COT 4210 DAILY PROOF #6 SOLUTIONS SUMMER 2012

1) A TM that decides ALL_{DFA} works as follows:

a) Verify that A is a valid DFA. (If not, reject.)
b) Run a breadth first search from the start state and mark all reachable states from the start state.
c) If all reachable states are accept states, accept, otherwise, reject.

An alternate solution makes use of the DFA minimization algorithm. Both of these are guaranteed to halt and both produce the correct answer. (If all strings are accepted by A, it should have no reachable state that is a non-accept state.)

2) A TM that decides $INFINITE_{DFA}$ works as follows:

a) Verify that A is a valid DFA. (If not, reject.)
b) Let n equal that number of states in A.
c) Run A on every string with a length in between n and 2n, inclusive.
d) If any of these strings are accepted by A, accept, otherwise, reject.

The rationale behind this TM is that if a DFA accepts an infinite number of strings, it must accept at least one string where its path in the DFA contains a cycle. No string that contains ONLY one cycle is longer than 2n. All strings of length n or longer contain at least one cycle.

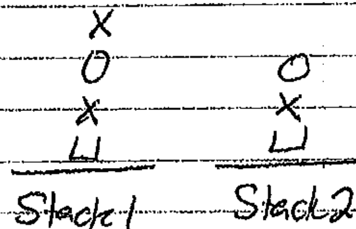
2con't) Namely, any string $xyz \in L$ where y refers to a substring that starts and ends in the same state of A, and where x and z contain no cycles has $|xz| < n$ and must have $n \leq |xy^iz| \leq 2n$ for some $i \in \mathbb{Z}^+$ because $|y| \leq n$. By testing all strings with lengths in between n and $2n$, inclusive, we guarantee that if a string is in the language described by A that contains a loop when read in, we will detect it by trying these strings. (Maybe not that particular string, but a version of it with a different number of loops.)

3) a) $L = \{0^n 1^n 2^n \mid n \geq 0\}$ can be accepted by a 2-PDA. When you read in each 0, push a 0 onto both stacks. When you read in each 1, pop of one 0 from the first stack. When you read in each 2, pop off one 0 from the second stack. Use extra markers so we can detect an empty stack. Accept only if each stack propelly empties out.

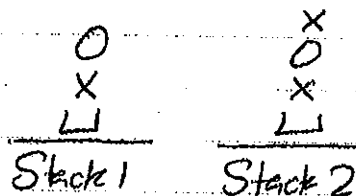
b) First, we must specify where in our 2-PDA the "input" would be "stored". Stack 1 will store everything to the left of the tape head. Stack 2 will store everything to the right of the tape head that has previously been "read in". Everything not yet "read in" will remain in our Input string to be processed. Reading and writing a symbol can be implemented by popping and pushing a stack symbol. To move right, we pop a symbol from Stack 2 if it exists, or read in the next input Symbol and push the appropriate value onto Stack 1. To move left, we transfer a symbol from Stack 1 to Stack 2.

Consider the ... [illegible]

Consider the status/configuration $\sqcup \times O \times q_5 O \times \sqcup$ from question 1c. In a 2-PDA our stacks would be as follows:



3 b cont) In the ensuing transition, we read the O (top of stack 2) and move left, writing the O back. For our 2-PDA, we will move left by popping X from Stack 1 and pushing it onto Stack 2:



Once the entire input has been processed by the PDA, we just continue computation by <u>ONLY</u> reading from the stacks and pushing and popping items as needed. At any point in time, Stack 1 represents the symbols on the TM tape to the left on the tape head, and Stack 2 represents the items to the right of the tape head with the top of this stack storing the next symbol to be processed.

4) Perfect is decidable. Given an input n, check to see if each integer from 1 to n-1 divides evenly into n or not. If it does, add this integer to an accumulator variable sum. Then, if sum equals n, accept, otherwise reject. Here is a C function that implements the algorithm. While it's relatively slow, it does indeed work, and on a Turing Machine would be guaranteed to halt and return the correct answer. (In C, of course, we would get an overflow error if tested with a value outside the range of an integer.)

```
int perfect(int n) {

    int i, sum = 0;

    for (i=1; i<n; i++)
        if (n%i == 0)
            sum += i;

    return (sum == n);
}
```

It is unknown whether or not it's regular. That's because right now, there are only a finitely many known Perfect Numbers and it is unknown whether or not there are an infinite number of them. Thus, if there are only a finite number, then it would definitely be a regular language. On the other hand, if there are a infinite number of them, it would likely not be regular, since there's no fixed constant that would bound the distance between successive perfect numbers.