Automated Generation of Exam Sheets for Automated Deduction

Petra Hozzová, Laura Kovács, and Jakob Rath

TU Wien, Austria
{petra.hozzova,laura.kovacs,jakob.rath}@tuwien.ac.at

Abstract. Amid the COVID-19 pandemic, distance teaching became default in higher education, urging teachers and researchers to revise course materials into an 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 teaching-related project paper we survey the 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 called for individual exam sheets focused on problem solving and deductive proofs; as such exam sheets using test grids were not a viable solution for written exams within our 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, recordings, streaming and online communication channels, coming up with best practices to assess course performance was far from trivial. Even with very sophisticated technical infrastructure (use of which, on the other hand, would be unethical to require from course participants), avoiding collusion in the virtual environment is very hard to achieve, if possible at all. While work on online feedback generation has already been initiated, see e.g. [7,15], not much work on online examinations has emerged so far.

In this paper we survey our teaching-related project work in organizing online written exams, where the exam solutions require rigorous logical reasoning and proofs rather than using mechanized test grids. In particular, we are 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

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

focusing on specialised algorithms for reasoning in various fragments of first-order logics, such as propositional logic, combinations of ground theories, and full first-order logic with equality. As such, topics of the course cover theoretical and practical aspects of SAT/SMT solving [4,13,5] and first-order theorem proving using superposition reasoning [12,9].

By no means are we claiming that the framework we developed for online examination is optimal. Given the time constraints of examination periods, we aimed for an online exam setting that (i) reduces collusion among students and (ii) requires the same workload on each participant. Note that there is a trade-off between (i) and (ii) – very similar problems require comparable effort to be solved, while solving very different problems requires unequal effort. Therefore our goal was to strike a balance between (i) and (ii).

The algorithmic reasoning developed within our course called for exam sheets focused on problem solving and deductive proofs; hence, exam sheets using test grids were 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 essentially the same in each generated exam sheet. As such, we have randomly generated individual exam problems on

– SAT solving, by imposing (mostly) syntactical constraints on randomly generated SAT formulas (Section 2.1);
– Satisfiability modulo theory (SMT) reasoning, by exploiting reasoning in a combination of theories and varying patterns of SMT problem templates (Section 3.1);
– First-order theorem proving, by adjusting simplification orderings in superposition reasoning and using redundancy elimination in first-order proving, both in the ground/quantifier-free and non-ground/quantified setting (Section 2.2 and Section 3.2).

For each of the SMT and first-order problems we generated, we used respective SMT and first-order solvers to perform an additional sanity check (Section 4). Our toolchain and the generated benchmarks/exams are available at

https://github.com/AutomatedDeductionTUW/exagen

We believe our framework is beneficial not only for other distance learning platforms, but also to researchers in automated reasoning as we provide a large set of randomly generated benchmarks in SAT/SMT solving and first-order theorem proving to our scientific community. While our teaching-related project delivery is specific to formal aspects of automated reasoning, we note that our work can be extended with further constraints to scale it to other courses in formal methods.

This paper is structured as follows. In Sections 2-3 we discuss the high-level approach to generating the exam problems. Section 4 surveys the main implementation principles supporting our solution. Finally, in Section 5 we compare the teaching outcomes of our online written exam with those coming from previous in-class examinations. Based on these outcomes, we believe our online examination maintained the overall course quality in the study curricula.

2 Random Problem Generation

We first describe our solution for generating automated reasoning benchmarks in a fully automated and random manner. We used this setting to generate exam problems on SAT solving and first-order theorem proving by filtering out problem instances that are either too hard or too easy. Throughout this paper, we assume basic familiarity with standard first-order logic and refer to the literature [2,9] for further details.

2.1 Boolean Satisfiability (SAT)

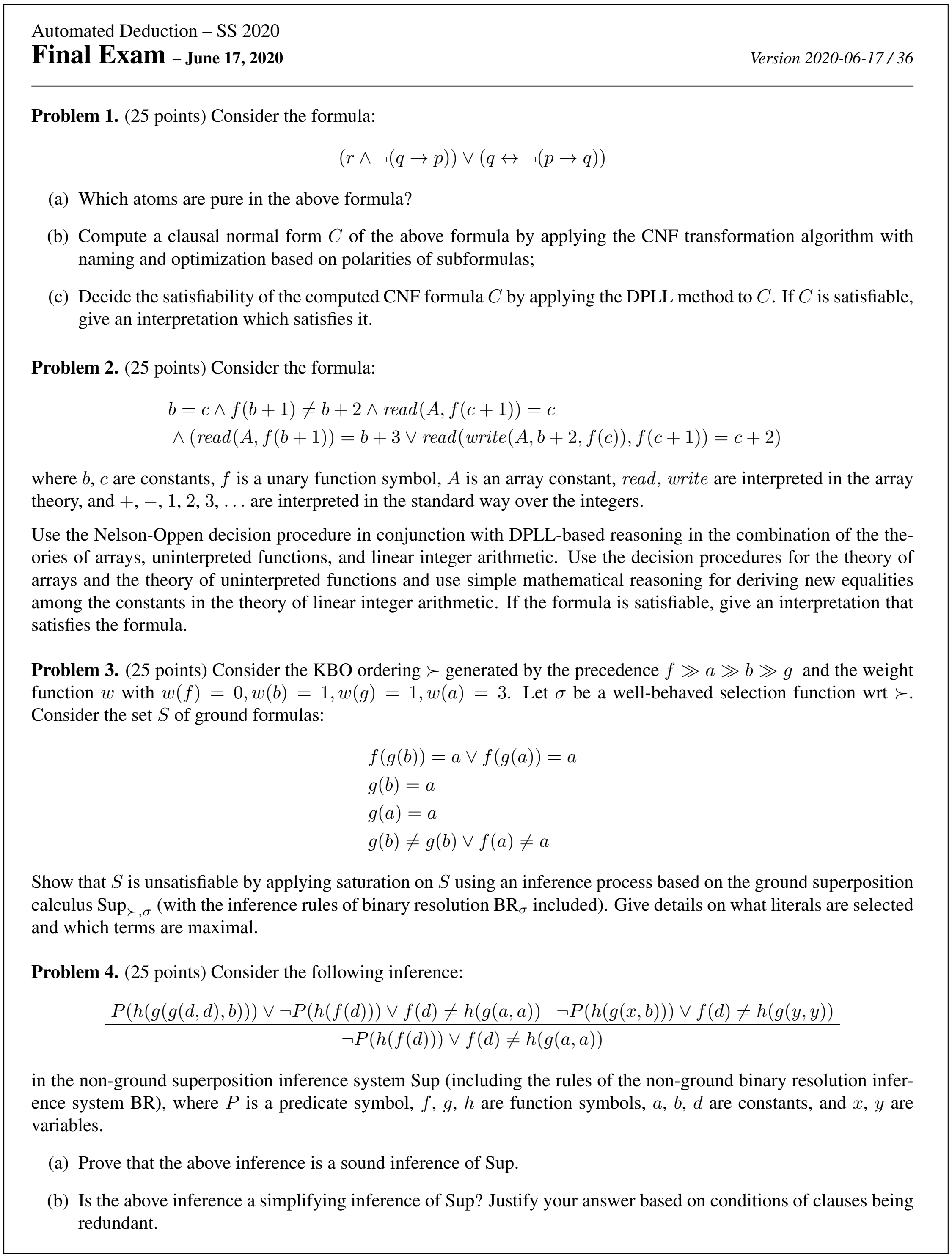
In our exam problem on SAT solving (Problem 1 of Figure 1), students were asked to (a) determine which atoms are of pure polarity in the formula, (b) compute a polarity-optimized clausal normal form (CNF) [14], and (c) decide satisfiability of the computed CNF formula by applying the DPLL algorithm.

Randomly generating propositional formulas in a naive setting would lead to a huge variety of formulas, spanning both formulas for which the above questions are trivial to answer (e.g., clauses as propositional tautologies) and others requiring much more effort (e.g., arbitrary formulas using only "$\leftrightarrow$"). More work was thus needed to ensure comparable workload for solving exam sheets.

To this end, we identified several syntactical characteristics that the exam problems on SAT solving should exhibit, and filtered the generated formulas by these, as summarized partially below.

(i) The SAT formula contains exactly seven logical connectives and exactly three different propositional variables.
(ii) There is at least one atom that appears with a pure polarity.
(iii) The connectives "$\leftrightarrow$", "$\rightarrow$", and "$\neg$" appear at least once, with "$\leftrightarrow$" appearing at most twice. At least one of "$\wedge$" and "$\vee$" appears.
(iv) Recall that the polarity-optimized clausal normal form involves a set of definitions, each of which is of the form $n \circ \varphi$ with $\circ \in \{\rightarrow, \leftarrow, \leftrightarrow\}$, a fresh propositional variable n, and a formula φ. We restrict the SAT formula such that at least two of the choices for $\circ$ appear in its CNF.
(v) The SAT formula has at most six models.

Our aim was to create problems of similar difficulty as in previous iterations of the course, which is why we used exams from previous years as a reference point. Some of the criteria, such as the number of connectives and variables, come from this previous experience. Other criteria, such as the restrictions on connectives and atom polarity, have been refined iteratively by checking the output for trivial or too complicated instances.



Automated Deduction – SS 2020

Final Exam **– June 17, 2020** *Version 2020-06-17 / 36*

Problem 1. (25 points) 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.

Problem 2. (25 points) Consider the formula:

$$\begin{aligned} & b = c \wedge f(b+1) \neq b+2 \wedge \mathit{read}(A, f(c+1)) = c \\ & \wedge (\mathit{read}(A, f(b+1)) = b+3 \vee \mathit{read}(\mathit{write}(A, b+2, f(c)), f(c+1)) = c+2) \end{aligned}$$

where b, c are constants, f is a unary function symbol, A is an array constant, read, write are interpreted in the array theory, and $+$, $-$, 1, 2, 3, ... are interpreted in the standard way over the integers.

Use the Nelson-Oppen decision procedure in conjunction with DPLL-based reasoning in the combination of the theories of arrays, uninterpreted functions, and linear integer arithmetic. Use the decision procedures for the theory of arrays and the theory of uninterpreted functions and use simple mathematical reasoning for deriving new equalities among the constants in the theory of linear integer arithmetic. If the formula is satisfiable, give an interpretation that satisfies the formula.

Problem 3. (25 points) Consider the KBO ordering $\succ$ generated by the precedence $f \gg a \gg b \gg g$ and the weight function w with $w(f) = 0, w(b) = 1, w(g) = 1, w(a) = 3$. Let σ be a well-behaved selection function wrt $\succ$. Consider the set S of ground formulas:

$$\begin{aligned} & f(g(b)) = a \vee f(g(a)) = a \\ & g(b) = a \\ & g(a) = a \\ & g(b) \neq g(b) \vee f(a) \neq a \end{aligned}$$

Show that S is unsatisfiable by applying saturation on S using an inference process based on the ground superposition calculus $\text{Sup}_{\succ,\sigma}$ (with the inference rules of binary resolution BR_{σ} included). Give details on what literals are selected and which terms are maximal.

Problem 4. (25 points) Consider the following inference:

$$\frac{P(h(g(g(d,d),b))) \vee \neg P(h(f(d))) \vee f(d) \neq h(g(a,a)) \quad \neg P(h(g(x,b))) \vee f(d) \neq h(g(y,y))}{\neg P(h(f(d))) \vee f(d) \neq h(g(a,a))}$$

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.

Fig. 1. An example of a randomly generated exam sheet of automated deduction.

Although the combination of the above conditions (i)-(v) might seem very restrictive, we note that there are 20 390 076 different SAT formulas satisfying the above criteria. Further, if we do not want to distinguish formulas that differ only by a permutation of atoms, 3 398 346 formulas remain. We are thus able to generate a large number of unique SAT formulas to be used in online examinations and beyond. Problem 1 of Figure 1 showcases one SAT reasoning challenge we automatically generated for one online examination sheet.

We finally note that, while experimenting with the different constraints (i)-(v) above, we encountered the following issues that may arise if the restrictions on the randomly generated formula are too strict:

– The sample space might be empty or very sparse. In practice, it seems to the user as if the problem generator got stuck, usually resulting in the process being killed by the user. For example, consider the restriction on polarities of propositional variables. Combined with the other restrictions, it is impossible to get a formula that contains atomic propositions of purely positive and purely negative polarity at the same time.
– The second issue manifests less drastically but is perhaps more problematic: the sample space may be too uniform, leading to the generation of trivial and/or very similar formulas. In particular, we encountered this problem when we restricted the number of models to exactly one, or zero. We note that there simply are not that many ways to rule out eight interpretations using only seven connectives.

2.2 Non-Ground Superposition with Redundancy

Moving beyond Boolean satisfiability, we developed a random problem generator for first-order formulas with equality, in the setting of superposition-based first-order theorem proving with redundancy elimination [12,9]. In this problem, a concrete inference[2] was given to the students, and their task was to (a) prove that the inference is sound and (b) that the inference is a simplification inference (Problem 4 of Figure 1).

We recall that a simplification inference is an inference that removes clauses from the proof search space, whereas a generating inference adds new clauses to the search space [9]. In our work, we considered the simplification inference of *subsumption resolution*

$$\frac{A \vee C \quad \neg B \vee D}{D} \qquad \text{or} \qquad \frac{\neg A \vee C \quad B \vee D}{D} \tag{1}$$

where A, B are atoms and C, D are clauses such that A and B are unifiable with the most general unifier θ, and we have $A\theta \vee C\theta \subseteq B \vee D$. Due to the last condition, the second premise $\neg B \vee D$ (or $B \vee D$) of (1) is redundant and can be deleted from the search space after applying (1) within proof search.

[2] I.e., an instance of an inference rule as opposed to the rule itself

We randomly generated first-order instances of the inference rule (1), as discussed next. Our setting could however be easily extended to other simplification inferences, such as subsumption demodulation [6], and even generating inferences.

(i) To randomly generate first-order terms and literals, we fixed a first-order signature consisting of predicate and function symbols and specified a set of logical variables. We controlled the shape of the generated terms by giving bounds on the *depth* of the term, that is 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).

(ii) To obtain random instances of (1), we first generated non-ground clauses $C_1 \coloneqq L_1 \vee L_2$ corresponding to an instance of the first premise of (1). To this end, we generated 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.

(iii) We next generated the clause $C_2 \coloneqq \overline{L_1\theta} \vee L_2\theta \vee L_3$ as an instance of the second premise of (1) where θ is a randomly generated grounding substitution, L_3 is a randomly generated ground literal, and $\overline{L}$ is the complementary[3] literal to L.

(iv) We set $C_3 \coloneqq L_3 \vee L_2\theta$ as an instance of the conclusion of (1), yielding thus the inference $\dfrac{C_1 \quad C_2}{C_3}$ as an instance of (1).

We found that with the concrete signature used for our exam, based on the above steps (i)-(iv), our approach can generate more than 10^{11} different instances of the inference (1). Problem 4 of Figure 1 lists one such an instance.

3 Random Variation of Problem Templates

We now describe our framework for generating random quantifier-free first-order formulas with and without theories, that was used in the SMT reasoning and ground superposition proving tasks of our exam. For both of these tasks, we used quantifier-free first-order formula templates and implemented randomization over these templates by considering theory reasoning and simplification orderings.

Using this approach we achieved highly controlled output: exam problems which did not require any additional filtering. However, we note that the number of generated problems was limited, and to obtain additional problems, we would have to modify the templates.

3.1 Satisfiability Modulo Theories (SMT)

We considered first-order formula templates in the combined, quantifier-free theories of equality, arrays and linear integer arithmetic, corresponding to the

[3] I.e., $\overline{L} = \neg L$ and $\overline{\neg L} = L$.

weight of: f g a b	precedence	weight of: f g a b	precedence
$w_{1,f}$: 1 3 2 1	$p_{1,f} : a \gg b \gg f \gg g$	$w_{1,g}$: 3 1 2 1	$p_{1,g} : a \gg b \gg g \gg f$
$w_{2,f}$: 0 3 2 1	$p_{2,f} : f \gg a \gg g \gg b$	$w_{2,g}$: 3 0 2 1	$p_{2,g} : g \gg a \gg f \gg b$
$w_{3,f}$: 0 1 3 1	$p_{3,f} : f \gg a \gg b \gg g$	$w_{3,g}$: 1 0 3 1	$p_{3,g} : g \gg a \gg b \gg f$
$w_{4,f}$: 1 2 3 1	$p_{4,f} : g \gg f \gg a \gg b$	$w_{4,g}$: 2 1 3 1	$p_{4,g} : f \gg g \gg a \gg b$

Table 1. Weights and precedences for the ground superposition problem.

logic AUFLIA of SMT-LIB [1]. We aimed at generating SMT formulas over which reasoning in all three theories was needed, by exploiting the DPLL(T) framework [13] in combination with the Nelson-Oppen decision procedure [11] (Problem 2 of Figure 1).

With naive random generation, it might however happen that, for example, array reasoning is actually not needed to derive (un)satisfiability of the generated SMT formula. We therefore constructed an SMT formula template and randomly introduced small perturbations in this template, so that the theory-specific reasoning in all generated SMT instances is different while reasoning in all theories is necessary. For doing so, we considered an SMT template with two constants of integer sort and replaced an integer-sorted constant symbol c by integer-sorted terms $c + i$, where $i \in \{-3, -2, \ldots, 3\}$ is chosen randomly. We flattened nested arithmetic terms such as $(c + i) + j$ to $c + k$, where i, j, k are integers and $k = i + j$. As a result, we generated 49 different SMT problems; we show one such formula, together with the corresponding reasoning tasks, in Problem 2 of Figure 1.

3.2 Ground Superposition

For generating quantifier-free first-order formulas with equalities, over which ground and ordered superposition reasoning had to be employed (Problem 3 of Figure 1), we aimed at (i) generating unsatisfiable sets S of ground formulas with uninterpreted functions symbols, such that (ii) refutation proofs of S had similar lengths and complexities. Similarly to Section 3.1, we fixed a template for S and only varied its instantiation and the Knuth-Bendix ordering (KBO) [8] $\succ$ to be used for refuting S within the superposition calculus. To this end, we considered variations of weight function w and symbol precedence $\gg$ over S, yielding thus different KBOs $\succ$ to be used for refuting S. The main steps of our approach are summarized below.

(i) We fixed the template for S to be the following set of four clauses

$$E(F(X)) = a \ \vee \ E(G(Y)) = a \tag{2}$$
$$F(X) = a\,[\vee H(b) \neq H(b)\,] \tag{3}$$
$$G(Y) = a\,[\vee H(b) \neq H(b)\,] \tag{4}$$
$$E(a) \neq a\,[\vee H(b) \neq H(b)\,], \tag{5}$$

where $E, F, G, H \in \{f, g\}$, $X, Y \in \{a, b\}$, and the literal in [ ] is added to the clauses optionally.

condition	i_1, I_1	i_2, I_2	i_3, I_3
$F \neq G$ and $X \neq Y$	$1, E$	$2, E$	$3, E$
$F \neq G$ and $X = Y$	$1, H$	$2, E$	$4, H$
$F = G$ and $X \neq Y$	$1, H$	$2, H$	$3, E$

Table 2. Assignment of KBOs to instances of the ground superposition problem.

(ii) We created instances of S of this template ensuring that no clause in S is redundant, by considering the following constraints.
- $E \neq H$ and $F(X) \neq G(Y)$;
- Either X or Y is not a. Similarly, either F or G is not E;
- The literal $H(b) \neq H(b)$ is in exactly one of the clauses (3), (4), (5).

As a result, we produced 12 instances of S satisfying the above properties.

(iii) We considered the term algebras induced by the generated instances of S and designed KBOs $\succ$ such that refuting the respective instances of S using $\succ$ requires ordering terms both using weight w and precedence $\gg$. In addition, we imposed that either $F(X) \succ a \succ G(Y)$ or $G(Y) \succ a \succ F(X)$ holds. With such orderings $\succ$, the shortest refutations of instances of S are of the same length, and in at least one application of superposition, a is replaced by either $F(X)$ or $G(Y)$ in the resulting clause. We generated eight different KBOs $\succ$ fulfilling these conditions. The weights and precedences used to generate the KBOs are displayed in Table 1. The table shows all weight and precedence combinations, denoted as $w_{i,I}, p_{i,I}$ for $i \in \{1, 2, 3, 4\}$ and $I \in \{f, g\}$.[4] Each instance of S was combined with three different KBOs, generated by pairs $(w_{i_1,I_1}, p_{i_1,I_1}), (w_{i_2,I_2}, p_{i_2,I_2}), (w_{i_3,I_3}, p_{i_3,I_3})$. The values of $i_1, I_1, i_2, I_2, i_3, I_3$ are chosen based on the values of F, G, X, Y, as expressed by the conditions in Table 2.

Ultimately, we obtained 36 different problems (combinations of instances of S and $\succ$) for the ground superposition reasoning task of our exam. Problem 3 of Figure 1 shows such an instance.

4 Implementation

We implemented our approach to randomly generating SAT, SMT, and non-ground first-order problems in Haskell, whereas our ground superposition problem generator was implemented in Python. All together, our toolchain involved about 2 300 lines of code, including additional scripts for putting parts together. We

[4] Note that for all values of i, $w_{i,f}(f) = w_{i,g}(g)$ and $w_{i,f}(g) = w_{i,g}(f)$, and the precedences $p_{i,f}, p_{i,g}$ are the same except for the precedence of f, g. However, for convenience, the table contains both $w_{i,f}$ and $w_{i,g}$, as well as $p_{i,f}$ and $p_{i,g}$ for all values of i.

encoded each randomly generated SMT and first-order formula into the SMT-LIB input format [1] and, for sanity checks, ran the SMT solver Z3 [10] and the first-order theorem prover Vampire [9] for proving the respective formulas. In addition, each formula has been converted to LaTeX, yielding randomly generated exam sheets – one such exam sheet is given in Figure 1.

Regarding the filtering of generated formulas using the constraints discussed in Section 2, we implemented restrictions on the shape of formulas (items (i) and (iii) in Section 2.1) as constraints during formula generation, while other critera were realized as post-generation filters. Regarding post-generation filtering, we did not require very efficient algorithms since the formulas under consideration are very small. For example, for the restriction on the number of models we used a naive satisfiability test based on evaluating the formula under each possible interpretation. Thanks to this approach it is easy to add new filters/constraints.

For the random problem generation setting of Section 2, we applied design principles of the Haskell library QuickCheck [3]. With QuickCheck, randomly generated data can easily be defined in an embedded generator language. However, because of our many filtering criteria, we wanted the generator to additionally support backtracking. We were also interested in determining the size of the filtered sample space. To this end, we created a simple typeclass `MonadChoose` in the style of the monad transformer library (mtl), with a single primitive operation `choose` for choosing an element from a list of possible choices:

```
class MonadPlus m => MonadChoose m where
  choose :: [a] -> m a
```

Our generator implementations are generic over the monad, constrained by `MonadChoose`. The following listing shows (a slightly simplified) part of the inference generator discussed in Section 2.2.

```
genExamInference :: MonadChoose m => m Inference
genExamInference = do
  -- Define signature (partially omitted)
  let vars = ["x", "y", "z"]
  let opts = GenOptions{ vars = vars, ... }

  -- Choose variables to appear in l1 and l2
  v1 <- choose vars
  v2 <- choose (filter (/= v1) vars)

  -- Generate literals
  -- l1: exactly one occurrence of v1
  l1 <- mfilter ((==1) . length . toListOf variables)
        $ genUninterpretedLiteral opts{ vars = [v1] }
  -- l2: at least two occurrences of v2
  l2 <- mfilter ((>=2) . length . toListOf variables)
        $ genEqualityLiteral opts{ vars = [v2] }
  -- l3: ground literal
  l3 <- genUninterpretedLiteral opts{ vars = [] }
  -- (rest omitted)
  return inference

genEqualityLiteral, genUninterpretedLiteral
  :: MonadChoose m => GenOptions -> m Literal
-- (literal generators omitted)
```

We used two concrete implementations to evaluate generators:

1. `RandomChoice`, a monad that implements `choose` as uniform random choice with backtracking support. Conceptually, this is like the standard list monad where `choose` works like the regular monadic bind for lists except that it first shuffles the list with a random permutation. This evaluation method is used to generate random exams.
2. The standard list monad to enumerate the sample space. This second evaluation method helps verifying that the sample space is sufficiently large.

5 Evaluation of Online Exam Outcomes

In Summer 2020, all together 31 students took the online written exam in "Automated Deduction". We note that in Summer 2018 and Summer 2019, there have been 17 and respectively 31 students taking the in-class exam of the course. We believe that the online lecturing and examination in Summer 2020 did not have negative impact on the students' course performance.

In the online written examination of Summer 2020, the students solved their respective unique exam assignments on paper and submitted scanned versions of their solutions online. The types of exam problems from Summer 2020 were the same as in previous editions of the course. However, contrary to previous years, different students had different exam assignments, to minimise opportunity for collusion between students.

While building the pipeline described in this paper required much more work than creating just one exam sheet, our approach was more efficient than it would be to create 31 different exam sheets manually. Additionally, our approach guaranteed that the exam problems were unique, yet required comparable effort to solve. Also, reusing our pipeline in the future requires only minimal changes.

Further, the types of the problems in our exam are not trivial to grade, since the solutions require applying complicated reasoning algorithms on paper, and the grade has to take into account the whole process, not just the result. However, the use of templates of Section 3 made the grading fairly similar to grading multiple solutions of the same problem by providing a clear pattern to follow. This observation extends to the problem on non-ground superposition (Subsection 2.2), because the argument required in the solution does not depend majorly on the generated parts, even though we did not use an explicit template. The situation is different for the problem on boolean satisfiability (Subsection 2.1). There, the solution varies greatly with the input formula, and grading a different instance requires mentally stepping through the problem again. One might suggest to also generate fully worked solutions to this problem, however it is not immediately clear that this would be helpful: at various points, the students may choose among multiple correct possibilities, each of which leads to differences in subsequent parts of the solution.

The average exam score was 79.9 %, compared to 80 % in 2019 and 76 % in 2018. Based on the comparable exam averages, we believe our online written examination from Summer 2020 did not bring any significant change in the overall course performances of students enrolled in the course.

Finally, eight students filled out a feedback survey for the course in Summer 2020. All of them reported high levels of satisfaction with the course, with one student explicitly praising the online exam format. Our course in Summer 2020 has been also nominated for the *Best Distance Learning Award 2020* of the TU Wien.

6 Conclusion

We describe a randomized approach and toolchain for generating exam problems in automated reasoning, in particular in the setting of SAT, SMT, and first-order theorem proving. Our approach was used to generate individual exam sheets focused on problem solving within automated deduction, and could be adapted to other constraints and course frameworks.

Acknowledgments

We acknowledge funding supporting this work, in particular the ERC CoG ARTIST 101002685, the ERC StG 2014 SYMCAR 639270 and the Austrian FWF research project LogiCS W1255-N23.

References

1. Barrett, C., Fontaine, P., Tinelli, C.: The SMT-LIB standard: Version 2.6. Tech. rep., Department of Computer Science, The University of Iowa (2017), available at `www.SMT-LIB.org`
2. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press (2009)
3. Claessen, K., Hughes, J.: Quickcheck: a lightweight tool for random testing of haskell programs. In: Proc. ICFP. pp. 268–279 (2000)
4. Davis, M., Logemann, G., Loveland, D.W.: A Machine Program for Theorem-Proving. Commun. ACM **5**(7), 394–397 (1962)
5. Ganzinger, H., Hagen, G., Nieuwenhuis, R., Oliveras, A., Tinelli, C.: DPLL(T): Fast Decision Procedures. In: Proc. of CAV. pp. 175–188 (2004)
6. Gleiss, B., Kovács, L., Rath, J.: Subsumption Demodulation in First-Order Theorem Proving. In: Proc. IJCAR (2020), to appear
7. Gulwani, S., Radicek, I., Zuleger, F.: Automated Clustering and Program Repair for Introductory Programming Assignments. In: Proc. PLDI. pp. 465–480 (2018)

8. Knuth, D.E., Bendix, P.B.: Simple word problems in universal algebras. In: Computational Problems in Abstract Algebra, pp. 263–297. Pergamon Press (1970)
9. Kovács, L., Voronkov, A.: First-Order Theorem Proving and Vampire. In: Proc. CAV. pp. 1–35 (2013)
10. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Proc. TACAS. pp. 337–340 (2008)
11. Nelson, G., Oppen, D.C.: Simplification by Cooperating Decision Procedures. ACM Trans. Program. Lang. Syst. **1**(2), 245–257 (1979)
12. Nieuwenhuis, R., Rubio, A.: Paramodulation-Based Theorem Proving. In: Handbook of Automated Reasoning, pp. 371–443 (2001)
13. Tinelli, C.: A DPLL-Based Calculus for Ground Satisfiability Modulo Theories. In: Porc. JELIA. pp. 308–319 (2002)
14. Tseytin, G.S.: On the Complexity of Derivation in Propositional Calculus, chap. Studies in Constructive Mathematics and Mathematical Logic, pp. 115–1125. Steklov Mathematical Institute (1970)
15. Wang, K., Singh, R., Su, Z.: Search, Align, and Repair: Data-Driven Feedback Generation for Introductory Programming Exercises. In: Proc. PLDI. pp. 481–495 (2018)