CSE 105:Introduction to the Theory of Computation, Fall 1999

Homework 3

For many of the problems there are multiple possible solutions. The answers presented below outline one such solution for each problem. They may not contain sufficient detail to receive full credit. In some of the diagrams, E, represents an empty transition.

PROBLEM 1 a)

To prove a 2-PDA is more powerful than a 1-PDA, it must be shown that a 2-PDA can do anything a 1-PDA can do, and more. It is clear that a 2-PDA can simulate a 1-PDA, so it need only be shown that there is something a 2-PDA can do that a 1-PDA cannot.

A 1-PDA cannot accept the language $\{a^n b^n c^n | n > o\}$. A 2-PDA can, by placing all a's on stack 1, all b's on stack 2, and then removing an a for each c read following the b's. The language is accepted if there are no a's remaining, and all the input has been read. Therefore, a 2-PDA is more powerful than a 1-PDA.

PROBLEM 1 b)

If a 2-PDA can be used to simulate a 3-PDA, it is clear that a 3-PDA is no more powerful than a Turing machine, as a Turing machine can simulate a 3-PDA. (theorem 3.8: "Every multitape TM has an equivalent single tape Turing machine." It is easy to see that a multi-tape turing machine can simulate a 3-PDA.) Thus, our proof will consist of showing the 2-PDA can simulate a Turing machine.

1. the first tape is used as a stack representing the symbols before the tape head. Thus, the first stack starts empty.
2. the second tape is used as a stack representing the symbols after the tape head. Thus, the second stack starts with all the input.
3. Left and Right movement can be duplicated in a 2-PDA model. For right movement, the top symbol on the second stack is popped, and pushed onto the second stack. The opposite is done for left movement.
4. to write, we pop of the symbol, and push on the replacement. To move.

Thus, it can be seen that a 2-PDA can simulate a turing machine. Therefore, given the logic listed above, the 3-PDA is no more powerful than a 2-PDA.

PROBLEM 2 a)

For a union, A(x) or B(x) must be true. By running x on both machines, and accepting if either accepts, and rejecting if both rejects, we have replicated union. For decidable languages, both A and B will accept or reject. Thus, union is closed for decidable languages.

PROBLEM 2 d)

If we run the machine A on x, and reject if A accepts, and accept if A rejects, we have achieved the complement. As A must accept or reject if it is decidable, complement is closed for decidable languages.

PROBLEM 2 e)

For intersection, if A(x) and B(x) both accept, we accept, if one rejects, we reject. As these are the only possibilities, intersection is closed for decidable languages.

PROBLEM 3 b)

For concatentation, we can non-deterministcally split the input into two, one for the first language, and one for the other. As this non-deterministic split basically gaurantees accuracy through simultaneously simulating all possible splits, we can accept if both machines accept, reject if both reject, and not halt if one does not halt. Thus, concatentation is closed for Turing recognizable languages.

PROBLEM 3 d)

By running both unioned languages on machines simultaneously, we can ensure that both accept, reject, or loop as would be expected if run singly. As union will accept if both accept, reject if one rejects, and loop if both loop, Turing-recognizable languages are closed under intersection.

PROBLEM 4

1. Search from the right to find the first symbol that matches the symbol at the beginning of the input string. If not found, reject
2. mark this sybol as well as the first symbol in the input string.
3. check if both the character to the right of the first mark matches the character to the left of the second mark. If so, mark them both off and repreat until they don't match or if a mark is to the right and left of the two positions.
4. If they don't match, unmark all the symbols and start at the 1, on the condition that each time we go back to 1, we search for the next occurence of the symbol matching the first symbol in the input string, skipping over those that we have already performed steps 1-3 on.
5. if step 3 was succesful then we have found one configuration of $\{ww^R\}$. We then must repeat steps 1-4 on the characters following the last symbol of our $\{w^R\}$.
6. if we find another such suitable configuration for $\{uu^R\}$, then we are done and we accept. If we don't, we start over and repeat until all possible combinations of $\{ww^R\}$ and $\{uu^R\}$ are exhausted. At this point if no combination has been accepted, we reject.

PROBLEM 5

Let A be a CFL. From the definition we know that A has a CFG, G generating it. We also know that any CFG can be represented by a 1-PDA. We know that with such grammers, we ahve a set of accept states in which a PDA representing the CFG will accept. For all other cases, we reject. We stated in HW problem 1 b) that Turing machines are more powerful than 1-PDAs and thus the contect-free languages are contained in the Turing-recognizable languages. Since CFGs and PDAs always accept or reject, we know that these languages could never cuase a Turing machine to loop indefinitely. Therefore, the contect free languages are contained in the Turing-decidable languages.