang, D. Kapur, and M. S. Krishnamoorthy. *A mechanizable induction principle for equational specifications.* 9^{th} CADE, LNCS 310, pp. 162–181. Springer, 1988.
13. `www-avenhaus.informatik.uni-kl.de/quodlibet/`