

Computing Tiny Clause Normal Forms

Noran Azmy and Christoph Weidenbach

Max Planck Institute for Informatics

Abstract. Automated reasoning systems which build on resolution or superposition typically operate on formulas in clause normal form (CNF). It is well-known that standard CNF translation of a first-order formula may result in an exponential number of clauses. In order to prevent this effect, renaming techniques have been introduced that replace subformulas by atoms over fresh predicates and introduce definitions accordingly. This paper presents *generalized renaming*. Given a formula and a set of subformulas to be renamed, it is suggested to use one atom to replace all instances of a generalization of a given subformula. A generalized renaming algorithm and an implementation as part of the SPASS theorem prover are described. The new renaming algorithm is faster than the previous one implemented in SPASS. Experiments on the TPTP show that generalized renaming significantly reduces the number of clauses and the average time taken to solve the problems afterward.

1 Introduction

Most automated reasoning systems operate on formulas in *clause normal form* (CNF). Many problem formulations, however, are given in full propositional or first-order logic. While every first-order formula can be translated into an equisatisfiable formula in CNF [2], the conversion leads to an exponential blow-up of the formula in the worst case. An obvious example is this formula in *disjunctive normal form* (DNF),

$$(X_1 \wedge Y_1) \vee (X_2 \wedge Y_2) \vee \cdots \vee (X_n \wedge Y_n),$$

with $O(n)$ literals, for which the CNF variant obtained by exhaustive application of distributivity laws has $O(2^n)$ literals.

A well-established workaround to this problem is *formula renaming*. Originally due to Tseitin [10], the idea is to introduce new predicate symbols to replace parts of the formula that might be “problematic”, i.e. that might be eventually responsible for exponential blow-up of the result. These replacements avoid the repeated application of distributivity laws. For example, a renamed variant of the above formula is,

$$(R_1 \vee \cdots \vee R_n) \wedge (R_1 \leftrightarrow (X_1 \wedge Y_1)) \wedge \cdots \wedge (R_n \leftrightarrow (X_n \wedge Y_n)).$$

In this manner, the maximum number of new symbols introduced is in $O(n)$, and the CNF translation also becomes linear in the size of the formula. It is

worth noting that renaming generates fewer clauses, without having an impact on the individual clause size. Because of the introduction of new symbols, formula renaming does not preserve logical equivalence. However, it is *satisfiability-preserving*; meaning that the resulting formula is satisfiable if and only if the original formula is satisfiable. This is not a problem in the context of automated theorem proving; CNF translation employs Skolemization and therefore is itself only satisfiability-preserving.

Existing work differs as to the criteria for choosing positions to rename. [8] suggests the replacement of every subformula, while [4] suggests only those replacements that would decrease the size of the CNF result. [7] shows an efficient method to find these “beneficial” replacements, which involves a linear-time computation of numbers and some arithmetic manipulation.

For first-order formulas, the authors of [7] mention the benefit of renaming several “compatible” subformulas using the same symbol. Two formulas φ_1, φ_2 are *compatible* if they are both instances of a common formula φ . We call such a φ a *generalization* of φ_1 and φ_2 . For k compatible subformulas with a generalization φ , we need only define one symbol with the “general” definition φ , as opposed to k symbols that standard renaming would introduce. As an example, standard renaming of the formula,

$$(P \vee (Q_1(a_1) \wedge Q_2(a_1))) \wedge \cdots \wedge (P \vee (Q_1(a_n) \wedge Q_2(a_n))),$$

would yield the following formula of $4n$ clauses,

$$\begin{aligned} & (P \vee R_1) \wedge (P \vee R_2) \wedge \cdots \wedge (P \vee R_n) \\ & \wedge R_1 \leftrightarrow Q_1(a_1) \wedge Q_2(a_1) \\ & \wedge \dots \\ & \wedge R_n \leftrightarrow Q_1(a_n) \wedge Q_2(a_n). \end{aligned}$$

Renaming of the compatible subformulas of the form $Q_1(a_i) \wedge Q_2(a_i)$ using one symbol yields a formula for which the CNF has only $n + 3$ clauses,

$$\begin{aligned} & (P \vee R(a_1)) \wedge (P \vee R(a_2)) \wedge \cdots \wedge (P \vee R(a_n)) \\ & \wedge \forall x [R(x) \leftrightarrow Q_1(x) \wedge Q_2(x)]. \end{aligned}$$

This inspires what we denote by *generalized formula renaming*. We assume the input to be a formula ψ , and a set Π of beneficial positions for renaming in ψ . For every beneficial position determined by the system, we would also like to rename all other compatible positions in ψ using the same predicate symbol.

Generalized renaming has several potential benefits. For redundant formulas, the clause set can be significantly reduced. New symbols are introduced minimally, and therefore there are fewer definition clauses. The individual clauses are also shortened by the replacements. Moreover, generalized renaming has the advantage of preserving structural information, which we will later show can have a significant effect on the time taken to find a proof.

This work is the first to study the generalized renaming problem. We present a formal generalized renaming algorithm with proof of correctness, and experiments based on our implementation of the algorithm as part of the SPASS automated theorem prover. Our tests show that generalized renaming is able to reduce the number of clauses in the CNF result by up to 92.9%, when compared to a standard renaming procedure where compatible subformulas are not considered. This significant reduction in size also reduces the time taken for CNF translation, and allows for potential reduction of the total run time of the prover. We will show cases where the performance of the prover is significantly enhanced with generalized renaming, and discuss the general effect of this approach on automated theorem proving.

We begin by fixing the notation and going over the necessary background in Section 2. Section 3 formally defines the problem and presents the algorithm. We show our experiments and results in Section 4, concluding with a discussion. All relevant proofs can be found in a technical report on [1].

2 Background

We rely mostly on the notation used in [7]. We assume a first-order language over a *signature* $\Sigma = (\mathcal{F}, \mathcal{R})$, where $\mathcal{F}$ and $\mathcal{R}$ are the sets of function and predicate symbols respectively. The *arity* of function (predicate) symbol f (P) is given by $\text{arity}(f)$ ($\text{arity}(P)$), with f/n denoting $\text{arity}(f) = n$. Additionally, we assume an infinite set $\mathcal{X}$ of *variable symbols* disjoint from the symbols in Σ . Terms, formulas, atoms, literals and clauses are defined in the usual way. A formula is in *clause normal form* if it is a conjunction of clauses. The set of positions of a term t (formula φ) is defined by: (1) $\epsilon \in \text{pos}(t)$ for any t , where $t|_\epsilon = t$, (2) if $t|_\pi = f(t_1, \dots, t_n)$ for $f \in \mathcal{F}$, then $\pi.i$ is a position in t for all $i = 1, \dots, n$, such that $t|_{\pi.i} = t_i$, (3) $\epsilon \in \text{pos}(\varphi)$ for any φ , where $\varphi|_\epsilon = \varphi$, (4) if $\varphi|_\pi = f(t_1, \dots, t_n)$ for $f \in \mathcal{F}$, then $\pi.i$ is a position in φ for all $i = 1, \dots, n$, such that $t\varphi|_{\pi.i} = t_i$, (5) if $\varphi|_\pi = P(t_1, \dots, t_n)$ for $P \in \mathcal{R}$, then $\pi.i$ is a position in φ for all $i = 1, \dots, n$, such that $\varphi|_{\pi.i} = t_i$, (6) if $\varphi|_\pi = \varphi_1 \odot \varphi_2$ for $\odot \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$, then $\pi.1$ and $\pi.2$ are positions in φ such that $\varphi|_{\pi.1} = \varphi_1, \varphi|_{\pi.2} = \varphi_2$, and (7) if $\varphi|_\pi = Qx\varphi'$ for $Q \in \{\forall, \exists\}$, or $\varphi|_\pi = \neg\varphi'$, then $\pi.1$ is a position in φ such that $\varphi|_{\pi.1} = \varphi'$.

We assume a *prefix order* $\leq$ on positions, with a corresponding strict ordering $<$. Two positions π_1, π_2 are *parallel*, denoted by $\pi_1 \parallel \pi_2$, if they are incomparable by $\leq$. In addition, there is the total lexicographic ordering $<_{lex}$. The size of a formula $|\varphi| = |\text{pos}(\varphi)|$. The *top symbol* of a term t (formula φ) is given by $\text{topsymbol}(t)$ ($\text{topsymbol}(\varphi)$). For a term t (formula φ), the function $\text{vars}(t)$ ($\text{vars}(\varphi)$) returns a set of variables that occur in t (φ). Similarly, $\text{freevars}(\varphi)$ is the set of free variables of φ .

The *polarity* of a position π in a formula ψ is denoted by $\text{pol}(\psi, \pi)$. Substitutions are defined in the usual way. We denote the identity substitution by σ_I . The *composition of two substitutions* σ, τ is denoted by $\sigma \circ \tau$. We denote the *most general unifier* by $\text{mgu}(t_1, t_2)$. For two formulas φ_1, φ_2 , $\text{subinst}(\varphi_1, \varphi_2) := \{\pi \mid \varphi_2|_\pi = \varphi_1\sigma \text{ for some substitution } \sigma\}$. The application of a substitution σ

is extended to terms by the rule $f(t_1, \dots, t_n)\sigma = f(t_1\sigma, \dots, t_n\sigma)$ for all $f \in \mathcal{F}$ with arity n . It is extended to formulas by (1) $P(t_1, \dots, t_n)\sigma = P(t_1\sigma, \dots, t_n\sigma)$ for every $P \in \mathcal{P}$, (2) $(\neg\varphi)\sigma = \neg(\varphi\sigma)$, (3) $(\varphi_1 \odot \varphi_2)\sigma = \varphi_1\sigma \odot \varphi_2\sigma$ for $\odot \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$, and (4) $(Qx\varphi)\sigma = Qy[(\varphi\{x \mapsto y\})\sigma]$ for $Q \in \{\forall, \exists\}$, where y is a fresh variable not in the domain or range of σ or $\text{freevars}(\varphi)$. An injective substitution σ for which $x\sigma \in \mathcal{X}$ for all $x \in \mathcal{X}$ is called a *variable renaming*. Syntactic equality on terms is defined in the usual way. For formulas, we define syntactic equality up to the names of quantified variables. Namely, for $Q \in \{\exists, \forall\}$, $Qx\varphi_1 = Qy\varphi_2$ if and only if $\varphi_1\{x \mapsto z\} = \varphi_2\{y \mapsto z\}$ for a fresh z . In all other cases, it is defined in the usual way.

We define formula renaming for first-order logic as in [7].

Definition 1. (Formula Renaming) *Let ψ, φ be first-order formulas, and π a non-term position such that $\psi|_\pi = \varphi$. Let $\text{freevars}(\varphi) = \{x_1, \dots, x_n\}$, and R a fresh predicate symbol of arity n . A formula renaming of ψ at π is the first-order formula $\psi[\pi/R(x_1, \dots, x_n)] \wedge \text{def}(\pi, \psi, R)$, where the formula $\text{def}(\pi, \psi, R)$ is a polarity-dependent definition of the new predicate R , defined as:*

$$\text{def}(\pi, \psi, R) := \begin{cases} \forall x_1, \dots, x_n [R(x_1, \dots, x_n) \rightarrow \psi|_\pi] & \text{if } \text{pol}(\psi, \pi) = 1, \\ \forall x_1, \dots, x_n [\psi|_\pi \rightarrow R(x_1, \dots, x_n)] & \text{if } \text{pol}(\psi, \pi) = -1, \\ \forall x_1, \dots, x_n [R(x_1, \dots, x_n) \leftrightarrow \psi|_\pi] & \text{if } \text{pol}(\psi, \pi) = 0. \end{cases} \quad (1)$$

The formula $\text{def}(\pi, \psi, R)$ defines the renaming predicate R in terms of $\psi|_\pi = \varphi$, according to the *polarity* of φ in ψ . This polarity-dependent definition is a well-known optimization, first suggested by [5], eliminating implications that are not useful for inferences. However, Equation 1 assumes that R is to be defined in terms of a subformula φ of ψ . In case of generalized renaming, we would like to define a renaming predicate R in terms of a *generalization* φ , where φ does not generally occur in ψ . We overload the function $\text{def}(\pi, \psi, R)$ with the following variant.

$$\text{def}(\varphi, p, R) = \begin{cases} \forall x_1, \dots, x_n [R(x_1, \dots, x_n) \rightarrow \varphi] & \text{if } p = 1, \\ \forall x_1, \dots, x_n [\varphi \rightarrow R(x_1, \dots, x_n)] & \text{if } p = -1, \\ \forall x_1, \dots, x_n [R(x_1, \dots, x_n) \leftrightarrow \varphi] & \text{if } p = 0, \end{cases} \quad (2)$$

where $\text{freevars}(\varphi) = \{x_1, \dots, x_n\}$ and $p \in \{-1, 0, 1\}$. In this context, p refers to the polarity of the *instances* of φ . Because different instances of the same generalization may occur with different polarities in ψ , we use one “general” polarity which is compatible with all these instances. For a formula ψ , the polarity of $\Pi \subset \text{pos}(\psi)$ in ψ , $\text{pol}(\psi, \Pi) = p$ if $\text{pol}(\psi, \pi) = p$ for all $\pi \in \Pi$, $\text{pol}(\psi, \Pi) = 0$ otherwise.

For two formulas φ_1 and φ_2 , we define the notion of generalization and compatibility as follows. A *generalization* of φ_1 and φ_2 is a formula φ such that for some substitutions σ_1, σ_2 , $\varphi\sigma_1 = \varphi_1$ and $\varphi\sigma_2 = \varphi_2$. The set of all generalizations of φ_1 and φ_2 is denoted by $\text{GEN}(\varphi_1, \varphi_2)$. Two formulas φ_1 and φ_2 are *compatible*, denoted by $\varphi_1 \sim \varphi_2$, if $\text{GEN}(\varphi_1, \varphi_2) \neq \emptyset$.

In order to minimize the introduction of new variables, and consequently the arity of renaming predicates, we are interested in *minimal* generalizations. A generalization $\varphi \in \text{GEN}(\varphi_1, \varphi_2)$ is called *minimal* if for all generalizations $\varphi' \in \text{GEN}(\varphi_1, \varphi_2)$, there exists some σ such that $\varphi = \varphi'\sigma$. It is clear to see that minimal generalizations are equal up to variable renaming.

We extend the notion of generalization to sets of positions. Let ψ be a formula and $\Pi \subset \text{pos}(\psi)$. A *generalization of Π in ψ* is a formula φ such that for all $\pi \in \Pi$, there is some σ such that $\psi|_\pi = \varphi\sigma$. The set of all generalizations of Π in ψ is denoted by $\text{GEN}(\psi, \Pi)$. A generalization $\varphi \in \text{GEN}(\psi, \Pi)$ is called *minimal* if for every $\varphi' \in \text{GEN}(\psi, \Pi)$ there is some σ such that $\varphi = \varphi'\sigma$.

Clearly, compatibility is an equivalence relation. For generalized renaming of compatible subformulas, we will cluster subformulas into equivalence classes with respect to compatibility. We denote these by *compatibility classes*. For a formula ψ , the *compatibility relation* $\sim_\psi$ on $\text{pos}(\psi)$ is an equivalence relation given by $\pi_1 \sim_\psi \pi_2$ if and only if $\psi|_{\pi_1} \sim \psi|_{\pi_2}$. The equivalence class of $\pi \in \text{pos}(\psi)$ with respect to $\sim_\psi$ is denoted by $[\pi]_\psi$. The *quotient set* of $\text{pos}(\psi)$ by $\sim_\psi$ is given by $Q_\psi := \{[\pi]_\psi \mid \pi \in \text{pos}(\psi)\}$.

3 Generalized Formula Renaming

The input to the renaming algorithm is a formula ψ and a set of positions $\Pi \subset \text{pos}(\psi)$ to be renamed in ψ . Consider the formula,

$$\begin{aligned} \psi := & \forall x, y [P(x) \leftrightarrow (Q(x) \leftrightarrow (P(y) \leftrightarrow (Q(y) \leftrightarrow (P(b) \leftrightarrow Q(b))))) \\ & \wedge ((P(a_1) \leftrightarrow Q(a_1)) \wedge \cdots \wedge (P(a_n) \leftrightarrow Q(a_n)))] \end{aligned}$$

We assume that $\Pi = \{111122, 1111222, 11112222\}$, since renaming at these positions eventually reduces the number of clauses in the CNF transformation. A *standard renaming* of ψ at Π is the formula,

$$[\psi[\pi_i/R_i(x_1, \dots, x_n)]] \wedge \text{def}(\pi_i, \psi, R_i)]_{\pi_i \in \Pi},$$

where $\{x_1, \dots, x_n\}$ are the free variables of $\psi|_{\pi_i}$. Applying standard renaming in our example yields the following formula,

$$\begin{aligned} & \forall x, y [P(x) \leftrightarrow (Q(x) \leftrightarrow R_2(y))] \\ & \wedge \forall x [R_2(x) \leftrightarrow (P(x) \leftrightarrow R_1(x))] \\ & \wedge \forall x [R_1(x) \leftrightarrow (Q(x) \leftrightarrow R_0)] \\ & \wedge [R_0 \leftrightarrow (P(b) \leftrightarrow Q(b))] \\ & \wedge (P(a_1) \leftrightarrow Q(a_1)) \wedge \cdots \wedge (P(a_n) \leftrightarrow Q(a_n)) \end{aligned}$$

for which the CNF translation has $2n + 16$ clauses. Individual renaming of any of the subformulas $P(a_i) \leftrightarrow Q(a_i)$ does not reduce the number of clauses due to the need to introduce clauses for the definitions. However, we notice that the atom R_0 is used to rename a compatible subformula $P(b) \leftrightarrow Q(b)$, and can be used to rename all subformulas $P(a_i) \leftrightarrow Q(a_i)$ as well, without the introduction

of more definitions. Now, $P(b) \leftrightarrow Q(b), P(a_1) \leftrightarrow Q(a_1), \dots, P(a_n) \leftrightarrow Q(a_n)$ are instances of $P(x) \leftrightarrow Q(x)$. We use this generalization to define the predicate R_0 , which is now assigned an arity of 1. This *generalized renaming* yields,

$$\begin{aligned} & \forall x, y [P(x) \leftrightarrow (Q(x) \leftrightarrow R_2(y))] \\ & \wedge \forall x [R_2(x) \leftrightarrow (P(x) \leftrightarrow R_1(x))] \\ & \wedge \forall x [R_1(x) \leftrightarrow (Q(x) \leftrightarrow R_0(b))] \\ & \wedge \forall x [R_0(x) \leftrightarrow (P(x) \leftrightarrow Q(x))] \\ & \wedge R_0(a_1) \wedge \dots \wedge R_0(a_n) \end{aligned}$$

for which the CNF translation has $n + 16$ clauses.

Formally, we define generalized renaming as follows.

Definition 2. (Generalized Formula Renaming) *Let ψ be a formula and $\Pi \subset \text{pos}(\psi)$ be a set of non-term positions in ψ . Then $\langle \psi, \Pi \rangle$ is a formula renaming pair. Let $\text{sort}_{\text{lex}}(\Pi) = [\pi_1, \dots, \pi_n]$ be a list of the elements in Π , sorted in descending lexicographic order. A generalized renaming of Π in ψ is given by $\text{rename}(\psi, \text{sort}_{\text{lex}}(\Pi))$, where,*

$$\begin{aligned} \text{rename}(\psi, \text{NIL}) &= \psi \\ \text{rename}(\psi, [\pi_i \mid \Pi]) &= \text{rename}(\text{rename}(\psi, \pi_i), \Pi) \\ \text{rename}(\psi, \pi_i) &= \psi \left[\pi_i^j / R_i(x_1, \dots, x_m) \sigma_i^j \right]_{\pi_i^j \in [\pi_i]_\psi} \wedge \text{def}(\varphi_i, p_i, R_i) \end{aligned}$$

where $\varphi_i \in \text{GEN}(\psi, [\pi_i]_\psi)$, $\psi|_{\pi_i^j} = \varphi_i \sigma_i^j$, $\text{freevars}(\varphi_i) = \{x_1, \dots, x_m\}$, $p_i = \text{pol}(\psi, [\pi_i]_\psi)$ and R_i is a fresh predicate symbol of arity m .

Let $\psi_{\text{SR}}, \psi_{\text{GR}}$ be the CNF translations of ψ produced by standard and generalized renaming respectively. It is worth mentioning that whenever there is no redundancy to be exploited in the input formula ψ , i.e. whenever $[\pi]_\psi = \{\pi\}$ for all $\pi \in \Pi$, generalized renaming reduces to standard renaming of ψ at all positions $\pi \in \Pi$, and $\psi_{\text{SR}} = \psi_{\text{GR}}$.

In general, ψ_{GR} contains at most as many clauses as ψ_{SR} . It is possible, however, that generalized renaming results in more clauses than standard renaming by the above definition. When compatible instances of different polarities are included in the renaming, a definition may expand into a full equivalence where only an implication would suffice in the standard version. For example, if only position $\pi = 1$ is beneficial in $\psi := \phi\sigma_1 \wedge \neg\phi\sigma_2$, standard renaming would yield $A \wedge \neg\phi\sigma_2 \wedge \forall x_1, \dots, x_n [A \rightarrow \phi\sigma_1]$ for some atom A , where $\text{freevars}(\phi\sigma_1) = \{x_1, \dots, x_n\}$. Generalized renaming includes the compatible subformula $\phi\sigma_2$, yielding $B\sigma_1 \wedge \neg B\sigma_2 \wedge \forall y_1, \dots, y_m [B \leftrightarrow \phi]$ for some atom B , where $\text{freevars}(\phi) = \{y_1, \dots, y_m\}$, doubling the size of the definition. If we are interested in minimizing the number of clauses, generalized renaming may be

restricted to include only those compatible instances that will not expand the definition. Both types of generalized renaming are included in our SPASS implementation.

3.1 A Generalized Renaming Algorithm

Detecting compatible subformulas and constructing the appropriate definitions involves a process of repeated generalization. There are several issues to note here. First, every replacement of the form $\psi[\pi/R(x_1, \dots, x_m)\sigma]$ requires the computation of the substitution σ . There are $|\psi|$ such substitutions in the worst case, each of size $O(|\psi|)$, which means that computing them requires quadratic time. To avoid this, we note that an upper replacement in the formula hides the effect of all lower replacements. Therefore, we need only perform those replacements at *top-most renaming positions*. A *top-most position in ψ* with respect to $\Pi \subseteq \text{pos}(\psi)$ is a position π such that for all $\pi' \in \Pi$, $\pi' > \pi$ or $\pi' \parallel \pi$. Top-most renaming positions are the top-most positions in Π . The sum of sizes of formulas at top-most renaming positions is at most linear.

However, nested renamings cause one renaming predicate to appear in the definition of another, which can be seen in our example, where $R_0(b)$ occurs in the definition of $R_1(x)$. This requires renaming steps to be performed in a bottom-up manner, as indicated in the definition. Compatibility checks on subformulas of a certain size can only be performed once all deeper subformulas (of smaller size) have been renamed. In order to employ top-most replacement, the definitions have to be constructed in such a way that is independent from the replacement steps. We will later show that, since dependencies between definitions correspond to similar dependencies between subformulas, we can use formula structure to compute the definitions correctly in a bottom-up manner, independent from the replacements.

We now present an algorithm for generalized renaming on a formula renaming pair $\langle \psi, \Pi \rangle$. Our algorithm performs replacements and generation of definitions in two separate stages.

The first step is to cluster subformulas of ψ into compatibility classes. The procedure described in COMPUTE-COMPATIBILITY performs this task. The result is a mapping M from subformulas to their respective generalizations.

COMPUTE-COMPATIBILITY relies on a process of repeated generalization. For this purpose, GENERALIZE describes a minimal generalization procedure for two input formulas φ_1, φ_2 . Temporary substitutions are kept in σ_1, σ_2 . A list L is used to keep track of bound variables, to prevent generalization above a bound variable position. It can be shown that the generalization returned by GENERALIZE is minimal.

At this point, every position $\pi \in \text{pos}(\psi)$ is assigned a minimal generalization $\varphi \in \text{GEN}(\psi, [\pi]_\psi)$. Every such φ is assigned the polarity $\text{pol}(\psi, [\pi]_\psi)$, and φ is marked for renaming if and only if $[\pi]_\psi \cap \Pi \neq \emptyset$.

Let $\varphi_1, \dots, \varphi_n$ be the set of constructed generalizations, where for $1 \leq i \leq n$, $\varphi_i \in \text{GEN}([\pi_i]_{\psi})$ for some π_i , and φ_i is assigned polarity p_i and renaming predicate R_i . Because of possible nested renamings, R_i cannot be defined directly in terms of φ_i . In order to define R_i correctly, we need to detect what we call *direct dependencies* of φ_i on φ_j , where $j \neq i$, $1 \leq j \leq n$, and φ_j is marked for renaming. A *direct dependency* is the case where $\varphi_i|_{\pi} = \varphi_j\sigma$ for some π, σ such that $\varphi_i|_{\pi'} = \varphi_k\sigma'$ for no φ_k marked for renaming, $1 \leq k \leq n$, $k \neq i \neq j$ and $\pi' < \pi$. For every direct dependency $\varphi_i|_{\pi} = \varphi_j\sigma$, φ_i should be replaced at π with $R_j(x_1, \dots, x_m)\sigma$, where $\text{freevars}(\varphi_j) = \{x_1, \dots, x_m\}$.

1 Procedure: COMPUTE-COMPATIBILITY $(\langle \psi, \Pi \rangle, \pi)$

Input: A formula renaming pair $\langle \psi, \Pi \rangle$ and a position $\pi \in \text{pos}(\psi)$.

Result: A mapping M from subformulas of $\psi|_{\pi}$ to the respective generalizations.

```

2 if  $\psi|_{\pi}$  is a formula then
3   if  $\psi|_{\pi}$  is not an atom then
4     | Recursively compute compatibility for subformulas of  $\psi|_{\pi}$ .
5   end
6   for all generalizations  $\varphi$  computed so far do
7     | Set  $\langle \varphi, \sigma_1, \sigma_2 \rangle$  to GENERALIZE  $(\psi|_{\pi}, \varphi, \emptyset, \emptyset, \emptyset)$ .
8     | if  $\varphi \neq \perp$  then
9       | Set  $M(\psi|_{\pi})$  to  $\varphi$ .
10      | if the polarity assigned to  $\varphi$  is different from  $\text{pol}(\psi, \pi)$  then assign
11        |  $\varphi$  a polarity 0.
12        | if  $(\pi \in \Pi)$  then mark  $\varphi$  for renaming.
13        | break.
14      end
15    end
16    if no generalization has been found for  $\psi|_{\pi}$  then
17      | Construct a new generalization  $\varphi = \psi|_{\pi}$ .
18      | Assign  $\varphi$  the polarity  $\text{pol}(\psi, \pi)$ .
19      | if  $\pi \in \Pi$  then mark  $\varphi$  for renaming.
20      | Set  $M(\psi|_{\pi})$  to  $\varphi$ .
21    end
22  end

```

1 **Procedure:** GENERALIZE($\varphi_1, \varphi_2, \sigma_1, \sigma_2, L$)

Input: Formulas or terms φ_1, φ_2 , substitutions σ_1, σ_2 , and a list L of variables.

Output: $\langle \varphi, \sigma_1, \sigma_2 \rangle$ where $\varphi_i = \varphi \sigma_i$ for $i \in \{1, 2\}$ if $\varphi_1 \sim \varphi_2$, $\langle \perp, \text{NIL}, \text{NIL} \rangle$ otherwise.

```

2 if topsymbol( $\varphi_1$ ) = topsymbol( $\varphi_2$ ) then
3   switch topsymbol( $\varphi_1$ ) do
4     case predicate/function symbol  $P$  where  $\varphi_1 = P(t_1, \dots, t_n)$  and
        $\varphi_2 = P(s_1, \dots, s_n)$ 
5       Create list  $A[1, \dots, n]$ .
6       for  $i = 1$  to  $n$  do
7         Set  $\langle t, \sigma_1, \sigma_2 \rangle$  to GENERALIZE( $t_i, s_i, \sigma_1, \sigma_2, L$ ).
8         if  $t = \perp$  then return  $\langle \perp, \text{NIL}, \text{NIL} \rangle$ .
9          $A[i] \leftarrow t$ .
10      end
11      return  $\langle P(A[1], \dots, A[n]), \sigma_1, \sigma_2 \rangle$ .
12     case negation symbol  $\neg$ 
13       Set  $\langle \varphi, \sigma_1, \sigma_2 \rangle$  to GENERALIZE( $\varphi_1|_1, \varphi_2|_1, \sigma_1, \sigma_2, L$ ).
14       if  $\varphi = \perp$  then return  $\langle \perp, \text{NIL}, \text{NIL} \rangle$ .
15       return  $\langle \neg \varphi, \sigma_1, \sigma_2 \rangle$ .
16     case  $\odot \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$ 
17       Set  $\langle \varphi'_1, \sigma_1, \sigma_2 \rangle$  to GENERALIZE( $\varphi_1|_1, \varphi_2|_1, \sigma_1, \sigma_2, L$ ).
18       Set  $\langle \varphi'_2, \sigma_1, \sigma_2 \rangle$  to GENERALIZE( $\varphi_1|_2, \varphi_2|_2, \sigma_1, \sigma_2, L$ ).
19       if  $\varphi'_1$  or  $\varphi'_2$  is  $\perp$  then return  $\langle \perp, \text{NIL}, \text{NIL} \rangle$ .
20       return  $\langle \varphi'_1 \odot \varphi'_2, \sigma_1, \sigma_2 \rangle$ .
21     case  $Q \in \{\forall, \exists\}$  where  $\varphi_i = Qx_i \varphi'_i$ 
22       Rename  $x_i$  in  $\varphi'_i$  to  $x \notin \text{vars}(\varphi_i) \cup \text{dom}(\sigma_i) \cup \text{range}(\sigma_i)$ . Add  $x$  to  $L$ .
23       Set  $\langle \varphi, \sigma_1, \sigma_2 \rangle$  to GENERALIZE( $\varphi'_1, \varphi'_2, \sigma_1, \sigma_2, L$ ).
24       if  $\varphi = \perp$  then return  $\langle \perp, \text{NIL}, \text{NIL} \rangle$ .
25       return  $\langle Qx\varphi, \sigma_1, \sigma_2 \rangle$ .
26     case variable  $x$  return  $\langle x, \sigma_1, \sigma_2 \rangle$ .
27   endsw
28 else if  $\varphi_1, \varphi_2$  are terms such that no  $x \in L$  occurs in  $\varphi_i$  then
29   if there is a variable  $x$  such that  $x\sigma_i = \varphi_i$  then return  $\langle x, \sigma_1, \sigma_2 \rangle$ .
30   else
31     Let  $x$  be a fresh variable not in  $\varphi_i, \text{dom}(\sigma_i), \text{range}(\sigma_i)$ .
32      $\sigma_i \leftarrow \sigma_i \cup \{x \mapsto \varphi_i\}$ .
33     return  $\langle x, \sigma_1, \sigma_2 \rangle$ .
34   end
35 else return  $\langle \perp, \text{NIL}, \text{NIL} \rangle$  // formulas with different top symbols

```

Instead of searching for these dependencies in a naive way, we notice that for generalizations φ_1, φ_2 of $\psi|_{\pi_1}, \psi|_{\pi_1\pi_2}$ respectively, $\varphi_1|_{\pi_2} = \varphi_2\sigma$ for some σ . This is due to the property of *coverage*. A formula φ *covers* another formula ψ if for every formula φ' and substitution σ such that $\varphi = \varphi'\sigma$, $|\text{subinst}(\psi, \varphi)| = |\text{subinst}(\psi, \varphi')|$. Coverage can be shown to hold for generalizations constructed by our procedure (see technical report). Therefore, we may solve the dependencies by constructing definitions in a bottom-up manner, guided by the structure of ψ . At every position π in a bottom-up traversal of ψ , let φ be the generalization assigned to $\psi|_{\pi}$. We compute the definition def_{φ} for φ , as well as a *representative* rep_{φ} . If φ is marked for renaming, rep_{φ} is simply the renaming atom for φ . Otherwise, it is identical to def_{φ} . The definition def_{φ} can be easily computed using $\text{rep}_{\varphi'}$, where φ' are the generalizations of the immediate subformulas of $\psi|_{\pi}$.

Using our previous example, let $\varphi_1 = Q(x)$, $\varphi_2 = P(x) \leftrightarrow Q(x)$, and $\varphi_3 = Q(x) \leftrightarrow (P(y) \leftrightarrow Q(y))$ be generalizations constructed for $n+3$ and $n+1$ and 1 instance(s) in ψ , respectively. Because φ_1 is an atom unmarked for renaming, $\text{def}_{\varphi_1} = \text{rep}_{\varphi_1} = \varphi_1$. Now, φ_2 is marked for renaming with renaming predicate symbol R_0 , and so $\text{def}_{\varphi_2} = P(x) \leftrightarrow Q(x)$, and $\text{rep}_{\varphi_2} = R_0(x)$. Similarly, φ_3 is marked for renaming with predicate symbol R_1 , and $\text{rep}_{\varphi_3} = R_1(x)$. We construct $\text{def}_{\varphi_3} = \text{rep}_{\varphi_1} \leftrightarrow (\text{rep}_{\varphi_2} \{x \mapsto b\})$, because $\varphi_3 = \varphi_1 \leftrightarrow \varphi_2 \{x \mapsto b\}$.

Finally, REPLACE-TOPMOST performs top-most replacement, by traversing ψ in a top-down manner. If the current position π is a renaming position, it is replaced by the corresponding renaming atom. Otherwise, recursive calls are made to the immediate subformulas $\pi.1$ and $\pi.2$, if any.

4 Experiments

The generalized formula renaming algorithm is implemented as part of SPASS version 3.8 [11]. The TPTP problem library [9], version 5.4.0, was used for testing. Tests were run on a cluster of computing nodes with quad-core processors 2x Intel Xeon E5620 @ 2.40GHz, and 48GB RAM. The input set of beneficial renaming positions Π is computed using the same method in SPASS 3.7, suggested in [7].

The first experiment compared generalized renaming to standard renaming on all 7807 problems given in full first-order logic, from 36 different domains. In this experiment, only the CNF translation was performed, and the number of clauses and time taken for CNF translation was compared. For the second experiment, we considered the subset of these problems for which CNF translation finished within the given time limit, and generalized renaming was able to reduce the number of clauses by at least 5%. These are 528 problems from 16 different domains.

The CNF translation results can be found on the SPASS homepage [1].

1 Procedure: GENERATE-DEFINITIONS (ψ, π)**Input:** A formula ψ and position $\pi \in \text{pos}(\psi)$.**Output:** A conjunction δ of definition formulas for all generalizations of subformulas at or below $\psi|_\pi$.

```

2 Set  $\delta$  to  $\top$ .
3 Let  $\varphi$  be the generalization assigned to  $\psi|_\pi$ .
4 if  $\varphi$  is not yet defined then
5   switch topsymbol ( $\psi|_\pi$ ) do
6     case negation symbol  $\neg$ 
7       Set  $\delta$  to GENERATE-DEFINITIONS ( $\psi, \pi.1$ ).
8       Set  $\varphi_1$  to the generalization of ( $\psi|_{\pi.1}$ ).
9       Set  $\text{def}_\varphi$  to  $\neg(\text{rep}_{\varphi_1})\sigma$ , where  $\varphi = \neg\varphi_1\sigma$ .
10    case one of  $\{\forall, \exists\}$ 
11      Set  $\delta$  to GENERATE-DEFINITIONS ( $\psi, \pi.1$ ).
12      Set  $\varphi_1$  to the generalization of ( $\psi|_{\pi.1}$ ).
13      Set  $\text{def}_\varphi$  to  $Qx(\text{rep}_{\varphi_1})\sigma$ , where  $\varphi = Qx[\varphi_1\sigma]$ .
14    case  $\odot \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$ 
15      Set  $\delta$  to
16        GENERATE-DEFINITIONS ( $\psi, \pi.1$ )  $\wedge$  GENERATE-DEFINITIONS ( $\psi, \pi.2$ ).
17      Set  $\varphi_i$  to the generalization of ( $\psi|_{\pi.i}$ ) for  $i \in \{1, 2\}$ .
18      Set  $\text{def}_\varphi$  to  $(\text{rep}_{\varphi_1})\sigma_1 \odot (\text{rep}_{\varphi_2})\sigma_2$  for  $\varphi = \varphi_1\sigma_1 \odot \varphi_2\sigma_2$ .
19    otherwise Set  $\text{def}_\varphi$  to  $\varphi$ . // predicate symbol
20  endsw
21  if  $\varphi$  is marked for renaming then
22    Set  $\text{rep}_\varphi$  to  $R(x_1, \dots, x_n)$ , where  $\text{freevars}(\varphi) = \{x_1, \dots, x_n\}$  and  $R/n$  is
23    a new predicate symbol.
24    Set  $\delta$  to  $\delta \wedge \text{def}(\text{def}_\varphi, p, R)$ , where  $p$  is the polarity assigned to  $\varphi$ .
25  else Set  $\text{rep}_\varphi$  to  $\text{def}_\varphi$ .
26  Mark  $\varphi$  as defined.
27 end
28 return  $\delta$ .

```

1 Procedure: REPLACE-TOPMOST (ψ, π)**Input:** A formula ψ and position π .**Result:** Every renaming position in $\psi|_\pi$ is replaced by the corresponding renaming predicate.

```

2 if  $\psi|_\pi$  is a non-atomic formula then
3   Let  $\varphi$  be the generalization assigned to  $\psi|_\pi$ .
4   if  $\varphi$  is marked for renaming then set  $\psi$  to  $\psi[\pi / (\text{rep}_\varphi)\sigma]$  for  $\psi|_\pi = \varphi\sigma$ .
5   else recursively perform top-most replacement on subformulas of  $\psi|_\pi$ .
6 end

```

4.1 Comparing the Size of CNF

In this experiment, we performed CNF translation on 7807 TPTP problems using both standard renaming (SR), and generalized renaming (GR). The respective number of clauses and time taken for CNF translation were compared. The number of problems for which both versions terminated within the time limit of one hour is 7231, leaving 576 particularly large problems. This is partly due to the fact that the SPASS 3.8 implementation is not targeted towards very large input files in general.

Fewer Clauses: Generalized renaming reduces the number of clauses in the CNF result for 1983 (27.4%) of the problems. Clause reduction may be as high as 93%, as in problem ALG207+1. In this problem, 147 occurrences of subformulas of the form $(\text{op}(x_1, y_1) \approx z_1) \vee \dots \vee \text{op}(x_7, y_7) \approx z_7$ are replaced by instances of $\text{SkP0}(x_1, y_1, z_1, \dots, x_7, y_7, z_7)$, reducing the number of clauses from 1098 to only 78. Average clause reduction over each problem domain also proves significant, reaching 15.4% for some domains (Table 1a). For 528 problems, there is at least 5% reduction in the number of clauses. We focus on these problems in our theorem proving experiment in the next section. Table 1b shows clause reduction for hard problems. For some of the hardest TPTP problems (with rating 1.0), generalized renaming was able to reduce the number of clauses by up to 87%. Although this reduction in size does not yet enable SPASS to solve these problems, the significant reduction in size means that generalized renaming is able to discover a hidden structure in these problems. Developing new reasoning mechanisms for the abbreviated structure is a new approach towards tackling these problems in the future (see our discussion on theorem proving mechanisms in the next section).

Less CNF Translation Time: On average, generalized renaming does not require more time for CNF translation. In fact, while the average improvement in CNF translation time is 0.6%, the time taken for CNF translation is significantly reduced in many cases. Naturally, this depends on the structure of the problem, the applicability of generalized renaming, compared to the overhead required for compatibility computation. Both the maximum improvement and deterioration in CNF translation time are around 30%, for problems CSR086+2 and CSR104+2 respectively.

4.2 Comparing Theorem Proving Time

For this experiment, SPASS was run on all 528 problems for which there was at least 5% clause reduction, to test if this significant size improvement corresponds to an improvement in proof times. For these problems, there was 30.9% average clause reduction, and 37.8% average reduction in CNF translation time. The time limit was set to 1 day. The total run time of the prover was compared.

For the problems solved by both versions, generalized renaming is able to outperform standard renaming, as shown in Table 1. The time taken for theorem proving is improved by 59.62%.

Table 1. Some of the best results according to problem domain/difficulty

(a) Problem domains with significant clause reduction.

DOMAIN	CLAUSES SAVED	CNF TIME SAVED
SWV	15.40%	93.14%
LCL	15.22%	16.28%
ALG	12.72%	3.87%
COM	3.35%	4.32%
MSC	2.93%	6%

(b) Hard problems with significant clause reduction.

PROBLEM COUNT	RATING	MAXIMUM CLAUSES SAVED
32	1.0	86.9%
9	[0.9, 1.0[	21.7%
21	[0.8, 0.9[	23.8%
28	[0.7, 0.8[	85.5%
27	[0.6, 0.7[	85.7%
62	[0.5, 0.6[	67.6%

Table 2. Problems where SPASS terminates within the time limit of 1 day, using both renaming versions. Generalized renaming significantly outperforms standard renaming with respect to theorem proving.

PROBLEM COUNT	AVERAGE CNF SIZE		AVERAGE TIME		TIME IMPROVEMENT
	SR	GR	SR	GR	
381	287.88	194.90	294.39	118.88	59.62%

However, the effect of generalized renaming on the time taken to find a proof for a specific problem is rather complex. Generalized renaming has two main effects on the resulting clause set. First, generalization results in clauses with more variables. This has many consequences. For one, ground clauses produced by standard renaming may correspond to non-ground ones produced by generalized renaming, which means that certain mechanisms such as splitting or reduction may no longer be possible. An example of this is problem ALG091+1, where generalized renaming succeeds in reducing the number of clauses by 90.5%. However, the CNF result after generalized renaming contains the clause $(\text{op}(u, u) \approx v \vee \text{SkP0}(w, x, y, v, u))$. This clause cannot be split, because of the shared variables between the two literals. It corresponds, however, to several ground clauses in the standard renaming version, each of which can be split. In this case, splitting is essential to finding the proof. A mechanism of splitting together with instantiation is needed, as suggested in [6]. However, this has not yet been implemented.

It may also happen that some reductions performed using standard renaming are not performed in the generalized version. For example, for ALG047+1 a ground reduction turns into an instance of contextual rewriting after translation

with generalized renaming. As this instance of contextual rewriting is currently not implemented in SPASS, the problem can no longer be solved.

Second, the ordering of literals within a clause in one version may be different from that within the corresponding clause in the other version, due to the reduction in size of literals at renaming positions. This means that some inferences may no longer be possible, and vice versa, because the maximal literal in the clause is different. When a renaming predicate SkP is introduced with an equivalence formula as a definition $\text{SkP} \leftrightarrow \varphi$, there are essentially two proof directions for the prover to take. One direction is to apply inferences involving SkP in order to *expand* the definition, i.e. revert the renaming. The other direction involves inferences that do not revert the renaming, but use information about SkP to find the proof. It is clear that these two proof attempts are complementary. While the first direction is suited for some problems, the second may be better-suited for a different class of problems. An example of this is problem NLP160+1, where the prover sidetracks to irrelevant inferences involving the renaming predicate SkP2 , marked as maximal in the respective clauses. By enforcing an ordering in which SkP2 is no longer maximal, the problem is solved.

Third, the change in the individual clause size and depth means that clauses are selected in different order by the prover. It may happen that useful clauses are only considered much later (or vice versa). In short, slight changes in the shape of the input clause set may lead the prover down a very different path. The outcome depends mostly on the problem, but also on the approach/heuristics used by the prover.

Based on the above, we have implemented two additional settings in our generalized renaming technique. The first setting, denoted by GC, prevents generalized renaming of closed subformulas (i.e. subformulas with no free variables). For any set of pairwise-compatible subformulas with no free variables, each subformula is renamed separately, i.e. only if its renaming is beneficial with respect to the number of clauses. This setting is well-suited for a number of TPTP problems where ground reductions are particularly helpful in finding a proof.

The second setting, denoted by GMX, enforces proof attempts that do not expand renaming predicate definitions. When enabled, renaming predicates whose definitions are expanded from an implication into a full equivalence by generalized renaming are assigned a minimal ordering with respect to other predicates.

Testing each setting individually against standard renaming on all 528 problems, we get the results shown in Table 3. Although SR manages to solve the highest number of problems in total, it solves fewer problems than the three versions of generalized renaming combined. This is because the different flavors of generalized renaming are suited for different types of input problems. For example, the 391 problems solved by plain GR are not the same as those solved by GC. These initial results suggest that a more robust renaming procedure should combine all these different versions.

By running a simple portfolio of GC for 30 seconds, GMX for 10 seconds and plain generalized renaming for the rest of the timeout of 1 day, we obtain the results summarized in Tables 4 and 5. Out of the problems solved by standard

Table 3. Performance of standard renaming along with three versions of generalized renaming.

	Solved	Unsolved
SR	414	114
PLAIN GR	391	137
GC	391	137
GMX	402	126

Table 4. Problems where SPASS terminates within the time limit of 1 day, using both renaming versions with GC and GMX settings. The generalized renaming ensemble significantly outperforms simple standard renaming.

PROBLEM COUNT	AVERAGE TIME		TIME IMPROVEMENT
	SR	GR/GC/GMX	
409	281.77	145.42	48.4%

Table 5. Problems where SPASS terminates within the time limit of 1 day, using both renaming versions with GC and GMX settings. SPASS does not terminate using simple SR.

PROBLEM	TIME (GR/GC/GMX)	PROBLEM	TIME (GR/GC/GMX)
LCL649+1.010	22.51	LCL659+1.020	0.3
LCL659+1.005	0.07	LCL660+1.005	1439.17
LCL659+1.010	0.17	LCL660+1.010	15080.06
LCL659+1.015	0.24	SWV024+1	2359.38

renaming, generalized renaming only fails to solve one problem, LCL684+1.005. This problem is solved by the standard procedure in 30.14 seconds. On the other hand, our technique is able to solve 8 problems that cannot be solved using standard renaming. Out of 528 problems, 110 remain unsolved by both procedures.

5 Conclusion

This paper provides a formal definition, algorithm and implementation for the generalized renaming problem. Given a formula ψ , our goal is to obtain a shorter

CNF result, by renaming ψ to an equisatisfiable variant ψ' . Instead of renaming every subformula of ψ , we assume a set Π of *renaming positions* is given as input. We rename ψ at all the given renaming positions, as well as positions of *compatible subformulas*, which can be renamed at no additional cost. We define compatible formulas as formulas that are instances of a common formula. Of course, our generalized renaming technique is also applicable to the translation of propositional formulas and SAT solving [3]. This is also an interesting starting point for further experiments, because in SAT solving the strengthening of the polarity of a definition often speeds up a subsequent solver run.

Our experiments show that generalized renaming reduces the number of CNF clauses by up to 92.9%. For some problem domains, the average clause reduction is as high as 15.4%. The time taken for CNF transformation is also significantly reduced in the average case. More importantly, generalized renaming works for some cases for which the standard renaming procedure fails. Although the time taken to find a proof is improved by up to 59.62% in the general case, the effect of introducing more variables is rather complex and may lead to an increase in time. With the introduction of some new mechanisms, however, our technique is able to outperform the standard procedure, with a time improvement of 48.4%.

Limitations and Future Work. Technically, logical operators may have arbitrary number of arguments. Suppose some subformula ϕ is to be renamed in $\psi := \phi_1 \wedge \dots \wedge \phi_i \wedge \phi_{i+1} \wedge \dots \wedge \phi_n$, represented as one conjunction operator with n arguments. If $\phi_i \wedge \phi_{i+1} \sim \phi$, this is not detected by our algorithm; the conjunction $\phi_i \wedge \phi_{i+1}$ does not exist from a technical point of view. In order to handle such cases, compatibility computation should rely on a more sophisticated pattern-matching procedure on term trees, instead of subformula checks. The algorithm also not aware of associativity/commutativity. If some subformula ϕ of $\psi := \phi_1 \wedge (\phi_2 \wedge \phi_3)$ is to be renamed and $\phi \sim \phi_1 \wedge \phi_2$, this compatible instance is not found in ψ . Since this is an NP problem, an efficient implementation modulo associativity/commutativity must resort to heuristics or approximations that provide accurate results without compromising efficiency.

As discussed in Section 4.2, there is potential to find more proofs/model by adapting the inference and reduction mechanisms in SPASS to the structure of the clauses generated by generalized renaming. For example, new variants of contextual rewriting can be implemented and dedicated to the renaming predicates introduced by generalized renaming.

References

1. SPASS Current Prototypes and Experiments,
<http://www.spass-prover.org/prototypes/index.html>
2. Baaz, M., Egly, U., Leitsch, A.: Normal Form Transformations. In: Voronkov, A., Robinson, A. (eds.) *Handbook of Automated Reasoning*, ch. 5, pp. 273–333
3. Biere, A., Heule, M., Maaren, H.V., Walsh, T. (eds.): *Handbook of Satisfiability*. IOS Press (2009)

4. Thierry Boy de la Tour. An Optimality Result for Clause Form Translation. *Journal of Symbolic Computation* 14(4), 283–301 (1992)
5. Henschen, L., Lusk, E., Overbeek, R., Smith, B.T., Veroff, R., Winker, S., Wos, L.: Challenge Problem 1. *SIGART Newsletter* (72), 30–31 (1980)
6. Hillenbrand, T., Weidenbach, C.: Superposition for bounded domains. In: Bonacina, M.P., Stickel, M.E. (eds.) *McCune Festschrift. LNCS (LNAI)*, vol. 7788, pp. 68–100. Springer, Heidelberg (2013)
7. Nonnengart, A., Weidenbach, C.: Computing Small Clause Normal Forms. In: Robinson, A., Voronkov, A. (eds.) *Handbook of Automated Reasoning*, ch. 6, pp. 337–367 (2001)
8. Plaisted, D.A., Greenbaum, S.: A Structure-preserving Clause Form Translation. *Journal of Symbolic Computation* 2(3), 293–304 (1986)
9. Sutcliffe, G., Suttner, C.: The TPTP Problem Library for Automated Theorem Provers (September 2010), <http://www.tptp.org/>
10. Tseitin, G.S.: On the Complexity of Derivation in Propositional Calculus. *Studies in Constructive Mathematics and Mathematical Logic* 8(115-125), 234–259 (1968)
11. Weidenbach, C., Dimova, D., Fietzke, A., Kumar, R., Suda, M., Wischniewski, P.: SPASS Version 3.5. In: Schmidt, R.A. (ed.) *CADE 2009. LNCS*, vol. 5663, pp. 140–145. Springer, Heidelberg (2009)