Gen2sat: an Automated Tool for Deciding Derivability in Analytic Pure Sequent Calculi*

Yoni Zohar[1] and Anna Zamansky[2]

[1] Tel Aviv University, Israel
[2] Haifa University, Israel
yoni.zohar@cs.tau.ac.il annazam@is.haifa.ac.il

Abstract. Gen2sat [1] is an efficient and generic tool that can decide derivability for a wide variety of propositional non-classical logics given in terms of a sequent calculus. It contributes to the line of research on computer-supported tools for investigation of logics in the spirit of the "logic engineering" paradigm. Its generality and efficiency are made possible by a reduction of derivability in analytic pure sequent calculi to SAT. This also makes Gen2sat a "plug-and-play" tool so it is compatible with any standard off-the-shelf SAT solver and does not require any additional logic-specific resources. We describe the implementation details of Gen2sat and an evaluation of its performance, as well as a pilot study for using it in a "hands on" assignment for teaching the concept of sequent calculi in a logic class for engineering practitioners.

1 Introduction

Logic Engineering [2] is a quickly developing field which studies ways to investigate and construct new logical formalisms with "nice" properties (such as decidability, appropriate expressive power, effective reasoning methods, etc.), for a particular need or application. Handling whole families (or "product lines") of non-classical logics calls for automatic methods for their construction and investigation, as well as for new approaches in teaching these topics to future logicians and practitioners. Such ideas of automatic support for the investigation of logics, first presented in [21], were realized in a variety of tools, such as MultLog [5], TINC [8], NESCOND [22], LoTREC [13], MetTeL [24], and many others.

The tool Gen2sat is a contribution to the above paradigm, which particularly aims to support investigators who use *sequent calculi* for the specification of logics. Sequent calculi are a prominent proof-theoretic framework, suitable for performing proof search in a wide variety of different logics. However, a great deal of ingenuity is required for developing efficient proof-search algorithms for sequent calculi (see, e.g., [12]). Aiming to support users with minimal background in programming and automated reasoning techniques, Gen2sat uses a *uniform* method for deciding derivability of a sequent in a given calculus using

* This research was supported by The Israel Science Foundation (grant no. 817-15).

the polynomial reduction of [16] to SAT. Shifting the intricacies of implementation and heuristic considerations to the realm of off-the-shelf SAT solvers, the tool is lightweight and focuses solely on the transformation of derivability to a SAT instance. As such, it also has the potential to serve as a tool that can enhance learning and research of concepts related to proof theory and semantics of non-classical logics, in particular those of sequent calculi. To demonstrate the educational potential of the tool, in this paper we report on a pilot on using it to enhance learning sequent calculi by graduate Information Systems students at the University of Haifa.

2 Analytic Pure Sequent Calculi

While this paper's focus is on the implementation and usage of Gen2sat, the theoretical background can be found in [16]. Below we briefly review the relevant results.

The variety of sequent calculi which can be handled by Gen2sat includes the family of *pure analytic calculi*. A derivation rule is called *pure* if it does not enforce any limitations on the context formulas. For example, the right introduction rule of implication in classical logic is pure, but not in intuitionistic logic, where only left context formulas are allowed. A sequent calculus is called pure if it includes all the standard structural rules:[3] weakening, identity and cut; and all its derivation rules are pure.

For a finite set $\circledcirc$ of unary connectives, we say that a formula φ is a $\circledcirc$-subformula of a formula ψ if either φ is a subformula of ψ, or $\varphi = \circ\psi'$ for some $\circ \in \circledcirc$ and proper subformula ψ' of ψ. A pure calculus is $\circledcirc$-*analytic* if whenever a sequent s is provable in it, s can be proven using only formulas from $sub^{\circledcirc}(s)$, the set of $\circledcirc$-subformulas of s. We call a calculus *analytic* if it is $\circledcirc$-analytic for some set $\circledcirc$. Note that $\emptyset$-analyticity amounts to the usual subformula property. Many well-known logics can be represented by analytic pure sequent calculi, including three and four-valued logics, various paraconsistent logics, and extensions of primal infon logic ([16] presents some examples).

In order to decide derivability in sequent calculi, Gen2sat adopts the following semantic view:

Definition 1. Let $\mathbf{G}$ be an analytic pure sequent calculus. A $\mathbf{G}$-*legal bivaluation* is a function v from some set of formulas to $\{0, 1\}$ that respects each rule of $\mathbf{G}$, that is, for every instance of a rule, if v assigns 1 to all premises, it also assigns 1 to the conclusion. This definition relies on the following extension of bivaluations to sequents: $v(\Gamma \Rightarrow \Delta) = 1$ iff $v(\psi) = 0$ for some $\psi \in \Gamma$ or $v(\psi) = 1$ for some $\psi \in \Delta$.

Theorem 1 ([16]). *Let $\circledcirc$ be a set of unary connectives, $\mathbf{G}$ a $\circledcirc$-analytic pure sequent calculus, and s a sequent. s is provable in $\mathbf{G}$ if and only if there is no $\mathbf{G}$-legal bivaluation v with domain $sub^{\circledcirc}(s)$ such that $v(s) = 0$.*

[3] We take sequents to be pairs of *sets* of formulas, and therefore exchange and contraction are built in.

Thus, given a $\circledcirc$-analytic calculus **G** and a sequent s as its input, Gen2sat does not search for a proof. Instead, it searches for a countermodel of the sequent, by encoding in a SAT instance the following properties of the countermodel: *1*) Assigning 0 to s; and *2*) Being **G**-legal with domain $sub^{\circledcirc}(s)$.

Gen2sat is capable of handling also impure rules of the form $(*i)\ \frac{\Gamma \Rightarrow \Delta}{*\Gamma \Rightarrow *\Delta}$ for *Next*-operators. This requires some adaptations of the above reduction, that are described in [16]. $(*i)$ is the usual rule for *Next* in LTL (see, e.g., [15]). It is also used as $\Box$ (and $\Diamond$) in the modal logic $KD!$ of functional Kripke frames (also known as KF and $KDalt1$). In primal infon logic [10] *Next* operators play the role of quotations.

3 Features and Usage

There is a variety of tools developed in the spirit of logic engineering, such as MultLog [5], TINC [8], NESCOND [22], LoTREC [13], and finally MetTeL [24], which generates a theorem prover for a given logic, as well as a source code for the prover, that can be further optimized. The aim of Gen2sat is similar, allowing the user to specify the logic and automatically obtain a decision procedure. In contrast to MetTeL which uses tableaux, in Gen2sat the logic is given by a sequent calculus. Moreover, the core of Gen2sat is a reduction to SAT, thus it leaves the "hard work" and heuristic considerations of optimizations to state of the art SAT solvers, allowing the user to focus solely on the *logical* considerations.

Gen2sat can be run both via a web interface and from the command line. In the web-based version the user fills in a form; in the command line a property file is passed as an argument. From the command line, Gen2sat is called by: `java -jar gen2sat.jar <path>`. The form has the following fields:

Connectives A comma separated list of connectives, each specified by its symbol and arity, separated by a colon.

***Next* operators** A comma separated list of the symbols for the next operators.

Rules Each rule is specified in a separate line that starts with "rule:". The rule itself has two parts separated by "/": the premises, which is a semicolon separated list of sequents, and the conclusion, which is a sequent.

Analyticity For the usual subformula property this field is left empty. For other forms of analyticity, it contains a comma separated list of unary connectives.

Input sequent The sequent whose derivability should be decided.

The web-based version includes predefined forms for some propositional logics (e.g. classical logic, primal infon logic and more). In addition, it allows the user to import sequent calculi from *Paralyzer*.[4]

If the sequent is unprovable, Gen2sat outputs a countermodel. If it is provable, Gen2sat recovers a sub-calculus in which the sequent is already provable (naturally, the full proof is unobtainable due to the semantic approach

[4] Paralyzer is a tool that transforms Hilbert calculi of a certain general form into equivalent analytic sequent calculi. It was described in [7] and can be found at `http://www.logic.at/people/lara/paralyzer.html`.

```
Input file
connectives: P:2, E:2
rule: =>a; =>b / =>aPb
rule: a=> / aPb=>
rule: b=> / aPb=>
rule: =>a; =>b / =>aEb
rule: =>b; a=> / aEb=>
analyticity:
inputSequent: (((m1 P m2 ) E k) E k),k=>m1



Output
provable
There's a proof that uses only these rules:
[=>b; a=> / a E b=>, a=> / a P b=>]
```

Fig. 1. A provable instance

```
Input file
connectives: AND:2,OR:2,IMPLIES:2,TOP:0
nextOperators: q1 said, q2 said, q3 said
rule: =>p1; =>p2 / =>p1 AND p2
rule: p1,p2=> / p1 AND p2=>
rule: =>p1,p2 / =>p1 OR p2
rule: =>p2 / =>p1 IMPLIES p2
rule: =>p1; p2=> / p1 IMPLIES p2=>
rule: / => TOP
analyticity:
inputSequent: =>q1said (p IMPLIES p)

Output
unprovable
Countermodel:
q1said p=false, q1said(p IMPLIES p)=false
```

Fig. 2. An unprovable instance

of Gen2sat). Thus, for a provable sequent Gen2sat outputs a subset of rules that suffice to prove the sequent.

Figures 1 and 2 present examples for the usage of Gen2sat. In Figure 1, the input contains a sequent calculus for the Dolev-Yao intruder model [9]. The connectives E and P correspond to encryption and pairing. The sequent is provable, meaning that given two messages m_1 and m_2 that are paired and encrypted twice with k, the intruder can discover m_1 if it knows k. The output also contains the only two rules that are needed in order to prove the sequent. In Figure 2, the input file contains a sequent calculus for primal infon logic, where the implication connective is not reflexive, and hence the input sequent is unprovable. Note that the rules for the next operators are fixed, and therefore they are not included in the input file. Both calculi are $\emptyset$-analytic, and hence the analyticity field is left empty. Gen2sat supports analyticity w.r.t. any number of unary connectives, and hence this field may include a list of unary connectives.

4 Implementation Details

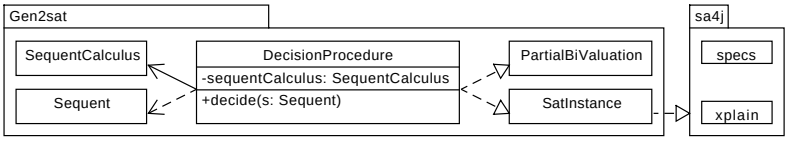

Fig. 3. A partial class diagram of Gen2sat

Gen2sat is implemented in Java and uses sat4j [17] as its underlying SAT solver. Since the algorithm from [16] is a "one-shot" reduction to SAT, no changes are needed in the SAT solver itself. In particular, sat4j can be easily replaced by other available solvers. Figure 3 includes a partial class diagram of Gen2sat. The two main modules of sat4j that we use are `specs`, which provides the solver itself, and `xplain`, which searches for an unsat core. The main class of Gen2sat is `DecisionProcedure`, that is instantiated with a specific `SequentCalculus`. Its main method `decide` checks whether the input sequent is provable. Given a `Sequent` s, `decide` generates a `SatInstance` stating that s has a countermodel,

by applying the rules of the calculus on the relevant formulas, as described above. `SatInstance` is the only class that uses sat4j directly, and thus it is the only class that will change if another SAT solver is used.

For satisfiable instances, the `specs` module returns a satisfying assignment, which is directly translated to a countermodel in the form of a `PartialBivaluation`. For unsatisfiable instances, the `xplain` module generates a subset of clauses that is itself unsatisfiable. Tracking back to the rules that induced these clauses, we are able recover a smaller sequent calculus in which s is already provable. Note however, that the smaller calculus need not be analytic, and then the correctness, that relies on Theorem 1 might fail. Nevertheless, correctness is preserved in this case, as the "if" part of Theorem 1 holds even for non-analytic calculi. Thus, although Gen2sat does not provide a proof of the sequent, we do obtain useful information about the rules that were used in it.

5 Performance

Gen2sat can be used in applications based on reasoning in non-classical logics, and especially paraconsistent logics [6], as there exist analytic pure sequent calculi for many of them [4]. For evaluating its performance, we have considered the well-known paraconsistent logic C_1 [11], using the $\{\neg\}$-analytic calculus from [4], and compared its running time with KEMS [18,19,20], a theorem prover that implements several KE tableau calculi for classical logic and paraconsistent logics.[5] For this comparison, a lighter version of Gen2sat was compiled, called Gen2sat$_m$, that only decides whether the input sequent is provable, without providing a countermodel or a smaller calculus. The experiments were made on a dedicated Linux machine with four dual-core 2.53 Ghz AMD Opteron 285 processors and 8GB RAM. We have used the problem families of *provable* sequents for benchmarking paraconsistent logics provers from [18,20], and extended them by creating similar problem families of *unprovable* sequents. For example, problems of the first family from [18] have the form:

$$\Phi_n^1 = \bigwedge_{i=1}^{n}(\neg A_i), \bigwedge_{i=1}^{n}(\circ A_i \rightarrow A_i), (\bigvee_{i=1}^{n} \circ A_i) \vee (\neg A_n \rightarrow C) \Rightarrow C$$

where $\circ$ is defined by $\circ A = \neg(A \wedge \neg A)$. Dismissing the first conjunct of the first formula leads to an unprovable sequent. Similarly, problems of the fourth family have the form:

$$\Phi_n^4 = \bigwedge_{i=1}^{n} A_i, \bigwedge_{j=1}^{n}((A_j \vee B_j) \rightarrow \circ A_{j+1}), (\bigwedge_{k=2}^{n} \circ A_k) \rightarrow A_{n+1} \Rightarrow \neg\neg A_{n+1}$$

Replacing $\neg\neg A_{n+1}$ with $\neg A_{n+1}$ leads to an unprovable sequent.

Table 1 includes the running times in miliseconds for the first and fourth problem families.[6] Similar results were obtained on the other families.

[5] The tableau calculus for C_1 in KEMS can be translated to a sequent calculus (the connection between the two frameworks was discussed e.g. in [3]). However, the translated sequent calculus is non-analytic, and thus cannot be used with Gen2sat.

[6] Out of 11 possible formula comparator choices of KEMS, Table 1 presents the results for the best performing one in each problem.

In our tests, KEMS performed better on smaller inputs, while Gen2sat performed better on larger ones. Also, provable sequents are easier than unprovable ones for KEMS (which searches for a proof), while the opposite holds for Gen2sat (which searches for a countermodel). For provable sequents, Gen2sat$_m$ performs much faster than the full version, as it does not call the `xplain` module of sat4j. In contrast, for unprovable sequents, the difference between the two versions is negligible. The table also contains the number of variables and clauses in the SAT instances that were generated by Gen2sat. In the case of C_1, the set of variables corresponds to the set of subformulas of the sequent and their negations.

	provable					unprovable				
	KEMS	Gen2sat	Gen2sat$_m$	#vars	#clauses	KEMS	Gen2sat	Gen2sat$_m$	#vars	#clauses
Φ^1_{10}	133	342	213	137	344	153	224	215	135	339
Φ^1_{20}	675	252	73	277	694	686	70	70	275	689
Φ^1_{50}	13934	747	143	697	1744	14247	146	159	695	1739
Φ^1_{80}	75578	1393	148	1117	2794	78212	175	203	1115	2789
Φ^1_{100}	175716	2178	235	1397	3494	182904	226	284	1395	3489
Φ^4_{10}	124	291	212	173	410	205	220	219	173	410
Φ^4_{20}	502	207	78	353	840	1416	83	78	353	840
Φ^4_{50}	8723	444	158	893	2130	36282	137	159	893	2130
Φ^4_{80}	45130	744	178	1433	3420	226422	194	190	1433	3420
Φ^4_{100}	123619	908	220	1793	4280	661078	227	227	1793	4280

Table 1. Benchmark results for provable and unprovable sequents in C_1

6 Gen2sat for Education: a Pilot

There is an ongoing debate on the appropriate way of teaching logic and formal methods to future software engineering practitioners, in particular on ways to *bridge* between the taught material and the software domain (see, e.g., [23]). As a contribution to the latter question, we have initiated a pilot of integrating a "hands-on" assignment based on Gen2sat into a logic course for Information Systems graduate students[7], exploring its potential to enhance learning of the concept of sequent calculi. The assignment aimed to allow them to experiment with different sequent calculi, discovering "a whole new world" of non-classical logics. To increase their engagement, the assignment had the "look and feel" of a software engineering assignment whose domain is non-classical logics.

After a two hour lecture on sequent calculi and the system LK, we introduced Gen2sat in class and explained its functionality and features. The students were then requested to play the role of *testers* of the tool. More concretely, they were requested to provide a test plan (as small as possible) which would cover all possible scenarios the tool could encounter. For a quantifiable measure for success

[7] The second author has been teaching the course for several years at the University of Haifa; see [25] for further details on the course design.

we used a standard approach of measuring code coverage, instructing them to install the Eclemma plug-in for Eclipse [14] for determining the percentage of code activated for a given input. Thus, basically the students' assignment was producing a minimal test plan that would achieve maximal code coverage. When analyzing different inputs to the tool, the students would potentially gain insights into the wide variety of non-classical logics defined in terms of sequent calculi.

The results of our pilot were encouraging: of eight students who participated in the assignment, all ended up submitting[8] test plans which achieved between 70% - 85% coverage, and included non-trivial sequent calculi for different languages. An anonymous feedback questionnaire showed that the students found the assignment helpful for understanding sequent calculi, as well as engaging and fun. This seems to us an indication of the potential of integrating "hands on" assignments in the spirit of logic engineering in illuminating educational logical content through experimenting with software tools.

7 Conclusions and Future Work

We have introduced Gen2sat, an efficient tool that decides derivability for a wide family of non-classical logics via the reduction to SAT given in [16]. In the spirit of the "logic engineering" paradigm, Gen2sat is *generic*: it receives as input the language and rules of a sequent calculus of a very general form. In addition, Gen2sat works on top of standard off-the-shelf SAT solvers, without requiring any additional logic-specific resources. Our preliminary experimental results show that the generality of Gen2sat does not come at the expence of its performance, making it appropriate for practical automated reasoning in non-classical logics. We plan to extend these experiments to include more provers for logics with analytic pure sequent calculi.

As a result of its semantic approach, Gen2sat currently does not provide actual proofs of provable sequents. This can be overcome by integrating Gen2sat with other existing theorem provers so that for unprovable sequents, the theorem prover will not have to search for a proof, while for provable sequents, the search space can be potentially reduced by exploiting gensat's capability of supplying a sufficient subset of rules. Our experience with teaching the concept of sequent calculi in a "hands-on" assignment on test design for Gen2sat shows its potential in educational settings. Another direction for further research is developing an educational version of the tool, in which learning concepts from non-classical logics could be achieved via interacting with the software.

References

1. Gen2sat website. `http://www.cs.tau.ac.il/research/yoni.zohar/gen2sat`.

[8] Interestingly, seven students employed new connectives with arity greater than 2 and three employed also 0-ary connectives (which indeed increased coverage), although they have not seen any such example in class.

2. C. E. Areces. *Logic Engineering: The Case of Description and Hybrid Logics*. Institute for Logic, Language and Computation, 2000.
3. A. Avron. Gentzen-type systems, resolution and tableaux. *Journal of Automated Reasoning*, 10(2):265–281.
4. A. Avron, B. Konikowska, and A. Zamansky. Efficient reasoning with inconsistent information using c-systems. *Information Sciences*, 296:219 – 236, 2015.
5. M. Baaz, C. G. Fermüller, G. Salzer, and R. Zach. Multlog 1.0: Towards an expert system for many-valued logics. In *Automated DeductionCade-13*, pages 226–230. Springer, 1996.
6. W. Carnielli, M. E. Coniglio, and J. Marcos. Logics of formal inconsistency. In *Handbook of philosophical logic*, pages 1–93. Springer, 2007.
7. A. Ciabattoni, O. Lahav, L. Spendier, and A. Zamansky. Automated support for the investigation of paraconsistent and other logics. In S. Artemov and A. Nerode, editors, *Logical Foundations of Computer Science*, volume 7734 of *Lecture Notes in Computer Science*, pages 119–133. Springer Berlin Heidelberg, 2013.
8. A. Ciabattoni and L. Spendier. Tools for the investigation of substructural and paraconsistent logics. In *Logics in Artificial Intelligence*, pages 18–32. Springer, 2014.
9. H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *Logic in Computer Science, 2003. Proceedings. 18th Annual IEEE Symposium on*, pages 271–280, June 2003.
10. C. Cotrini and Y. Gurevich. Basic primal infon logic. *Journal of Logic and Computation*, 2013.
11. N. C. Da Costa. *Sistemas formais inconsistentes*, volume 3. Editora UFPR, 1993.
12. A. Degtyarev and A. Voronkov. The inverse method. *Handbook of Automated Reasoning*, 1:179–272, 2001.
13. O. Gasquet, A. Herzig, D. Longin, and M. Sahade. *Automated Reasoning with Analytic Tableaux and Related Methods: 14th International Conference, TABLEAUX 2005, Koblenz, Germany, September 14-17, 2005. Proceedings*, chapter LoTREC: Logical Tableaux Research Engineering Companion, pages 318–322. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.
14. M. Hoffmann and G. Iachelini. Code coverage analysis for eclipse. *Eclipse Summit Europe*, 2007.
15. H. Kawai. Sequential calculus for a first order infinitary temporal logic. *Mathematical Logic Quarterly*, 33(5):423–432, 1987.
16. O. Lahav and Y. Zohar. Sat-based decision procedure for analytic pure sequent calculi. In S. Demri, D. Kapur, and C. Weidenbach, editors, *Automated Reasoning*, volume 8562 of *Lecture Notes in Computer Science*, pages 76–90. Springer International Publishing, 2014.
17. D. Le Berre and A. Parrain. The sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64, 2010.
18. A. Neto and M. Finger. Effective prover for minimal inconsistency logic. In M. Bramer, editor, *Artificial Intelligence in Theory and Practice*, volume 217 of *IFIP International Federation for Information Processing*, pages 465–474. Springer US, 2006.
19. A. Neto and M. Finger. Kems-a multi-strategy tableau prover. *Proceedings of the VI Best MSc Dissertation/PhD Thesis Contest (CTDIA 2008), Salvador*, 2008.
20. A. Neto, C. A. A. Kaestner, and M. Finger. Towards an efficient prover for the paraconsistent logic C1. *Electronic Notes in Theoretical Computer Science*, 256:87–102, 12 2009.

21. H. J. Ohlbach. Computer support for the development and investigation of logics. *Logic Journal of IGPL*, 4(1):109–127, 1996.
22. N. Olivetti and G. Pozzato. Nescond: An implementation of nested sequent calculi for conditional logics. In S. Demri, D. Kapur, and C. Weidenbach, editors, *Automated Reasoning*, volume 8562 of *Lecture Notes in Computer Science*, pages 511–518. Springer International Publishing, 2014.
23. R. L. Page. Software is discrete mathematics. In *ACM SIGPLAN Notices*, volume 38, pages 79–86. ACM, 2003.
24. D. Tishkovsky, R. A. Schmidt, and M. Khodadadi. Mettel2: Towards a tableau prover generation platform. In *PAAR@ IJCAR*, pages 149–162, 2012.
25. A. Zamansky and E. Farchi. Teaching Logic to Information Systems Students: Challenges and opportunities. In *Fourth International Conference on Tools for Teaching Logic, TTL*, 2015.