The Efficiency of Automated Theorem Proving by Translation to Less Expressive Logics

Negin Arhami and Geoff Sutcliffe

University of Miami, Coral Gables, USA
n.arhami@umiami.edu, geoff@cs.miami.edu

Abstract

Many Automated Theorem Prover (ATP) systems for different logics, and translators for translating different logics from one to another, have been developed and are now available. Some logics are more expressive than others, and it is easier to express problems in those logics. On the other hand, the ATP systems for less expressive logics have been under development for many years, and are more powerful and reliable. There is a trade-off between expressivity of a logic, and the power and reliability of the available ATP systems. Translators and ATP systems can be combined to try to solve a problem. In this research, an experiment has been carried out to compare the performance of different combinations of translators and ATP systems.

1 Introduction

Automated Theorem Proving (ATP) [9] is concerned with the development of automatic techniques and computer programs for checking whether the conjecture of a logic problem is a theorem of the axioms of the problem. An *ATP system* is a program that automatically tries to determine whether the conjecture of a logic problem is a theorem. The TPTP [19] provides support for a range of logics for writing logic problems, including, in increasing order of expressivity, Propositional Logic (PL), Effectively Propositional form (EPR), Conjunctive Normal Form (CNF), First Order Form (FOF), Typed First order Form - monomorphic (TF0), Typed First order Form - polymorphic (TF1), and Typed Higher order Form - monomorphic (TH0). Note that the TPTP does not support Description Logic (DL), which sits between PL and EPR in terms of expressivity. A separate CNF to DL translator was developed as part of this research, as described in Section 1.2. Expressing a problem in a more expressive logic is easier because it is more natural and compact. On the other hand, ATP systems for less expressive logics are typically more powerful than ATP systems for more expressive logics, because they have been under development for many more years (e.g., ATP systems for first-order logic, along the lines of Otter and Prover9 [10], Vampire [13], E [16] have been developed since the 1960's, while ATP systems for high-order logic, such as LEO-II [2] and Satallax [3] have been developed only in the last 10 years).

It is often possible to soundly translate a problem written in one logic to another logic. Translations from more expressive logics to less expressive ones are of interest in this work. (Translation from less expressive logics to more expressive ones, e.g., [15], are also interesting in general.) A problem can be written in a more expressive logic to benefit from the higher expressivity, then an ATP system for that logic can be used to try solve the problem, or the problem can be translated down to a less expressive logic where a more powerful ATP system might be available. If a problem is translated to a less expressive logic, again the same two options of using an ATP system for that logic, or translating down again (if possible), are available. Figure 1 illustrates these alternatives for the logics listed above, for currently available translators and ATP systems. In the process of translating between two logics, extra

complexity might be added to the problem, which affects the performance of the corresponding ATP systems. There is is a trade-off between power of the ATP systems for a logic and the effect of translating to the destination logic. The choice of using an ATP system for the original logic, or translating to a less expressive logic, is the major concern of this research.

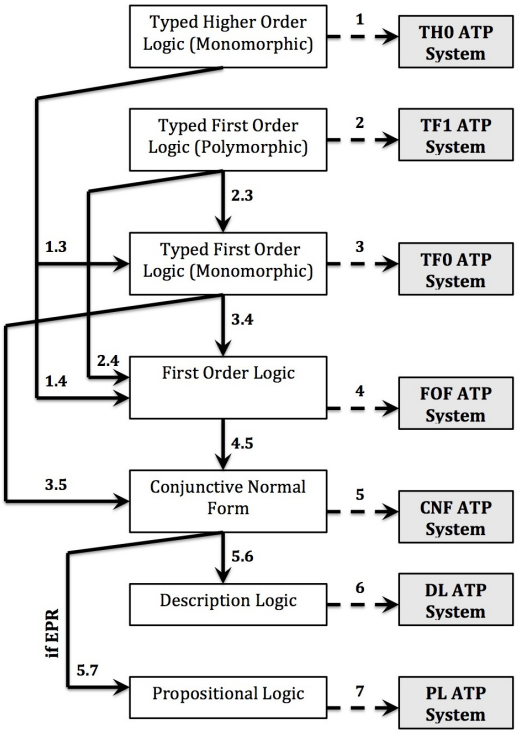

Figure 1: Different Combinations of Translators and ATP Systems to Solve a Problem

1.1 Processes and Tools

Table 1 lists the translators and ATP systems used in this research, corresponding to the labels on the arrows in Figure 1. The translators and ATP systems represent the state-of-the-art at the time of this research - most of the the ATP systems were winners of recent competitions, such as CASC [11]. If a CNF problem is EPR then iProver is used as the ATP system, otherwise Vampire is used (as motivated by the results of recent CASCs).

1.2 The Saffron CNF to DL Translator

No CNF to DL translator was available when this research was started, and the translator Saffron was implemented in this research. Saffron is implemented in Prolog. The input is a CNF problem in TPTP syntax, and the output is a DL problem in RDF syntax. Saffron can translate only those clauses that contain only constants, unary predicates, and binary predicates. It also might not be possible to translate some clauses with only constants, unary predicates, and binary predicates to DL, because there might not be equivalent DL semantics for the clause.

Label	Action			Tool	
1	TH0	⇒	Proof	Isabelle-HOT 2013	[12]
1.3	TH0	⇒	TF0	Isabelle-2TF0 2013	[12]
1.4	TH0	⇒	FOF	Isabelle-2FOF 2013	[12]
2	TF1	⇒	Proof	Alt-Ergo 0.94	[5]
2.3	TF1	⇒	TF0	Why3-TF0 0.71	[7]
2.4	TF1	⇒	FOF	Why3-FOF 0.71	[7]
3	TF0	⇒	Proof	Princess 120604	[14]
3.4	TF0	⇒	FOF	Monotonox-2FOF e3c1636	[4]
3.5	TF0	⇒	CNF	Monotonox-2CNF e3c1636	[4]
4	FOF	⇒	Proof	Vampire 3.0	[13]
4.5	FOF	⇒	CNF	ECNF 1.8	[16]
5	CNF	⇒	Proof	Vampire 3.0 or iProver 1.0	[13, 8]
5.6	CNF	⇒	DL	Saffron 1.0	[1]
5.7	CNF	⇒	PL	EGround 1.8	[17]
6	DL	⇒	Proof	HermiT 1.3.8	[18]
7	PL	⇒	Proof	MiniSat 2.2.0	[6]

Table 1: ATP Systems and Translators used in this research

Clauses that can be translated are referred to as *DL-able* clauses. The translation of a CNF problem to DL might not be complete because there might be non-DL-able clauses in the problem. However, the unsatisfiability of a partially translated CNF problem is preserved if an unsatisfiable subset of the clauses is translated to DL, so that partial translation can still be useful in the context of the standard approach of proof by refutation.

Constants, unary predicates, and binary predicates in CNF correspond to individuals, classes, and roles that are the building blocks of DL. Every DL axiom defines a characteristic of an individual, class, or role, or describes the default class `Thing`. Translation from CNF to DL is done by considering the intended semantics of each clause. For example, the CNF clause `~p(X)|q(X)` includes two unary predicates, `p` and `q`, that correspondingly introduce two DL classes `P` and `Q`. This clause expresses the fact that for every element `e` of an interpretation domain, if `p(e)` then `q(e)`. The corresponding DL axiom of this clause defines a characteristic of the class `P`, which is that the class `P` is a subclass of the class `Q`. CNF clauses that describe the default class `Thing` are expressed in DL as a restriction on, or a description of the class `Thing`. For example, `~p(X)` is expressed in DL as an axiom that says the class `Thing` is equivalent to the complement of the class `P`.

For every constant appearing in a CNF problem, an individual with the same name is defined in DL. The characteristics of individuals that are covered in this translation are an individual belonging to a class, an individual not belonging to a class, an individual belonging to the union of classes and the complements of classes, and an individual being in a binary relationship with another individual. An equality or negative equality between two constants in a CNF clause are translated to "same individual" and "different individual" DL axioms.

For every unary predicate appearing in a CNF problem, a class with the same name (with the first letter capitalized) is defined in DL. The characteristics of classes that are covered in this translation are a class being a subclass of another class, and a class being equivalent to class `Thing`. In CNF equivalency is expressed by two clauses, each of which expresses that one of the classes is a subclass of the other, which are then translated to DL. DL reasoners recognize

such a pair of axioms as expressing equivalency. For every binary predicate appearing in a CNF problem, a role with the same name is defined in DL. The role characteristics that are covered in this translation are reflexivity, irreflexivity, symmetry, asymmetry, transitivity, and a role being the inverse of another role.

In RDF syntax, each individual, each class, each role, and class `Thing` is expressed as an RDF tag, with their characteristics (if any exist) as sub-tags. When the clause-by-clause translation of a whole CNF problem is completed, for each individual, each class, each role, and class `Thing`, all the characteristics, that were found during the translation are gathered and combined into in an RDF tag.

2 Experiments

An experiment has been carried out to compare the reasoning processes through all available paths for problems in different logics (TH0, TF1, TF0, FOF and CNF), in terms of the number of problems solved and the CPU time taken (including the time for translation), over standard sample problems. A 30s CPU time limit was imposed on each proof attempt. The experiment was done on a computer with the following specifications.

Number of CPUs : 4
CPU Model: Intel(R) Xeon(TM) CPU 2.80GHz
RAM per CPU: 756MB
OS: Linux 2.6.32.26-175.fc12.i686.PAE

The translators and ATP systems used are those listed in Table 1.

Notation: A *path* denoted by $logic_1 \cdot logic_2 \cdot \ldots \cdot logic_n$, with a chain of translators and an ATP system $translator_1 \rhd translator_2 \rhd \ldots \rhd translator_{n-1} \rhd atp_n$, means that the logic problem is originally in logic $logic_1$, and for $1 \leq i \leq n-1$, problems in the logic $logic_i$ is translated to $logic_{i+1}$ using the translator $translator_i$, and eventually the problem in $logic_n$ is solved using the ATP system atp_n.

In the result tables, the # column shows the number problems that could be translated by that path, and are submitted to the ATP system. As is evident from the result tables, translation was not always successful. The failures are mostly due to timeouts and memory errors. The *Solved* column shows the number of problems that were solved, and the percentage solved of the number translated. The *Time* column shows the average CPU time taken over the problems solved, in seconds.

2.1 Conjunctive Normal Form

Some CNF problems can be translated to DL, and CNF EPR problems can be translated to PL. Two sets of sample problems were used to compare the possible paths for CNF problems. One set is the problems from the CASC-J6 CNF division, none of which are EPR. The other is the set of CNF problems in the TPTP that include only constants and unary and binary predicates, which are necessarily EPR problems. Table 2 shows the results for the first set of CNF problems. As these problems are not EPR, none of them were translated to PL. The translation of all of these CNF problems to DL were all partial, with an average of only 17% of the clauses translated to DL. This incompleteness is presumed to be the reason why none of the resultant DL problems could be shown to be unsatisfiable by `HermiT`, and further work to extend the scope of the translator is necessary for this kind of problem. Table 3 shows the results for the second set of problems. As illustrated in this table, iProver can solve more problems than

EGround▷MiniSAT, and EGround▷MiniSAT can solve more problems than Saffron▷HermiT. Only 11 of the 262 can be solved through all three paths.

Although Saffron▷HermiT solved less problems than iProver and EGround▷MiniSAT, there are 46 problems that are uniquely solved by Saffron▷HermiT, i.e., problems that cannot be solved through the other two paths. Similarly, there are 15 problems that are uniquely solved by iProver. These are significant numbers of uniquely solved problems, suggesting that when more than one CPU is available, different paths should be used in parallel to increase the chance of the problem being solved. Therefore, with N processors, the N paths that provide the maximum number of problems solved are chosen. Table 4 shows which path is the best to be picked as an additional path to maximize the number of problems solved. The number of uniquely solved problems in each row is the number of problems that can be solved by only that path, compared to all paths up to that row. The number solved is the total number of problems that can be solved by all the N paths up to that row. Out of 262 sample problems in CNF, 175 problems can be solved through all paths together. This is a significant gain, and illustrates the benefit of integrating DL-based reasoning with standard first-order ATP. The two paths shown in Table 4 can solve all of these problems, so using more than two CPUs does not increase the number of problems solved.

Path	Tools	#	Solved	Time
CNF	Vampire	150	123 (82%)	2.2
CNF.DL	Saffron▷HermiT	150	0 (0%)	0.0

Table 2: Paths from 150 CNF Problems from CASC-J6

Path	Tools	#	Solved	Time
CNF	iProver	262	129 (49%)	2.2
CNF.PL	EGround▷MiniSAT	165	102 (61%)	0.0
CNF.DL	Saffron▷HermiT	262	69 (26%)	0.1

Table 3: Paths from 262 CNF Completely DL-able Problems

CPUs	Path	Unique	Solved	Time
1	CNF	129	129	2.2
2	CNF.DL	46	175	0.6

Table 4: Paths from 262 CNF Completely DL-able Problems in Parallel

2.2 First Order Form (FOF)

FOF problems can be translated to CNF. If a problem is translated to CNF then it can be translated to DL or PL, in the way explained in Section 2.1. The sample problems are from the CASC-J6 FOF division. Table 5 shows the results. Table 6 splits the results between EPR and non-EPR CNF problems.

Table 7 shows that by increasing the number of CPUs, which path is the best to be picked as an additional path to maximize the number of problems solved. Out of the 500 sample problems, 425 problems can be solved through all paths together. This is a significant gain over the 396 solved by Vampire alone. Among the 16 problems that are exclusively solved by

the FOF.CNF path, 13 are EPR that are solved using iProver Only three of the problems on the FOF.CNF.DL path are completely translated to DL, and HermiT confirmed their unsatisfiability. On average 21% of the clauses of problems on the FOF.CNF.DL path are translated to DL, but HermiT still confirmed the unsatisfiability of 17 partially translated problems. This shows that partial translation really can be useful. The three paths shown in Table 7 can solve all of these problems, so using more than three CPUs will not increase the number of problems solved.

Paths	Tools	#	Solved	Time
FOF	Vampire	500	396 (79%)	3.1
FOF.CNF	ECNF▷(Vampire or iProver)	445	294 (58%)	1.5
FOF.CNF.DL	ECNF▷Saffron▷HermiT	441	20 (4%)	0.5
FOF.CNF.PL	ECNF▷EGround▷MiniSAT	13	13 (100%)	0.1

Table 5: Paths from 500 FOF Problems

Paths	Tools	#	Solved	Time
FOF.EPR	ECNF▷iProver	30	21 (70%)	4.0
FOF.non-EPR	ECNF▷Vampire	415	273 (65%)	1.3

Table 6: Paths from 500 FOF Problems to CNF

CPUs	Path	Unique	Solved	Time
1	FOF	396	396	3.0
2	FOF.CNF	16	412	2.4
3	FOF.CNF.DL	13	425	2.1

Table 7: Paths from 500 FOF Problems in Parallel

2.3 Typed First-order Form - monomorphic

TF0 problems can be translated to FOF or CNF. If a problem is translated to FOF, then it can be translated to CNF in the way explained in Section 2.2. If a problem is translated to CNF, then it can be translated to DL or PL, in the way explained in Section 2.1. The sample problems are the TF0 problems that were in the TPTP when the research was initiated. Table 8 shows the results. There is a clear advantage to translating to FOF or CNF, rather than using the ATP system for TF0. The lower CPU times of the FOF and CNF-based paths is particularly better. The TF0 logic is relatively new in the TPTP, and the results show that ATP systems for this logic have not matured to the extent of FOF and CNF based reasoning. There is no benefit to translating to even less expressive logics.

Table 9 shows that by increasing the number of CPUs, which path is the best to be picked as an additional path to maximize the number of problems solved. Out of 97 sample problems, 44 can be solved through all paths together. This provides a useful gain of around 20% more problems solved. It is interesting that TF0.FOF.CNF.DL path, which solves only 5 problems, provides the most unique contribution after the top ranked system. On average 58% of the clauses of problems on the TF0.FOF.CNF.DL path are translated to DL The four paths shown in Table 9 can solve all of these problems, so using more than four CPUs will not increase the number of problems solved.

Paths	Tools	#	Solved	Time
TF0.FOF	Monotonox-2FOF▷Vampire	88	36 (40%)	1.4
TF0.FOF.CNF	Monotonox-2FOF▷ECNF▷Vampire	88	31 (35%)	1.0
TF0.CNF	Monotonox-2CNF▷Vampire	96	24 (25%)	1.8
TF0	Princess	97	12 (12%)	12.0
TF0.FOF.CNF.DL	Monotonox-2FOF▷ECNF▷Saffron▷HermiT	10	5 (50%)	0.2
TF0.CNF.DL	Monotonox-2CNF▷Saffron▷HermiT	15	2 (13%)	0.2
TF0.CNF.PL	Monotonox-2CNF▷EGround▷MiniSAT	3	0 (0%)	0.0
TF0.FOF.CNF.PL	Monotonox-2FOF▷ECNF▷EGround▷MiniSAT	3	0 (0%)	0.0

Table 8: Paths from 97 TF0 Problems

CPUs	Path	Unique	Solved	Time
1	TF0.FOF	36	36	1.4
2	TF0.FOF.CNF.DL	5	41	1.2
3	TF0	2	43	1.3
4	TF0.CNF.DL	1	44	1.3

Table 9: Paths from 97 TF0 Problems in Parallel

2.4 Typed First-order From - polymorphic

TF1 problems can be translated to TF0 or FOF. If a problem is translated to TF0, then it can be translated to FOF or CNF in the way explained in Section 2.3. If a problem is translated to FOF, then it can be translated to CNF in the way explained in Section 2.2. If a problem is translated to CNF then it can be translated to DL or PL, in the way explained in Section 2.1. The sample problems are the TF1 problems that were in the TPTP when the research was initiated. Table 10 shows the results. As for TF0, there is a clear advantage to translating to FOF or CNF, rather than using the ATP system for TF1. Like TF0, the TF1 logic is relatively new in the TPTP, and the results show that ATP systems for this logic have not matured to the extent of FOF and CNF based reasoning. However, the effect is less severe compared to TF0, thanks to the strength of the relatively mature Alt-Ergo ATP system.

Table 11 shows that by increasing the number of processors which path is the best to be picked as an additional path to the path set to maximize the number of problems solved. Out of the 987 sample problems, 385 problems can be solved by all alternatives together. This provides a useful gain of around 10% more problems solved. The four paths shown in Table 11 can solve all of these problems, so using more than four CPUs will not increase the number of problems solved.

2.5 Typed Higher-order Form - monomorphic

TH0 problems can be translated to TF0 and FOF. If a problem is translated to TF0 then it can be translated to FOF or CNF, in the way explained in Section 2.3. If a problem is translated to FOF then it can be translated to CNF, in the way explained in Section 2.2. If a problem is translated to CNF then it can be translated to DL or PL, in the way explained in Section 2.1. The sample problems are from the CASC-J6 TH0 division. Table 12 shows the results. There is only a slight time benefit of translating to FOF.

Table 13 shows that by increasing the number of processors which path is the best to be picked as an additional path to the path set to maximize the number of problems solved. Out

Paths	Tools	#	Solved	Time
TF1.FOF.CNF	Why3-FOF▷ECNF▷Vampire	987	348 (35%)	0.8
TF1.TF0.FOF	Why3-TF0▷Monotonox-2FOF▷Vampire	957	329 (34%)	0.9
TF1.FOF	Why3-FOF▷Vampire	987	317 (32%)	0.8
TF1	Alt-Ergo	987	312 (31%)	1.0
TF1.TF0.CNF	Why3-TF0▷Monotonox-2CNF▷Vampire	957	276 (28%)	1.7
TF1.TF0	Why3-TF0▷Princess	987	33 (3%)	12.6
TF1.TF0.FOF.CNF	Why3-TF0▷Monotonox-2FOF▷ECNF ▷Vampire	48	16 (33%)	0.8
TF1.TF0.FOF.CNF.DL	Why3-TF0▷Monotonox-2FOF▷ECNF ▷Saffron▷HermiT	957	2 (0%)	0.1
TF1.TF0.CNF.DL	Why3-TF0▷Monotonox-2CNF▷Saffron ▷HermiT	957	2 (0%)	0.5
TF1.FOF.CNF.DL	Why3-FOF▷ECNF▷Saffron▷HermiT	957	0 (0%)	0.0
TF1.⋆.CNF.PL	⋆▷EGround▷MiniSAT	0	0 (0%)	0.0

Table 10: Paths from 987 TF1 Problems

CPUs	Path	Unique	Solved	Time
1	TF1.FOF.CNF	348	348	0.8
2	TF1	23	371	0.6
3	TF1.TF0.FOF	9	380	0.5
4	TF1.FOF	5	385	0.5

Table 11: Paths from 987 TF1 Problems in Parallel

of the 200 sample problems, 114 problems can be solved through all paths together. This is a significant gain over direct reasoning or translation to FOF. The four paths shown in Table 13 can solve all of these problems, so using more than four CPUs will not increase the number of problems solved.

Paths	Tools	#	Solved	Time
TH0.FOF	Isabelle-2FOF▷Vampire	196	78 (39%)	9.0
TH0	Isabelle-HOT	200	78 (39%)	11.9
TH0.TF0.FOF	Isabelle-2TF0▷Isabelle-2FOF▷Vampire	192	77 (40%)	11.4
TH0.TF0.CNF	Isabelle-2TF0▷Monotonox-2CNF▷Vampire	194	52 (26%)	5.7
TH0.TF0	Isabelle-2TF0▷Princess	196	24 (12%)	14.9
TH0.TF0.FOF.CNF	Isabelle-2TF0▷Isabelle-2FOF▷ECNF ▷Vampire	192	0 (0%)	0.0
TH0.FOF.CNF	Isabelle-2FOF▷ECNF▷Vampire	193	0 (0%)	0.0
TH0.⋆.CNF.PL	⋆▷EGround▷MiniSAT	0	0 (0%)	0.0
TH0.⋆.CNF.DL	⋆▷Saffron▷HermiT	0	0 (0%)	0.0

Table 12: Paths from 200 TH0 Problems

CPUs	Path	Unique	Solved	Time
1	TH0.FOF	78	78	8.8
2	TH0	33	111	9.4
3	TH0.TF0.FOF	2	113	9.4
4	TH0.TF0.CNF	1	114	8.1

Table 13: Paths from 200 TH0 Problems in Parallel

3 Conclusion

There are different ways of solving a problem expressed in a logical form. It can be solved directly using the ATP system of that logic or, if possible, it can be soundly translated to a less expressive form. When it is translated to a less expressive form, again the same two options are available. In this research, different ways of solving a problem have been compared. The results show that more TF0 and TF1 problems are solved by translation to less expressive logics than by using the ATP system of the source logic. Generally, the best thing to do seems to be to translate to FOF or CNF, and take advantage of the highly mature and optimised ATP systems for those logics. When more than one CPU is available, parallelization over multiple paths can provide significant gain, and the experiment shows this to be most true for completely DL-able CNF problems, FOF problems, and TH0 problems.

The development of the Saffron translator has provided a new tool that offers new possibilities for reasoning by translation to DL. DL-based paths provided significant numbers of uniquely solved CNF, FOF, and TF0 problems. Saffron and HermiT have been combined into an ATP system called SafHer, which is available in the SystemOnTPTP interface [20]. Some initial experiments on EPR problems in the HWV domain of the TPTP suggests that translation to DL might provide a valuable alternative to using a ATP system such as iProver, which is already well tuned for EPR problems. Further experiments are being performed to evaluate this. Saffron is also being extended to increase the extent to which CNF problems can be translated to DL, which will increase the completeness of paths that use Saffron.

Other future work includes use of other ATP systems, to determine the extent to which these results are dependent on the particular ATP systems used. Use of other ATP systems might result in different unique solutions, which would it possible to effectively use more CPUs for alternative processing options.

References

[1] N. Arhami. The Efficiency of Automated Theorem Proving by Translation to Less Expressive Logics. Master's thesis, Department of Computer Science, University of Miami, Miami, USA, 2014.

[2] C. Benzmüller, L. Paulson, F. Theiss, and A. Fietzke. LEO-II - A Cooperative Automatic Theorem Prover for Higher-Order Logic. In P. Baumgartner, A. Armando, and D. Gilles, editors, *Proceedings of the 4th International Joint Conference on Automated Reasoning*, number 5195 in Lecture Notes in Artificial Intelligence, pages 162–170. Springer-Verlag, 2008.

[3] C.E. Brown. Satallax: An Automated Higher-Order Prover (System Description). In B. Gramlich, D. Miller, and U. Sattler, editors, *Proceedings of the 6th International Joint Conference on Automated Reasoning*, number 7364 in Lecture Notes in Artificial Intelligence, pages 111–117, 2012.

[4] K. Claessen, A. Lilliestrom, and N. Smallbone. Sort It Out with Monotonicity - Translating between Many-Sorted and Unsorted First-Order Logic. In N. Bjorner and V. Sofronie-Stokkermans,

editors, *Proceedings of the 23rd International Conference on Automated Deduction*, number 6803 in Lecture Notes in Artificial Intelligence, pages 207–221. Springer-Verlag, 2011.

[5] S. Conchon, E. Contejean, J. Kanig, and S. Lescuyer. CC(X): Semantic Combination of Congruence Closure with Solvable Theories. In S. Krstic and A. Oliveras, editors, *Proceedings of the 5th International Workshop on Satisfiability Modulo Theories*, volume 198 of *Electronic Notes in Computer Science*, pages 51–69. Elsevier, 2008.

[6] N. Eén and N. Sörensson. An Extensible SAT-solver. In E. Giunchiglia and A. Tacchella, editors, *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing*, number 2919 in Lecture Notes in Computer Science, pages 502–518. Springer-Verlag, 2004.

[7] J-C. Filliatre and A Paskevich. Why3 - Where Programs Meet Provers. In M. Felleisen and P. Gardner, editors, *Proceedings of the 22nd European Symposium on Programming*, number 7792 in Lecture Notes in Computer Science, pages 125–128. Springer, 2013.

[8] K. Korovin. iProver - An Instantiation-Based Theorem Prover for First-order Logic (System Description). In P. Baumgartner, A. Armando, and D. Gilles, editors, *Proceedings of the 4th International Joint Conference on Automated Reasoning*, number 5195 in Lecture Notes in Artificial Intelligence, pages 292–298, 2008.

[9] D.W. Loveland. *Automated Theorem Proving : A Logical Basis*. Elsevier Science, 1978.

[10] W.W. McCune. Otter 3.3 Reference Manual. Technical Report ANL/MSC-TM-263, Argonne National Laboratory, Argonne, USA, 2003.

[11] R. Nieuwenhuis, editor. *Special Issue: The CADE ATP System Competition*, volume 15, 2002.

[12] T. Nipkow, L. Paulson, and M. Wenzel. Isabelle/HOL: A Proof Assistant for Higher-Order Logic. http://www.cl.cam.ac.uk/research/hvg/Isabelle/dist/Isabelle/doc/tutorial.pdf.

[13] A. Riazanov and A. Voronkov. The Design and Implementation of Vampire. *AI Communications*, 15(2-3):91–110, 2002.

[14] P. Rümmer. A Constraint Sequent Calculus for First-Order Logic with Linear Integer Arithmetic. In N. Bjorner and A. Voronkov, editors, *Proceedings of the 18th International Conference on Logic for Programming Artificial Intelligence and Reasoning*, number 7180 in Lecture Notes in Artificial Intelligence, pages 274–289. Springer-Verlag, 2012.

[15] M. Schneidner and G. Sutcliffe. Reasoning in the OWL 2 Full Ontology Language using First-Order Automated Theorem Proving. In N. Bjorner and V. Sofronie-Stokkermans, editors, *Proceedings of the 23rd International Conference on Automated Deduction*, number 6803 in Lecture Notes in Artificial Intelligence, pages 461–475. Springer-Verlag, 2011.

[16] S. Schulz. System Abstract: E 0.81. In M. Rusinowitch and D. Basin, editors, *Proceedings of the 2nd International Joint Conference on Automated Reasoning*, number 3097 in Lecture Notes in Artificial Intelligence, pages 223–228. Springer-Verlag, 2004.

[17] S. Schulz and G. Sutcliffe. System Description: GrAnDe 1.0. In A. Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction*, number 2392 in Lecture Notes in Artificial Intelligence, pages 280–284. Springer-Verlag, 2002.

[18] R. Shearer, B. Motik, and I. Horrocks. HermiT: A Highly-Efficient OWL Reasoner. In A. Ruttenberg, U. Sattler, and C. Dolbear, editors, *Proceedings of the 5th International Workshop on OWL: Experiences and Directions*, number 432 in CEUR Workshop Proceedings, 2008.

[19] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure. The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[20] G. Sutcliffe. The TPTP World - Infrastructure for Automated Reasoning. In E. Clarke and A. Voronkov, editors, *Proceedings of the 16th International Conference on Logic for Programming Artificial Intelligence and Reasoning*, number 6355 in Lecture Notes in Artificial Intelligence, pages 1–12. Springer-Verlag, 2010.