Automated Reasoning in Generating Exam Sheets for Automated Deduction

Petra Hozzová, Laura Kovács, and Jakob Rath

TU Wien, Austria

Abstract. Amid the COVID-19 pandemic, distance teaching became default in world-wide higher education, urging teachers and researchers to revise course materials into a more accessible online content for a diverse audience. Probably one of the hardest challenges came with online assessments of course performance, for example by organizing online written exams. In this short paper we describe the online setting we organized for our master's level course "Automated Deduction" in logic and computation at TU Wien. The algorithmic and rigorous reasoning developed within our course calls for exam sheets focused on problem solving and deductive proofs; as such exam sheets using test grids are not a viable solution for written exams within our course. We report on the automated reasoning framework we developed for generating individual online exam sheets for students enrolled in the course. We believe the toolchain of automated reasoning tools we have developed for holding online written exams could be beneficial not only for other distance learning platforms, but also to researchers in automated reasoning, by providing our community with a large set of randomly generated benchmarks in SAT/SMT solving and first-order theorem proving.

1 Motivation

Amid the COVID-19 pandemic, higher education has moved to distance teaching. While online lecturing was relatively fast to implement via webinars, recording, streaming and online communication channels, coming up with best practices to assess course performances was far from trivial. Even with very sophisticated technical infrastructure, which on the other hand is unethical to aspect from course participants, avoiding collaborative course work in a virtual environment is very hard to achieve, if possible at all. In addition, course assessment is very diverse and depends on the available resources institutions have to implement individual oral exams or large-scale written exams.

In this short paper we report on our solution for organizing online written exams, where solutions to written exams require rigorous

logical reasoning and proofs rather than using mechanized test grids. In particular, we were faced with the challenge of organizing online written exams for our master's level course "Automated Deduction" in logic and computation at TU Wien[1]. This course introduces algorithmic techniques and fundamental results in automated reasoning, by focusing on specialised algorithms for reasoning in various fragments of first-order logics, such as propositional logic, combination of ground theories, and full first-order logic with equality. As such, topics of the course cover from theoretical and practical aspects of SAT/SMT solving [**?**,**?**] and first-order theorem proving using superposition reasoning [**?**,**?**,**?**]

We claim by no means that the framework we developed for online examinaion is optimal. is actually very specific to the logo Given the time constraints of examination periods, we aimed for an online exam setting that (i) reduces collaborative work but (ii) requires the same workload on each participants. The algorithmic and rigorous reasoning developed within our course called for exam sheets focused on problem solving and deductive proofs; as such exam sheets using test grids are not a viable solution for written exams within our course. We have therefore used and adapted the automated reasoning approaches introduced in our course to automate the generation of individual exam sheets for students enrolled in our course, by making sure that the exam tasks remain the same in each exam sheet. As such

- SAT
- SMT
- FO

While our proposal is very specific to the formal aspects of automated reasonig, we believe our framework can be extended with further constraints to scale it to other courses in formal methods.

We believe the toolchain of automated reasoning tools we have developed for holding online written exams could be beneficial not only for other distance learning platforms, but also to researchers in automated reasoning, by providing our community with a large set of randomly generated benchmarks in SAT/SMT solving and first-order theorem proving.

[1] https://tiss.tuwien.ac.at/course/courseDetails.xhtml?dswid=2002&dsrid=601&courseNr=184774&semester=2020S

2 todo

Challenge: problem instances should be different but of similar difficulty to make sure the exam is fair to students.

3 Method 1: Varying Templates and Fixed Patterns

3.1 Satisfiability Modulo Theories (SMT)

relatively easy: provided template, do small random perturbations that don't change the solution

3.2 Ground Superposition

4 Method 2: Full Random Generation

more sophisticated: full random generation, filter out "too hard"/"too easy" instances. Note that we don't need very efficient implementation of filters since the instances are very small. so we can use naive satisfiability tests or model counting.

4.1 Boolean Satisfiability (SAT)

Example 1. Consider the formula:

$$(r \wedge \neg(q \rightarrow p)) \vee (q \leftrightarrow \neg(p \rightarrow q))$$

(a) Which atoms are pure in the above formula?
(b) Compute a clausal normal form C of the above formula by applying the CNF transformation algorithm with naming and optimization based on polarities of subformulas;
(c) Decide the satisfiability of the computed CNF formula C by applying the DPLL method to C. If C is satisfiable, give an interpretation which satisfies it. □

Simply generating propositional formulas fully randomly would lead to a huge variety of formulas, spanning both formulas for which the above questions are trivial to answer and others where much more

we could also give some more examples (5?) if we have space (just the formula)

work is required. This is obviously undesirable in an exam setting, where the tasks should ultimately be challenging, but still solvable by hand. Furthermore, the variation in difficulty between different exams should be kept as small as possible, to make the setting as fair to the examinees as possible.

To this end, we identified several characteristics that the exam formulas should exhibit, and filtered the generated formulas by these.

The criteria are the following:

1. The formula contains exactly seven connectives.
2. The formula contains exactly three different atomic propositions (but each of these may appear multiple times).
3. There exists at least one atom that appears with pure polarity (i.e., either only in positive position or only in negative position).
4. There is no subformula for the form $\neg\neg\varphi$ for any formula φ.
5. The connective "$\leftrightarrow$" appears at least once but at most twice.
6. The connectives "$\rightarrow$" and "$\neg$" appear at least once.
7. At least one of the connectives "$\wedge$" or "$\vee$" appears in the formula.
8. If a binary connective has a literal as argument, the other argument cannot also be a literal containing the same atomic proposition. For example, this excludes subformulas such as $p \wedge \neg p$ and $p \vee p$, but not $p \rightarrow q$.
9. The polarity-optimized clausal normal form (also known as Tseitin transformation) to be found in subtask (b) results in a set of definitionsm each of which is of the form $n \rightarrow \varphi$, $n \leftarrow \varphi$, or $n \leftrightarrow \varphi$, where n is a fresh atomic proposition and φ is a formula. To ensure this CNF is non-trivial, we force the formula such that at least two of these definition types appear in the CNF. (this basically means there must be two non-atomic subformulas of different polarity.)
10. The formula has at most 6 models. Note that there are $2^3 = 8$ different interpretations of our formula, so this means the formula is not valid.
11. To reduce difficulty introduced by visual complexity, the LaTeX rendering of the formula should have a parenthesis nesting level of at most two.

Issues that we encountered with too strict restrictions:

cite tseitin paper?

the value of this is restriction isn't clear to me anymore... I wanted to control DPLL branching somewhat but now I don't think it does anything for that.

– the sample space might be empty. may lead to the generator getting "stuck" for a minute until the user kills it.
  For example, a note about restriction 3: with all the other restrictions, it is impossible to get a formula that contains atomic proposition of purely positive and purely negative polarity at the same time.
– Less drastic but perhaps more problematic: the sample space may be restricted too much, leading to the generation of boring and similar formulas.
  happens, for example, if we restrict the number of models to exactly one, or to no model.
  there simply aren't that many ways to rule out 8 interpretations using only 7 connectives.

We also tried enumerating the sample space to see how much variety we can expect. With five connectives there are 111 060 formulas satisfying the above criteria, with six connectives there are already 2 050 524 such formulas, while for seven connectives our computation did not finish in time. (our implementation was very little optimized for speed)

Finally, we convert the generated formula into LaTeX format and write it to a separate *.tex-file.

4.2 Non-Ground Superposition and Redundancy

Example 2. Consider the following inference:

$$\frac{\begin{array}{c} P(h(g(g(d,d),b))) \vee \neg P(h(f(d))) \vee f(d) \not\approx h(g(a,a)) \\ \neg P(h(g(x,b))) \vee f(d) \not\approx h(g(y,y)) \end{array}}{\neg P(h(f(d))) \vee f(d) \not\approx h(g(a,a))}$$

in the non-ground superposition inference system Sup (including the rules of the non-ground binary resolution inference system BR), where P is a predicate symbol, f, g, h are function symbols, a, b, d are constants, and x, y are variables.

(a) Prove that the above inference is a sound inference of Sup.

(b) Is the above inference a simplifying inference of Sup? Justify your answer based on conditions of clauses being redundant. □

Here, we fixed the pattern of the inference to be an instance of *subsumption resolution* (also known as *contextual literal cutting*).

To randomly generate first-order terms and literals, we first specify a signature of symbols from which the random generator may choose. Our signature consists of three sets: the predicate symbols (with arity), function symbols (with arity), and variables. We further control the shape of generated terms by specifying bounds on the *depth* of the term, i.e., the maximal nesting level of function calls (e.g., a constant symbol b has depth 0, while the term $g(f(x), d)$ has depth 2).

We first generate the second premise, which should always be non-ground. To this end, we generate a random uninterpreted literal L_1 containing exactly one variable occurrence, and a random equality literal L_2 containing at least two occurrences of a different variable. The second premise is then $C_2 \coloneqq L_1 \vee L_2$. The restrictions on variable occurrences are there to 1. ensure the clause is non-ground and 2. to make discovery of the mgu be of similar difficulty for all examinees.

Following this, we generate another uninterpreted literal L_3. Here, we also check that at least one function symbol of arity 2 appears in at least one of the literals.

Following this, we generate a ground substitution θ by randomly generating two ground terms. Note that is very easy to restrict the term generation to ground terms: we simply fix the set of variables in the desired signature to the empty set before calling the generator. The first premise is then $C_1 \coloneqq \overline{L_1\theta} \vee L_3 \vee L_2\theta$, where $\overline{L}$ is the complementary[2] literal to L.

Finally, the conclusion is $C_3 \coloneqq L_3 \vee L_2\theta$. Now we write the inference to a *.tex-file. As the last step, we extract the actual signature from the generated inference and output it into a separate *.tex-file.

After generating, we also output SMT-LIB problem files for soundness ($C_1, C_2 \models C_3$) and part of the redundancy condition ($C_3, C_2 \models C_1$, in our case, actually $C_3 \models C_1$ already holds). To

[2] i.e., $\overline{A} = \neg A$ and $\overline{\neg A} = A$.

double-check our generated inference, we then run Vampire on these files, expecting it to prove the entailment.

5 Implementation

Various programs/scripts tied together by a control script. Running the control script generates all problems and puts them together, resulting in n different exams in PDF format.

Latex template: leave holes for formulas/signatures that will be inserted using the command "\input".

Problem templates: (todo: describe impl of smt and groundsup)

For the full random generation of the SAT and redundancy problems, we were initially inspired by Haskell's QuickCheck library. We used QuickCheck for the first prototype, but realized that we need backtracking in the generator.

cite? Claessen, Koen and Hughes, John (2000). "QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs" (PDF). Proceedings of the International Conference on Functional Programming (ICFP), ACM SIGPLAN.

Added mtl-style typeclass 'MonadChoose' with a primitive operation 'choose' for choosing an element from a list of choices. Our generator implementations are generic over the monad, constrained by this typeclass.

We used two concrete implementations to evaluate generators: 1. 'RandomChoiceT', a monad transformer that implements 'choose' as uniform random choice with backtracking support (can be thought of as 'ListT' where 'choose' is like the normal monadic bind but shuffles the list with a random permutation first). This is what we actually use to generate random exams. 2. a standard list monad (or similar) to enumerate the sample space. This second evaluation method lets us ensure that the sample space is sufficiently large.

6 Conclusion