CS 511
Formal Methods for High-Assurance Software Engineering
Homework Assignment 03

Out: 18 September 2020
Due: Friday, 25 September 2020, by 11:59 pm

- There are six problems in this assignment. The first 4 problems have to be solved by hand. The last 2 problems are implementation/coding problems.
- You should submit a single "`.pdf`" file to Gradescope, where you include links to your scripts for Problem 5 and Problem 6. Your scripts should be stored in the repository of your Github account.
- For full credit in the homework, you need to complete 4 out of 6 problems in this assignment. Each is worth 4 points. Of course, you may want to try all 6 problems. You will get credit for all extra exercises you do (correctly!).

Problem 1 Do the second exercise on page 19 of Lecture Slides 09, entitled ***Resolution in Propositional Logic*** (there are three exercises on that page). Follow the provided *Hint* closely.

Problem 2 Do the top three exercises on page 27 of Lecture Slides 10, entitled ***Binary Decision Diagrams***. Ignore the last and fourth exercise at the bottom of the same page.

Hint: Make sure you understand the definition of ROBDD(φ). Once you understand this definition, the three exercises are straightforward. To get full credit, make the determination in each of the second exercise and third exercise as efficient as you can. To get efficiency in those two latter exercises, you have to precisely define how a dag should be traversed, starting at the root, assuming that all the edges of ROBDD(φ) are directed downwards to form a dag with a single source node (which is the root at the top).

Problem 3 and Problem 4 in this assignment continue the analysis you started in Problem 3 and Problem 4 in Homework Assignment 02.

Problem 3 Go to the set of Lecture Notes with file name `2020-09-07.fCtC.pdf`, posted on Piazza under the "Resources" webpage. Do part 3 of Exercise 13 on pages 12-13.

Problem 4 This continues the preceding problem. Again, go to the set of Lecture Notes with file name `2020-09-07.fCtC.pdf`. Do part 4 of Exercise 13 on page 13.

Problem 5 Read and understand the handout entitled, "A Scheduling Problem", which is posted with file name `scheduling_problem.pdf` under Resources/Lecture Notes on the Piazza website for the course. At the end of the handout, there is a script in the conventions of Z3. Your task in this problem is to translate this Z3 script into an equivalent Z3Py script.

PS: As mentioned in lecture and again in the handout, there is an error message when you run my Z3 script, which seems to be the result of the last command: `(get-value (A B C D E F))`. If you break this last command into three commands:

```
(get-value (A B)) (get-value (C D)) (get-value (E F))
```

the error message disappears. I will appreciate it, if you find out and let me know what is causing this unexpected error.

Problem 6 A *weighted graph* G is an undirected graph (no cycles, no self-loops, no more than a single edge between two nodes) where every node is assigned a *weight*. An example is shown in Figure 1, which depicts a weighted graph with 5 nodes $\{A, B, C, D, E\}$ each associated with a weight; in this example, the assigned weights are $\{A : 2,\ B : 4,\ C : 4,\ D : 7,\ E : 7\}$.

In this problem we ask you to write a Z3Py script (not a Z3 script) which, given the constraints specifying a weighted graph G, returns a *maximum-weight clique* in G.

We ask you to write the script so that it satisfies the following conditions:

1. Your script can handle an arbitrary weighted graph G with n nodes where $n \geqslant 1$.
2. The weighted graph G is specified by n lists, with one list for every node in G, and with every list on a separate line. The list corresponding to node x has 3 entries: (i) the name x, (ii) the weight assigned to x, and (iii) the list of nodes to which x is adjacent.

   For example, for the weighted graph in Figure 1, its specification is given by the following 5 lists, with one list per line:

   ```
   (1, 2, [ 2, 3, 4, 5 ])
   (2, 4, [ 1, 4 ])
   (3, 4, [ 1, 5 ])
   (4, 7, [ 1, 2, 5 ])
   (5, 7, [ 1, 3, 4 ])
   ```

   where we identify the 5 nodes: A, B, C, D, E, by the five integers: $1, 2, 3, 4, 5$, respectively, to simplify the specification a little.

Warning: Of course, you can solve this problem by writing an appropriate program in Python, or any other general-purpose programming language that you like (Java, C++, OCaml, etc.). However, the point of this problem is *not* to test your ability to think algorithmically and to produce a program that computes a correct solution. Rather, we want you to formally specify the problem by a set of constraints, and then let Z3Py do the hard work for you and figure out how to find a solution that satisfies the constraints.

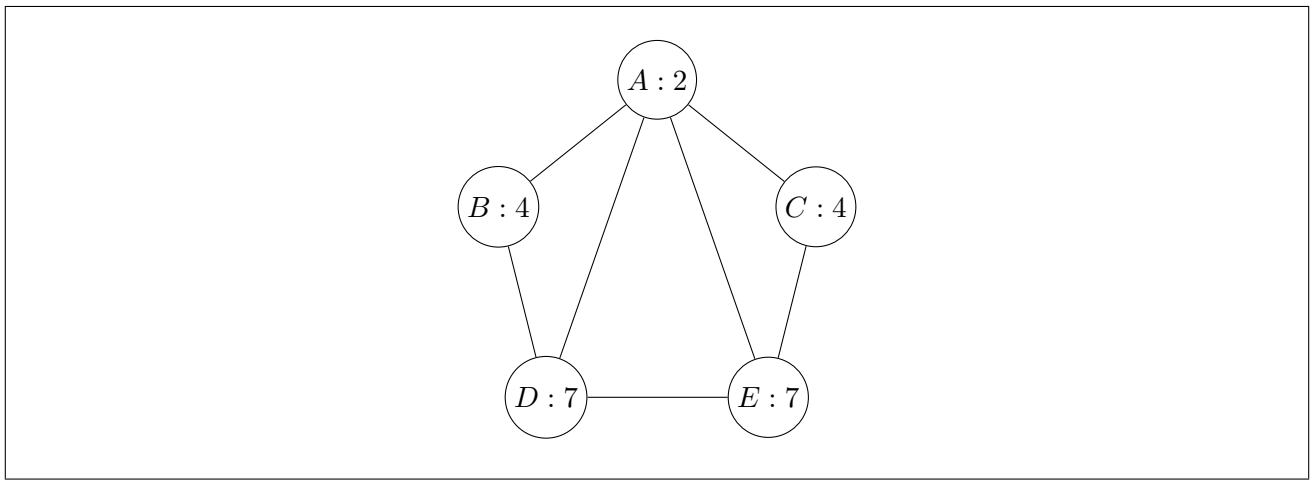

Figure 1: An example of a weighted graph with 5 nodes.