Rusu Wu

11694764

HW1

CPTS_516

1.

(1)We can prove that L1 // L2 can be accepted by a FA called M, and then L1 // L2 is a regular language. Defining a FA called N1 can recognize L1, and a FA called N2 can recognize L2.

Each time M reads a letter form *w*, M guesses whether the letter comes from N1 or N2. If the letter comes form N1, then M simulates N1. On reverse, if M guesses that the letter comes from N2, then it simulates N2. Once M finishes reading, than M says YES . If both N1 and N2 finish reading and say YES on the guesses, L1//L2 is a regular language.

(2) We can prove that there is a PDA called M can recognize L1 // L2, such that L1// L2 is context-free language. In the case that L1 is a regular language and L2 is a context-free language, so defining a FA called N1 to recognize L1 and a PDA called N2 to recognize L2.

Each time M reads an input letter form *w*, M guesses whether the letter is form L1 or L2. If M guesses the letter comes from N1, than it uses the temporary memory to simulate N1. On reverse, if M guesses the letter is form N2, M uses the stack to simulate N2. After M reads the last letter of *w* and says YES, if both N1 and N2 finish reading and say YES on the guesses, L1//L2 is a context-free language.

2.

(1) If there is a FA can recognize h(L), then we can prove that h(L) is a regular language, and we can define the FA is M. Also, there is a FA called N can recognize L since L is a regular language.

Next, replace all the symbols of h(L) to L, such that,

N = {Q1, Σ, δ1, q1, F1}

M = {h(Q1), Σ, h(δ1), h(q1), h(F1)}

Since M is a regular expression for h(L).

Such that h(L) is regular language.

(2) Base on the previous answer, if there is a non-regular language L, M only can recognize some of h(L), so that h(L) is a regular language for some h.

3.

There is a FA M1 recognizing w_1 w, and there is a FA M2 recognizing w_2 w. Also, there is a FA M recognizing *w* from L1♥ L2.

Before M read the input from *w*, M guesses all the w_1 from M1 and starts simulating M1. After M read all the guesses of w_1 and start to read *w*, M1 is also going to read the *w* from the $w_1 w$. Right at this moment, M2 starts to read *w* from $w\, w_2$, and M starts to simulate M2. When all the *w* has been read by M M1and M2, M2 keep reading w_2 form $w\, w_2$, and M guesses all the w_2 and keep simulating. After M reads all guesses of w_2 and say YES to $w_1 w\, w_2$, if M1 says YES to w_1 w and also M2 says YES to $w\, w_2$, we can conclude that L1♥ L2 is a regular language.

4.

A B-bounded PDA can be seen as a FA, so that it only can recognize regular language.

However, the language L= $\{0^n 1^n : n \geq 1\}$ is not regular language. We can proof it by using Pumping Lemma. Assume that L= $\{0^n 1^n : n \geq 1\}$ is a regular language, so it satisfies pumping property. There is a pumping length *p* and we pick the string s = $0^p 1^p$. We can consider for any s = xyz, with |y|>0 and |xy| ⩽ p. For x= 0^m , y = 0^n , z = $0^r 1^p$ with m+n+r=p, j>0, we pick i=2, while $xy^i z$ =xyyz= $0^m\ 0^n\ 0^n\ 0^r 1^p$, where $xy^i z$ is not in L= $\{0^n 1^n : n \geq 1\}$. Therefore, L is not a regular language.

Hence, the language $\{0^n 1^n : n \geq 1\}$ can not be accepted by a B-bounded PDA for any B.

5.

If PDA M has an infinite execution, it means that M runs the same finite section repeatedly. For example, if there is a state p that the PDA has reached, the PDA will reach the state p over and over again later. Therefore, if there is a loop in M, and the starting state of the loop is state p. Also, the stating popping word to the stack with the loop is a special word *a*. Such that, Every time M starts the loop, the top word of the stack should be *a*. Therefore, we only need to make sure that there is a input word that can reach the state p, or make sure the top word of the stack is *a*, then M will execute the loop infinitely.

6. The fair walk refers to there are infinitely many prefixes of the walk on which the number of red arcs, green arcs and blue are same. In order to decide whether a ω-walk is fair walk, we can define 3 PDAs called M1, M2, M3. All of M1, M2 and M3 read the same prefix of ω, and the number of prefix symbol i should satisfy with $i \bmod(6)=0$.

If M1 reads Red color arc, push a R to the sack. At the same time, M2 pop a G from its stack.

If M2 reads Green color arc, push a G to the sack. At the same time, M3 pop a B from its stack.

If M3 reads Blue color arc, push a B to the sack. At the same time, M1 pop a R from its stack.

After M1, M2 and M3 finish the reading of ω, if the stacks of M1, M2 and M3 and say YES to the empty of its own stack, such that the ω-walk is fair walk.

7.

A finite automata can be defined as M = {Q, Σ, δ, q0, F}. In order to add timers to the definition of FA, the transaction state δ should be under the constraint of clock. Such that the timer finite automata can be defined as a tuple M = {Q, Σ, δ_T, q0, F, T}, in which $\delta_T = \delta \times T$, and T is a finite set of clocks.

8.

We can define one PDA for each kid, we call the PDAs are M and N. Also, there is a string *w1 = c, w2=d.* Since the two kids could not hold the same apple at the same time, we need to set restriction; w1 and w2 can not be read by the same NFA at the same time.

If A pick c, M reads w1, push c into M's stack;

If A pick d, M reads w2, push d into M's stack;

If B pick c, N reads w1, push c into N's stack;

If B pick d, N reads w2, push d into N's stack;

If A or B put c back on the table, M or N pop c;

If A or B put d back on the table, M or N pop d;

Excepting from the initial state, w1 or w2 only can be read when c or d being popped from M or N.

The existing events are :

1) A idles: M is free;

2) B idles: N is free;

3) A picks up c: M is reading read w1 and pushing c;

4) A picks up d: M is reading read w2 and pushing d;

5) B picks up c: N is reading w1and pushing c;

6) B picks up d: N is reading w2 and pushing d;