Effective Normalization Techniques for HOL*

Max Wisniewski, Alexander Steen, Kim Kern and Christoph Benzmüller

Dept. of Mathematics and Computer Science, Freie Universität Berlin, Germany
max.wisniewski|a.steen|kim.kern|c.benzmueller@fu-berlin.de

Abstract. Normalization procedures are an important component of most automated theorem provers. In this work we present an adaption of advanced first-order normalization techniques for higher-order theorem proving which have been bundled in a stand-alone tool. It can be used in conjunction with any higher-order theorem prover, even though the implemented techniques are primarily targeted on resolution-based provers. We evaluated the normalization procedure on selected problems of the TPTP using multiple HO ATPs. The results show a significant performance increase, in both speed and proving capabilities, for some of the tested problem instances.

1 Introduction

Problem normalization has always been an integral part of most automated theorem proving (ATP). Whereas early ATP systems relied heavily on external normalization and clausification, the normalization task has gradually been transferred to the prover themselves. The influence and success of FLOTTER [16] underlined the importance of careful employment of pre-processing techniques. Current state-of-the-art first-order ATP systems can spend a large portion of their execution time on pre-processing. Higher-order (HO) ATPs have not yet developed as sophisticated methods as their first-order counterparts and use hardly any sophisticated pre-processing techniques regarding clausification.

In this paper we present adaptations of prominent first-order techniques that improve clause normal form (CNF) calculations [12] first analyzed by Kern [8] in the context of higher-order logic (HOL; cf. [1] and the references therein). These adaptions are further augmented with HOL specific techniques and bundled in different normalization procedures. These procedures are intended as pre-processing routines for the new Leo-III theorem prover [17].

The effectiveness of these procedures is evaluated using a benchmark suite of over 500 higher-order problems. The measurements are conducted using the HO ATP systems LEO-II [5], Satallax [7] and Isabelle/HOL [10].

Furthermore, the normalization techniques are implemented in a stand-alone tool, called *Leonora*, ready to use with any TPTP-compliant HO ATP system.

* This work has been supported by the DFG under grant BE 2501/11-1 (Leo-III).

2 Normalization Techniques

The first two techniques, *simplification* and *extensionality treatment*, are already implemented in most systems. Nonetheless, we briefly survey them in the following. They are essential to the overall normalization process since they allow, when combined with further techniques, a more thorough in-depth normalization in some cases (see e.g. §2.4).

2.1 Simplification and Extensionality Treatment

Simplification is a procedure invoked quite often during proof search. It resolves simple syntactical tautologies and antinomies, removes trivial quantifiers, and eliminates the constant symbols for truth and falsehood from a formula. In general, simplification can be used to minimize formulas and reduce the number of applicable inference rules.

Extensionality Treatment. In comparison to FOL, equalities in HOL can occur between terms of any type, especially between terms of Boolean type or functional type. To guarantee completeness, these equalities must also comply to the extensionality principle. This is often dealt with using special extensionality rules in the underlying calculus. In our context, we employ an adaption of the extensional RUE calculus rules as implemented in LEO-II [3,5]. Intuitively, the rule for equality on Booleans $\Phi = \Psi$ replaces the equality by the equivalence $\Phi \Leftrightarrow \Psi$ using the fact, that the domain of Booleans only contains truth and falsehood. For equality on functions as in $f = g$ the rule states that two functions are equal if and only if they agree on each argument, hence we have $\forall X \,.\, f\, X = g\, X$ as result. We included both rules in the normalization framework for enabling deeply normalizing formulas: In some of the normalization procedures below, a rule can only be applied to a non-nested formula, i.e. not occurring at argument position. Using extensionality treatment, formulas can be lifted to top-level and then subsequently processed by other normalization steps.

2.2 Formula Renaming

Formula Renaming is a technique to reduce the size of the CNF [12]. Essentially, the idea is to split a clause into two separate clauses and to logically link them via a freshly introduced symbol that is added to both new clauses.

Definition 1 (Formula Renaming).
Let Φ be a formula and $\Psi_1 \circ \Psi_2$ a subterm of Φ, where $\circ$ is a binary Boolean connective. We replace Ψ_2 by $r(X_1, \ldots, X_n)$, where $\{X_1, \ldots, X_n\} = \mathit{free}(\Psi_2)$, r is a fresh predicate symbol (of appropriate type), and add a new clause D with

$$D = \begin{cases} r(X_1, \ldots, X_n) \supset \Psi_2 & , \mathit{if}\ \mathit{polarity}(\Psi_2) = 1 \\ \Psi_2 \supset r(X_1, \ldots, X_n) & , \mathit{if}\ \mathit{polarity}(\Psi_2) = -1 \end{cases}$$

if the size of the CNF (denoted $\#\mathrm{CNF}$) is decreasing, i.e.

$$\#\mathrm{CNF}(\Phi) > \#\mathrm{CNF}(\Phi[\Psi_2 \setminus r(X_1, \ldots, X_n)]) + \#\mathrm{CNF}(D).$$

The definition of $free(.)$ and $polarity(.)$ are hereby straight-forward adaptions of the their usual first-order counterparts (cf. e.g. [12]). We intentionally omitted the case of polarity$(\Psi_2) = 0$, since it is subsumed by a technique in §2.3.

Renaming reduces the size[1] of the CNF tremendously. In the example of a formula in disjunctive normal form, e.g. $(a \wedge b \wedge c) \vee (d \wedge e \wedge f)$, we obtain the nine multiplied cases $(a \vee d)$, $(a \vee e)$, ..., $(c \vee f)$. First renaming, however, yields the two clauses $(a \wedge b \wedge c) \vee r$ and $r \supset (d \wedge e \wedge f)$, which are normalized to six clauses $(a \vee r), (b \vee r), \ldots, (\neg r \vee e), (\neg r \vee f)$. This effect of formula renaming is mostly present in the multiplicative case of a β-rule, where the size of the CNF is reduced from a product (of the subterm sizes) to a sum. Thus, we can eliminate cases of exponential blowup in the transformation to CNF.

Reducing the search space in this manner can greatly boost the search process, as shown for first-order problems by Nonnengart et al. [11].

2.3 Argument Extraction

One major difference between higher-order and first-order logic is the shallow term-formula structure: Whereas in FOL we have the well-known distinct constructs of *formulas* and *terms*, in HOL there exist only terms (terms of Boolean type are still referred to as formulas). This allows in HOL the notion of nested formulas, that is, formulas p occurring at argument position of, e.g., an uninterpreted function symbol (in which case the polarity of p is 0). A treatment of these nested formulas is not immediately possible, since calculus rules dealing with Boolean formulas, such as clausification rules, cannot be applied to subterms. In order to apply these rules, the nested formulas have to be lifted to the top level, e.g. by decomposition rules as part of common unification procedures.

To allow immediate processing of nested formulas this lifting can be done in a pre-processing step [8]. Possible duplicated normalizations of nested formulas at a later proof search phase can thus be avoided. This *argument extraction* can be seen as a special higher-order case of *formula renaming* [12].

Definition 2 (Argument extraction).
Let Φ be a formula with $f(p)$ occurring as subterm. We replace p if its head symbol is a logical connective. More precisely, let $\{X_1, \ldots, X_n\} = free(p)$. We introduce a new function symbol s (of appropriate type) and return $\Phi[p \setminus s(X_1, \ldots, X_n)]$ together with the definition $\forall X_1 \ldots X_n \,.\, p = s(X_1, \ldots, X_n)$.

Consider the following theorem of HOL, which LEO-II is not able to solve:

$$\vdash \forall R \,.\, (R(\bot \Leftrightarrow (b \Leftrightarrow c)) \Rightarrow R((c \Leftrightarrow b) \Leftrightarrow \forall X \,.\, (X \wedge \neg X)))$$

In this formula the arguments of both occurrences of the Boolean connective R can be extracted, resulting in two axioms and the remaining conjecture:

$$\begin{aligned} & s_1 \Leftrightarrow (\bot \Leftrightarrow (b \Leftrightarrow c)), \\ & s_2 \Leftrightarrow ((c \Leftrightarrow b) \Leftrightarrow \forall X \,.\, (X \wedge \neg X)) \\ \vdash\ & \forall R \,.\, R(s_1) \Rightarrow R(s_2) \end{aligned}$$

[1] With *#CNF* we denote the number of clauses generated by transforming the given formula into clause normal form.

It is easy to see that further normalization steps are enabled by argument extraction. In other words, some challenging HOL aspects have been eliminated from the given proof problem in a pre-processing step. In fact, the processed problem is now easily provable for LEO-II.

2.4 Extended Prenex Normal Form

A term Φ is in prenex normal form, if it is of the form $Q_1 X_1 \ldots Q_n X_n \,.\, \Psi$ where Q_i are quantifier symbols and Ψ does not contain any quantifier. Many proof calculi, especially unification-based calculi, work on clauses with implicitly bound variables. The quantifier for such an implicitly bound variable is (implicitly) always enclosing the whole formula. Hence, it is necessary to move the quantifiers outwards.

We first adapted the normalization of Nonnengart et al. [12] that first skolemizes existential quantifiers. A higher-order formula

$$\vdash (\forall X \,.\, a(X) \wedge \exists Y \,.\, Y) \wedge \forall Y \,.\, p(\forall X \,.\, b(Y) \supset a(X))$$

treated with the adopted algorithm for prenex normal form yields

$$\vdash \forall X \,.\, \forall Y \,.\, ((a(X) \wedge sk_1(X)) \wedge p(\forall X \,.\, b(Y) \supset a(X)).$$

With this simple adaption, a prenex form cannot be reached in HOL. As in §2.3 we have to cope with nested formulas. Moving the quantifiers out of the nested application is not possible. In fact, even skolemizing is impossible, since we loose track of the polarity inside the application. However, applying argument extraction will introduce a new axiom containing the nested Boolean argument. Subsequently processed with extensionality – forcing a hard polarity distinction – all quantifiers will now appear at top-level and can be treated with the standard adaption to transform the problem into a *pure* higher-order prenex form. We call this approach *extended prenex normal form* which is, up to the author's knowledge, novel in the context of HOL. Normalizing the example finally yields

$$\begin{gathered} \forall Y \,.\, \forall X \,.\, \neg ek_1(Y) \vee (\neg b(Y) \vee a(X)), \\ \forall Y \,.\, ek_1(Y) \vee (b(Y) \wedge \neg a(sk_2(Y))) \\ \vdash \forall X \,.\, \forall Y \,.\, ((a(X) \wedge sk_1(X)) \wedge p(ek_1(Y)). \end{gathered}$$

3 Evaluation and Discussion

We have conducted several experiments to evaluate the potential benefits of the afore described normalization procedures. These experiments are designed to benchmark the number of problems that can be solved with the respective ATP as well as the time spent by the ATP on solving each individual problem.

The benchmark suite consists of overall 537 higher-order problems divided in eight domains from a broad field of application domains. The problems were

	AGT (23 Prob.)			**CSR** (123 Prob.)			**GEG** (18 Prob.)			**LCL** (139 Prob.)			**PHI** (10 Prob.)			**PUZ** (59 Prob.)			**QUA** (20 Prob.)			**SET** (145 Prob.)		
	Org	N_1	N_2	Org	N_1	N_2	Org	N_1	N_2	Org	N_1	N_2	Org	N_1	N_2	Org	N_1	N_2	Org	N_1	N_2	Org	N_1	N_2
LEO-II																								
Solved	23	23	23	61	69	45	12	13	11	104	93	104	6	5	5	31	32	31	0	0	0	134	134	135
–THM	23	23	23	57	65	41	12	13	11	99	88	99	6	5	5	31	32	31	0	0	0	134	134	135
Σ *[s]*	4.9	4.8	4.6	417	98	200	5.2	5.3	16.4	22.2	13.1	22.2	18.3	0.6	0.6	8.8	21	9.2	—	—	—	14.9	16.4	15.3
Avg. [s]	0.2	0.2	0.2	6.8	1.4	4.4	0.4	0.4	1.5	0.2	0.1	0.2	3	0.1	0.1	0.3	0.7	0.3	—	—	—	0.1	0.1	0.1
Satallax																								
Solved	18	19	19	58	78	51	17	17	15	113	114	112	7	7	7	35	37	33	2	2	2	138	136	138
–THM	18	19	19	54	74	47	17	17	15	104	105	103	7	7	7	31	34	30	2	2	2	138	136	138
Σ *[s]*	97	157	157	290	52	252	204	202	192	267	361	276	52	76	77	42	71	68	3.5	3.5	14.6	153	119	201
Avg. [s]	5.4	8.3	8.3	5	0.7	4.9	12	11.9	12.8	2.4	3.2	2.5	7.5	10.9	10.9	1.2	1.9	2.0	1.7	1.7	7.3	1.1	0.9	1.5

Table 1: Measurement results for normalizations N_1 and N_2 over all benchmark domains

taken from the TPTP library [14,15] (version 6.3.0) and coincide with the complete higher-order subsets of the corresponding TPTP problem domains[2] AGT, CSR, GEG, LCL, PHI, PUZ, QUA and SET.

For assessing the effectiveness of the proposed normalization pre-processing, we run the ATP systems on the original problems first and then on a series of differently normalized versions of the respective problems. These versions differ hereby in the number and combination of enabled normalization transformations from §2. More specifically, we investigated four different normalization procedures, denoted $N_1, \ldots, N_4$:

- N_1 Enabled routines: Prenex form, argument extraction, formula renaming, simplification and extensionality processing (*i.e. full normalization*)
- N_2 Enabled routines: Argument extraction, formula renaming, simplification and extensionality processing
- N_3 Enabled routines: Prenex form, argument extraction, simplification and extensionality processing
- N_4 Enabled routines: Argument extraction, simplification and extensionality processing

We chose to investigate different combinations of normalization techniques as pre-processing step to take into account that different ATP systems can use fundamentally different calculi and thus may benefit from different input conditions.

The measurements were primarily taken using the higher-order ATP systems LEO-II [5] and Satallax [7]. While the former system is based on higher-order resolution, the latter uses a sophisticated tableau-like approach. Selected benchmark results are additionally investigated using the automated and interactive theorem prover Isabelle/HOL [10]. In its automatic proof mode, Isabelle employs different solving tools such as the counter model finder Nitpick [6], the first-order tableau prover Blast [13], the SMT solver CVC4 [2] and several more.

[2] A comprehensive presentation of the different TPTP problem domains and their application domain can be found at `http://www.cs.miami.edu/~tptp/cgi-bin/SeeTPTP?Category=Documents&File=OverallSynopsis`.

As indicated before, for each original problem and each of the four normalized versions (by $N_1, \ldots, N_4$ respectively) we measure whether the system were able to solve the input problems as well as the time taken to do so. The CPU limit (timeout) for each problem and each ATP system is limited to $60s$. The measurements were taken on a 8 core (2x AMD Opteron Processor 2376 Quad Core) machine with 32 GB RAM.

Pre-processing time is not considered in the results below, since it still carries essentially little weight and does not considerably contribute to the overall CPU time.

Results and Discussion. Table 1 displays the benchmark result summary. For the two ATP systems LEO-II and Satallax, the number of solved problems and the CPU time is shown for the original problems (denoted Org) in the respective domain and the pre-processed problem domains (denoted N_1 and N_2 respectively). The results for the remaining two normalization procedures are omitted since they are very similar to the ones shown.[3] The number of solved problems (thereof theorems) is denoted *Solved* (*–THM*). The sum (average) of CPU time spent on solving all input problems (that could be solved using the respective normalization procedure) is denoted Σ (*Avg.*).

As can be seen, in the case of the LEO-II prover, the normalization procedure N_1 results in more solved problems in benchmark domains CSR, GEG and PUZ. A decrease in solved problems can be observed in domains LCL (only 90% solved) and PHI (83% solved). The results of the remaining three benchmark domains only differ in the overall (and average) solving time. For the Satallax prover, the results are even better: More problems were solved in domains AGT, CSR, LCL and SET using N_1. Only in domain SET there are some problems that could not be proven anymore (see remark on domain SET below). In all cases, the normalization procedure N_2 did not improve the reasoning effectivity.

The most striking increase in solved problems can be observed in domain CSR where 8 problem were additionally solved due to N_1 (roughly 13%) by LEO-II. Also, the overall (and average) proving time in N_1 only takes approximately a quarter of the original time (while proving more problems in that time). These observations also apply for Satallax, where 20 more problems (34%) were solved by using N_1 while reducing the reasoning time to roughly one quarter of the original time. Detailed measurement results for problem domain CSR are shown in Table 2, where the fifteen best speed-ups are displayed for each employed ATP system. In order to provide additional evidence for the practicability of the presented normalization procedures, we included Isabelle/HOL in these measurements. The average speed-up for the Isabelle system is approx. 66%. Here, some problems that were originally provable by Isabelle's CVC4 routine become provable by Blast after normalization with N_1, hence the speed-up.

The above results shows a significant increase in reasoning effectivity for problems of the CSR (*c*ommon*s*ense *r*easoning) domain. In the investigated

[3] The average time results for normalization procedure N_3 are in nearly all cases within a range of 0.1% of the results for N_1. Likewise results apply for N_4 and N_2.

Problem	Time [s] Orig.	N_1	N_2
CSR153^2	38.254	0.054	†
CSR138^1	9.858	0.029	9.875
CSR153^1	5.492	0.038	5.493
CSR126^2	31.425	0.676	31.451
CSR139^1	10.022	0.266	10.027
CSR137^2	1.351	0.039	4.270
CSR134^1	9.713	0.338	0.359
CSR122^2	19.307	0.683	19.266
CSR143^2	2.714	0.238	†
CSR153^3	7.743	0.988	†
CSR119^3	26.588	3.672	†
CSR120^3	26.609	3.678	†
CSR137^1	0.242	0.042	0.246
CSR152^3	14.960	3.671	†
CSR151^3	14.939	3.668	27.075

(a) Satallax

Problem	Time [s] Orig.	N_1	N_2
CSR139^2	5.331	0.146	5.633
CSR132^2	5.331	0.283	†
CSR139^1	1.196	0.055	1.142
CSR150^1	1.633	0.091	1.604
CSR141^2	1.229	0.202	†
CSR148^1	0.295	0.057	†
CSR149^2	1.002	0.194	1.479
CSR123^2	0.930	0.192	†
CSR124^2	0.731	0.190	†
CSR122^2	0.725	0.193	†
CSR125^2	0.757	0.327	†
CSR119^2	0.355	0.173	†
CSR138^1	0.104	0.054	0.122
CSR120^2	0.388	0.202	†
CSR127^2	0.342	0.190	0.529

(b) LEO-II

Problem	Time [s] Orig.	N_1	N_2
CSR128^2	52.286	14.305	49.489
CSR153^2	50.682	14.056	51.087
CSR131^2	49.519	13.983	52.484
CSR133^2	48.984	13.902	50.028
CSR148^2	43.199	14.172	42.063
CSR149^2	40.294	14.310	38.664
CSR138^2	39.387	15.305	40.481
CSR150^1	29.746	11.767	29.844
CSR130^2	49.632	21.136	46.207
CSR132^2	55.217	24.252	56.829
CSR129^2	47.401	20.979	45.682
CSR119^1	26.858	12.259	12.343
CSR141^2	52.501	26.366	44.813
CSR123^2	52.272	26.337	54.192
CSR127^2	27.227	14.248	26.311

(c) Isabelle

Table 2: The 15 best relative time improvements with normalization N_1 in CSR domain for the respective prover (ordered by speed-up). Normalization N_2 is shown for comparison. A timeout result of a system is denoted †.

THF subset of that domain, a majority of problems represent HOL embeddings of SUMO [9] reasoning tasks.

Another major observation is that a TPTP rating 1.0 problem (i.e. a problem that could not be solved by any ATP system) became provable after normalization with procedure N_1. Here, the problem PUZ145^1 from the puzzles domain can be shown to be a theorem in $5.8s$ by Satallax.

Another interesting aspect of the normalization procedures is the impact on occurrences of defined equalities in problems. As discussed before, some problems in our benchmark suite became unprovable after normalization. This could be due to the fact that the HOL ATPs under consideration provide some special techniques for the manipulation of defined equalities, for example, Leibniz equalities. Leibniz equalities have the form $\forall P.Pa \Leftrightarrow Pb$, $\forall P.Pa \supset Pb$, $\forall P.\neg Pa \vee Pb$, etc., for arbitrary terms a and b.[4] The special techniques are aiming at a more goal directed equality handling as is possible with the above formulas, cf. [4]. However, our implemented normalization procedures do not yet support a similar detection and treatment of defined equalities. Thus, executing a normalization strategy can alter the structure of a (sub-)formula in a way that the HOL ATPs do subsequently not recognize it as an instance of a defined equality anymore. Augmenting our procedures with special techniques for defined equalities might therefore further improve the above results. Similar techniques might be useful for description and choice.

[4] The prover LEO-II, for example, is able to detect such (sub-)formulas and to replace them by primitive equalities $a = b$.

Implementation. For the above experiments, the described normalization techniques were implemented into a stand-alone pre-processing tool, called *Leonora* (for *Leo*'s *nor*m*a*lization) that can be used to normalize any higher-order problem file in THF format [15], ready to employ in conjunction with any TPTP-compliant HO ATP system [14]. The selection of normalization steps to apply on the input problem can be controlled individually for each technique via flags, e.g. `-a` for enabling argument extraction or `-r` for formula renaming. A preliminary version of Leonora is freely available under MIT license and can be found at GitHub[5]. For the experiments we used version 1 of Leonora.

4 Further Work

Even though we have not yet included special techniques for defined equalities, the results show a significant improvement in some problem domains. Introducing such techniques has large potential to further improve these results.

Future work is to additionally support normalization techniques that are more suited for non-CNF based calculi. While it was indeed possible to improve Satallax's performance in both speed and proving capabilities in some problem domains, the system is using a quite different approach, i.e. a tableau method with an iterative queuing, SAT solver and special treatment for existentially quantified formulas. Applying the standard prenex algorithm could complicate the problems for Satallax. In order to deal with provers that are not suited for working with Skolem variables, we will additionally implement a procedure that moves all quantifiers outside and omits the skolemization.

Additionally there is a lot of potential for further improvements, such as finding meaningful size functions for the formula renaming procedure (cf. #CNF from §2.2). One example would be size functions that aim at minimizing the number of tableau branches created by the input problem.

5 Conclusion

In this work, we have adopted prominent first-order normalization techniques for higher-order logic. First benchmark results of these techniques on a set of HO problems indicate promising results. For each of the employed ATP systems the normalization procedures enables an improvement in both speed as well as number of solved problems for most benchmark domains. In some domains, up to 20 (34%) more problems could be solved. Additionally, a problem that was not solved by any ATP system before could be solved by Satallax in less than six seconds after normalization.

We have observed that a straight-forward adaption of FOL techniques is not enough for the HOL case. Especially the treatment of nested formulas has great potential. Additionally, we have identified that a simple application of these techniques can interfere with occurrence of defined equalities, which is a

[5] The Leonora repository can be found at `https://github.com/Ryugoron/Leonora`

problem specifically arising in HOL. By lifting nested formulas to top-level, we have on the one hand established a prenex normal form for HOL. On the other hand, the lifting contributes to the improvement of a prover's performance.

References

1. P. Andrews. Church's type theory. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Stanford University, Spring 2014 edition, 2014.
2. C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli. Cvc4. In *23rd Int. Conference on Computer Aided Verification*, volume 6806 of *LNCS*, pages 171–177. Springer, 2011.
3. C. Benzmüller. Higher-order automated theorem provers. In D. Delahaye and B. Woltzenlogel Paleo, editors, *All about Proofs, Proof for All*, Mathematical Logic and Foundations, pages 171–214. College Publications, London, UK, 2015.
4. C. Benzmüller, Brown C, and M. Kohlhase. Cut-simulation and impredicativity. *Logical Methods in Computer Science*, 5(1:6):1–21, 2009.
5. C. Benzmüller, L. C. Paulson, N. Sultana, and F. Theiß. The Higher-Order Prover LEO-II. *J. Automated Reasoning*, 2015.
6. J. C. Blanchette and T. Nipkow. Nitpick: A counterexample generator for higher-order logic based on a relational model finder. *ITP*, 6172:131–146, 2010.
7. Chad. E. Brown. Satallax: An Automatic Higher-Order Prover. In *Automated Reasoning*, volume 7364 of *LNCS*, pages 111–117. Springer Berlin Heidelberg, 2012.
8. Kim Kern. Improved Computation of CNF in Higher-Order Logics. Bachelor thesis, Freie Universität Berlin, 2015.
9. I. Niles and A. Pease. Towards a standard upper ontology. In *Int. Conf. on Formal Ontology in Information Systems, Proceedings*, pages 2–9. ACM, 2001.
10. Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
11. A. Nonnengart, G. Rock, and C. Weidenbach. On Generating Small Clause Normal Forms. In C. Kirchner and H. Kirchner, editors, *15th Int. Conf. on Automated Deduction, Proc.*, volume 1421 of *LNCS*, pages 397–411, Germany, 1998. Springer.
12. A. Nonnengart and C. Weidenbach. Computing Small Clause Normal Forms. In J. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume 1, pages 335–367. Gulf Professional Publishing, 2001.
13. L. C Paulson. A generic tableau prover and its integration with isabelle. *J. Universal Computer Science*, 5(3):73–87, 1999.
14. G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure. *J. Automated Reasoning*, 43(4):337–362, 2009.
15. G. Sutcliffe and C. Benzmüller. Automated Reasoning in Higher-Order Logic using the TPTP THF Infrastructure. *J. Formalized Reasoning*, 3(1):1–27, 2010.
16. C. Weidenbach, B. Gaede, and G. Rock. SPASS & FLOTTER version 0.42. In M. A. McRobbie and J. K. Slaney, editors, *13th Int. Conf. on Automated Deduction, Proc.*, volume 1104 of *LNCS*, pages 141–145, USA, 1996. Springer.
17. M. Wisniewski, A. Steen, and C. Benzmüller. The Leo-III Project. In A. Bolotov and M. Kerber, editors, *Joint Automated Reasoning Workshop and Deduktionstreffen*, page 38, 2014.