
Problem 3.9. Let a k-PDA be a pushdown automaton that has k stacks. Thus a 0-PDA is an NFA and a 1-PDA is a conventional PDA. You already know that 1-PDAs are more powerful (recognize a larger class of languages) than 0-PDAs.

Part a. Show that 2-PDAs are more powerful than 1-PDAs.

Proof. A 2-PDA can simulate any 1-PDA by using only one of it's stacks. Therefore, 2-PDAs can recognize all the languages that are recognized by 1-PDAs. Additionally, we show that 2-PDAs can also recognize more languages, which are not recognized by 1-PDAs. For example, the language $A = \{a^n b^n c^n \mid n \geq 0\}$. The language A is not a context free language[1], but we can construct a 2-PDA to recognize A as follows:

1. Read and push a's on both stacks until a b is read.
2. Once a b is read, match it with an a by popping an a from the first stack. Keep reading and matching any subsequent b's. Reject if the number of a's and b's are not equal.
3. Next, match c's with a's on the second stack. Accept if the number of a's and c's are equal, reject otherwise.
4. In above steps, also make sure the input string is in $a^*b^*c^*$.

□

Part b. Show that 3-PDAs are not more powerful than 2-PDAs.

Proof. To show that 3-PDAs are not more powerful than 2-PDAs, we show how two stacks can be used to simulate a Turing machine tape.

1. Consider an arrangement of the two stacks, where the two stacks are placed horizontally next to each other. The right hand side (RHS) stack grows to the right, whereas the left hand side (LHS) stack grows to the left.
2. Mark the end of stacks by pushing special symbols before any input is processed. Let π and λ be two distinct symbols, which are not part of the input language:
   (a) Push π on the LHS stack.
   (b) Push λ on the RHS stack.
3. The read/write head is always at the top of the RHS stack. For now, assume that the input symbols are available on the RHS stack. Later we show how to load input string on the RHS stack.
   (a) To read, pop from the RHS stack.

[1] Example 2.36, Chapter 2.

(b) To write, push on the RHS stack.

4. Move left and right are simulated by push and pop operations on the two stacks:

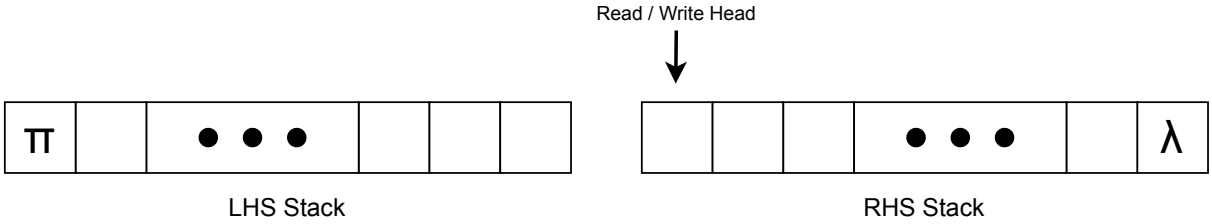
   (a) To move right, pop a symbol from the RHS stack and push it to the LHS stack.

   (b) To move left, pop a symbol from the LHS stack and push it to the RHS stack.

5. Special cases for the left-most and right-most end of the tape:

   (a) If π is popped from the RHS, then this indicates that the head is at the left-most position of the tape. In this case, push π back on the LHS stack and read again.

   (b) If λ is popped from the RHS, then this indicates that the head is at the right-most position of the tap. To simulate infinite tape on the right, in this case the input symbol is assumed to be the blank and the λ is pushed back on the RHS stack. The written symbol is pushed on the LHS stack.



Configuration of the two stacks. RHS stack grows to the right, and the LHS stack grows to the left. The ends of LHS and RHS stacks are marked by the symbols π and λ respectively.

Load input string on the RHS stack as follows:

1. Push π on the LHS stack, and λ on the RHS stack to mark their ends.

2. Repeatedly, read a symbol from the input, push it on the LHS stack and non-deterministically go to step 3.

3. Pop symbols from LHS stack and push them on the RHS stack. Once the LHS stack is emptied, non-deterministically go to step 4.

4. Process the input string on the RHS stack as discussed earlier.

□