

Encoding First Order Proofs in SMT

Jeremy Bongio¹ Cyrus Katrak² Hai Lin³ Christopher Lynch⁴
Ralph Eric McGregor⁵

Mathematics and Computer Science Department, Clarkson University, Potsdam, New York, United States

Abstract

We present a method for encoding first order proofs in SMT. Our implementation, called ChewTPTP-SMT, transforms a set of first order clauses into a propositional encoding (modulo theories) of the existence of a rigid first order connection tableau and the satisfiability of unification constraints, which is then fed to Yices. For the unification constraints, terms are represented as recursive datatypes, and unification constraints are equations on terms. The finiteness of the tableau is encoded by linear real arithmetic inequalities.

We compare our implementation with our previous implementation ChewTPTP-SAT, encoding rigid connection tableau in SAT, and show that for Horn clauses many fewer propositional clauses are generated by ChewTPTP-SMT, and ChewTPTP-SMT is much faster than ChewTPTP-SAT. This is not the case for our non-Horn clause encoding. We explain this, and we conjecture a rule of thumb on when to use theories in encoding a problem.

Keywords: SMT, first-order, tableau, Yices

1 Introduction

Recent techniques in SAT solving have resulted in extremely fast procedures for solving propositional satisfiability problems[8], based on the DPLL method[4]. As an application of these techniques, we have developed an automated theorem prover called ChewTPTP-SAT[6], which encodes rigid first order theorem proving problems as SAT problems, and solves those SAT problems using Minisat[8].

Rigid unsatisfiability has been studied as early as [3,1]. A set of first order clauses is rigidly unsatisfiable if and only if there exists a closed rigid connection tableau for that set of clauses[10]. Our encoding uses this fact and solves the satisfiability of a set of rigid clauses by encoding the existence of a rigid connection tableau in SAT.

¹ Email:bonjiojp@clarkson.edu

² Email:katrakc@clarkson.edu

³ Email:linh@clarkson.edu

⁴ Email:clynch@clarkson.edu

⁵ Email:mgregre@clarkson.edu

A set of Horn clauses is encoded by creating propositional clauses representing the following requirements of a tableau T : (1) The root of the tableau must be a clause with only negative literals. (2) If a clause is in the tableau, then all its negative literals are in the tableau. (3) If a negative literal is in the tableau, then it must be extended by some clause. (4) If a negative literal $\neg A$ is extended by a clause C , then A must unify with the positive literal in C . (5) All unifications must be consistent with each other. (6) The tableau must be finite, i.e., there is no cycle.

For connection tableaux for non-Horn clauses, literals are either extended or complementary to an ancestor literal in its branch. For efficiency reasons, we choose to encode a clause tableau as a DAG. So there may be many branches from the root to a node. Therefore, we encode the fact that each literal L in the tableau must either be extended or all paths from the root to that node must contain a literal complementary to L . A tableau may have the same clause on different branches, and those branches may be closed with different literals. Therefore, we may have to add more instances of clauses to find a closed tableau. This cannot be avoided, since rigid Horn clause satisfiability is NP -complete, but rigid non-Horn clause satisfiability is Σ_2^P -complete[9]. However, because of the DAG structure, we can often encode many instances of a clause with just one instance.

Since we encode rigid proofs, the proof of unsatisfiability of a set of clauses may require repeating the encoding with fresh variants of each clause. However, there are also applications which really require rigid proofs[5].

Our original ChewTPTP-SAT implementation[6] performed well on some problems, but some of the encodings created huge sets of clauses. Some parts of our encoding represented choices made, such as which clause to extend each literal with. But other parts of our encoding represented deterministic procedures, such as deciding the consistency of unification constraints and deciding the acyclicity of the DAG, which verifies that a particular property holds of the DAG. Furthermore, in experimental results with Horn clauses, approximately 99% of the clauses generated were encoding the deterministic procedures, and only about 1% represented the choices. We had an eager encoding of unification and acyclicity. We decided the implementation would be more efficient if unification and acyclicity were encoded lazily and implemented these changes in ChewTPTP-SMT. It makes sense to express choices involved in building the tableau using SAT, and verification of unification and acyclicity using underlying theories. Therefore, we chose to encode our problem as Satisfiability modulo Theories[12], and we replaced Minisat[8] with Yices[7].

Yices has a theory for recursive datatypes, which can be used to represent terms. A term can be defined by using function symbols as constructors. Each function symbol of arity n is a constructor with n arguments. Constants are constructors with no arguments. Predicate symbols are viewed the same as function symbols. Variables are instances of terms. Then unification is represented as equality of terms. We represent acyclicity using linear arithmetic. Consider a graph $G = (V, E)$. If an edge (u, v) exists in E , then we assert an inequality $x_u < x_v$ for some real numbers x_u and x_v . Then G is acyclic if and only if the set of inequalities is consistent.

In this paper, we describe our implementation of ChewTPTP-SMT, and compare our results with ChewTPTP-SAT. We show that in the Horn encoding, ChewTPTP-

SMT produces far fewer clauses than ChewTPTP-SAT. The time needed to decide the satisfiability is also drastically reduced. This is not the case for non-Horn clauses. We explain why this is the case and give a rule of thumb for when theories should be used for encoding.

2 Clausal Tableau

See [2] for a detailed description of first order logic and a background discussion on the validity of a first order logic formula.

We use the following definition of tableau [10].

Definition 2.1 *Clausal tableaux are trees with nodes labeled with literals and branches labeled either open or closed. Clausal tableaux are inductively defined as follows. Let $S = \{C_1 \dots C_n\}$ be a set of clauses. If T is a tree consisting of a single unlabeled node N then T is a clausal tableau for S . The branch consisting of only the root node N is open. If N is a leaf node on an open branch B in the tableau T for S and one of the following inference rules are applied to T then the resulting tree is a clausal tableau for S .*

(Expansion rule) Let C_k be a clause in S . Replace each variable in C_k with a new variable not appearing in T . Suppose $L_{k1} \vee \dots \vee L_{ki}$ is the resulting clause. Construct a new tableau T' by adding i nodes as children of N and labeling them L_{k1} through L_{ki} . Label each of the i branches open. T' is a clausal tableau for S .

(Closure rule) Suppose L_{ij} is the literal at N and for some predecessor node with literal L_{pq} such that L_{ij} and $\neg L_{pq}$ are unifiable. Construct T' from T by applying the unifier to T and labeling the branch containing L_{ij} as closed. T' is a clausal tableau for S .

A clause which is added to the root node is called the *start clause* and we say that a clause is *in* a tableau if the clause was used in an application of the expansion rule.

Definition 2.2 *A clausal tableau is tightly connected if each clause (except the start clause) in the tableau contains some literal which is unifiable with the negation of its predecessor.*

Connected clausal tableaux use an additional rule called *extension rule*.

Definition 2.3 (Extension Rule) *Let N be a node in the tableau T and let C_k be a clause in S such that there exists a literal L_{ik} in C_k which is unifiable with the negation of N . Apply the expansion rule with C_k and immediately apply the closure rule with L_{ik} .*

Definition 2.4 *The calculus for connection tableaux consists of the expansion rule (for the start clause only), the closure rule, and the extension rule.*

We call a tableau closed if each leaf node has been closed by an application of the closure rule. By [11] we can require that the start clause is a negative clause since there exists a negative clause in any minimally unsatisfiable set.

2.1 Rigid Unsatisfiability

The main problem in Automated Theorem Proving is to determine if a set of hypotheses implies a conclusion, or equivalently that a formula F is unsatisfiable. We will assume that F is in CNF. The problem of rigid unsatisfiability of F is to determine whether there exists a ground instance of F which is unsatisfiable. A rigid tableau is a tableau in which multiple instances of a clause appearing in the tableau are identical copies of the clause appearing in F . One result of Tableau Theory is the completeness and soundness of closed connection tableaux.

Theorem 2.5 *There exists a closed connection (rigid) tableau for F iff F is (rigidly) unsatisfiable[10].*

3 Tableau Encoding

Our method to determine the rigid unsatisfiability of F generates a set S of propositional logic clauses modulo the theories of unification and arithmetic for F which encodes a rigid closed connection tableau for F and tests the satisfiability of S with a SMT solver.

We provide two encodings, the first for problems containing only Horn clauses and the second for those containing non-Horn clauses. Given F we enumerate each of the clauses in F and each of the literals in each clause. We denote clause i by C_i and denote the j^{th} literal in clause i by L_{ij} . We denote A_{ij} to be the atom of L_{ij} . Therefore L_{ij} is either of the form A_{ij} or $\neg A_{ij}$.

3.1 Encoding for Horn Clauses

Let F be a set of first order logic formulas.

We define a set of propositional variables c_m, l_{mn}, e_{mnq} , disjoint from the symbols in F , as follows: Define $c_m = T$ iff C_m appears in the tableau. Define $l_{mn} = T$ iff L_{mn} is an internal node in the tableau. Define $e_{mnq} = T$ iff C_q is an extension of L_{mn} . For each pair of clauses C_i and C_j we define $x_i < x_j = T$ (where x_i and x_j do not exist in F) iff there exists a path from C_i to C_j . For each pair of atoms A_i and A_j in F , we define $(A_i = A_j) = T$ iff A_i and A_j are the two atoms involved in an application of the closure rule.

Below we list the set of clauses that we generate and provide their meaning.

At least one clause containing only negative literals appears in the tableau:

$$(1) \quad \bigvee_{C_m \text{ is a negative clause}} c_m$$

If C_m appears in the tableau and L_{mn} is a negative literal then L_{mn} is an internal node in the tableau:

$$(2) \quad c_m \Rightarrow l_{mn}$$

If L_{mn} is an internal node in the tableau then for some q_j , C_{q_j} is an extension of L_{mn} :

$$(3) \quad l_{mn} \Rightarrow (e_{mnq_1} \vee \dots \vee e_{mnq_k})$$

where $\{C_{q_1} \dots C_{q_k}\}$ represent the set of all clauses whose positive literals are unifiable with L_{mn}

If C_q is an extension of L_{mn} then C_q exists in the tableau:

$$(4) \quad e_{mnq} \Rightarrow c_q$$

If C_q is an extension of L_{mn} and L_{qr} is the positive literal in C_q then A_{mn} and A_{qr} are unifiable:

$$(5) \quad e_{mnq} \Rightarrow (A_{mn} = A_{qr})$$

If C_q is an extension of L_{mn} then there is a path from C_m to C_q :

$$(6) \quad e_{mnq} \Rightarrow (x_m < x_q)$$

The encoding is satisfiable if and only if the original set of first order Horn clauses is rigidly unsatisfiable. We encode non-rigid unsatisfiability by continually adding new instances of each clause, renamed apart.

3.2 Encoding for Non-Horn Clauses

For non-Horn problems we use a different set of variables and generate a different set of clauses. Note: we say that two literals are *complementary* if they have opposite signs and their atoms are unifiable.

We define the variables, disjoint from the symbols in F , s_m , c_{mn} , l_{mn} , e_{mnqj} , o_{ijkl} and q_{mnij} as follows: Define $s_m = T$ iff C_m is the start clause. Define $c_{mn} = T$ iff C_m appears in the tableau and L_{mn} is complementary to its parent. Define $l_{mn} = T$ iff L_{mn} is a node in the tableau and is not a leaf node created by an application of the extension rule. Define $e_{mnqj} = T$ iff C_q is an extension of L_{mn} and L_{qj} is the complement of L_{mn} . Define $o_{ijkl} = T$ iff L_{ij} and L_{kl} are a pair of literals used in a closure but not by the extension rule. If a path to a node N contains the complement of N , then we say that the path is closed. Define $q_{mnij} = T$ iff L_{mn} is a leaf and L_{ij} is a node on a path from the root node to L_{mn} and every path from the root to L_{ij} contains a complement of L_{mn} . For each pair of clauses C_i and C_j we define $x_i < x_j = T$ (where x_i and x_j do not exist in F) iff there exists a path from C_i to C_j . For each pair of atoms A_i and A_j in F , we define $(A_i = A_j) = T$ iff A_i and A_j are the two atoms involved in an application of the closure rule.

The clauses are as follows.

There exists a start clause in the tableau which only contains negative literals:

$$(7) \quad \bigvee_{s_m \text{ is a negative clause}} s_m$$

If C_m is the start clause in the tableau then each literal L_{mn} of C_m is in the tableau:

$$(8) \quad s_m \Rightarrow l_{mn}$$

If C_i appears in the tableau and L_{ij} is the complement of a literal in its parent then all other literals of C_i are in the tableau:

$$(9) \quad c_{ij} \Rightarrow l_{ik} \text{ where } j \neq k$$

If L_{ij} exists in the tableau and is not a leaf node created by an application of the closure rule then either every branch ending at L_{ij} is closed or there is an extension

of L_{ij} :

$$(10) \quad l_{ij} \Rightarrow (q_{ijij} \vee (\bigvee_{k,l} e_{ijkl}))$$

If L_{ij} is extended with C_k then C_k is in the tableau and some L_{kl} of C_k is the complement of L_{ij} :

$$(11) \quad e_{ijkl} \Rightarrow c_{kl}$$

If clause C_m is an extension of L_{ij} and literals L_{ij} and L_{ml} are complements then A_{ij} and A_{ml} are unifiable.

$$(12) \quad e_{ijml} \Rightarrow (A_{ij} = A_{ml})$$

If L_{ij} and L_{kl} are a pair used in a closure then they must be unifiable:

$$(13) \quad o_{ijkl} \Rightarrow (A_{ij} = A_{kl})$$

If L_{ij} has the same sign as L_{kl} or their respective atoms are not unifiable then they are not complements:

$$(14) \quad \neg o_{ijkl} \text{ where } L_{ij} \text{ and } L_{kl} \text{ are not unifiable}$$

If every path through L_{kl} to leaf L_{ij} is closed and C_k is an extension of L_{mn} then either L_{ij} is a complement of L_{mn} or every path through L_{mn} to L_{ij} is closed:

$$(15) \quad q_{ijkl} \Rightarrow (e_{mnkp} \Rightarrow (o_{ijmn} \vee q_{ijmn}))$$

If C_k is an extension of L_{ij} then there is a path from clause C_i to clause C_k :

$$(16) \quad e_{ijkl} \Rightarrow (x_i < x_k)$$

If C_i is the start clause then there are no inferences into any of the literals in C_i :

$$(17) \quad s_i \Rightarrow \neg e_{kl ij}$$

If C_i is the start clause, L_{mn} is a leaf, and all paths that traverse L_{ij} to L_{mn} are closed, then L_{ij} and L_{mn} are complementary:

$$(18) \quad s_i \Rightarrow (q_{mni j} \Rightarrow o_{mni j})$$

We represent our tableau as a DAG, so there is some structure sharing. But even with the structure sharing, a non-Horn clause tableau may need more than one instance of the same clause. Rigid unsatisfiability could be determined by continually adding identical instances of a clause. Non-Horn encoding could also be extended to the non-rigid case in the same way as the Horn encoding.

4 Implementation and Experimental Results

We have implemented our tableau encoding in our theorem prover ChewTPTP-SMT, which is an extension of ChewTPTP-SAT[6]. In ChewTPTP-SAT, instead of using theories, we encoded the consistency of the unifiers and the acyclicity of the tableau with additional propositional clauses. To encode the consistency of the unifiers, we encoded the equations that would be created if a unification algorithm was run. We do not know ahead of time which unifiers we will have to create, so we encode everything that can possibly occur when the unification algorithm is run. To encode the absence of a cycle, we encode the existence of a path from one clause

Table 1
ChewTPTP Times For Horn Problems

| Name | SAT-M/Y | | SMT-Y | | SAT-M | SAT-Y | SMT-Y |
|----------------|---------|-----|--------|-----|-------|-------|-------|
| | Clause | Gen | Clause | Gen | Total | Total | Total |
| PUZ008-1.p | 1 | | 0 | | 1.06 | 0.89 | 0.11 |
| NLP106-1.p | 2 | | 0 | | 1.8 | 1.9 | 0.06 |
| NLP104-1.p | 2 | | 0 | | 1.82 | 1.9 | 0.05 |
| NLP105-1.p | 2 | | 0 | | 1.83 | 1.89 | 0.06 |
| NLP107-1.p | 2 | | 0 | | 2.47 | 1.99 | 0.06 |
| GRP033-3.p | 1 | | 0 | | 2.48 | 1.8 | 0.28 |
| NLP109-1.p | 1 | | 0 | | 2.49 | 1.99 | 0.05 |
| NLP113-1.p | 2 | | 0 | | 2.51 | 2.01 | 0.06 |
| NLP110-1.p | 2 | | 0 | | 2.74 | 1.84 | 0.07 |
| NLP112-1.p | 2 | | 0 | | 2.92 | 1.92 | 0.07 |
| NLP111-1.p | 1 | | 0 | | 2.94 | 1.93 | 0.06 |
| NLP108-1.p | 2 | | 0 | | 2.94 | 1.94 | 0.07 |
| PUZ036-1.005.p | 3 | | 0 | | 4.33 | 2.92 | 0.03 |
| RNG037-2.p | 4 | | 0 | | 5.33 | 5.35 | 6.2 |
| RNG038-2.p | 4 | | 0 | | 5.34 | 3.89 | 19.94 |
| RNG001-5.p | 4 | | 0 | | 6.93 | 5.32 | 0.84 |
| SWV015-1.p | 9 | | 0 | | 9.64 | 10.08 | 0.08 |
| SWV017-1.p | 11 | | 0 | | 10.82 | 11.27 | 0.1 |
| RNG006-2.p | 7 | | 0 | | 11.19 | 7.53 | 6.03 |

to another and the fact that there is no path from a clause to itself. This requires encoding all possible transitivity and irreflexivity axioms that may occur.

Our implementation allows the user to decide whether ChewTPTP encodes the problem as a SAT problem or an SMT problem. If the user chooses SMT, our implementation uses Yices to test the satisfiability of the encoding. If the user chooses SAT, then the user can also choose whether to test the satisfiability using Yices or Minisat, with a DIMACS encoding of SAT.

We tested our prover in all three settings on a subset of TPTP[13] problems. Tables 1-4 provide empirical data from these tests.

SMT-Y denotes our prover run in SMT mode, SAT-Y is SAT mode using Yices, and SAT-M is SAT mode using Minisat. For Horn clauses, we ran ChewTPTP on all the Horn problems in the TPTP database, but for non-Horn we only had time

Table 2
ChewTPTP Clause and Variable Count For Horn Problems

| Name | SAT-M/Y Cls Ct | SMT-Y Cls Ct | SAT-M/Y Var Ct | SMT-Y Var Ct | Result |
|----------------|-------------------|-----------------|-------------------|-----------------|--------|
| PUZ008-1.p | 52957 | 323 | 207608 | 216 | sat |
| NLP106-1.p | 130174 | 338 | 513774 | 392 | unsat |
| NLP104-1.p | 130724 | 344 | 515712 | 398 | unsat |
| NLP105-1.p | 130724 | 344 | 515712 | 398 | unsat |
| NLP107-1.p | 137380 | 315 | 542996 | 370 | unsat |
| GRP033-3.p | 115013 | 737 | 445065 | 383 | sat |
| NLP109-1.p | 137380 | 315 | 542996 | 370 | unsat |
| NLP113-1.p | 137897 | 319 | 544836 | 374 | unsat |
| NLP110-1.p | 128150 | 296 | 506951 | 350 | unsat |
| NLP112-1.p | 135667 | 287 | 537099 | 342 | unsat |
| NLP111-1.p | 135667 | 287 | 537099 | 342 | unsat |
| NLP108-1.p | 135667 | 287 | 537099 | 342 | unsat |
| PUZ036-1.005.p | 185292 | 45 | 729464 | 91 | unsat |
| RNG037-2.p | 221760 | 1524 | 876393 | 714 | sat |
| RNG038-2.p | 230063 | 1522 | 910786 | 718 | sat |
| RNG001-5.p | 258888 | 1527 | 1026821 | 725 | sat |
| SWV015-1.p | 559284 | 1047 | 2105121 | 532 | unsat |
| SWV017-1.p | 625119 | 1137 | 2354882 | 578 | unsat |
| RNG006-2.p | 432194 | 2058 | 1702459 | 925 | sat |

to run it through the GRP problems. We report all problems that both provers solved within five minutes but SAT-M took greater than one second. We believe the problems in these tables are representative of the overall results. Columns in the table show the running time of each method, the clause generation time rounded off to the nearest second, the number of clauses generated, and the number of variables generated for each method. We also show whether or not the problem is rigidly satisfiable. For these experiments, we only tested rigid satisfiability with one instance of each clause.

We wanted to see if working modulo theories would improve the performance of ChewTPTP. In the Horn case the running time was reduced significantly, except for a small percentage of exceptions. In the non-Horn case, working modulo theories often increased the running time. Generally, Yices was faster than Minisat on SAT

Table 3
ChewTPTP Times For Non-Horn Problems

| Name | SAT-M/Y | | SMT-Y | | SAT-M | SAT-Y | SMT-Y |
|----------------|---------|-----|--------|-----|-------|-------|-------|
| | Clause | Gen | Clause | Gen | Total | Total | Total |
| ANA025-2.p | 1 | | 0 | | 1.02 | 1.04 | 2.43 |
| COL121-2.p | 0 | | 1 | | 1.02 | 0.92 | 1.41 |
| ANA004-4.p | 1 | | 0 | | 1.33 | 1.87 | 2.77 |
| GRA001-1.p | 2 | | 2 | | 1.92 | 1.74 | 4.08 |
| ANA029-2.p | 2 | | 2 | | 2.05 | 2.08 | 4.68 |
| ANA005-2.p | 2 | | 1 | | 2.38 | 2.31 | 4.72 |
| ANA004-2.p | 2 | | 1 | | 2.39 | 2.3 | 5.06 |
| ANA003-2.p | 3 | | 1 | | 2.96 | 2.81 | 5.53 |
| GRP123-1.003.p | 3 | | 2 | | 3.41 | 3.76 | 18.11 |
| ANA001-1.p | 4 | | 2 | | 4 | 3.84 | 7.94 |
| GRP123-2.003.p | 4 | | 3 | | 5.55 | 5.37 | 17.66 |
| ANA002-2.p | 5 | | 3 | | 5.73 | 5.34 | 10.56 |
| ANA002-1.p | 5 | | 3 | | 6.17 | 5.67 | 11.84 |
| GRP124-2.004.p | 9 | | 6 | | 10.51 | 11.4 | 43.91 |
| GRP033-3.p | 15 | | 6 | | 20.11 | 15.69 | 23.18 |
| GRP123-3.003.p | 28 | | 20 | | 30.63 | 30.73 | 80.84 |
| ALG002-1.p | 1 | | 1 | | 43.51 | 64.92 | 75.33 |
| ANA004-5.p | 2 | | 1 | | 47.25 | 21.5 | 83.54 |
| GRP124-3.004.p | 46 | | 31 | | 88.23 | 83.83 | 171 |
| COM003-2.p | 82 | | 49 | | 88.72 | 84.54 | 168.1 |

problems without theories.

We believe we have an explanation for our results. In the Horn problems the number of clauses is reduced by an order of magnitude, whereas in the non-Horn problems the number of clauses is not reduced by much. This implies that working modulo theories is only useful when the clauses size is reduced significantly.

In the Horn encoding, everything can be encoded in $O(n^2)$ except for the encoding of unification and acyclicity, which require $O(n^3)$ space. When we remove the clauses used to represent unification and acyclicity, the number of clauses is now $O(n^2)$. However, for the encoding of non-Horn clauses, we must encode the fact of a leaf node having a complementary literal as an ancestor. This encoding is $O(n^3)$. We do not know how to encode this using the theories of Yices, so we have kept

Table 4
ChewTPTP Clause and Variable Count For Non-Horn Problems

| Name | SAT-M/Y Cls Ct | SMT-Y Cls Ct | SAT-M/Y Var Ct | SMT-Y Var Ct | Result |
|----------------|-------------------|-----------------|-------------------|-----------------|--------|
| ANA025-2.p | 41129 | 36020 | 2655 | 2286 | sat |
| COL121-2.p | 47725 | 20335 | 2322 | 1538 | sat |
| ANA004-4.p | 44142 | 36844 | 3160 | 2631 | sat |
| GRA001-1.p | 64222 | 60849 | 3292 | 3161 | sat |
| ANA029-2.p | 79860 | 66884 | 4107 | 3388 | sat |
| ANA005-2.p | 93806 | 68206 | 4907 | 3802 | unsat |
| ANA004-2.p | 93806 | 68206 | 4907 | 3802 | unsat |
| ANA003-2.p | 114945 | 78930 | 5654 | 4243 | unsat |
| GRP123-1.003.p | 111866 | 94335 | 4589 | 3596 | unsat |
| ANA001-1.p | 154246 | 113596 | 6680 | 5185 | unsat |
| GRP123-2.003.p | 180783 | 154243 | 6723 | 5450 | unsat |
| ANA002-2.p | 226149 | 151313 | 7457 | 5436 | unsat |
| ANA002-1.p | 229871 | 151313 | 7544 | 5437 | unsat |
| GRP124-2.004.p | 339070 | 283967 | 10854 | 8953 | unsat |
| GRP033-3.p | 699160 | 301901 | 15989 | 8961 | sat |
| GRP123-3.003.p | 1003831 | 934044 | 17763 | 15377 | unsat |
| ALG002-1.p | 54559 | 32731 | 3524 | 2460 | unsat |
| ANA004-5.p | 101166 | 44953 | 4981 | 3196 | unsat |
| GRP124-3.004.p | 1596801 | 1468732 | 25314 | 21981 | unsat |
| COM003-2.p | 2920669 | 2365922 | 46818 | 36051 | sat |

the propositional encoding. Therefore, when we remove the encoding of unification and acyclicity, the entire coding of the problem is still $O(n^3)$. We conjecture a good rule of thumb for deciding when it is useful to encode properties using theories. We conjecture that if the number of clause can be reduced by a factor of n , then the coding is useful, but if the asymptotic complexity remains the same, then it is not a good idea.

5 Conclusion

We have given an application of SMT to theorem proving in first order logic by encoding the existence of a rigid connection tableau in SMT. We have implemented the

SMT encoding in our theorem prover ChewTPTP-SMT. We compared it with our initial version of ChewTPTP-SAT, where a rigid connection tableau was encoded in SAT.

Compared to our encoding in SAT, the encoding in SMT is more natural and more efficient. As part of our encoding, we need to encode the solving of unification problems and the acyclicity of the tableau. In SAT, it was necessary to add cubically many clauses to encode the solving of unification. In addition, it was necessary to add cubically many clauses to encode the acyclicity of the tableau. However, when encoding this information in SMT, there was no need to encode the solving of unification, since this was accomplished directly with the Yices recursive datatype theory. The number of unification clauses was reduced from a cubic to a quadratic number. Similarly for acyclicity of tableau, we did not need to encode the transitivity and irreflexivity of the path relation. We only needed to express edges in the tableau as inequalities. The number of clauses to represent acyclicity also dropped from a cubic number to a quadratic number.

In the Horn encoding, all the other information in the tableau can also be encoded with a quadratic number of clauses. Therefore the entire encoding of the existence of a tableau dropped from a cubic number of clauses in SAT to a quadratic number in SMT. This drastically reduced the number of clauses, and simultaneously decreased the time needed to decide the satisfiability of the clauses. There was only a small reduction in number of clauses for non-Horn clauses, because we still need to encode the fact that all paths in the tableau can be closed. Therefore the entire encoding is still cubic, and the running time was actually worse. We conjecture a rule of thumb saying that it is worthwhile to use theories if the number of clauses is reduced by a factor of n , but not worthwhile if the asymptotic number remains the same.

For future work, we hope to be able to use SMT to further reduce the representation for non-Horn clauses, ideally cutting it down to a quadratic number of clauses. It would be possible to define a theory to do this directly, but we have not yet figured out how to do it with the existing theories in Yices. In addition, in order to prove the general first order problem we also need to find a good way to decide exactly which clauses should be copied. We would like a method to decide satisfiability from rigid satisfiability. It would be useful to have an encoding of rigid clauses modulo a non-rigid theory, as discussed in [5]. This way, we could immediately identify some clauses as non-rigid, and work modulo those clauses.

This paper shows the usefulness of SMT to theorem proving in first order logic. We suspect there are other logics which could also be solved efficiently using SMT.

Acknowledgement

We would like to thank to Leonardo de Moura for his explanation of how to express unification problems in Yices using recursive datatypes.

References

- [1] Andrews P. B. [1981], Theorem Proving via General Matings, *Journal of the Association for Computing Machinery*, Vol. 28, No. 2, pp.193-214
- [2] Bell J.L. and Slomson A.B. [1969], *Models and Ultraproducts, An Introduction*, Dover
- [3] Chang, C. and Lee, C.R. [1973], *Symbolic Logic and Mechanical Theorem Proving*. Academic Press New York and London.
- [4] Davis M., Logemann D. and Loveland D. [1962], A Machine Program For Theorem Proving, *Communications of the ACM*, Volume 5, Issue 7, pp. 394-397
- [5] Delaune S., Lin H. and Lynch C. [2007], Protocol Verification Via Rigid/Flexible Resolution, submitted
- [6] Deshane T., Hu W., Jablonski P., Lin H., Lynch C. and McGregor R.E. [2007], CADE, *Lecture Notes in Computer Science*, Springer, Vol. 4603, pp. 476-491
- [7] Dutertre B. and deMoura L., Yices, <http://yices.csl.sri.com>
- [8] Eén N. and Sörensson N. [2003], An Extensible Sat-Solver, In *SAT*, pp. 502-518
- [9] Goubault J. [1994], The Complexity of Resource-Bounded First-Order Classical Logic, *Lecture Notes In Computer Science, Proceedings of the 11th Annual Symposium on Theoretical Aspects of Computer Science*, Vol. 775, Springer-Verlag, pp. 59-70
- [10] Hähnle R. [2001], Tableaux and Related Methods, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. 1, Elsevier Science, chapter 3, pp. 101-177
- [11] Letz R. and Gernot S. [2001], Model Elimination and Connection Tableau Procedures, in A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. 2, Elsevier Science, chapter 28, pp. 2015-2113
- [12] Nieuwenhuis R., Oliveras A. and Tinelli C. [2006], Solving SAT and SAT Modulo Theories: From an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T), *Journal of the ACM*, 53(6), 937-977, November 2006.
- [13] Sutcliffe G. and Suttner C.B. [1998], The TPTP Problem Library: CNF Release v1.2.1, *Journal of Automated Reasoning*, Vol. 21, No. 2, pp. 177-203