On Generating Small Clause Normal Forms

Andreas Nonnengart[1], Georg Rock[2], and Christoph Weidenbach[1]

[1] Max-Planck-Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany, email: {nonnenga, weidenb}@mpi-sb.mpg.de
[2] German Research Center for Artificial Intelligence GmbH, Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany, email: rock@dfki.de

Abstract. In this paper we focus on two powerful techniques to obtain compact clause normal forms: Renaming of formulae and refined Skolemization methods. We illustrate their effect on various examples. By an exhaustive experiment of all first-order TPTP problems, it shows that our clause normal form transformation yields fewer clauses and fewer literals than the methods known and used so far. This often allows for exponentially shorter proofs and, in some cases, it makes it even possible for a theorem prover to find a proof where it was unable to do so with more standard clause normal form transformations.

1 Introduction

Theorem provers for first-order predicate logic usually operate on sets of clauses. However, many problem formulations, in particular problems from application domains such as formal program analysis (verification, model checking) are given in full first-order logic and thus require a transformation into clause normal form (CNF). It is well-known that the quality of such a translation has a great impact on the success of an afterwards applied theorem prover. A CNF of some formula is better than another one if it enables a theorem prover to find a proof/counter-model in a shorter period of time. Since this is an undecidable criterion, there cannot be an optimal algorithm in this sense. Therefore, we employ the following heuristics in our paper: The smaller a set of clauses is the easier a proof or a counter-model can be found. Even the design of an algorithm producing "smallest" CNFs is non-trivial. Nevertheless, we shall show that heading towards small CNFs pays off.

The approach we propose in this paper emphasizes on two major aspects in the CNF transformation of first-order predicate logic formulae. These are *Renaming* and *Skolemization*.

By *Renaming* we mean the replacement of subformulae with some new predicates and adding a suitable definition for these new predicate. This is done whenever there is some evidence that such a step will ultimately lead to fewer and smaller clauses [1]. It shows that the clause set obtained by this method in accordance with the other techniques described below is usually smaller than the corresponding result of any other method used so far, c.f. [6,14].

Moreover, we propose the two *Skolemization* techniques described in [12] and [11], which turned out to be superior to the standard Skolemization approaches, i.e., they usually result in smaller and/or more general clauses. In particular, we show how these methods can be turned into algorithms showing a reasonable behaviour for all examples we tested.

A further aspect of our CNF generator lies in the elimination of redundant clauses. All generated clauses are checked for subsumption and condensation. In addition, tautologies are removed.

We thus finally obtain clause sets which, in many cases (e.g., the halting problem [6], Pelletier's Problem 38 [13]), are smaller than any other CNF known so far.

Our CNF translation procedure FLOTTER consists of the following steps:

1. Obvious simplifications.
2. Renaming: see Section 2.
3. Building negation normal form.
4. Anti-Prenexing: see Section 3.
5. Skolemization: see Section 3.
6. Transformation into conjunctive normal form.
7. Redundancy tests: see Section 4

The first step eliminates redundant operators, quantifiers, the logical constants for truth and falsity and simple, syntactic tautologies. The steps 3 and 6 include the standard procedures for building these normal forms [2,9], except that we use polarity dependent linearization, see Section 2. We call a CNF translation *standard* if step 2 is missing and step 5 is replaced by standard Skolemization [2,9].

The paper mainly contributes to two aspects of CNF translation: (i) it shows how techniques already established theoretically can be turned into well-behaved algorithms in practice (ii) by an exhaustive experiment we show that these methods outperform standard CNF translations and simple definitional translations.

The paper is now organized as follows: Section 2 turns the renaming technique due to Thierry Boy de la Tour [1] into an effective algorithm. In Section 3 we discuss our Skolemization techniques and present methods to make them tractable in practice. Concerning redundancy tests, we explain in Section 4 improvements for subsumption/condensation tests. The procedure FLOTTER incorporating all these techniques is put to the test in Section 5. We conclude the paper with a small summary and an outlook in Section 6.

Preliminaries

We assume the reader to be familiar with the syntax, semantics and common notions of first-order logic and resolution-based theorem proving.

Clauses are often denoted by their respective multisets of literals. A clause C is said to *subsume* a clause D, if there exists a substitution σ with $C\sigma \subseteq D$. A clause C is a *condensation* of a clause D if there exists a substitution σ such that C is obtained from $D\sigma$ by the deletion of duplicate literal occurrences,

$C \neq D$ and C subsumes D, or, equivalently, if C is a proper factor of D that subsumes D.

A *position* is a word over the natural numbers. The set $pos(\phi)$ of positions of a given formula ϕ is defined as follows: (i) the empty word $\epsilon \in pos(\phi)$ (ii) for $1 \leq i \leq n$, $i.p \in pos(\phi)$ if $\phi = \phi_1 \circ \ldots \circ \phi_n$ and $p \in pos(\phi_i)$ where $\circ$ is a first-order operator. Now, if $p \in pos(\phi)$ we define $\phi|_\epsilon = \phi$ and $\phi|_{i.p} = \phi_i|_p$ where $\phi = \phi_1 \circ \ldots \circ \phi_n$. We write $\psi[\phi]_p$ for $\psi|_p = \phi$. With $\psi[p/\phi]$ where $p \in pos(\psi)$ we denote the formula obtained by replacing $\psi|_p$ with ϕ at position p in ψ.

The polarity of a formula occurring at position π in a formula ψ is denoted by $Pol(\psi, \pi)$ and defined in the usual way: $Pol(\psi, \epsilon) = 1$; $Pol(\psi, \pi.i) = Pol(\psi, \pi)$ if $\psi\,|_\pi$ is a conjunction, disjunction, formula starting with a quantifier or an implication with $i = 2$; $Pol(\psi, \pi.i) = -Pol(\psi, \pi)$ if $\psi\,|_\pi$ is a formula starting with a negation symbol or an implication with $i = 1$ and, finally, $Pol(\psi, \pi.i) = 0$ if $\psi|_\pi$ is an equivalence.

2 Renaming

First, let us illustrate the problem with the help of a simple example. Consider the formula

$$\phi_1 \vee \forall x\, \phi_2$$

If n is the number of clauses generated by ϕ_1 and m is the number of clauses generated by $\forall x\, \phi_2$ then the overall above formula generates nm clauses. Thus, a nesting of such disjunctions easily leads to an exponential number of clauses. The same holds for nested equivalences. The reason for this exponential explosion is the (exponential) duplication of subformulae obtained by the exhaustive application of the distributive law. The solution to this problem is *Renaming*, the replacement of subformulae using new predicates. For the above formula, such a renaming could be

$$[\phi_1 \vee \forall x\, P(x)] \wedge \forall y\, (P(y) \supset \phi_2)$$

where P is a new one-place predicate. The renamed formula is not logically equivalent to the first formula, but preserves both satisfiability and unsatisfiability and this suffices for resolution based theorem proving. Furthermore, the renamed formula generates only $n + m$ clauses. Thus, using a renaming technique, the worst case exponential explosion of the number of clauses generated by a formula can be avoided.

The renaming idea goes back to Tseitin [19] who showed that the introduction of new propositional symbols can result in exponentially shorter proofs. In the context of CNF translation important contributions have been made by Plaisted and Greenbaum [15] and Boy de la Tour [1]. These approaches differ in their respective criteria to decide which subformulae are to be replaced by new predicates. Whereas Plaisted and Greenbaum basically suggested to rename all subformulae up to literals, Boy de la Tour only replaces a subformula if this would decrease the number of eventually generated clauses. Since our goal is to produce short CNFs, we follow the approach of Boy de la Tour. In the following we present the key ideas of Boy de la Tour's technique and show how these ideas can be turned into effective algorithms. Furthermore, we use the notion of formula positions for our definitions which avoids some ambiguities contained in Boy de la Tour's work [1,16].

Let us start with a precise definition of a renaming. Let ψ be a formula and $\phi = \psi|_\pi$ be a subformula of ψ that we want to rename. Let $x_1, \ldots, x_n$ be the free variables in ϕ and let R be an n-place predicate new to ψ. Then the formula

$$\psi[\pi/R(x_1, \ldots, x_n)] \wedge \mathit{Def}^\psi_\pi$$

is a *renaming* of ψ at position π. The formula Def^ψ_π is a polarity dependent definition of the new predicate R:

$$\mathit{Def}^\psi_\pi = \begin{cases} \forall x_1, \ldots, x_n\, [R(x_1, \ldots, x_n) \supset \phi] & \text{if } \mathit{Pol}(\psi, \pi) = 1 \\ \forall x_1, \ldots, x_n\, [\phi \supset R(x_1, \ldots, x_n)] & \text{if } \mathit{Pol}(\psi, \pi) = -1 \\ \forall x_1, \ldots, x_n\, [R(x_1, \ldots, x_n) \equiv \phi] & \text{if } \mathit{Pol}(\psi, \pi) = 0 \end{cases}$$

The first crucial point in the approach is to calculate the number of clauses generated by a standard CNF. This can be done by the function p defined in Table 1, where $\bar{p}(\psi)$ stands for $p(\neg\psi)$ for any formula ψ. The first column shows the form of ψ and the next two columns the corresponding recursive, top-down calculations for $p(\psi)$ and $\bar{p}(\psi)$, respectively.

ψ	$p(\psi)$	$\bar{p}(\psi)$
$\phi_1 \wedge \ldots \wedge \phi_n$	$\sum_{i=1}^n p(\phi_i)$	$\prod_{i=1}^n \bar{p}(\phi_i)$
$\phi_1 \vee \ldots \vee \phi_2$	$\prod_{i=1}^n p(\phi_i)$	$\sum_{i=1}^n \bar{p}(\phi_i)$
$\phi_1 \supset \phi_2$	$\bar{p}(\phi_1)p(\phi_2)$	$p(\phi_1) + \bar{p}(\phi_2)$
$\phi_1 \equiv \phi_2$	$p(\phi_1)\bar{p}(\phi_2) + \bar{p}(\phi_1)p(\phi_2)$	$p(\phi_1)p(\phi_2) + \bar{p}(\phi_1)\bar{p}(\phi_2)$
$\forall x_1, \ldots x_n\, \phi_1$	$p(\phi_1)$	$\bar{p}(\phi_1)$
$\exists x_1, \ldots x_n\, \phi_1$	$p(\phi_1)$	$\bar{p}(\phi_1)$
$\neg\phi_1$	$\bar{p}(\phi_1)$	$p(\phi_1)$
atomic	1	1

Table 1. Calculating the number of clauses

The calculation for equivalences assumes a polarity dependent linearization: A formula $\psi \equiv \phi$ is transformed into $(\psi \wedge \phi) \vee (\neg\psi \wedge \neg\phi)$ if its polarity is -1. It is transformed into $(\psi \supset \phi) \wedge (\phi \supset \psi)$ if its polarity is $+1$. This linearization avoids generating redundant clauses that are hardly recognizable once the CNF is built.

Second, in order to check whether a renaming yields fewer clauses, we need to calculate the difference between the number of clauses generated with and without a renaming. So let us assume that we want to rename a subformula at position π within a formula ψ. The condition to be checked is

$$p(\psi) \geq p(\psi[\pi/R(x_1, \ldots, x_n)]) + p(\mathit{Def}^\psi_\pi).$$

The obvious problem with this condition is that the function p cannot be effectively computed in general, since it grows exponentially in the size of the input formula. The rest of this section is concerned with solving this problem. Obviously, the formulae ψ and $\psi[\pi/R(x_1,\ldots,x_n)]$ only differ at position π, the other parts of the formulae remain identical. We can make use of this fact by an abstraction of those parts of ψ that do not influence the changed position. This is done by the notion of a coefficient as shown in Table 2, where $\mathcal{Q}$ is an arbitrary

π	$\psi\vert_\tau$	a^ψ_π	b^ψ_π
$\tau.i$	$\phi_1 \wedge \ldots \wedge \phi_n$	a^ψ_τ	$b^\psi_\tau \prod_{j\neq i} \bar{p}(\psi\vert_{\tau.j})$
$\tau.i$	$\phi_1 \vee \ldots \vee \phi_n$	$a^\psi_\tau \prod_{j\neq i} p(\psi\vert_{\tau.j})$	b^ψ_τ
$\tau.1$	$\phi_1 \supset \phi_2$	b^ψ_τ	$a^\psi_\tau p(\psi\vert_{\tau.2})$
$\tau.2$	$\phi_1 \supset \phi_2$	$a^\psi_\tau \bar{p}(\psi\vert_{\tau.1})$	b^ψ_τ
$\tau.i$	$\phi_1 \equiv \phi_2$	$a^\psi_\tau \bar{p}(\psi\vert_{\tau.(3-i)}) + b^\psi_\tau p(\psi\vert_{\tau.(3-i)})$	$a^\psi_\tau p(\psi\vert_{\tau.(3-i)}) + b^\psi_\tau \bar{p}(\psi\vert_{\tau.(3-i)})$
$\tau.1$	$\neg\phi_1$	b^ψ_τ	a^ψ_τ
$\tau.1$	$\mathcal{Q}x_1,\ldots,x_n\phi_1$	a^ψ_τ	b^ψ_τ
ϵ	ψ	1	0

Table 2. Calculating the coefficients

quantor ($\mathcal{Q} \in \{\forall, \exists\}$). The coefficients determine how often a particular subformula and its negation is duplicated in the course of a standard CNF translation. The coefficient a^ψ_π is the factor for the eventual multiplication of $p(\psi|_\pi)$ in the CNF and the factor b^ψ_π for the multiplication of $\bar{p}(\psi|_\pi)$. The first column of Table 2 shows the form of π, the second column the form of ψ directly above position π (ψ itself if $\pi = \epsilon$) and the next two columns the corresponding recursive, bottom-up calculations that have to be done to compute a^ψ_π and b^ψ_π, respectively. For our starting example formula $\psi = \phi_1 \vee \forall x\, \phi_2$ where we renamed position 2.1, i.e., the subformula ϕ_2, the coefficients are $a^\psi_{2.1} = p(\phi_1)$ (Table 2, seventh, second and last row, first column) and $b^\psi_{2.1} = 0$ (seventh, second and last row, second column). Note that a^ψ_π (b^ψ_π) is always 0 if $Pol(\psi,\pi) = -1$ ($Pol(\psi,\pi) = 1$).

Using the notion of a coefficient, the previously stated condition can be reformulated as

$$a^\psi_\pi p(\phi) + b^\psi_\pi \bar{p}(\phi) \geq a^\psi_\pi + b^\psi_\pi + p(Def^\psi_\pi)$$

where we still assume that $\phi = \psi|_\pi$. Note that since ϕ is replaced by an atom, the coefficients a^ψ_π, b^ψ_π are multiplied by 1 in the renamed version. Depending on the polarity of $\psi|_\pi$ the disequation is equivalent to one of the three disequations:

$$\begin{array}{ll} a^\psi_\pi p(\phi) \geq a^\psi_\pi + p(\phi) & \text{if } Pol(\psi,\pi) = 1 \\ b^\psi_\pi \bar{p}(\phi) \geq b^\psi_\pi + \bar{p}(\phi) & \text{if } Pol(\psi,\pi) = -1 \\ a^\psi_\pi p(\phi) + b^\psi_\pi \bar{p}(\phi) \geq a^\psi_\pi + b^\psi_\pi + p(\phi) + \bar{p}(\phi) & \text{if } Pol(\psi,\pi) = 0 \end{array}$$

Let us examine the most complicated third case in more detail. The other cases can be treated accordingly. The third case can be equivalently transformed into the disequation

$$(a_\pi^\psi - 1)(p(\phi) - 1) + (b_\pi^\psi - 1)(\bar{p}(\phi) - 1) \geq 2$$

Let us abbreviate the product $(a_\pi^\psi - 1)(p(\phi) - 1)$ with p_a and $(b_\pi^\psi - 1)(\bar{p}(\phi) - 1)$ with p_b. Since neither p_a nor p_b can become negative, the disequation holds if (i) $p_a \geq 2$ or (ii) $p_b \geq 2$ or (iii) $p_a \geq 1$ and $p_b \geq 1$. In order to check these conditions, it is sufficient to check whether the coefficients a_π^ψ, b_π^ψ and number of clauses $p(\phi)$, $\bar{p}(\phi)$ are strictly greater than 1, 2, respectively. This can always be checked in linear (!) time with respect to the size of ψ. For example, $p(\phi) > 1$ holds iff ϕ contains an equivalence or a conjunct with positive polarity or a disjunct with negative polarity or an implication with negative polarity. To sum up, we turned the renaming condition due to Boy de la Tour [1] that requires the computation of exponentially growing functions into a condition that can be checked in linear time and does not require any arithmetic calculation at all.

In fact, all the above is relevant in practice. An older version of FLOTTER [20] computed the values for the number of clauses and coefficients. It turned out that the computation of coefficients and number of clauses significantly shows up in the profiling of FLOTTER. There were even some examples for which the exponential explosion did prevent FLOTTER to work properly. After the above explained reformulation all these problems disappeared.

3 Skolemization

A further important step in the clause normal form transformation which is commonly neglected is the so-called *Skolemization* [17] of formulae. It is used to get rid of existentially quantified variables and that by replacing each such occurrence with a Skolem function application. There are usually two kinds of Skolemization techniques described in the literature (see, e.g., [2,9]) which we call *Inner* and *Outer Skolemization* respectively. The two differ mainly in the choice of variables the Skolem functions get as arguments. Outer Skolemization gets those universally quantified variables as arguments in whose scopes the (existentially quantified) subformula under consideration occurs, whereas Inner Skolemization receives all free variables of this very subformula as arguments. For both kinds of Skolemization it has been shown to be valuable to initially transform the given problem into anti-prenex normal form [5], i.e., quantifiers are moved inwards as far as possible.[1]

Not much effort had been spent until recently to improve these standard Skolemization techniques. Nevertheless, even Skolemization leaves possibilities for improvements. In this paper we want to introduce two such proposals, the Optimized Skolemization [12] and the Strong Skolemization [11].

Definition 1 (Optimized Skolemization). Let Δ be a sentence in negation normal form, i.e., it contains no implications or equivalences and negations do only occur directly in front of atoms. Moreover, let $\exists x_1, \ldots, x_k\,(\phi \wedge \psi)$ be a

[1] There is one exception however: Existential quantifiers are not distributed over disjunctions in order to avoid generating unnecessarily many Skolem functions.

subformula of Δ at position π and assume that $\Delta \models \forall y_1, \ldots, y_n \, \exists x_1, \ldots, x_k \, \phi$ where $\{y_1, \ldots, y_n\}$ denotes the set of free variables of $\exists x_1, \ldots, x_k \, (\phi \wedge \psi)$. Finally, let $f_1, \ldots, f_k$ be new (Skolem) function symbols. We then say that

$$\forall y_1, \ldots, y_n \, \phi\{x_i \mapsto f_i(y_1, \ldots, y_n)\} \wedge \Delta[\pi/\psi\{x_i \mapsto f_i(y_1, \ldots, y_n)\}]$$

can be obtained by a *single optimized Skolemization step* from Δ.

In [12] it is shown that Optimized Skolemization behaves as desired, i.e., Δ is satisfiable if and only if $\forall y_1, \ldots, y_n \, \phi\{x_i \mapsto f_i(y_1, \ldots, y_n)\} \wedge \Delta[\pi/\psi\{x_i \mapsto f_i(y_1, \ldots, y_n)\}]$ is satisfiable.

Note that Optimized Skolemization requires a theorem prover on its own in order to prove the preliminary condition $\Delta \models \forall y_1, \ldots, y_n \, \exists x_1, \ldots, x_k \, \phi$.

As an example consider the (sub-)problem

$$\forall x, y, z \, (R(x, y) \wedge R(x, z) \supset \exists u \, (R(y, u) \wedge R(z, u)))$$

and assume that $\forall x \, \exists y \, R(x, y)$ is provable from the whole problem under consideration. With standard Skolemization we would obtain the clauses

$$\begin{array}{l} \neg R(x, y) \vee \neg R(x, z) \vee R(y, f(y, z)) \\ \neg R(x, y) \vee \neg R(x, z) \vee R(z, f(y, z)) \end{array}$$

With Optimized Skolemization, however, we would end up with the clause set

$$\begin{array}{l} R(y, f(y, z)) \\ \neg R(x, y) \vee \neg R(x, z) \vee R(z, f(y, z)) \end{array}$$

which is definitely superior to the standard result. In short, the effect of Optimized Skolemization compared to standard (Inner) Skolemization is that some of the literals in the standard result are deleted.

Strong Skolemization differs from Optimized Skolemization is many respects. First of all, like standard (Inner) Skolemization, it is a local method, i.e., it applies only to the subformula under consideration and does not take the whole problem into account. Moreover, it does not require a theorem prover on its own to perform its task. Now, in order to define what Strong Skolemization is about let us first introduce the notion of a *free variable splitting*. Consider the (sub-)formula $\Delta = \exists x_1, \ldots x_k \; (\phi_1 \wedge \ldots \wedge \phi_n)$. Then Δ's free variable splitting $\langle \mathbf{z}_1, \ldots, \mathbf{z}_n \rangle$ consists of n sequences of free variables of Δ such that the sequence $\mathbf{z}_1$ consists of exactly the free variables occurring in ϕ_1 (without the $x_1, \ldots x_k$) and for each i with $1 < i \leq n$, $\mathbf{z}_i$ contains all free variables of ϕ_i (again without the $x_1, \ldots x_k$) which did not yet occur in any of the subformulae $\phi_1, \ldots, \phi_{i-1}$. Evidently, the union of all these sequences contains all the free variables of the given subformula.

Notice that standard Inner Skolemization could then be defined as follows: Replace the subformula $\exists x_1, \ldots x_k \; (\phi_1 \wedge \ldots \wedge \phi_n)$ with $\phi_1^\star \wedge \ldots \wedge \phi_n^\star$ where $\phi_i^\star = \phi_i\{x_j \mapsto f_j(\mathbf{z}_1, \ldots, \mathbf{z}_n) \mid 1 \leq j \leq k\}$ and $\langle \mathbf{z}_1, \ldots, \mathbf{z}_n \rangle$ is the corresponding free variable splitting.[2]

[2] For simplicity, we assume uniqueness of existentially quantified subformulae.

The effect of Strong Skolemization now lies in the replacement of some of these variable sequences with sequences of fresh universally quantified variables. The following definition specifies this.

Definition 2 (Strong Skolemization). Let Δ be a first-order sentence in negation normal form. Strong Skolemization replaces existentially quantified subformulae of the form $\exists x_1, \ldots x_k\, (\phi_1 \wedge \ldots \wedge \phi_n)$ with

$$\begin{array}{r}
\forall \mathbf{w}_2, \mathbf{w}_3, \mathbf{w}_4, \ldots, \mathbf{w}_n\ \phi_1\{x_i \mapsto f_i(\mathbf{z}_1, \mathbf{w}_2, \mathbf{w}_3, \mathbf{w}_4, \ldots, \mathbf{w}_n)\} \wedge \\
\forall \mathbf{w}_3, \mathbf{w}_4, \ldots, \mathbf{w}_n\ \phi_2\{x_i \mapsto f_i(\mathbf{z}_1, \mathbf{z}_2, \mathbf{w}_3, \mathbf{w}_4, \ldots, \mathbf{w}_n)\} \wedge \\
\vdots \qquad\qquad\qquad\qquad \\
\forall \mathbf{w}_n\ \phi_{n-1}\{x_i \mapsto f_i(\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_{n-1}, \mathbf{w}_n)\} \wedge \\
\phi_n\{x_i \mapsto f_i(\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_{n-1}, \mathbf{z}_n)\}\ \,
\end{array}$$

where $\langle \mathbf{z}_1, \ldots, \mathbf{z}_n \rangle$ is the free variable splitting of $\exists x_1, \ldots x_k\, (\phi_1 \wedge \ldots \wedge \phi_n)$, each variable sequence $\mathbf{w}_i$ has length equal to the length of the variable sequence $\mathbf{z}_i$, and the $f_i, 1 \leq i \leq k$, are new (Skolem) function symbols. In [11] it is shown that Strong Skolemization preserves both satisfiability and unsatisfiability.

Evidently, the Strong Skolemization result subsumes what we could possibly get from standard Inner Skolemization (simply instantiate the $\mathbf{w}_i$ with the corresponding $\mathbf{z}_i$). Briefly, the effect of Strong Skolemization compared to standard Inner Skolemization is that some of the arguments of the new Skolem functions are replaced with new, universally quantified variables.

Let us again have a look at the example

$$\forall x, y, z\, (R(x, y) \wedge R(x, z) \supset \exists u\, (R(y, u) \wedge R(z, u))).$$

Strong Skolemization would lead us to the clause set

$$\begin{array}{l}
\neg R(x, y) \vee \neg R(x, z) \vee R(y, f(w, y)) \\
\neg R(x, y) \vee \neg R(x, z) \vee R(z, f(x, y))
\end{array}$$

which is almost identical to the standard Skolemization outcome, with one exception, however, the new variable w in the first clause. In fact, this tiny change allows us to perform a condensation step on the first two literals, so that we finally end up with

$$\begin{array}{l}
\neg R(x, y) \vee R(y, f(w, y)) \\
\neg R(x, y) \vee \neg R(x, z) \vee R(z, f(x, y))
\end{array}$$

This result is not quite as strong as the one we obtained from Optimized Skolemization. But notice that it did not require to prove the intermediate goal $\forall x\, \exists y\, R(x, y)$ which actually might be hard to be proved, provided it is at all provable. In fact, proving such preliminary lemmata can be arbitrarily complicated and so, in case of Optimized Skolemization, resource restrictions are necessary to ensure that such intermediate steps ultimately terminate. Also, given for instance a subformula of the form $\exists x\ (\phi_1 \wedge \ldots \wedge \phi_n)$, it would be necessary for Optimized Skolemization to consider up to $n!$ (n faculty) intermediate proofs for best performance. This is intractable in practice. Therefore, in FLOTTER only up to n proof attempts are made in such cases, one for each conjunct ϕ_i. Furthermore, all proof attempts performed in the context of optimized Skolemization are restricted to resolution inferences where the resolvent is strictly shorter than its longest parent clause and the term depth of the resolvent is at most equal to the maximal term depth of a parent clause. Together with subsumption and condensing this restriction guarantees termination of the inference process. Moreover, it is guaranteed to detect each situation where the formula to be proven is already contained in the input formula. Strong Skolemization, on the other hand, is almost as cheap as standard (Inner) Skolemization and requires no intermediate theorems to be proved.

Now recall the example from above. It shows the superiority of both Strong and Optimized Skolemization over standard Skolemization. Also, it suggests to prefer Optimized Skolemization provided the intermediate goal $\forall y, z\, \exists u\ R(y, u)$ is (more or less easily) provable. For this reason, the current FLOTTER implementation first attempts to perform an Optimized Skolemization step and only if this one was not successful a Strong Skolemization step is performed. Such a heuristics has shown to be sensible in practice, although there are examples where an opposite direction would be more effective.

For instance consider the formula $\forall x, y\ \ R(x, y) \supset \exists z\ \ (P(z) \wedge Q(x, z) \wedge S(x, y, z))$ and suppose that the relation R is provably non-empty. Standard Skolemization would then lead us to the clause set

$$\begin{array}{l}\neg R(x,y) \vee P(f(x,y))\\ \neg R(x,y) \vee Q(x,f(x,y))\\ \neg R(x,y) \vee S(x,y,f(x,y))\end{array}$$

whereas Strong Skolemization would come up with

$$\begin{array}{l}\neg R(x,y) \vee P(f(v,w))\\ \neg R(x,y) \vee Q(x,f(x,w))\\ \neg R(x,y) \vee S(x,y,f(x,y))\end{array}$$

Because of the assumption that R is provably non-empty, sooner or later it will become possible to simplify the first clause of the Strong Skolemization outcome to the unit clause $P(f(v, w))$.

In case of Optimized Skolemization the attempt to prove that $\exists z\, P(z)$ would obviously be successful and so we would finally end up with the clause set

$$\begin{array}{l}P(f(x,y))\\ \neg R(x,y) \vee Q(x,f(x,y))\\ \neg R(x,y) \vee S(x,y,f(x,y))\end{array}$$

which is a little less general than what we obtained from Strong Skolemization.

Thus there are examples which suggest to first try an Optimized Skolemization step and then a Strong Skolemization step (if the former was not successful in proving any of the intermediate goals), but also there are examples which propose the other way round. It certainly would be desirable to combine the two techniques more tightly. This however, is a matter of future work.

4 Subsumption and Condensation

Our CNF translation removes tautologies and also checks for subsumption and condensation. Since it is well known that testing subsumption between two clauses is an *NP*-complete problem, we have to be very careful what the construction of our subsumption algorithm is concerned to obtain an acceptable behaviour. Gottlob and Leitsch [8] refined standard subsumption algorithms with polynomial complexity on certain classes of clauses. We implemented their algorithms, however, as it turned out that for all problems we considered so far their techniques are superfluous. It showed that the addition of two filters and efficient data structures makes the standard Stillman algorithm (see [8]) a tractable subsumption test. The first filter considers the number of symbols in the clauses. A necessary condition for a clause C to subsume a clause D is that the number of symbols in C is smaller or equal to the number of symbols in D. Recall that we consider clauses to be multisets. In practice, this test allows us to reject about 95% of all subsumption queries. The second filter independently checks for every literal in clause C whether it can be matched to a literal in D. This reduces the remaining queries by another 70%.

Our condensation test is based on matching and subsumption according to the algorithm *fastcondense* presented by Gottlob and Leitsch [8].

5 Experiments

We apply the techniques introduced in the previous sections to all TPTP-v2.1.0 [18] first-order problems and to two problems that have been discussed in the literature, namely the halting problem [3,6] and problem 38 from the Pelletier problem collection [13,14]. Our CNF translator FLOTTER, its restriction to standard CNF generation, the CNF translator of ILF (see below) and the theorem provers SPASS-v.0.80[3] and OTTER-v3.0.4 [10] are run on Sun Sparc Ultra 170 workstations with 128MB of main memory. All timings are given in seconds and refer to this machine setup.

The halting problem [3] was first automatically solved by Egly and Rath [6]. They also used a renaming technique for their disjunctive normal form translation and obtained a normal form with 75 conjuncts. Pelletier's problem 38 was discussed in detail by Pelletier and Sutcliffe [14] where they present a CNF with 31 clauses. Applying FLOTTER to these problems yielded the results below, where the Halting Problem is problem COM003+1 and Pelletier's problem 38 is problem SYN67+1 of the TPTP. For the Halting Problem FLOTTER generates 28 clauses and for Pelletier's problem 38 it generates 27 clauses, the smallest

[3] SPASS and FLOTTER can be obtained from `ftp.mpi-sb.mpg.de`.

CNFs we know so far for these problems. This is due to the fact that FLOTTER made three renamings and applied six times optimized Skolemization to the Halting Problem and applied no renamings but five times optimized Skolemization to Pelletier's problem 38.

Furthermore, the below table shows the results of applying FLOTTER, standard CNF translation, and the normal form translation contained in ILF [4] to all first-order problems of the TPTP. For the CNF translation of ILF we used the settings definitions-on, recognize_equal_subformulas-on, polarity-on, optimized_skolemization-on, tautology_detection-on. These settings cause a CNF translation close to the one proposed by Plaisted and Greenbaum [15] that is also somewhere between the normal forms generated by the settings p-def. and p-def. red suggested by Egly and Rath [7]. Therefore we call this normal form *definitional* in the sequel. Unfortunately, it was not possible to obtain the original normal form procedure of Egly and Rath. On the other hand they considered disjunctive normal forms, so our results are not easily comparable with their results anyway.

We checked all 347 first-order examples from the TPTP. Below only those 95 problems are listed where the number of symbols/clauses generated by FLOTTER and the standard CNF differs. The table shows the problem name the number of clauses/symbols generated by the CNF translators FLOTTER, the standard translation and the definitional translation, the time spent by the CNF translators and the time spent by the provers SPASS and OTTER to solve the respective normal forms. Here, SFL, SST, SDE abbreviates SPASS applied to the FLOTTER, standard and definitional CNF, respectively and OFL, OST, ODE abbreviates the same relations for OTTER. We set a time limit of 120 seconds for the CNF translation and a time limit of 300 seconds for the provers. A "-" indicates that a CNF could not be generated in the time limit and therefore, in the respective row the proof attempts to this CNF are not available, marked "n.a.".

It holds on all 347 examples that the number of clauses/symbols generated by the definitional translation is greater than the number of clauses/symbols generated by FLOTTER. Furthermore, except for the examples SYN007+1.014, SYN522+1 and SYN532+1 even the standard CNF transformation needs less time and produces fewer clauses than the definitional transformation. This is mostly a result of our careful implementation in C compared to a straightforward implementation of the definitional transformation in PROLOG.

Another interesting point to note is that FLOTTER not in all cases produces a shorter clause normal form than the standard CNF. This occurs for the examples SYN069+1, SYN327+1, SYN351+1 and SYN393+1. This is due to the effect that without renaming these problems produce a highly redundant set of clauses such that the effects of renaming are compensated by the redundancy tests. The introduction of new predicate symbols prevents the redundancy tests from being successful for the renamed version. However, the redundancy is later on detected by both provers such that there are no significant difference in their time spent to solve these four problems.

For 65 problems out of the presented 95 problems, FLOTTER produced fewer clauses than the standard transformation. For our comparison, we do not consider problems where the timing differences of the proof attempts are less than 0.5 seconds or less than 20%. With respect to this restriction, comparing the FLOTTER output with the standard CNF output, SPASS performed worse on 4 problems and performed better on 26 problems. In particular, there are examples SPASS could only solve using the CNF generated by FLOTTER, but no examples there it was possible for SPASS to solve the standard translation but not the FLOTTER translation. The 4 problems SYN514+1, SYN518+1, SYN520+1 and SYN546+1 where SPASS performed worse are all satisfiable problems. The timing of SPASS on these problems depends crucially on the ordering in which SPASS applies its Splitting rule [20].

Similar observations can be made by comparing the results produced by OTTER or by comparing the behaviour of the provers on the definitional CNF with the FLOTTER generated clause sets. With respect to some few exceptions, the heuristic that smaller clause sets enable easier proofs or counter-models holds over the TPTP problem domain. This corresponds to our experience on other problem domains like formal software-reuse problems or problems generated in the course of program verification.

	FLOTTER			STANDARD			DEFINITIONAL			SFL	SST	SDE	OFL	OST	ODE
Problem	#cl	#sy	time	#cl	#sy	time	#cl	#sy	time	time	time	time	time	time	time
COM003+1	28	311	0.05	50	1030	0.22	114	788	3.32	0.16	5.31	1.80	1.30	>300	77.14
COM003+2	43	409	0.07	43	445	0.06	192	1609	6.65	0.06	0.07	0.38	0.12	0.11	0.44
COM003+3	19	230	0.03	27	531	0.11	71	536	2.43	0.12	1.08	0.30	0.66	>300	5.57
GRP194+1	12	139	0.13	12	143	0.12	52	430	1.28	0.14	0.13	4.60	5.33	0.88	>300
MGT019+2	12	114	0.04	12	118	0.02	41	292	1.16	0.04	0.05	0.12	>300	>300	>300
MGT023+1	9	119	0.20	9	137	0.03	31	237	0.90	0.05	0.13	0.09	0.54	2.51	3.01
MGT023+2	14	186	0.07	17	352	0.07	49	388	1.49	0.08	1.35	0.14	131.01	111.34	256.80
MGT027+1	16	196	0.10	16	209	0.07	69	520	1.97	3.30	3.74	0.49	0.06	0.10	0.42
MGT028+1	13	203	0.16	14	259	0.04	45	343	1.34	0.09	0.11	0.12	0.06	0.08	0.27
MGT029+1	22	369	0.24	22	379	0.12	70	525	1.94	8.56	5.70	>300	>300	>300	>300
MGT030+1	13	163	0.04	16	322	0.06	45	359	1.38	0.08	0.56	0.13	0.09	0.18	0.33
MGT032+2	8	73	0.02	8	87	0.02	29	223	1.02	0.01	0.02	0.07	0.03	0.02	0.10
MGT035+1	25	424	0.28	25	434	0.14	84	691	2.62	7.97	9.17	>300	1.08	1.54	86.32
MGT035+2	45	667	0.22	45	677	0.16	167	1335	9.72	184.43	144.49	>300	0.48	0.73	>300
MGT038+1	12	103	0.05	12	116	0.03	45	291	1.12	0.15	0.19	0.27	0.08	0.07	0.24
MGT038+2	30	459	0.76	30	465	0.16	121	894	4.93	>300	>300	>300	0.28	0.31	6.45
MGT039+1	19	190	0.06	19	203	0.04	75	504	1.78	0.60	0.76	>300	2.55	2.70	>300
MGT039+2	28	350	0.14	28	363	0.12	116	857	5.14	>300	>300	>300	>300	>300	>300
MSC009+1	26	197	0.04	26	200	0.02	144	967	1.91	0.07	0.06	0.26	0.05	0.05	0.19
PUZ031+1	25	177	0.14	26	186	0.13	104	567	2.60	0.16	0.19	0.65	0.11	0.13	0.36
SET046+1	3	23	0.00	3	26	0.00	17	131	0.63	0.00	0.01	0.03	0.01	0.03	0.06
SYN007+1.014	104	468	0.12	-	-	>120	393	2064	1.05	54.57	n.a.	76.45	48.00	>300	39.83
SYN036+1	22	128	0.03	72	1032	0.42	125	752	1.04	0.06	1.32	0.18	0.19	>300	146.43
SYN036+2	24	152	0.04	86	1282	0.47	125	752	1.05	0.09	1.31	0.27	0.54	>300	>300
SYN055+1	8	38	0.01	8	43	0.01	26	116	0.69	0.00	0.01	0.03	0.03	0.02	0.07
SYN056+1	8	52	0.01	9	62	0.00	46	240	0.70	0.02	0.03	0.07	0.03	0.05	0.14
SYN058+1	8	30	0.00	9	40	0.00	26	111	0.66	0.02	0.01	0.03	0.01	0.02	0.06
SYN059+1	9	40	0.02	16	120	0.02	57	319	0.76	0.02	0.02	0.07	0.03	0.09	0.27
SYN066+1	6	42	0.00	6	50	0.00	26	161	0.79	0.02	0.01	0.02	0.02	0.03	0.07
SYN067+1	27	457	03.55	55	1076	0.69	115	851	1.68	20.33	>300	0.56	>300	>300	0.47
SYN068+1	7	34	0.01	7	40	0.01	22	117	0.66	0.01	0.02	0.06	0.02	0.03	0.06
SYN069+1	12	67	0.00	11	81	0.01	36	208	0.84	0.01	0.03	0.07	0.03	0.05	0.13
SYN070+1	9	60	0.01	9	65	0.02	31	176	0.79	0.01	0.00	0.07	0.03	0.04	0.12
SYN074+1	7	80	0.01	7	85	0.01	42	427	1.00	0.13	0.08	0.12	0.37	0.76	>300
SYN075+1	7	80	0.01	7	85	0.01	42	427	1.01	0.14	0.09	0.11	0.42	0.54	>300
SYN076+1	17	182	0.05	34	906	0.11	102	966	1.86	>300	>300	>300	>300	>300	>300
SYN077+1	7	68	0.03	7	79	0.01	46	370	0.87	0.12	0.17	0.98	22.94	251.63	70.34
SYN078+1	5	31	0.01	7	52	0.01	33	201	0.62	0.01	0.02	0.07	0.04	0.15	0.15
SYN082+1	4	39	0.01	4	47	0.00	24	202	0.62	0.01	0.02	0.04	0.03	0.04	0.10
SYN084+1	7	49	0.03	11	130	0.03	51	357	0.70	0.02	0.03	0.07	0.07	0.56	0.30
SYN327+1	6	47	0.01	4	28	0.00	14	112	0.64	0.00	0.00	0.03	0.03	0.02	0.15
SYN348+1	24	684	0.19	64	2944	0.43	70	905	1.60	0.36	0.40	0.29	>300	>300	>300
SYN349+1	10	144	0.05	10	208	0.02	43	532	1.05	0.04	0.04	0.09	168.64	0.46	>300

	FLOTTER			STANDARD			DEFINITIONAL			SFL	SST	SDE	OFL	OST	ODE
Problem	#cl	#sy	time	#cl	#sy	time	#cl	#sy	time	time	time	time	time	time	time
SYN351+1	6	74	0.02	4	52	0.01	21	256	1.05	0.01	0.03	0.50	0.03	0.02	0.43
SYN365+1	6	33	0.02	6	36	0.03	19	102	0.57	0.01	0.01	0.03	0.10	0.12	0.33
SYN374+1	5	34	0.00	6	44	0.01	49	302	0.68	0.01	0.01	0.05	0.03	0.06	4.49
SYN375+1	3	15	0.01	3	17	0.00	45	246	0.59	0.01	0.01	0.04	0.02	0.02	0.08
SYN377+1	3	16	0.01	3	18	0.00	45	246	0.61	0.01	0.01	0.02	0.01	0.02	0.07
SYN393+1	16	84	0.01	8	36	0.01	63	326	0.59	0.03	0.01	0.03	0.05	0.03	0.10
SYN414+1	7	48	0.03	11	93	0.02	55	392	0.84	0.02	0.01	0.07	0.02	0.06	0.35
SYN415+1	7	44	0.02	10	96	0.02	49	294	0.72	0.04	0.10	0.14	0.11	86.69	0.80
SYN418+1	504	5480	3.60	1158	22515	30.63	-	-	>120	3.87	32.63	n.a.	>300	>300	n.a.
SYN419+1	436	4636	2.61	1040	19949	21.94	-	-	>120	3.43	31.03	n.a.	>300	>300	n.a.
SYN420+1	605	6871	5.32	1617	30557	71.45	-	-	>120	22.59	114.18	n.a.	>300	>300	n.a.
SYN421+1	547	5508	3.83	-	-	>120	-	-	>120	5.20	n.a.	n.a.	>300	>300	n.a.
SYN422+1	529	6221	4.26	902	17375	18.21	-	-	>120	7.29	75.47	n.a.	>300	>300	n.a.
SYN423+1	629	6379	5.44	1190	21004	28.06	-	-	>120	20.06	66.92	n.a.	132.68	>300	n.a.
SYN424+1	765	8402	8.67	-	-	>120	-	-	>120	13.12	225.90	n.a.	>300	226.19	n.a.
SYN425+1	531	5645	4.72	1103	20830	33.29	-	-	>120	4.69	29.82	n.a.	>300	>300	n.a.
SYN426+1	724	8247	8.87	1673	37067	82.60	-	-	>120	44.38	164.36	n.a.	>300	>300	n.a.
SYN427+1	765	8385	9.21	-	-	>120	-	-	>120	87.14	n.a.	n.a.	>300	>300	n.a.
SYN428+1	700	7470	6.88	1641	31373	60.20	-	-	>120	9.51	98.31	n.a.	>300	>300	n.a.
SYN429+1	735	8436	7.72	1210	23473	33.43	-	-	>120	24.11	32.07	n.a.	>300	>300	n.a.
SYN513+1	392	4124	2.27	894	17379	19.30	-	-	>120	2.46	11.98	n.a.	>300	>300	n.a.
SYN514+1	279	3606	2.16	438	7898	6.57	-	-	>120	>300	73.74	n.a.	>300	>300	n.a.
SYN515+1	89	920	0.22	216	5007	2.54	239	1307	5.67	0.16	0.92	1.66	0.18	7.75	0.43
SYN516+1	71	561	0.13	97	1078	0.22	157	817	2.49	0.14	0.21	0.27	0.15	5.71	0.42
SYN517+1	68	654	0.13	76	815	0.16	181	912	2.92	0.18	0.13	0.45	0.34	0.35	0.44
SYN518+1	471	5136	3.20	1184	30854	119.90	-	-	>120	>300	58.35	n.a.	>300	>300	n.a.
SYN519+1	526	5638	3.81	1107	17815	16.40	-	-	>120	>300	>300	n.a.	>300	>300	n.a.
SYN520+1	528	5674	4.71	1008	15536	15.26	-	-	>120	214.52	42.92	n.a.	>300	>300	n.a.
SYN521+1	102	797	0.17	224	4532	2.21	239	1243	4.53	0.15	1.05	0.80	0.17	2.50	0.50
SYN522+1	111	1065	0.24	376	8070	4.86	302	1617	8.10	0.26	2.35	2.80	0.79	6.37	1.47
SYN523+1	53	439	0.08	54	587	0.13	146	716	1.97	0.07	0.10	0.35	0.12	0.16	0.30
SYN524+1	101	1166	0.34	132	2309	0.60	297	1578	8.35	0.32	0.59	2.72	0.27	0.47	2.20
SYN525+1	104	867	0.21	154	1652	0.34	263	1356	5.65	0.22	0.27	1.74	0.55	0.66	0.58
SYN526+1	113	1218	0.28	256	5201	2.01	308	1698	9.98	0.31	1.35	5.27	0.28	1.53	0.67
SYN527+1	84	686	0.15	92	1038	0.21	220	1077	3.56	0.13	0.15	0.70	0.17	0.25	0.48
SYN528+1	121	1180	0.32	202	3020	1.07	>300	1566	7.70	0.30	0.48	3.72	0.29	0.64	0.64
SYN529+1	133	1074	0.31	251	3807	1.37	319	1623	7.61	0.24	0.93	1.90	1.06	3.59	1.30
SYN530+1	95	916	0.21	110	1494	0.39	261	1375	6.21	0.28	0.41	3.08	0.40	0.74	0.62
SYN531+1	82	886	0.19	117	1763	0.45	235	1251	5.81	0.22	0.43	2.04	0.29	2.30	1.37
SYN532+1	101	737	0.17	389	6501	9.16	263	1275	4.60	0.15	1.17	1.43	0.37	1.72	0.55
SYN533+1	92	618	0.13	152	1413	0.31	207	1040	3.49	0.14	0.24	0.65	0.17	0.37	0.46
SYN534+1	77	908	0.19	99	1421	0.28	247	1334	6.67	0.19	0.31	5.02	0.22	0.30	0.85
SYN535+1	100	840	0.18	143	2028	0.51	266	1356	5.89	0.16	0.43	1.73	0.18	0.57	0.56
SYN536+1	87	907	0.21	130	1873	0.51	275	1542	9.12	0.19	0.41	5.44	0.36	4.35	1.16
SYN537+1	114	1134	0.29	118	1606	0.40	396	2078	13.69	0.32	0.36	5.71	0.35	0.34	10.79
SYN538+1	138	1769	0.66	208	4562	2.38	456	2536	24.31	0.61	1.31	>300	1.44	1.61	2.67
SYN539+1	152	1756	0.51	203	4053	1.59	499	2749	26.24	0.48	0.92	>300	>300	>300	16.56
SYN540+1	142	1417	0.42	221	2764	0.79	436	2331	18.54	0.40	0.55	13.13	0.41	1.83	2.05
SYN541+1	145	1680	0.54	242	5027	2.82	497	2717	26.22	0.54	1.49	>300	0.36	1.67	2.73
SYN544+1	363	4429	2.64	1139	18275	13.55	-	-	>120	3.82	12.61	n.a.	1.74	233.17	n.a.
SYN545+1	510	5878	4.00	835	16120	16.73	-	-	>120	4.09	11.54	n.a.	>300	>300	n.a.
SYN546+1	521	5983	4.20	1028	21117	28.21	-	-	>120	>300	78.04	n.a.	>300	>300	n.a.
SYN547+1	482	6069	5.80	919	20212	29.25	-	-	>120	7.43	28.82	n.v	>300	>300	n.a.

6 Conclusion

This paper consists of two major parts: It describes the main ingredients of FLOTTER, which is the realization of a sophisticated clause normal form generator and also it reports on the experimental results obtained from the application of FLOTTER to comparatively huge problem sets.

FLOTTER's most important features are (i) fast Renaming, (ii) Optimized and Strong Skolemization, and (iii) efficient redundancy tests.

Renaming is based on the method introduced by Boy de la Tour. It is further refined such that no computation of the worst case exponentially growing number of clauses needs to be performed.

Optimized and Strong Skolemization are variants of standard Skolemization that in general lead to fewer, smaller, and more general clauses.

The redundancy test consists mainly of subsumption and condensation. These are based on Stillman's and on Gottlob and Leitsch's algorithms, respectively, with the addition of some effective filters.

Experiments (e.g., on the TPTP library) show the great impact of this kind of CNF generation. FLOTTER outperforms all other considered CNF generators in any sense: It produces fewer, smaller, and more general clauses with fewer symbols in less time. This has a remarkable positive effect on the theorem provers applied to the respective CNF results. Only in a few cases of satisfiable formulae the clause normal form produced by FLOTTER behaves worse than the results that are obtained from a standard or a straightforward definitional translation. The performance of FLOTTER is one of the reasons why SPASS won the first-order division of CASC-14, the CADE theorem proving contest by solving all problems.

In the near future the algorithm will be applied to further problem sets. Also there is some further work to be done in the combination of Optimized and Strong Skolemization. In the current implementation Strong Skolemization is only invoked when Optimized Skolemization would end up in a result identical to the standard one. There is some evidence that a more tight combination of the two approaches is possible and valuable. Moreover, there are several techniques not touched in this paper that seem to be valuable for further improvements. This includes, for example, potential reuse of Skolem functions and/or predicate symbols introduced by renaming as it is described in [6].

References

1. Thierry Boy de la Tour. An optimality result for clause form translation. *Journal of Symbolic Computation*, 14:283–301, 1992.
2. Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving.* Computer Science and Applied Mathematics. Academic Press, 1973.
3. Li Dafa. The formulation of the halting problem is not suitable for describing the halting problem. *Association for Automated Reasoning Newsletter*, 27:1–7, 1994.
4. Ingo Dahn, J. Gehne, Thomas Honigmann, and Andreas Wolf. Integration of automated and interactive theorem proving in ilf. In *Proceedings of the 14th International Conference on Automated Deduction, CADE-14*, volume 1249 of *LNAI*, pages 57–60, Townsville, Australia, 1997. Springer.
5. Uwe Egly. On the value of antiprenexing. In *Logic Programming and Automated Reasoning, 5th International Conference, LPAR'94*, volume 822 of *LNAI*, pages 69–83. Springer, July 1994.
6. Uwe Egly and Thomas Rath. The halting problem: An automatically generated proof. *AAR Newsletter*, 30:10–16, 1995.
7. Uwe Egly and Thomas Rath. On the practical value of different definitional translations to normal form. In M.A. McRobbie and J.K. Slaney, editors, *13th International Conference on Automated Deduction, CADE-13*, volume 1104 of *LNAI*, pages 403–417. Springer, 1996.
8. Georg Gottlob and Alexander Leitsch. On the efficiency of subsumption algorithms. *Journal of the ACM*, 32(2):280–295, 1985.

9. Donald W. Loveland. *Automated Theorem Proving: A Logical Basis*, volume 6 of *Fundamental Studies in Computer Science*. North-Holland, 1978.
10. William McCune and Larry Wos. Otter. *Journal of Automated Reasoning*, 18(2):211–220, 1997.
11. Andreas Nonnengart. Strong Skolemization. Technical Report MPI-I-96-2-010, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 1996. http://www.mpi-sb.mpg.de/~nonnenga/publications/.
12. Hans Jürgen Ohlbach and Christoph Weidenbach. A note on assumptions about skolem functions. *Journal of Automated Reasoning*, 15(2):267–275, 1995.
13. Francis Jeffry Pelletier. Seventy-five problems for testing automatic theorem provers. *Journal of Automated Reasoning*, 2(2):191–216, 1986. Errata: *Journal of Automated Reasoning*, 4(2):235–236,1988.
14. Francis Jeffry Pelletier and Geoff Sutcliffe. An erratum for some errata to automated theorem proving problems. *Association for Automated Reasoning Newsletter*, 31:8–14, December 1995.
15. David A. Plaisted and Steven Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2:293–304, 1986.
16. Georg Rock. Transformations of first-order formulae for automated reasoning. Diplomarbeit, Max-Planck-Institut für Informatik, Saarbrücken, Germany, April 1995. Supervisors: H.J. Ohlbach, C. Weidenbach.
17. Thoralf Skolem. Logisch-kombinatorische untersuchungen über die erfüllbarkeit oder beweisbarkeit mathematischer sätze nebst einem theoreme über dichte mengen. *Skrifter utgit av Videnskappsellkapet i Kristiania*, 4:4–36, 1920. Reprinted in: *From Frege to Gödel, A Source Book in Mathematical Logic, 1879–1931*, van Heijenoort, Jean, editor, pages 252–263, Harvard University Press, 1976.
18. Geoff Sutcliffe, Christian B. Suttner, and Theodor Yemenis. The TPTP problem library. In Alan Bundy, editor, *Twelfth International Conference on Automated Deduction, CADE-12*, volume 814 of *Lecture Notes in Artificial Intelligence, LNAI*, pages 252–266, Nancy, France, June 1994. Springer.
19. G.S. Tseitin. On the complexity of derivations in propositional calculus. In A.O. Slisenko, editor, *Studies in Constructive Mathematics and Mathematical Logic*. 1968. Reprinted in: *Automation of Reasoning: Classical Papers on Computational Logic*, J. Siekmann and G. Wrightson, editors, pages 466–483, Springer, 1983.
20. Christoph Weidenbach, Bernd Gaede, and Georg Rock. Spass & flotter, version 0.42. In M.A. McRobbie and J.K. Slaney, editors, *13th International Conference on Automated Deduction, CADE-13*, volume 1104 of *LNAI*, pages 141–145. Springer, 1996.