Formal Verification
Coursework 2: SAT/SMT
Autumn 2021

Lecturers: Elizabeth Polgreen and Paul Jackson

Date issued: Monday 18th October 2021
Submission date: Monday 8th November 2021, 4pm
Mark total: 50. **Weight:** 20%
Submission Instructions: See Section 2.
Coursework Regulations: See Section 3.

You will be using the MiniSAT SAT solver, the Z3 SMT solver and a BDD package to solve satisfiability problems. All installation instructions are provided within the questions, inside boxes like this one.

Many of the following questions ask you to include your written answers in a PDF called `answers.pdf`. You can create this PDF using LaTeX or your favourite word processor (e.g., MS Word).

1 SAT/SMT questions

1. This question is **not assessed** and uses MiniSAT and the Z3 SMT solver.

   To install MiniSAT run the following:

   ```
   sudo apt update
   sudo apt install minisat
   ```

   Alternatively, you can download the pre-compiled binaries found at `http://minisat.se/MiniSat.html`.

The easiest way to install Z3 is via the python package manager, assuming you already have python and pip installed (If you don't already have pip, the pip website is at `https://packaging.python.org/key_projects/#pip`). To install Z3 run the following command:

```
pip install z3-solver
```

Consider 3 persons A, B and C who need to be seated in a row but

- A does not want to sit next to C
- A does not want to sit in the left chair
- B does not want to sit to the right of C

(a) Write a propositional formula that is satisfiable if and only if there is a seat assignment for the three persons that satisfies all constraints.

(b) Encode the formula into DIMACS CNF [1] in a file called `question1.cnf`. Use MiniSAT to check if the formula is satisfiable or not.

(c) Encode the same formula into SMT-LIB[2][3] in a file called `question1.smt2` and check if the formula is satisfiable or not using Z3.

(not assessed)

2. The expression `x ? y : z` is equivalent to:

```
if(x){ y; } else { z; }
```

Consider the following two expressions, assuming all variables have only one bit.

```
!(a || b) ? h : !(a == b) ? f : g
!(!a || !b) ? g : (!a && !b) ? h : f
```

[1] `https://people.sc.fsu.edu/~jburkardt/data/cnf/cnf.html`

[2] `http://smtlib.cs.uiowa.edu/`

[3] `https://web.archive.org/web/20210119175613/https://rise4fun.com/Z3/tutorial/guide`

(a) Show manually, by rewriting, that the two expressions are equivalent (this should be included `answers.pdf`)

(2 points)

(b) encode the problem into an SMT-LIB and save it in a file called `question2.smt2`. Run Z3 using the command `z3 question2.smt2`. Include the SMT-LIB file and copy the output from Z3 into `answers.pdf`.

(3 points)

3. The following problems are written in SMT-LIB but I forgot to include the `set-logic` command. It is good practice to set the logic to the least permissive valid logic as it often leads to better solver performance. A list of possible logics can be found here: `http://smtlib.cs.uiowa.edu/logics.shtml`.

   Choose the least permissive valid logic for each problem, i.e., a logic that does not allow theories that are not used in the problem, and explain your choice (e.g., "I chose `QF_LRA` (quantifier-free linear real arithmetic) because this problem uses only real variables, has no quantifiers and only linear mathematical operations"). In addition, briefly describe in words what the problem is asking (e.g., "Does there exist an assignment to integer variable x and y such that they are equal?"). Include the answers in `answers.pdf`:

   (a)
   ```
   (declare-fun x0 () Int)
   (declare-fun x1 () Int)
   (assert (>= x0 0))
   (assert (>= x1 0))
   (assert (= (- x0 x1) 1))
   (check-sat)
   ```

   (1 point)

   (b)
   ```
   (declare-fun v0 () (_ BitVec 4))
   (declare-fun v1 () (_ BitVec 4))
   (declare-fun f ((_ BitVec 4)) (_ BitVec 4))
   (declare-fun g ((_ BitVec 4)) (_ BitVec 4))
   (assert (= (f v0) (g v0)))
   (assert (not (= (f v1) (g v1))))
   (check-sat)
   ```

   (1 point)

   (c)
   ```
   (declare-fun x2 () (_ BitVec 32))
   (declare-fun x1 () (_ BitVec 32))
   (declare-fun b1 () Bool)
   ```

```
(assert (= ((_ extract 0 0) (bvxor x1 x2)) #b1))
(check-sat)
```

(1 point)

(d)
```
(declare-fun x0 () Real)
(declare-fun x1 () Real)
(assert (>= x0 0))
(assert (>= x1 0))
(assert (= (/ x0 x1) 2))
(check-sat)
```

(1 point)

(e)
```
(declare-fun x () String)
(declare-const I Int)
(assert (= x "\0\1\2\3\04\005\x06\7\8\9ABC\\""\t\a\b"))
(assert (= I (str.len x)))
(check-sat)
```

(1 point)

(f)
```
(declare-fun a () (Array Int Int))
(assert (forall ((i Int)) (= (select a i) 0)))
(check-sat)
```

(1 point)

(g)
```
(declare-fun a () (Array Int Int))
(declare-fun b () (Array Int Int))
(assert (forall ((i Int))
    (exists ((j Int))
        (= (select a i) (select b j)))))
(check-sat)
```

(1 point)

4. Consider the problem of allocating n pigeons to m pigeonholes such that no hole contains more than one pigeon[4]. We can write a propositional formula for this problem. Let x_{ij} be a propositional variable that is true when pigeon i is placed in hole j.

$$\bigvee_{j=1}^{m} x_{ij} \quad \text{for all } i \in 1, 2, \ldots n \qquad \bigwedge$$

$$(i \neq j) \implies (\neg x_{ik} \vee \neg x_{jk}) \quad \text{for all } k \in \{1, 2, \ldots m\}, i, j \in \{1, \ldots n\}$$

[4] `https://en.wikipedia.org/wiki/Pigeonhole_principle`

Figure 1: Pigeons! These pigeons do not satisfy the propositional formulae given in Question 4.

(a) Describe in words the meaning of the two parts of the formula above (in `answers.pdf`).

(2 points)

(b) Write a script that generates CNF in the DIMACs format for the formula above, for parameters m and n. You can write the script in any language you choose, but please call it `question4b.*` where `*` is the appropriate extension for your scripting language (e.g.,`question4b.py` for python). Include a CNF file generated by your script for $n = 3, m = 2$, called `question4b.cnf`. Check with MiniSAT that the formula is unsatisfiable when there are more pigeons than holes.

(6 points)

(c) Run MiniSAT for $n = 4, 5, 6, \ldots 15$ and $m = n - 1$ and plot how the runtimes vary with n. Describe your observations. Include the plot and your observations in `answers.pdf`.

(6 points)

5. We will now repeat the above for BDDs, using the Python-based dd package which contains a fully Pythonic implementation of Binary Decision Diagrams.

> Install the dd package with pip: `pip install dd`. You may need to run this command with superuser privileges (`sudo pip install dd`). The `pigeons.py` script can be found on Learn in the same place as the PDF instructions for this coursework, and has been tested with Python 2.7 and Python 3.5.

Test your installation of pip and the dd package by running `python pigeons.py\verb`. There are two examples provided in the code to introduce you to the dd package. The documentation for the dd package is on GitHub: `https://github.com/johnyf/dd/blob/master/doc.md`. The usage instructions for `pigeons.py` are

```
$ python pigeons.py --help

usage: pigeons.py [-h] [--example {1,2,3}] [--n N] [--pdf PDF]
optional arguments:
-h, --help          show this help message and exit
--example {1,2,3}   Example to run (1-3). Default=1.
--n N               Value of n (default=2). (Only for examples 2 and 3.)
--pdf PDF           PDF image output filename (Only for example 1.)
```

(a) Using the skeleton code in `pigeons.py`, construct the BDD for the CNF formula (replace example 3 in `pigeons.py` with your solution) and verify it simplifies to false.

(6 points)

(b) Describe (in `answers.pdf`) what happens to the size of the BDD and the run-time when you increase n? You can use the unix command `time`.

(2 point)

(c) Does the variable ordering affect runtime? If so, provide two different variable orderings and describe the difference between the two behaviours. Include the answer to this part in `answers.pdf`.

(3 points)

(d) The encoding described above uses one variable for each pigeon/hole combination, resulting in $n \times (n-1)$ variables in total. Can you find an encoding that uses $\mathcal{O}(n \times log\ n)$ variables? Does this change the scalability (i.e., the runtime as n increases)? Include the answer to this part in `answers.pdf`.

(5 points)

6. This problem uses Z3. Consider the following problem. Suppose a warehouse always needs to have two employees on duty. The warehouse operates 24/7 and each day is divided into three shifts: morning, afternoon, night. There are nine employees and we want to come up with a schedule for these employees.

(a) Model this as an SMT problem such that the following constraints are satisfied and the schedule repeats with a period of 7 days:

- Each employee covers an equal number of shifts
- No employee has to work 2 shifts back-to-back

Write a python script `question6.py` using the Z3 python API to generate the SMT query. Add comments to the script explaining the meaning of each variable in your encoding. Include the script file in your submission.

(6 points)

(b) Is the problem satisfiable? If not, find a repeating period for the schedule that does allow the constraints to be satisfied. Include your answer to this question in `answers.pdf`.

(2 points)

2 Submission

2.1 Packaging your submission

You will need to submit the following:

- A PDF file called `answers.pdf` with the answers to questions 2a, 2b, 3, 4a, 4c,5b, 5c, 5d, and 6b
- `question1.cnf`
- `question2.smt2`
- A script file called e.g., `question4b.py` (or with a different extension if you use a different scripting language), and `question4b.cnf`
- `pigeons.py`
- `question6.py`

Place all files you have used in one folder called `fv-cw2`. Make a compressed version of your project folder using zip compression:

- On Linux systems use the command `zip -r fv-cw2.zip fv-cw2`
- On Windows systems use `Send to > Compressed (zipped) folder`.
- On Mac systems use `File > Compress "fv-cw2"`.

You should now have a file called `fv-cw2.zip`.

2.2 How to submit

Ensure that you are LEARN-authenticated by visiting http://learn.ed.ac.uk. Go to the Formal Verification LEARN page. Click on the Assessment link in the left-hand margin bar and then the link that says **Coursework 2: SAT/SMT - submission**. Use the Browse Local Files option to find and upload your ZIP file.

In order to streamline the processing of your submissions, and help avoid lost submissions, please use exactly these filenames. When finished, make sure that you click Submit.

This submission mechanism should allow you to make multiple submissions. Later submissions will over- write earlier ones. Submissions which arrive after the coursework deadline will be subject to the School's late submission penalties as detailed at

`http://web.inf.ed.ac.uk/infweb/student-services/ito/admin/coursework-projects/late-coursework-extension-requests`.

Extension Rule 1 will be applied for submissions "Extensions are permitted (7 days) and Extra Time Adjustments (ETA) for extensions are permitted". The complete statement of this rule is available at the URL above.

3 Coursework Regulations

3.1 Good scholarly practice

Please remember the good scholarly practice requirements of the University regarding work for credit. You can find guidance at the School page:

`https://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct`

This also has links to the relevant University pages. You are required to take reasonable measures to protect your assessed work from unauthorised access. For example, if you put any such work in a source code repository then you must set access permissions appropriately, limiting access to at most yourself and members of the formal verification course team.

You can collaborate with otherAll work that you submit for assessment must be your own.

3.2 Late submission policy

It may be that due to illness or other circumstances beyond your control that you need to submit work late. Submissions which arrive after the coursework deadline will be subject to the School's late submission penalties as detailed at

`http://web.inf.ed.ac.uk/infweb/student-services/ito/admin/coursework-projects/late-coursework-extension-requests`.

Extension Rule 1 will be applied for submissions for Coursework 1 and Coursework 2. This states that "Extensions are permitted (7 days) and Extra Time Adjustments (ETA) for extensions are permitted." The complete statement of this rule is available at the URL above.