Reusing Contributing Proofs in Reductive Inference Systems Working on Proof Trees

Tobias Schmidt-Samoa

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

Abstract. We consider reductive inference systems, where a goal is reduced to a conjunction of subgoals, such as sequent calculi. On the one hand, the search space is not only infinite, but grows dramatically with the depth of the search. On the other hand, checking and analyzing given proofs is very efficient. As the search space also has a high redundancy, it is therefore reasonable to reuse subproofs found during the search. Besides reuse (1) in parallel parts of the proof (lemmatization), it is possible to reuse (2) a proof for a subgoal for solving the original goal itself. To be able to reuse proofs in both ways, we introduce new notions for the contribution of proof steps to a proof. These notions allow the derivation of pruned proofs and the identification of superfluous subformulas in theorems. The case studies in the sequent-based inductive theorem prover QuodLibet show substantial speed-ups, especially for critically time-consuming proofs.

1 Introduction

We are interested in the automation of (inductive) theorem proving for clausal first-order logic performed on proof trees. Given a specification `spec` and a clause Γ, we want to prove the validity of Γ in a suitable class of models of `spec`. We assume that we perform proofs within a sound inference system. Each inference rule reduces a *goal*[1] to a (possibly empty) sequence of new subgoals. The application of an inference rule is called a *proof step*. A proof is represented by a *proof tree* consisting of *goal* and *inference nodes*. The root goal node of a proof tree consists of a goal containing the clause Γ to be proved. An inference node represents the inference rule applied to its parent which is a goal node. Its n children ($n \geq 0$) are again goal nodes and represent the new subgoals created by the inference rule.[2] A proof tree is *closed* if all its leaves are inference nodes. In this case, the soundness of the inference rules guarantees that Γ is valid provided that all applied lemmas are valid. For brevity, we call a closed proof tree simply a *proof*.

Primarily, the automatic proof control *searches* for a proof by applying inference rules to goals based on heuristics. This proof search is often very time-consuming. Since we do not have perfect heuristics, we cannot decide a priori

[1] The use of the notion *goal* emphasizes that the inference system may depend on additional information beside the clauses themselves.

[2] For brevity, we will abuse the notions e.g. by calling goal nodes just goals.

whether a proof step will contribute to a proof, is superfluous, or—even worse—prevents a successful proof. In the latter case, we have to backtrack to revise a former proof step. Contrariwise, if we can *reuse* an old proof, we are able to perform only contributing proof steps without any backtracking. Therefore, reuse is much more efficient than proof search. But the reuse capabilities are often rather limited. This paper may be seen as a contribution to enhance these capabilities.

Usually, *arbitrary* proofs are searched for. *Minimal* proofs are not considered unless proofs are presented to human users or proof checkers. This seems to be reasonable because finding minimal proofs may be more difficult than finding arbitrary proofs at all. Nevertheless, minimal proofs may indicate that some of the premises in Γ are not needed for the validity of Γ. Eliminating these unused literals may deliver a stronger theorem with higher potential for reuse. This is especially useful for lemmas that have been speculated automatically.

In this paper, we define new notions for *contributing proof steps* and *contributing elements* in a goal which mutually depend on each other. Instead of *searching* for minimal proofs, we analyze and *prune* each proof found w.r.t. its contribution. We derive a pruned proof by deleting each non-contributing proof step. Every non-contributing literal of goal G for proof P is superfluous for the validity of G. Therefore, the notion of contribution extracts the essence of a proof. These notions are independent of a concrete inference system. Instead, they only depend on some abstract properties of the inference system. Roughly speaking, the applicability of an inference rule should only depend on some appropriate elements in the goal. If these elements are the same for two goals G_1 and G_2, then the application of the inference rule will generate similar subgoals for G_1 and G_2. The application of an inference rule will be described by partitioning a goal into two parts: a *principal* part and a *context*.

We do not only apply this pruning for the whole proof of the root goal but for every subgoal. We may reuse the proof for a goal if the following *reuse property* (RP) holds: Proof P for goal G is *reusable* for goal G' if G' contains every contributing element of G for P. Reuse can be exploited in two different ways: On the one hand, we can check property (RP) directly. This is sensible if we transfer a proof *sidewards* in the proof tree. On the other hand, we can propagate a proof for a subgoal *upwards* in the proof tree as long as on the path upwards none of its contributing elements has been introduced by the parent inference rule of the subgoal, i.e. the parent goal contains all contributing elements of the subgoal. This deletes proof steps that turned out to be superfluous and automatically improves the proof in a restricted way.

Sideward reuse may be interpreted as a restricted form of lemma application. Note that we do not allow any modifications to the contributing elements in property (RP) as e.g. instantiation with a substitution σ. Therefore, we can check this property very efficiently. The usefulness of our reuse mechanism depends on the fact that similar subgoals with the same contributing elements will be generated within one proof. Our case studies provide evidence for this.

The rest of the paper is organized as follows: After a summary of related work in Section 1.1 and some basic notions in Section 2, we motivate our approach with an abstract example in Section 3. In the following sections, we work out the details. Section 4 contains a formal framework for our approach defining adaptable inference systems. Section 5 provides algorithmic definitions for our new notions of contribution. In Section 6, we describe our reuse mechanism based on contribution. We evaluate our new approach within our inductive theorem prover QuodLibet [1] in Section 7 and conclude with further work in Section 8.

1.1 Related Work

Interestingly, similar ideas have been proposed and successfully implemented in other research areas as e.g. `SAT` (*boolean satisfiability*) based on `DPLL`, `CSP` (*constraint satisfaction*) and `LP` (*logic programming*). The analogon for upward propagation is called e.g. *non-chronological backtracking* for `SAT`, *conflict-directed backjumping* for `CSP` and *intelligent backtracking* for `LP`. The analogon for sideward reuse is called e.g. *conflict-driven learning* for `SAT` and *no-goods* for `CSP`. Due to the depth of literature in these fields, we can only scratch the surface and refer to articles that can be used as a starting point for further research.

The simplest of these problems is `SAT`. It is concerned with deciding whether a given propositional formula is satisfiable. Provers based on the `DPLL` procedure [5] systematically search the space of all variable assignments for a satisfying one. This search may be illustrated with a tree. On each level, one variable is assigned a truth value. The problem arises in the exponential blow-up of the search space w.r.t. the number of different variables. *Non-chronological backtracking* and *conflict-driven learning* introduced in [14] and [8] help to prune the search space. These techniques have dramatically increased the scope of `SAT` solvers based on `DPLL`. See [19] for a survey.

A generalization of `SAT` is `CSP`. Instead of the two truth values, each variable may be assigned a value from a finite set. Nevertheless, the search space for `CSP` is still finite. *Conflict-driven backjumping* [15] and the use of *no-goods* [16] laid the foundations for the approaches in `SAT`.

A more general problem is addressed in `LP`. In engines for logic programming languages such as `PROLOG`, a query is handled by resolution and unification of horn clauses over first-order logic. *Intelligent backtracking* aims at pruning the resulting search tree without losing any solutions. This technique can also be extended to non-horn clauses. See [4] for a survey.

In this paper, we transfer these approaches to reductive inference systems working on proof trees. In these systems the search space is usually infinite and the underlying theory is not decidable. For special calculi, similar approaches have been proposed before such as the *factorization* and *folding up* rule for tableaux. See [11] for a survey. We, however, do not restrict ourselves to concrete inference systems but present abstract requirements on inference systems that allow the application of our approach.

To provide an efficient reuse mechanism, we restrict ourselves to reusing pruned proofs without any modifications such as instantiation. Other reuse mechanisms such as the one described in [10] allow slight modifications of proofs. The combination of these approaches may be subject of further research. But it is questionable whether a reuse mechanism with higher complexity will pay off.

Note that our notion of contribution is closely related to that of relevance of proof steps, which plays a major role in tutoring systems for theorem proving. Currently, the only means to check for relevance is a static set of "golden proofs", where a step of a pupil is irrelevant if it is not found within this set. On p. 18 of MI 3 in [18] we find the following proposal to overcome the shortcomings of the golden-proof approach by checking each proof step of the pupil dynamically:

> "We dynamically support relevance analysis with domain reasoning. For this, we test whether a proof can still be obtained from the new proof situation (using an abstract-level proof system)."

And on p. 19 in [18]:

> "Proof step relevance is an interesting and ambitious challenge, and we will start research in this direction. Generally, the relevance challenge may even stimulate new research in proof theory. For instance, it would be nice to compactly and tractably represent the proofs in the proof space explicitly. Ideally only 'normal form' proofs (i.e. proofs without detours representing the whole equivalence class of proofs that reduce to the same normal form) would be stored in a (semi-) lattice structure that would encode some notion of quality of proofs and thereby support relevance analysis."

Our upward propagation of proof steps is to our knowledge the best approach towards such a normal form of proofs. Moreover, notice that our notion of contribution seems to be far more useful for the tutoring purpose than the question whether a proof can still be obtained. Indeed, the latter is only related to unsafe weakening steps of the pupil and cannot detect superfluous proof steps. Note that the proposal MI 3 was accepted by DFG for another three years of funding.

2 Basic Notions

We have originally developed our approach for the inductive theorem prover QuodLibet [1]. Our approach, however, is independent of a concrete specification style or inference system. Instead, we only require that the new subgoals generated by an inference rule are computed in a uniform way. Therefore, we will present our approach as abstractly as possible.

In general, a goal G consists of n components $\langle g_1; \ldots; g_n \rangle$. Many inference systems just have one component—the clause Γ to be proved. QuodLibet uses a weight $\mathcal{W}$ as second component to explicitly represent the induction order.

A clause Γ is a disjunction of literals l_i. A literal is an atom a or a negated atom $\neg a$. Clauses may be represented as lists, multisets or sets of literals. Lists can be interpreted as special multisets that allow to refer to literals by their positions in the clause. In contrast to these representations, a set contains each literal at most once. In QuodLibet clauses are represented as lists.

We expect that the following operations—inspired by set theory—are defined on goals in a suitable way: A subset relation $G_1 \subseteq G_2$, the union $G_1 \cup G_2$, the intersection $G_1 \cap G_2$, the sum $G_1 + G_2$, the difference $G_1 - G_2$ of two goals G_1 and G_2, and a partition relation $G = G_1 \uplus \cdots \uplus G_m$ on goals G and G_i, $i \in \{1, \ldots, m\}$. We can define them on the n components of the goals.

If the component is a set, we define $\subseteq$ as the subset relation on sets, $\cup$ and $+$ as the union, $\cap$ as the intersection, $-$ as the difference of sets, and $A =$

$A_1 \uplus \cdots \uplus A_m$ as the usual partition relation on sets, i.e. $A_1 \uplus \cdots \uplus A_m$ is a partition of set A if $A = A_1 + \cdots + A_m$ and A_i are pairwise disjoint sets.

If the component is a multiset $M(A)$ for set A—i.e. a function $M : A \to \mathbb{N}$ such that $\{x \in A \mid M(x) > 0\}$ is finite—we define the operations as follows:

- $M_1 \subseteq M_2$ iff $M_1(x) \leq M_2(x)$ for all $x \in A$;
- $(M_1 \cup M_2)(x) = \max(M_1(x), M_2(x))$ for all $x \in A$;
- $(M_1 \cap M_2)(x) = \min(M_1(x), M_2(x))$ for all $x \in A$;
- $(M_1 + M_2)(x) = M_1(x) + M_2(x)$ for all $x \in A$;
- $(M_1 - M_2)(x) = \max(0, M_1(x) - M_2(x))$ for all $x \in A$;
- $M = M_1 \uplus \cdots \uplus M_m$ iff $M = M_1 + \cdots + M_m$.

If the component consists of a single element, we may interpret it as singleton set containing this element. This is done for the weight component in QuodLibet.

3 A Motivating Example

In this section we illustrate our approach on an abstract level. Figure 1 contains a proof tree for an abstract inference system given by inference rules I. Goals are illustrated by their clauses given in rectangular boxes. For reference purposes, we display a unique number for each goal in the right upper corner enumerating goals in the tree in preorder. A clause is represented by the literals l_i it contains. The applied inference rules are illustrated in rounded boxes. For reference purposes, we annotate each inference rule with an index that corresponds to the goal it is applied to. Note that the tree contains the open goals G_{10} and G_{12}. Thus, the clause in the root goal G_1 is not proved. Our approach restructures the given proof tree automatically. It propagates a pruned subproof upwards in the proof tree to eliminate goal G_{10}, and reuses another pruned subproof to prove goal G_{12}. Thus, it creates a closed proof tree. Furthermore, it indicates that literal l_4 is superfluous. Figure 2 displays the pruned proof tree for this strengthened clause. The marking of the goals in this proof tree refers to the corresponding goals in the original proof tree—e.g. goals $G_{9'}$ and $G_{9''}$ in Figure 2 are derived from goal G_9 in Figure 1. The same holds true for inference rules.

Example 1. To exemplify inference rules, consider I_4 in Figure 1. Assume that we are given a conditional rewrite lemma $\{ \neg l_9, l_8, l_7, s = t \}$ which may be interpreted as an implication $l_9 \wedge \neg l_8 \wedge \neg l_7 \Rightarrow s = t$. Furthermore, assume that $s = t$ may be used to rewrite l_3 to l'_3, i.e. l_3 contains a subterm s and l'_3 is generated from l_3 by replacing one occurrence of s with t. Then we may apply this lemma to goal G_4 because G_4 contains l_3. But we have to guarantee that the conditions of the lemma are fulfilled. Therefore, our inference rule generates a case split for every condition that is not directly present in G_4 resulting in condition subgoals G_5 and G_6. The rewrite subgoal G_9 establishes all conditions of the lemma. Thus, we can replace l_3 with l'_3 in G_9. Note that all other literals in G_4 are transfered to subgoals G_5, G_6 and G_9 without any modifications.

We illustrate our requirements on inference rules in Figure 3. The notions are partially inspired by Gentzen's notions on sequent calculi [7]. They are motivated in this section and formally defined as *adaptable* inference systems in Section 4.

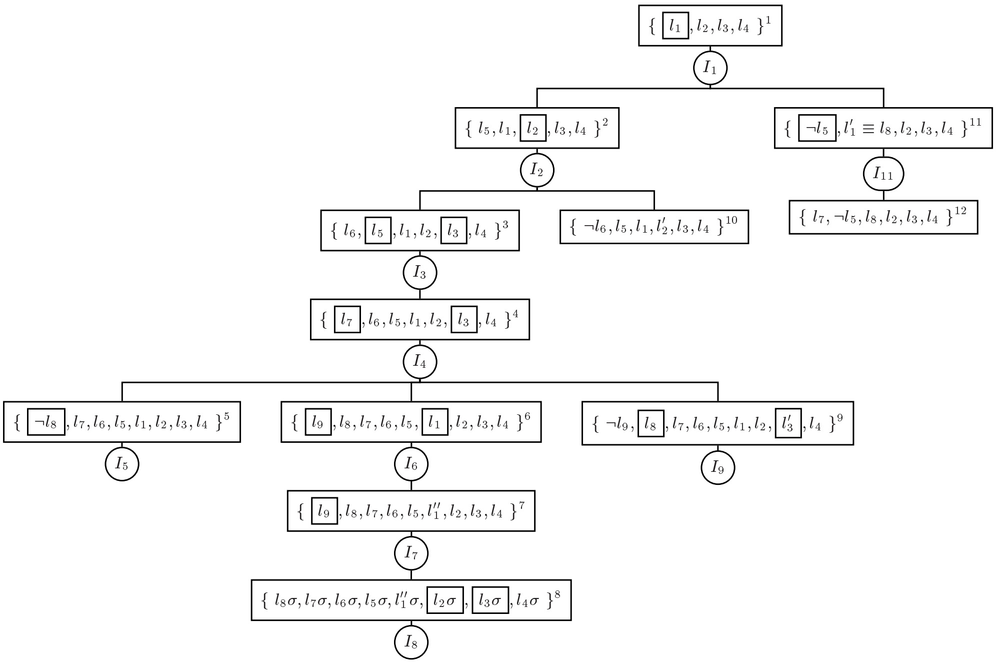

Fig. 1. proof tree with two open goals

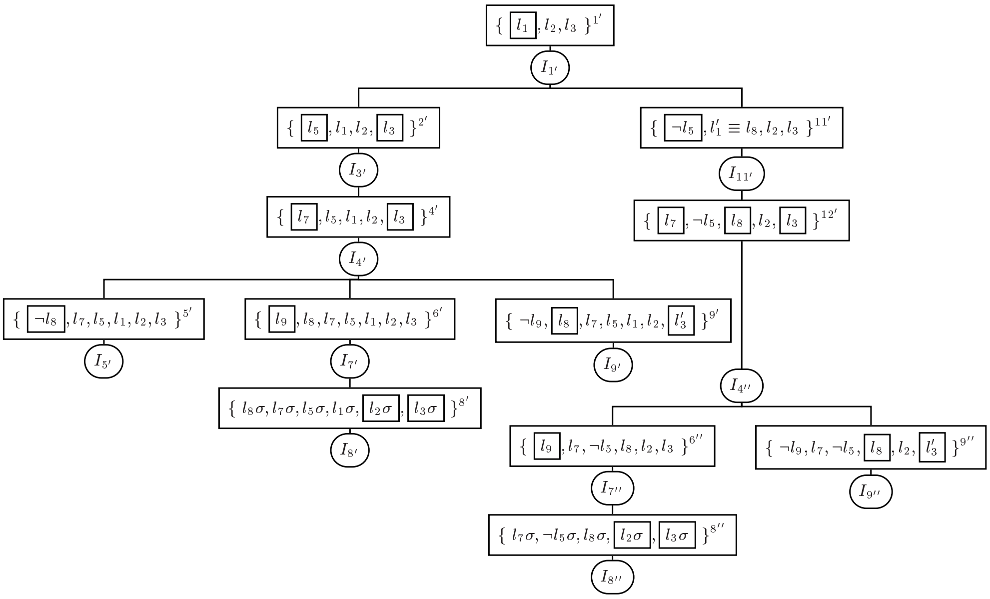

Fig. 2. proof tree closed with upward propagation and sideward reuse

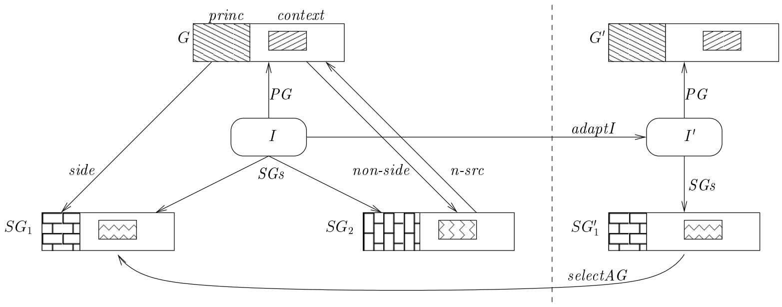

Fig. 3. Illustration of the Notions for Adaptable Inference Systems

An *inference rule* I is applied to a *parent* goal $PG(I)$. The application results in new subgoals $SGs(I)$. For our approach it is essential that the applicability of I to the parent goal G does not depend on the whole goal but only on a part of it—called the *principal* part ($princ(I) \subseteq G$). The remaining part of the goal is called *context* ($context(I) \subseteq G$). Furthermore, the new subgoals must not be generated in an arbitrary way but their derivation from G is fixed. Thus, we get similar new subgoals if we adapt I to another goal G' that contains the principal part for I. The principal part determines the essential appearance of the new subgoals. Some elements in the principal part may be removed or changed, new elements may be added. We model this by replacing the whole principal part with a *new* part in each new subgoal generated by function *side*. Furthermore, the principal part determines how the context will be changed in the new subgoals. Often the context will be transfered to the new subgoals without modifications but it may also be changed in a uniform way e.g. by applying a substitution to each element. We model this modification with a monotonic function *non-side*. This modification has to be done in such a way that we can determine the *source* within the context that generates certain elements in the *non-side* part of a subgoal. This is done with function *n-src*.

In Example 1, literals l_3 and l_7 have to be principal. Literal l_3 is rewritten in G_9, literal l_7[3] prevents the creation of one condition subgoal. We could mark further literals as principal to restrict the application of the lemma. Thereby, we could restrict the proof search. But as the principal part has to be present in a goal G' if we want to adapt a proof for a goal G to G', we aim at defining the principal part with as few elements as possible. In Figures 1 and 2, the principal literals of each goal are framed. The other literals form the context. If a literal is derived from a principal literal in the parent goal, we use a primed version of this principal literal in the subgoal. For new literals we use new indices. But this distinction is not essential. In G_{11}, literal l'_1 is derived from l_1 in G_1 whereas $\neg l_5$ is a new literal. We assume that l'_1 happens to be the same as l_8 indicated by $l'_1 \equiv l_8$ in G_{11}. For all inference rules in Figure 1 except for I_7, *non-side* and *n-src* are identity functions, i.e. the context is transfered to the subgoals unmodified. The remaining literals in the subgoals are generated by function *side*. In G_8, the

[3] In a finer modeling, we may call such a principal literal a *cut-off* literal as it cuts off some of the subgoals and literals that otherwise would have been created.

principal literal l_9 of G_7 is removed. All other literals in G_8 are modified in a uniform way applying substitution σ. This is done by function *non-side*. In this case, for a set L' of literals in G_8 generated from $context(I_7)$, function *n-src* may return an arbitrary set $L \subseteq context(I_7)$ such that $L\sigma = L'$.

To state the adaptation of an inference rule more precisely, we use functions *adaptI* and *selectAG*. An inference rule I with parent goal G can be adapted to goal G' if G' contains the principal part of G (for I). Function *adaptI* returns an adapted inference rule I' that can be applied to G'. For each new subgoal of I', function *selectAG* selects a subgoal of I that generates the same elements for the common parts of G and G'. In Figure 3, this is illustrated by hatching the common parts of the goals. Usually, inference systems are given by parameterized inference schemes. The adaptation of an inference rule may be achieved by changing some of the parameters of the inference scheme, e.g. the positions of the principal literals. In Figure 2, we can adapt inference rule $I_{4'}$ to $G_{12'}$ because $G_{12'}$ contains the principal literals l_3 and l_7 of $G_{4'}$. Note that the adapted inference rule $I_{4''}$ applied to $G_{12'}$ generates one subgoal less than the application of $I_{4'}$ to $G_{4'}$. Furthermore, l_8 in goals $G_{6''}$ and $G_{9''}$ is not newly created but inherited from $G_{12'}$. The literals common in $G_{12'}$ and $G_{4'}$, however, generate the same literals in the corresponding subgoals.

How do we find out whether a proof P for G can be reused for G' in our setting for adaptable inference system? We can adapt the first proof step of P if G' contains at least the principal literals of G. There will at most as many new subgoals be created for G' as for G. If we can reuse the proofs of the subgoals of G for those of G' we are done. For this, we have to guarantee that the subgoals of G' contain the principal literals of the corresponding subgoals of G. Furthermore, we have to recursively guarantee this property for all offspring goals.

With function *n-src* we can determine the source in the parent goal of those literals in a subgoal SG that are required for the proof of SG. Thus, we can precompute in a bottom-up style all the literals in G that are required for P. We call these literals *contributing literals* of G for P. During their computation we may recognize that some proof steps are not really needed for the proof: If a goal G contains all the contributing literals of one of its subgoals SG, we can reuse the proof of SG for G eliminating the proof step originally applied to G. The remaining proof steps are called *contributing proof steps*. The computations of contributing literals and proof steps mutually depend on each other.

The following table contains the contributing literals and contributing proof steps for the proof of G_3 in Figure 1:

Goal	Contributing Literals	Contributing Proof Steps
G_5	$\neg l_8$	I_5
G_8	$l_2\sigma, l_3\sigma$	I_8
G_7	l_9, l_2, l_3	I_7
G_6	l_9, l_2, l_3	I_7
G_9	l_8, l_3'	I_9
G_4	l_7, l_2, l_3	I_4
G_3	l_5, l_2, l_3	I_3

The contributing literals of G_5, G_8 and G_9 are the principal literals. In G_7, literals l_2 and l_3 are contributing as they are responsible for generating the contributing literals in G_8. Proof step I_6 is not contributing since the contributing literals of G_7 are present in G_6. Eventually, we get l_5, l_2 and l_3 as contributing literals of G_3 which are present in G_2. Therefore, we can propagate the proof of G_3 upwards to G_2. Note that this implies that we need not find a proof for G_{10}. This shows that pruning a proof may save time since unnecessary proof obligations are detected. Since the contributing literals of G_4 are present in G_{12}, we can reuse its pruned proof to close the whole proof tree as presented in Figure 2. Note that we cannot reuse the whole proof of G_4 for G_{12} because the non-contributing proof step I_6 requires literal l_1 which is not present in G_{12}.

4 Adaptable Inference Systems

Motivated by Section 3, we present abstract requirements on inference systems that allow the application of our approach. These requirements are defined in three steps with a couple of functions described in Section 3 (see Figure 3).

Definition 1 (Inference System). An inference system $\mathcal{IS} = (\mathcal{I}, PG, SGs)$ consists of a set $\mathcal{I}$ and two functions PG and SGs. The elements $I \in \mathcal{I}$ are called *inference rules*, the goal $PG(I)$ is called *parent goal* of I and the (possibly empty list of) finitely many goals $SGs(I)$ are called *(new) subgoals* of I. Let $nbSGs(I)$ be the number of new subgoals of I, and $SG_i(I)$ be the ith subgoal in $SGs(I)$ for $i \in \{1, \ldots, nbSGs(I)\}$.

An inference rule I with parent goal G and new subgoals $SG_1, \ldots, SG_n$ is often represented as $I = \dfrac{G}{SG_1 \quad \ldots \quad SG_n}$.

Usually, the new subgoals $SG_1, \ldots, SG_n$ of the inference rule I are derived from parent goal G by possibly adding a new part, changing some old part, or removing some old part. According to Section 3 we divide G into two parts. The generation of the new part that replaces the principal part is modeled with function *side*. The transformation of the context is modeled with a monotonic function *non-side*. To propagate the contributing part of a subgoal upwards in a proof tree to its parent goal (see Sections 3 and 5), we also require a function *n-src*. This function computes those elements within the context that are responsible for the creation of a given set of elements in a new subgoal.

Definition 2 (Defined with Principal Part and Context). An inference system $\mathcal{IS} = (\mathcal{I}, PG, SGs)$ is *defined with principal part and context* if there exist functions *princ*, *context*, *side*, *non-side* and *n-src* such that the following properties are fulfilled for each inference rule $I \in \mathcal{I}$:

(a) $PG(I) = princ(I) \uplus context(I)$;
(b) for each $i \in \{1, \ldots, nbSGs(I)\}$, $SG_i(I)$ is generated w.r.t. *side*, *non-side* as follows:
$$SG_i(I) = side(I, i) + non\text{-}side(I, i, context(I));$$
(c) function *non-side* is monotonic in the last argument, i.e., for each $i \in \{1, \ldots, nbSGs(I)\}$ and goals G, G' with $G \subseteq G' \subseteq context(I)$:
$$non\text{-}side(I, i, G) \subseteq non\text{-}side(I, i, G');$$
(d) for each $i \in \{1, \ldots, nbSGs(I)\}$, function *n-src* is defined such that for each $SG' \subseteq non\text{-}side(I, i, context(I))$:

– $n\text{-}src(I, i, SG') \subseteq context(I)$ and
– $non\text{-}side(I, i, n\text{-}src(I, i, SG')) = SG'$.

Note that every inference system can be defined with principal part and context if we define $princ(I) = PG(I)$, $context(I) = \emptyset$, and $side(I, i) = SG_i$ for each $i \in \{1, \ldots, nbSGs(I)\}$. But as we have to use the principal part as contributing part we should define this part with as few elements as possible to enhance the reusability of the proof for the goal.

For adaptable inference systems, we also require that, if there is an inference rule I and a goal G' that contains the principal part of I, then there exists an inference rule I' with parent goal G'. Furthermore, each of the new subgoals of I' "corresponds" to a new subgoal of I. Roughly speaking, the same elements in the new subgoals are generated for the common part of G and G'. We use function *adaptI* to perform the adaptation of an inference rule to another. Function *selectAG* relates the new subgoals of the two applications.

Definition 3 (Adaptable Inference Systems). Let $\mathcal{IS} = (\mathcal{I}, PG, SGs)$ be an inference system defined with principal part and context as in Definition 2. $\mathcal{IS}$ is *adaptable* if there exist functions *adaptI* and *selectAG* such that the following properties are fulfilled for each inference rule $I \in \mathcal{I}$ and for each goal G' with $princ(I) \subseteq G'$:

(a) $adaptI(I, G') = I'$ such that $I' \in \mathcal{I}$ and $PG(I') = G'$;
(b) let $G'' := (PG(I) \cap G') - princ(I)$ be the common non-principal part of $PG(I)$ and G'; then, for each $SG' \in SGs(I')$:
  – $selectAG(I, I', SG') \in \{1, \ldots, nbSGs(I)\}$;
  – $side(I, i) + non\text{-}side(I, i, G'') \subseteq SG'$ where $i = selectAG(I, I', SG')$.

In Section 5, we only require inference systems defined with principal part and context to compute the contribution of proof steps. For reusing proofs in Section 6, we require adaptable inference systems.

5 Contributing Proofs

A proof tree P consists of goal and inference nodes which refer to goals and inference rules. They additionally contain information about their position in the tree i.e. their parent and children. We will abuse these notions and identify goals with goal nodes and inference rules with inference nodes.

To simplify the presentation, we restrict ourselves to single and complete proofs that contain exactly one successive proof step for each goal. The extension for arbitrary proof trees is easy. Let $RG(P)$ be the root goal of proof P. Let $SI(G)$ be the inference rule applied to G in the proof tree and $PI(G)$ the parent inference rule of G (unless G is the root goal of a proof P).

Let P be a proof for G, and $I = SI(G)$ the proof step performed for G. If G' is a goal with $princ(I) \subseteq G'$, we can apply $I' = adaptI(I, G')$ to G'. If we can recursively apply this adaptation for every new subgoal $SG_j(I')$ using $SG_i(I)$ with $i = selectAG(I, I', SG_j(I'))$, then we can reuse the whole proof.

We aim at defining a *contributing* part $contrib(G)$ of G such that a (sub)proof of proof P for G can be applied to G' if $contrib(G) \subseteq G'$. During the analysis of

Procedure `calc-contrib`(G)

1 $I \leftarrow SI(G)$;
2 **if** $nbSGs(I) = 0$ **then**
3     $contribI(G) \leftarrow I$;
4     $contrib(G) \leftarrow princ(I)$;
5 **else**
6   **foreach** $SG \in SGs(I)$ **do**
7         $calc\text{-}contrib(SG)$;
8     $\mathcal{G} \leftarrow \{SG \in SGs(I) \mid contrib(SG) \subseteq G\}$;
9   **if** $\mathcal{G} \neq \emptyset$ **then**
10         $SG \leftarrow selectCG(\mathcal{G})$;
11         $contribI(G) \leftarrow contribI(SG)$;
12         $contrib(G) \leftarrow contrib(SG)$;
13   **else**
14         $contribI(G) \leftarrow I$;
15         $contrib(G) \leftarrow princ(I) + \bigcup_{i=1}^{nbSGs(I)} n\text{-}src(I, i, contrib(SG_i(I)) - side(I, i))$;

proof P, we may recognize proof steps that are superfluous, i.e. the contributing part of a subgoal is present in one of its ancestor goals. We use this information to extract from P a *pruned* subproof P'. This subproof is identified by storing in $contribI(G)$ the *contributing inference rule* that can be adapted to G.

Procedure *calc-contrib* calculates $contrib(SG)$ and $contribI(SG)$ for each offspring SG of a proved goal G in a bottom-up style. $contrib(SG)$ and $contribI(SG)$ are stored in global hashes as a side-effect of the procedure. If G is proved with a single inference rule, i.e. no new subgoals are created, then $princ(I)$ defines the contributing part, and the inference rule is contributing (Lines 2–4). Otherwise, we first calculate the contribution of each subgoal (Lines 6–7). If the contributing part of at least one subgoal is present in goal G we select one such subgoal SG with function *selectCG* and transfer its contributing part and inference rule to G (Lines 9–12). Otherwise, the inference rule applied to G is contributing. Beside $princ(I)$ we have to adapt the contributing elements of each subgoal that are generated by $context(I)$ (Lines 13–15). This is done using function *n-src*.

Function *selectCG* in Line 10 provides heuristics for selecting a subgoal whose contribution is propagated upwards in the proof tree. In our current implementation, we choose a subgoal whose contributing part consists of as few elements as possible. Thereby, we hope to increase the probability that we can further propagate the proof upwards in the proof tree. From these subgoals, we choose one with as few proof steps as possible to get shorter proofs.

Obviously, we can improve procedure *calc-contrib* as we do not have to recalculate the contribution for a goal SG if we call *calc-contrib* for its ancestor G. Instead, we can just use the hashed values $contrib(SG)$ and $contribI(SG)$. We may also use procedure *calc-contrib* for proof trees with multiple proof attempts. In this case, we just have to iterate over all inference rules applied.

Let $\prec$ be defined as $N_1 \prec N_2$ if N_1 is an offspring of N_2 in a proof tree P. Then, $\prec$ is a wellfounded order on goal and inference nodes.

Procedure `reuse-proof`(G, G') // Precondition: contrib$(G) \subseteq G'$

$I \leftarrow contribI(G)$;
$I' \leftarrow adaptI(I, G')$;
apply I' *to* G' *and remove former proof attempts for* G';
for $j = 1$ **to** $nbSGs(I')$ **do**
  $i \leftarrow selectAG(I, I', SG_j(I'))$;
  reuse-proof$(SG_i(I), SG_j(I'))$;

Let $\mathcal{IS}$ be an inference system defined with principal part and context as in Definition 2. Then, procedure *calc-contrib* has the following properties which can be proved with induction w.r.t. $\prec$:

Lemma 1. *Procedure calc-contrib is terminating.* □

Lemma 2. *For each goal G $contribI(G) \prec G$.* □

Lemma 3. *Let $I = contribI(G)$. Then the following holds true:*

(a) $contrib(PG(I)) = contrib(G) \subseteq G$;
(b) $princ(I) \subseteq contrib(G)$;
(c) for each $i \in nbSGs(I)$:
$contrib(SG_i(I)) \subseteq side(I, i) + non\text{-}side(I, i, contrib(PG(I)) - princ(I))$. □

Definition 4 (Contributing Part / Contributing Inference Rule). Let $\mathcal{IS}$ be an inference system defined with principal part and context as in Definition 2. Let P be a proof for goal G. Let *contrib*(G) and *contribI*(G) be computed with *calc-contrib*(G) (and heuristics *selectCG*). Then, *contrib*(G) is called the *contributing part* of G for P. The inference rule $I = SI(G)$ applied to G is called *contributing* for P if *contribI*$(G) = I$. Otherwise, I is called *non-contributing*.

6 Reusing Contributing Proofs

Due to Definition 3, we can exploit Lemma 3 to reuse the proof of a goal G for G' if *contrib*$(G) \subseteq G'$. For this, procedure *reuse-proof* adapts inference rule I = *contribI*(G) to G'. For each new subgoal $SG_j(I')$, we find one subgoal $SG_i(I)$ such that the precondition *contrib*$(SG_i(I)) \subseteq SG_j(I')$ for procedure *reuse-proof* is fulfilled. Thus, we can apply it recursively to $SG_i(I)$ and $SG_j(I')$. The following lemma states the soundness of procedure *reuse-proof*.

Lemma 4. *Let $\mathcal{IS}$ be an adaptable inference system as defined in Definition 3. If $contrib(G) \subseteq G'$ then the following properties hold true:*

(a) each recursive call reuse-proof$(SG_i(I), SG_j(I'))$ obeys the invariant $contrib(SG_i(I)) \subseteq SG_j(I')$;
(b) the call of reuse-proof(G, G') is terminating;
(c) reuse-proof(G, G') creates a closed proof tree for G'. □

For the usefulness of our approach it is crucial that we can efficiently find a proved goal G and an unproved goal G' such that $\mathit{contrib}(G) \subseteq G'$. We do not perform this search arbitrarily, but we fix one goal and perform this search in two directions: upwards and sidewards in the proof tree. Our upward propagation fixes the proved goal G. It is very efficient because we only try to propagate the proof of G to its unproved ancestor goals as far as possible. During sideward reuse, however, we fix one unproved goal G' and test for each proved goal G whose first proof step $\mathit{SI}(G)$ is contributing whether its proof can be reused for G'. Note that we do not have to consider a goal G whose first proof step $\mathit{SI}(G)$ is non-contributing because the same proof is performed for $PG(\mathit{contribI}(G))$, i.e. for the father of the contributing inference rule $\mathit{contribI}(G)$ that can be adapted to G. To perform the sideward check efficiently, we use indexing techniques [17]. In our current implementation, we use a simple trie-like data structure [6]. The leaves of the trie are marked with the proved goals. The path to goal G is marked with the elements of the contributing part $\mathit{contrib}(G)$. Since in a trie common prefixes are shared, we may exclude many goals at once if one of their contributing elements is not present in G'.

Our approach can be easily integrated into a waterfall model used in successful inductive theorem provers such as `NQTHM` [2] and its successor `ACL2` [9]. In a waterfall model, the goals to be proved are collected in a *pool* and the proof process is divided into several *phases*. Each goal passes through each phase successively until one of them can be applied. For a successful phase, the subgoals are put into the pool again starting the proof process from the beginning. The idea of a waterfall model is to start with the cheapest phases that promise the highest profit. Whereas sideward reuse can be integrated into the waterfall model as separate phase, the integration of upward propagation is more difficult as a proof may be propagated for more than one level. We have to take care that we remove those goals from the pool that are affected by the propagation, and that we transfer control to the right place in the recursive control structure of the waterfall. This can be achieved with an exception handling mechanism.

7 Case Studies

We perform our case studies with QuodLibet [1]. QuodLibet is an equality-based inductive theorem prover for clausal first-order logic with implicitly universally quantified variables. It admits *partial* definitions of operators over *free constructors* using (possibly *non-terminating*) 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.

The inference system of QuodLibet is adaptable. Therefore, our approach is applicable. Due to lack of space, we can only present a few case studies. The following table contains statistics for two proofs that $\sqrt{2}$ is irrational, based on the ideas of Hippasos of Metapont (H) and Euclid of Alexandria (E), respectively; a formal proof of the equivalence of different implementations of the lexicographic path order `Lpo` [12]; a termination proof for McCarthy's `f91` function [13]; and a soundness proof for a majority voting protocol [3]:

Example	#Lemmas	Reuse	#Appl.	#Del.	#UP	#SR	Runtime	
$\sqrt{2}$ (E)	0 / 26	—	459	6	—	—	1.23	+4.0%
		x	464	11	1	28	1.28	
$\sqrt{2}$ (H)	0 / 10	—	346	64	—	—	1.47	-20.4%
		x	292	52	0	16	1.17	
`Lpo`	23 / 273	—	20220	2555	—	—	274.12	-29.0%
		x	18582	2686	41	2005	194.38	
`f91`	1 / 3	—	488	0	—	—	1.65	-32.7%
		x	369	0	0	36	1.11	
`mjrty`	0 / 8	—	825	162	—	—	3.50	-56.0%
		x	556	12	1	67	1.54	

For each case study, column "#Lemmas" contains two entries. The second entry is the number of lemmas proved, the first one the number of lemmas that can be strengthened by eliminating non-contributing literals. The remaining columns are split into two lines. The first (resp. second) line presents the statistics without (resp. with) reuse enabled. Column "#Appl." contains the number of inference rules applied during the whole proof search, column "#Del." the number of inference rules deleted, column "#UP" the number of applications of upward propagation, column "#SR" the number of applications of sideward reuse, and column "Runtime" the runtime in seconds measured by a CMU Common Lisp system on a machine with a 1330 MHz AMD processor and 512 MB RAM. An inference rule may be deleted for two reasons: firstly, if the proof search gets stuck and has to backtrack; secondly, if a proof is propagated upwards.

With our implementation of the reuse mechanism, we aim at reducing the runtime for creating closed proof trees as much as possible. On the one hand, we only prune a proof tree by propagating a proof upwards if this eliminates some proof obligations. Therefore, we may derive much shorter proofs by further pruning. On the other hand, we do not only mark a goal as proved if we can reuse another proof but adapt this other proof. Thus, we may speed up the computation if we are only interested whether a lemma is valid but not in a proof itself. Instead of "replaying" a proof each time it can be reused, we could also introduce new lemmas according to the contributing parts of the reusable proof. This may speed up the computation if a proof is reused more than once. But as this blows up the lemma data base, we have not investigated this variant.

Our reuse mechanism introduces little overhead. Thus, the runtime may increase with reuse if the heuristics for proof search are well suited for a case study such as $\sqrt{2}$ (E). But typically, with our reuse mechanism, we get much shorter proofs and the runtime decreases by 20 to 30% for our case studies. For some case studies, we even get speed-ups of more than 50%. Note that sideward reuse is by far more often applicable than upward propagation. Nevertheless, for the most complicated case study `Lpo`, upward propagation is also frequently applicable.

8 Conclusion

In this paper, we have presented abstract requirements on reductive inference systems that allow the transmission of techniques successfully used in other research areas such as SAT, CSP and LP. Roughly speaking, these techniques allow to prune the search space by analyzing the part of the search space that has already been explored. In our approach, we analyze a closed subproof w.r.t. its contribution. We may propagate a (pruned) subproof upwards in the proof tree to eliminate an open subgoal, or reuse it sidewards to close an open subgoal. The case studies performed with the inductive theorem prover QuodLibet show substantial speed-ups when applying these reuse techniques.

Contribution may also be used for guiding proof search. A preliminary version of this idea has been implemented in QuodLibet. Therefore, we expect even better speed-ups if our reuse mechanism is integrated into other theorem provers whose proof search is not influenced by contribution. Both, the exploitation of contribution to guide proof search and the integration of our reuse mechanism into other theorem provers, are subject of further research.

References

1. J. Avenhaus, U. Kühler, T. Schmidt-Samoa, and C.-P. Wirth. How to prove inductive theorems? QuodLibet! In *CADE-19*, LNAI 2741, pp. 328–333. Springer, 2003.
2. R.S. Boyer and J S. Moore. *A Computational Logic Handbook*. Acad. Press, 1988.
3. R.S. Boyer and J S. Moore. MJRTY: A fast majority vote algorithm. In *Automated Reasoning: Essays in Honor of Woody Bledsoe*, pp. 105–118, 1991.
4. M. Bruynooghe. Intelligent backtracking revisited. In *Computational Logic - Essays in Honor of Alan Robinson*, pp. 166–177, 1991.
5. M. Davis, G. Logemann, and D.W. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.
6. E. Fredkin. Trie memory. *Commun. ACM*, 3(9):490–499, 1960.
7. G. Gentzen. Untersuchungen über das logische Schliessen. *Mathematische Zeitschrift*, 39:176–210 and 405–431, 1934f.
8. R.J. Bayardo Jr. and R. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *AAAI/IAAI*, pp. 203–208, 1997.
9. M. Kaufmann, P. Manolios, and J S. Moore. *Computer-Aided Reasoning: An Approach.* Kluwer Academic Publishers, 2000.
10. T. Kolbe and C. Walther. Reusing proofs. In *ECAI*, pp. 80–84, 1994.
11. R. Letz and G. Stenz. Model elimination and connection tableau procedures. In *Handbook of Automated Reasoning*, volume II, pp. 2015–2114. Elsevier, 2001.
12. B. Löchner. Things to know when implementing LPO. In *ESFOR '04*, ENTCS. Elsevier, 2004. Ext. version to appear in *Int. J. on Artificial Intelligence Tools.*
13. Z. Manna and J. McCarthy. Properties of programs and partial function logic. In *Machine intelligence 5*, pp. 27–37. 1970.
14. J.P. Marques-Silva and K.A. Sakallah. Conflict analysis in search algorithms for satisfiability. In *ICTAI*, pp. 467–469, 1996.
15. P. Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9:268–299, 1993.
16. T. Schiex and G. Verfaillie. Nogood recording for static and dynamic constraint satisfaction problems. In *ICTAI*, pp. 48–55, 1993.
17. R. Sekar, I.V. Ramakrishnan, and A. Voronkov. Term indexing. In *Handbook of Automated Reasoning*, volume II, pp. 1853–1964. Elsevier, 2001.
18. C.-P. Wirth (ed.). Proposal: January 2005–December 2007. Collaborative Research Center SFB 378 "Resource-Adaptive Cognitive Processes", Saarland Univ., 2004.
19. L. Zhang and S. Malik. The quest for efficient boolean satisfiability solvers. In *CADE-18*, LNCS 2392, pp. 295–313, Springer, 2002.

A Proofs of the Lemmas

For the "suitable definition" of the operations on goals in Section 2, we, e.g., assume that $+$ is commutative and that the following properties hold for goals $G_1, \ldots, G_n$ and G:

$$\bigcup_{i=1}^{n} G_i \subseteq G \quad \text{if for each } i \in \{1, \ldots, n\} : G_i \subseteq G \tag{1}$$

$$G_j \subseteq \bigcup_{i=1}^{n} G_i \text{ for each } j \in \{1, \ldots, n\} \tag{2}$$

$$G_1 - G_3 \subseteq G_2 - G_3 \quad \text{if } G_1 \subseteq G_2 \tag{3}$$

$$G_1 + G_3 \subseteq G_2 + G_3 \quad \text{if } G_1 \subseteq G_2 \tag{4}$$

$$G_1 \subseteq G_2 + G_3 \quad \text{iff} \quad G_1 - G_2 \subseteq G_3 \tag{5}$$

$$G = G_1 + G_2 \quad \text{if } G = G_1 \uplus G_2 \tag{6}$$

$$(G_1 + G_2) - G_1 = G_2 \quad \text{if there exist } G_3, G_4 \text{ with } G_3 = G_1 \uplus G_4 \text{ and } G_2 \subseteq G_4 \tag{7}$$

For the definition of the operations on set and multiset components in Section 2, these properties hold true.

For the proofs, we use the wellfounded order $\prec$ defined in Section 5.

Lemma 1. *Let $\mathcal{IS}$ be an inference system defined with principal part and context as in Definition 2. Then, procedure calc-contrib is terminating.*

Proof. We have to prove that each call *calc-contrib*(G) is terminating. We perform induction on goal G w.r.t. $\prec$: The only recursive call is in Line 7 for subgoals $SG \in SGs(I)$. As $SG \prec I \prec G$, each recursive call terminates by induction hypothesis. Since all loops terminate, procedure *calc-contrib* is terminating. □

Lemma 2. *Let $\mathcal{IS}$ be an inference system defined with principal part and context as in Definition 2. Then, for each goal G $contribI(G) \prec G$.*

Proof. This is proved with induction on G w.r.t. $\prec$: $contribI(G)$ is set in Lines 3, 11 and 14 of procedure *calc-contrib*(G), respectively.

- In Lines 3 and 14, $contribI(G)$ is set to $I = SI(G) \prec G$. Thus, $contribI(G) \prec G$ in these cases.
- In Line 11, $contribI(G)$ is set to $contribI(SG)$ for a subgoal $SG \in \mathcal{G} \subseteq SGs(I)$. Thus, $SG \prec I \prec G$. By induction hypothesis for SG, we get $contribI(G) = contribI(SG) \prec SG \prec G$. □

Lemma 3. *Let $\mathcal{IS}$ be an inference system defined with principal part and context as in Definition 2 and $I = contribI(G)$. Then the following holds true:*

(a) $contrib(PG(I)) = contrib(G) \subseteq G$;
(b) $princ(I) \subseteq contrib(G)$;
(c) for each $i \in nbSGs(I)$:
$$contrib(SG_i(I)) \subseteq side(I, i) + non\text{-}side(I, i, contrib(PG(I)) - princ(I)).$$

Proof. We perform induction on G w.r.t. $\prec$: $contrib(G)$ is set in Lines 4, 12 and 15 of procedure *calc-contrib*(G), respectively.

(a) We first prove $contrib(PG(I)) = contrib(G)$:
   - In Lines 4 and 15, $PG(I) = G$ because $contribI(G)$ is set to $SI(G)$ in Lines 3 and 14, respectively. Thus, $contrib(PG(I)) = contrib(G)$ in these cases.
   - In Line 12, $contrib(G)$ is set to $contrib(SG)$ for a subgoal $SG \in \mathcal{G} \subseteq SGs(I)$. Thus, $SG \prec I \prec G$. By induction hypothesis for SG, we get $contrib(PG(contribI(SG))) = contrib(SG)$. Due to Line 11, $PG(contribI(SG)) = PG(contribI(G)) = PG(I)$. Thus, we get $contrib(PG(I)) = contrib(G)$.

   It remains to prove $contrib(G) \subseteq G$:
   - In Line 4, $contrib(G)$ is set to $princ(I) \subseteq PG(I) = G$. Thus, $contrib(G) \subseteq G$ in this case.
   - In Line 12, $contrib(G)$ is set to $contrib(SG)$ for a subgoal SG with $contrib(SG) \subseteq G$. Thus, $contrib(G) \subseteq G$ in this case.
   - In Line 15, $contrib(G)$ is set to $princ(I) + \bigcup_{i=1}^{SGs(I)} n\text{-}src(I, i, contrib(SG_i(I)) - side(I, i))$. Because of $PG(I) = G$, it suffices to prove $princ(I) + \bigcup_{i=1}^{SGs(I)} n\text{-}src(I, i, contrib(SG_i(I)) - side(I, i)) \subseteq PG(I)$. Due to Definition 2(a) and Properties (4) and (6), this can be further reduced to $\bigcup_{i=1}^{SGs(I)} n\text{-}src(I, i, contrib(SG_i(I)) - side(I, i)) \subseteq context(I)$. Due to Property (1), it suffices to prove $n\text{-}src(I, i, contrib(SG_i(I)) - side(I, i)) \subseteq context(I)$. By induction hypothesis for $SG_i(I) \prec G$, $contrib(SG_i(I)) \subseteq SG_i(I)$. With Definition 2(b), we get $contrib(SG_i(I)) \subseteq side(I, i) + non\text{-}side(I, i, context(I))$. Because of Property (5), $contrib(SG_i(I)) - side(I, i) \subseteq non\text{-}side(I, i, context(I))$. With Definition 2(d), the claim follows.

(b)
   - In Line 4, $contrib(G)$ is set to $princ(I)$. Thus, $princ(I) \subseteq contrib(G)$ in this case.
   - In Lines 11 and 12, $contribI(G)$ is set to $contribI(SG)$ and $contrib(G)$ is set to $contrib(SG)$ for a subgoal SG. By induction hypothesis for $SG \prec G$, we get $princ(contribI(SG)) \subseteq contrib(SG)$. Therefore, $princ(I) \subseteq contrib(G)$ in this case.
   - In Line 15, $contrib(G)$ is set to $princ(I) + \bigcup_{i=1}^{nbSGs(I)} n\text{-}src(I, i, contrib(SG_i(I)) - side(I, i))$. Thus, $princ(I) \subseteq contrib(G)$.

(c)
   - In Line 4, $nbSGs(I) = 0$, i.e., there are no subgoals. Thus, nothing has to be shown.
   - In Line 12, the claim follows from induction hypothesis for $SG \prec G$ since $contrib(G) = contrib(SG)$ and $contribI(G) = contribI(SG)$.
   - Due to (a) and Property 3, $contrib(SG_i(I)) - side(I, i) \subseteq SG_i(I) - side(I, i)$. With Definition 2(b) and Property 5, we get $contrib(SG_i(I)) - side(I, i) \subseteq non\text{-}side(I, i, context(I))$. Because of Definition 2(d), we get

$$n\text{-}src(I, i, contrib(SG_i(I)) - side(I, i)) \subseteq context(I) \tag{8}$$

$$\begin{aligned} &non\text{-}side(I, i, n\text{-}src(I, i, contrib(SG_i(I)) - side(I, i))) \\ &\qquad = contrib(SG_i(I)) - side(I, i) \end{aligned} \tag{9}$$

In Line 15, $PG(I) = G$ and $contrib(PG(I)) = princ(I) +$
$\bigcup_{i=1}^{nbSGs(I)}$ $n\text{-}src(I, i, contrib(SG_i(I)) - side(I, i))$.
Thus, $princ(I) + n\text{-}src(I, i, contrib(SG_i(I)) - side(I, i)) \subseteq contrib(PG(I))$ due to Properties (2) and (4). Because of (8) and Definition 2(a), we can apply Property (7). Together with Property (3), this results in
$n\text{-}src(I, i, contrib(SG_i(I)) - side(I, i)) =$
$(princ(I) + n\text{-}src(I, i, contrib(SG_i(I)) - side(I, i))) - princ(I) \subseteq$
$contrib(PG(I)) - princ(I)$.
Thus, we get
$contrib(SG_i(I)) - side(I, i) \subseteq non\text{-}side(I, i, contrib(PG(I)) - princ(I))$
from (9) using Definition 2(c). The claim follows from Property 5. □

Lemma 4. *Let $\mathcal{IS}$ be an adaptable inference system as defined in Definition 3. If $contrib(G) \subseteq G'$ then the following properties hold true:*

(a) each recursive call reuse-proof$(SG_i(I), SG_j(I'))$ obeys the invariant $contrib(SG_i(I)) \subseteq SG_j(I')$;
(b) the call of reuse-proof(G, G') is terminating;
(c) reuse-proof(G, G') creates a closed proof tree for G'.

Proof. (a) Due to Lemma 3(b), $princ(I) \subseteq contrib(G)$. From the assumption $contrib(G) \subseteq G'$, we get $princ(I) \subseteq G'$. According to Definition 3, function $adaptI(I, G')$ returns an inference rule I' that can be applied to G'. Furthermore, $selectAG(I, I', SG_j(I'))$ chooses for each subgoal $SG_j(I')$ an index i such that for the subgoal $SG_i(I)$ the following holds true: $side(I, i) + non\text{-}side(I, i, (PG(I) \cap G') - princ(I)) \subseteq SG_j(I')$.
We have to prove: $contrib(SG_i(I)) \subseteq SG_j(I')$. Because of Lemma 3(c), it suffices to prove $side(I, i) + non\text{-}side(I, i, contrib(PG(I)) - princ(I)) \subseteq SG_j(I')$.
Due to Lemma 3(a), $contrib(PG(I)) \subseteq PG(I)$. Because of Lemma 3(a) and the assumption, $contrib(PG(I)) = contrib(G) \subseteq G'$.
Therefore, $contrib(PG(I)) \subseteq PG(I) \cap G'$.
Due to Property (3), $contrib(PG(I)) - princ(I) \subseteq (PG(I) \cap G') - princ(I)$.
Thus, the claim follows from Definition 2(c) and Property (4). Note that $(PG(I) \cap G') - princ(I) \subseteq PG(I) - princ(I) \subseteq context(I)$ because of Definition 2(a) and Properties (6) and (5).
(b) This holds true since in each recursive call, the first argument decreases w.r.t. $\prec$, i.e. $SG_i(I) \prec G$.
(c) This holds true as G' and each of its subgoals is handled with recursive calls to procedure *reuse-proof* and this computation terminates. □