EECS 219C: Formal Methods **S. A. Seshia and K. Cheang**

Homework 2: SMT

Assigned: February 27, 2019 *Due in class: March 13, 2019*

Note: For this and subsequent homeworks, if a problem requires you to come up with an algorithm, you should prove your algorithm's correctness as well as state and prove its asymptotic running time. See the webpage for rules on collaboration.

This homework uses the Python API of the Z3 SMT solver. There is a tutorial for the Z3 Python API here: `https://ericpony.github.io/z3py-tutorial/guide-examples.htm`. The official documentation for the API is here: `http://z3prover.github.io/api/html/namespacez3py.html`. You might want to run an example or two from the tutorial to make sure that your Z3 setup is working correctly.

1. **Bit Twiddling Hacks** (QF_BV) *(30 points)*

   In this question, you will use an SMT solver for finite-precision bit-vector arithmetic (QF_BV) to check whether two programs are functionally equivalent or not. Two program pairs are provided below in parts (a) and (b).

   You will need to use an QF_BV SMT solver to answer this question. Apart from the answer to this question, you should hand in on bCourses the SMT input files you create and the output from the SMT solver. If the programs are not equivalent, you should give an input value on which they differ — all SMT solvers have an option to generate models. (For fun: if the programs are not equivalent, suggest how one or both could be fixed so that they are equivalent.)

   Assume that an `int` is 32 bits for this assignment. It represents a signed quantity — the programs below are C code.

   (a) Are the functions `f1` and `f2` in Figure 1 equivalent?
   Assignment: In your solution:
   - Provide your SMT-LIB format input file to check equivalence.
   - Provide the counterexample input if these programs are not equivalent.

   (b) Are the functions `f3` and `f4` in Figure 2 equivalent?
   Assignment: In your solution:
   - Provide your SMT-LIB format input file to check equivalence.
   - Provide the counterexample input if these programs are not equivalent.

```c
int f1(int x) {
    int v0;

    if (x > 0) v0 = x;
    else v0 = -x;

    return v0;
}
```

```c
int f2(int x) {
    int v1, v2;

    v1 = x >> 31;
    v2 = x ^ v1;

    return (v2 - v1);
}
```

Figure 1: Functions for Question 1(a)

```c
int f3(int x, int y) {
    int v0;

    if (x >= y) v0 = x;
    else v0 = y;

    return v0;
}
```

```c
int f4(int x, int y) {
    int v1, v2, v3;

    v1 = x ^ y;
    v2 = (- (x >= y));
    v3 = v1 & v2;
    return (v3 ^ y);
}
```

Figure 2: Functions for Question 1(b)

While you can use any QF_BV solver for this assignment, it is probably easiest to use the Z3 SMT solver which can installed from: `https://github.com/Z3Prover/z3`.[1].

Learn about the SMT-LIB format by looking at examples from `http://www.smtlib.org/`.

2. **Sum-Sudoku** (Quantified Formulas) *(40 points)*

Consider the game of (simplified) (n, m) Sum-Sudoku. The game consists of an $n \times n$ square-grid. Each entry in the grid has to be filled using digits between 1 and m (both inclusive). The usual rules of Sudoku apply where a digit must appear only once in each row and each column.[2] In addition, Sum-Sudoku requires that each row and each column sum to a specified value.

	8	8	12
8			
10			
10			

(a) $(3, 5)$ Sum-Sudoku puzzle.

	8	8	12
8	1	4	3
10	5	1	4
10	2	3	5

(b) $(3, 5)$ Sum-Sudoku puzzle with solution.

Figure 3: Sum Sudoku

Figure 3a shows a $(3, 5)$ Sum-Sudoku puzzle. A valid solution requires that the rows sum to 8, 8 and 10 respectively, while the columns should sum to 8, 8 and 12 respectively. Figure 3b shows its solution.

In this problem, we will use the Z3 SMT solver's Python API to investigate properties of Sum-Sudoku puzzles. Skeleton code for Python (tested for v2.7 and v3.6) that demonstrates use of the API is provided in the bCourses folder Homeworks/HW2 under the menu option Files. You will implement specific functions in this skeleton code.

(a) Formulate an SMT instance that finds a solution to Sum-Sudoku puzzles. What theories are used in your formulation?

Assignment: In your solution:

- Describe your encoding and list the constraints in it.

[1] If you are installing the solver from source, enable the Python API by passing '`--python`' to the `mk_make.py` script as you will need this for Problems 2 and 3.

[2] In this simplified version of the game, we will ignore the "box"-constraint that Sudoku usually has.

- Encode your formulation using the Z3 API by completing the implementation of the functions `var`, `val`, and `valid` in the file `sumsudoku.py`.

(b) A Sum-Sudoku puzzle may not have a unique solution. Formulate a (quantified) SMT query that finds an assignment to the row and column sums such that the resulting puzzle has a unique solution.[3]

Assignment: In your solution:

- Describe this formulation and list its constraints.
- Encode your formulation using the Z3 API by completing the implementation of `create_puzzle` in the file `createpuzzle.py`.
- How scalable is your solution? Play around with the values of m and n and discuss your findings.

(c) A more scalable method of generating Sum-Sudoku puzzles with unique solutions would be to start with a fully-filled out puzzle and repeatedly remove entries as long as the resulting puzzle has a unique solution.

Assignment: In your solution:

- Describe the algorithm for this formulation and list the constraints that are checked in each iteration.
- Encode your formulation using the Z3 API by completing the implementation of `make_puzzle_unique` in the file `makepuzzleunique.py`.

3. **Bag of Chips** (Program Termination)[4] *(30 points)*

Consider the following puzzle. You are given a bag of red, yellow and blue chips. If the bag contains only one chip, you remove it from the bag. Otherwise, you remove two chips from the bag at random and:

(i) If one of the removed chips is red, you do not put any chips into the bag.

(ii) If both removed chips are yellow, you put one yellow chip and five blue chips into the bag.

(iii) If one of the removed chips is blue, and the other is not red, you put ten red chips in the bag.

(a) Prove that this process always halts: by following the above steps, you will eventually be left with an empty bag of chips.

[3] You can assume that the grid itself is empty and only the row and column sums are specified in the puzzle that you will be generating.

[4] Based on an exercise from Bradley and Manna. *The Calculus of Computation: Decision Procedures with Applications to Verification.* Springer, 2007.

To do this, let us represent state of the above process at step i of the above process by the tuple $(\#red_i, \#blue_i, \#yellow_i)$. We can prove that this process by terminates by defining a well-founded relation $\succ$ (see definition that follows) over the state. For each step the system takes, from state $(\#red_i, \#blue_i, \#yellow_i)$ to a different state $(\#red_{i+1}, \#blue_{i+1}, \#yellow_{i+1})$, you will prove that:

$$(\#red_i, \#blue_i, \#yellow_i) \succ (\#red_{i+1}, \#blue_{i+1}, \#yellow_{i+1})$$

Definition: A binary predicate $\succ$ over a set S is a well-founded relation iff there does not exist an infinite sequence $s_1, s_2, s_3, \ldots$ of elements in S that each $s_i \succ s_{i+1}$.

Assignment: In your solution:

- In your write-up, define your well-founded relation $\succ$ and informally describe why it proves termination.
- In your Python code, encode $\succ$ in the function `relation` in the file `chips_a.py`. Prove termination using the code in `chips_a.py`. This file contains skeleton code for Python (tested for v2.7/v3.6) and can be found in the bCourses folder Homeworks/HW2 under the menu option Files.

(b) Now consider the following variation of the problem. As before, if the bag contains only one chip, you remove it from the bag. Otherwise, you remove two chips from the bag at random and:

(i) If one of the removed chips is red, and the other is yellow, you return the red chip to the bag.

(ii) If both removed chips are yellow, you put five blue chips into the bag.

(iii) If one of the removed chips is blue, and the other is red, you put ten blue chips in the bag.

(iv) In all other cases, you do not put any chips into the bag.

Assignment: In your solution:

- In your write-up, define a well-founded relation $\succ$ which proves termination of this version of the process and informally describe why it proves termination.
- In the Python code, model the conditions (i), (ii) and (iii) in the functions `case1`, `case2` and `case3` respectively in the file `chips_b.py`. Implement the function `relation` that defines a well-founded relation over the program state and prove that this process too terminates.

  As before, the file `chips_b.py` contains skeleton code for Python (v2.7/v3.5) and can be found in the bCourses folder Homeworks/HW2 under the menu option Files.

(c) Suppose we want to prove that $(A_1 \wedge A_2 \wedge \cdots \wedge A_k) \implies B$, the code in `chips_a.py` and `chips_b.py` uses the following strategy: we assert $A_1, A_2, \ldots, A_k, \neg B$ in the SMT solver and check if this conjunction is satisfiable. If it is unsatisfiable, then the statement is considered "proven."

What pitfalls are associated with this strategy? What steps might one take to mitigate these pitfalls? (Hint: consider what might happen if one of the A_i's is incorrectly encoded.)