A GIRL HAS NO NAME
Computer Science 81
Homework 8
3/28/17

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12

1. [1 point] In the lecture slides, we showed how to define various primitive recursive predicates, such as **lesseq** (less-than-or-equal) and propositional connectives.
Using the approach described in the slides, specifically explicit definition, show that the function unequal defined below is primitive recursive.

$$\textbf{unequal}(x, y) = 0' \text{ if } x \neq y \text{ otherwise } 0$$

■

2. [1 point] Transcribe your definition above into Prover9, so that

$$\textbf{unequal}(x, y, z) \text{ gives } z = 0' \text{ if } x \neq y \text{ otherwise } 0$$

For this, and all Prover9 problems, show the result of the tests found in the starter file a08a.p9. Examples of tests for unequal are:

```
%-unequal(0''', 0''', z) #answer(z). % expect 0
%-unequal(0'', 0'''', z) #answer(z). % expect 0'
%-unequal(0'''', 0'', z) #answer(z). % expect 0'
```

These tests are commented out using %. Run the tests one at a time by uncommenting that test, leaving the others commented. Having more than one test active will confuse Prover9 on which test is being run. Using a table as shown below, copy and paste the results of each of your tests into your submission. (You may include a reasonable number of additional tests. These are just the minimum required

Test	**Last line of proof**
-unequal(0”’, 0”’, z) #answer(z). % expect 0	138 $F # answer(0). [ur(75,a,137,a),unit_del(a,134)].
-unequal(0”, 0””, z) #answer(z). % expect 0’	183 $F # answer(0’). [ur(74,a,181,a),unit_del(a,151)].
-unequal(0””, 0”, z) #answer(z). % expect 0’	182 $F # answer(0’). [ur(74,a,180,a),unit_del(a,151)].

■

3. [1 point] Define **less** so that **less**(x, y) means **lesseq**(x, y) and **unequal**(x, y).

■

4. [1 point] Transcribe your definition above into Prover9 and verify the tests in the same manner as in problem 2.

Note: In some problems you may want to employ an auxiliary (helper) function.

■

5. [3 points] Show that the function **rem** (remainder) is primitive recursive:

$$\boldsymbol{rem}(x, y) = 0 \text{ if } y = 0$$

$$\boldsymbol{rem}(x, y) = \text{ the remainder of dividing natural number } x \text{ by natural number } y \text{ if } y \neq 0$$

The reason for having a value if $y = 0$ is so that the function is total, as required of every primitive recursive function. (Advice: Consider using x as the induction variable, rather than y. Deal with the 0 exclusion separately.)

■

6. [3 points] Transcribe your definition of **rem** into logic for Prover9. To make testing more readable, we provide in the starter file a conversion predicate code that converts decimal numerals from 0 to 20 into the unary numerals used in the theory. An example of using code in testing your **rem** function is shown here:

```
test_rem(x,y,z)<-code(x,xc)&code(y,yc)&rem(xc,yc,zc)&code(z,zc).

-test_rem(20, 7, z) #answer(z). % expect 6
```

The definitions of code look like this:

```
        code( 0, 0).
        code( 1, 0').
        code( 2, 0'').
        code( 3, 0''').
              . . .
        code(20, 0'''''''''''''''''''').
```

These definitions can be used to encode, e.g. code(3, x) or decode, e.g. code(x, 0'''). They are not part of the primitive recursive formalism. They are just there for testing. Verify the tests in the same manner as in problem 2.

■

7. [3 points] Show that the function **quot** (quotient) is primitive recursive:
$\boldsymbol{quot}(x, y) = 0$ if $y = 0$
$\boldsymbol{quot}(x, y)$ = the whole-number part of the quotient of dividing natural number x by natural number y if $y \neq 0$

The reason for having a value if y = 0 is the same as in problem 1.

■

8. [3 points] Transcribe your definition of **quot** into logic for Prover9 and verify the tests in the same manner as in problem 2.

■

9. [6 points] Show that the 1-ary predicate **is_prime**, which returns 0' if its argument is prime and 0 otherwise, is primitive recursive. Here 0 and 1 are considered not prime. Any other number is prime iff it has no divisors other than 1 and the number itself.

■

10. [5 points] Transcribe your definition of **is_prime** into logic for Prover9 and show the result of the tests found in the starter file.

■

11. [3 points] The i^{th} triangular number is defined by primitive recursion thus:

$$\textbf{triangle}(0) = 0\textbf{triangle}(y') = y' + \textbf{triangle}(y)$$

Use the μ operator, as defined in the notes, to give a formal definition of the partial function that is the *inverse* of the function **triangle**. That is **invtriangle**(z) = the value of y such that **triangle**$(y) = z$. If there is no such y, then **invtriangle**(z) is undefined and can diverge. Show that your definition works in Prover9.

■

12. [20 points] Design a Turing machine that produces the **square** of the number on its tape. The input and the result are both encoded as a sequence of n contiguous 1s. For example, if the input is 1111 then the result is 1111111111111111 ($4^2 = 16$). Use the convention that the head is initially at the leftmost digit of the input. Thus if the input is empty, the head will be over a blank, and the result will also be empty (since 0 squared is 0).

a. [7 points] Describe in prose how your machine works.

b. [8 points]List all of the rules for your machine, in the Prover9 framework to be provided. Cut and paste your entire p9 file into your submission so that we can run it.

c. [5 points] Provide a trace of the machine running on the input 111, either from Prover9 or from Prolog. Paste in output from your simulation. (As an example, the output from the machine BB4 provided in the sample is given here. After preliminaries, starting at line 50 or so, the clauses in the proof are seen to mimic the states of the Turing machine, although the order may be slightly different due to choices Prover9 makes.

■