CS 511
Formal Methods for High-Assurance Software Engineering
Homework Assignment 06

Out: 9 October 2020
Due: Friday, 16 October 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!).

For the first two problems you will need to download the file `first-order-definability-exercises.pdf` (here, referred to by FODE), which is posted under Resources/Lecture Notes on the Piazza website for this course.

Problem 1 Do Exercise 7 in FODE. Write the wff's for the first-order definability of gcd in two different ways: φ_{gcd} and φ'_{gcd}, just as it is done for lcm in two different ways in Exercise 6 in FODE.

Problem 2 There are two parts:

1. Do exercise 10 in FODE.
2. Do exercise 11 in FODE.

Problem 3 [LCS, page 164], Exercise 2.4.12. Do parts (a), (b), and (c) only.

Hint: Two of these three parts are valid (*i.e.*, deducible from the empty set of premises using natural deduction), and one is not. For the wff among those three which is not valid, you will need to define a model which does not satisfy it.

Problem 4 Go to the end of Lecture Slides 20 (`HD20.extended-example.pdf` on the Piazza website). Show that "↑" (exponentiation) is first-order definable in the model $(\mathbb{N}, \approx, <, +, \cdot, S, 0)$.

Problem 5 Write a script in the conventions of Z3Py to solve part 1 only (not part 2) of Problem 6 in Homework Assignment 05. We will grade this problem on the basis of what you write in your script; there will be no input and no output to check you on.

In Homework Assignment 05, you had to write a script to show that:

$$(\mathtt{in}_{a,0} \approx \mathtt{in}_{b,0}) \wedge \varphi_a \wedge \varphi_b \ \vdash \ (\mathtt{out}_{a,2} \approx \mathtt{out}_{b,0})$$

The preceding is equivalent to showing:

$$\vdash \ (\mathtt{in}_{a,0} \approx \mathtt{in}_{b,0}) \wedge \varphi_a \wedge \varphi_b \ \rightarrow \ (\mathtt{out}_{a,2} \approx \mathtt{out}_{b,0})$$

Let $\psi \stackrel{\text{def}}{=} (\mathtt{in}_{a,0} \approx \mathtt{in}_{b,0}) \wedge \varphi_a \wedge \varphi_b \ \rightarrow \ (\mathtt{out}_{a,2} \approx \mathtt{out}_{b,0})$. You want to show that, for all possible valuations of the variables in ψ, the wff ψ is satisfied. This is equivalent to saying that for all valuations of the variables, the wff $\neg\psi$ is not satisfied, *i.e.*, $\neg\psi$ is unsatisfiable.

Your Z3Py script should determine that $\neg\psi$ is unsatisfiable.

Problem 6 Write a script in the conventions of Z3Py which reads from an input file 4 positive integers, corresponding to the 4 input variables used in the two programs in Figure 1, listed vertically:

```
in_a
m
in_b
n
```

Each of the 4 values will be on a separate line in the input file that we will use to test your script. Variables `in_a` and `m` are used by program `power`, and variables `in_b` and `n` are used by program `power_new`. Your script should return one Boolean value: *true* or *false*, depending on whether the two programs return the same value in their respective output variables, `out_a` and `out_b`.

The two programs `power` and `power_new` are simple enough that we can determine by inspection that they compute:

$$(\texttt{in\_a})^{\texttt{m+1}} \quad \text{and} \quad (\texttt{in\_b})^{2^{\texttt{n}}}$$

respectively. But, in this problem, we want to be lazy and want Z3Py to do the work for us! Namely, we want Z3Py to tell us when the two output values are equal and when they are not.

Hint: You will find it easier to tackle this problem after finishing Problem 5 above.

```
int power (int in_a, m )
{
    int i, out_a ;
    out_a = in_a ;
    for (i = 0; i < m ; i++)
      out_a = out_a * in_a  ;
    return out_a ;
}
```

```
int power_new (int in_b, n )
{
    int i, out_b ;
    out_b = in_b ;
    for (i = 0; i < n ; i++)
      out_b = out_b * out_b  ;
    return out_b ;
}
```

Figure 1: Two different implementations of the power function, `power` and `power_new`. Differences between the two programs are highlighted.